Efficient Error Estimation for High-Level Design Space Exploration of Approximate Computing Systems

Marzieh Vaeztourshizi and Massoud Pedram, *Fellow, IEEE*

***Abstract*— This article presents an error estimation technique for a data-flow graph (DFG) representation of an approximate computing (AC) circuit. The technique, which may be used during the high-level design of digital circuits, estimates error metrics for the outputs of the approximate circuit. The proposed technique receives as an input, output error characterizations of arithmetic modules in a high-level library. The error modeling of a library module is accomplished by dividing the input (operand) ranges into intervals and then characterizing the output error for different combinations of these input intervals. Subsequently, the error for each combination is stored in a lookup table (LUT). The module error models are integrated into a design space exploration (DSE) framework to evaluate different combinations of exact and approximate realizations of various operations in the DFG. The DSE, which performs error calculation and propagation from inputs to outputs of the target DFG, may be used to explore trade-offs between the output error and other design metrics of the approximate circuit, e.g., energy efficiency. The efficacy of the proposed method is assessed for three image processing benchmarks. Results for these benchmarks demonstrate that the framework can efficiently generate the Pareto frontier (PF) in the trade-off space of accuracy versus energy efficiency for the targeted benchmarks. Compared to a purely simulation-based exploration, the proposed technique provides an average of 92× speed improvement.**

***Index Terms*— Approximate circuits, data-flow graphs (DFGs), design space exploration (DSE), error estimation, look up tables (LUTs), Pareto frontier (PF).**

I. Introduction

POWER efficiency is a critical requirement in the design of digital systems, especially those that are included in battery-powered embedded and mobile systems. Running compute-intensive, data-hungry tasks such as machine learning (ML) applications on these systems necessitates very high energy efficiency to be of practical interest [1].

Many applications (including ML ones) have intrinsic tolerance to some degree of inaccuracy in their outputs. One reason for this error tolerance is the fact that the users of these applications, i.e., humans, may not be able to perceive small imperfections in the quality of the outputs (e.g., the human visual system does not detect infrequent drop of video frames) [2]. This application's tolerance to error opens an avenue for using the approximate computing (AC) approach to reduce energy and/or power consumption. In this design approach, the exactness of the computations can be abandoned in favor of improving energy efficiency. The extent of employing the AC approach is determined by a minimum quality of service (QoS) requirement.

AC can be applied at different levels of design abstraction. At the hardware level, one may employ functional approximation to reduce the delay and power consumption of the circuit, where an exact operation may be substituted with an alternative simplified one. The functional approximation technique may very well be applied to the design of arithmetic units (e.g., adders and multipliers) which consume a large portion of the computing system power [3]. Alternatively, one may lower the supply voltage (voltage overscaling) of the exact circuit of the function [4].

At this level of design, different degrees of approximation may be used. Examples include the use of a different number of approximate least significant bits (LSBs) in a multibit adder or applying different voltage overscaling levels in a digital design. Different accuracy levels lead to different energy dissipation values, where a higher degree of approximation (DOA) (lower accuracy level) tends to result in higher energy efficiency. Indeed, one may characterize and store a set of (accuracy, energy) value pairs for each arithmetic operation in a high-level design library and use them in a design consisting of these units to obtain diverse combinations of these two metrics for the whole design. There is, therefore, a compelling reason to develop electronic design automation of approximate circuits, which has been the focus of much attention in the past few years (e.g., [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]).

The automatic mapping of a high-level description of an input application to the register-transfer level (RTL) representation under predefined circuit constraints (e.g., latency and energy consumption) is called *high-level synthesis* (HLS). When using approximate units in this mapping to obtain an approximate implementation of the target application, the mapping is called *approximate high-level synthesis* (AHLS). Similarly, the automatic generation of approximate circuits at the gate-level is called *approximate logic synthesis* (ALS) (e.g., [5], [6]).

Manuscript received 3 August 2022; revised 13 November 2022; accepted 23 December 2022. Date of publication 24 May 2023; date of current version 28 June 2023. This work was supported in part by the Software Hardware Foundations of the National Science Foundation under Award 1901440. *(Corresponding author: Marzieh Vaeztourshizi.)*

The authors are with the Electrical and Computer Engineering Department, University of Southern California, Los Angeles, CA 90089 USA (e-mail: vaeztour@usc.edu; pedram@usc.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2023.3273478.

Digital Object Identifier 10.1109/TVLSI.2023.3273478

At a higher-level design abstraction, an input application is typically represented by a data-flow graph (DFG) that is a directed graph with the nodes representing the operations (e.g., arithmetic units) and edges capturing the data dependencies between the operations. For a given DFG of an input application, one may replace some exact arithmetic operations with their approximate counterparts, resulting in an *approximate DFG* (ADFG). Depending on the choice of exact versus approximate modules (with different accuracy levels) for the DFG nodes, many designs with different energy and accuracy levels may be realized. Obviously, a subset of these designs constitutes a Pareto-optimal curve. The designer of an approximate circuit should select a "noninferior" design from among the set of designs that reside on the said Pareto frontier (PF) in the trade-off space of energy consumption and QoS.

To determine the Pareto-optimal curve, the whole design space should be explored. In fact, the search operation, which is called *design space exploration* (DSE), forms an intractable problem [7]. Many heuristic techniques for exploring the design space of approximate circuits have been proposed in the literature ([8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]). These approaches are usually iterative, where a simulation engine is invoked on a set of input test vectors to evaluate the quality of each approximate design visited in the design space. To obtain the other design metrics (e.g., energy consumption), other calculators (which are typically integrated into the synthesis tool) may be invoked. In fact, a key task in the DSE framework of ADFGs is the process of determining the accuracy/energy of the designs using a simulation/synthesis tool [8]. To reduce the computation time, one should resort to efficient accuracy estimation of ADFGs by employing less time-consuming techniques. More specifically, one should replace the brute-force circuit or logic simulation engines with estimation models when possible.

In this work, we present an efficient technique for estimating the quality of the output of an ADFG. The technique relies on output error estimation for different library modules by using interval-based (piecewise) models. A set of look up tables (LUTs) are employed to store the error metrics of the library units. These tables are then employed to quickly evaluate the output quality of the ADFGs visited in a DSE framework. Key contributions of this work are the following[1]:

1) Present an efficient approach to estimate the error at the output of an ADFG.
2) Incorporate the error model in a DSE framework to extract the PF of the trade-off space of energy consumption and QoS.
3) Study the efficacy of two error metrics and two error estimation techniques on the quality of the extracted PFs of the ADFGs.

The remainder of the article is organized as follows. Section II presents a brief review of the error metric definitions. The works related to error estimation techniques at the application/DFG level and the DSE algorithms employed in approximate high-level designs are reviewed in Section III. The proposed error model is described in Section IV, and Section V describes the integration of the model in a DSE framework. The simulation setup and results are discussed in Section VI. Finally, the article is concluded in Section VII.

[1]This work is the extension of the work described in [19] and [21].

II. Error Metric Definitions

The error characteristic of an approximate design may be determined by using simulations where the outputs of the approximate and exact implementations (EMs) of the design are compared with each other for the same inputs. Ideally, all combinations of input values should be used in the error characterization process. For designs with large input dimensions, the characterization is obtained by choosing and applying a random subset of input vectors (X).

Next, we describe error metrics that are used in this work. *Error distance* (ED), is defined by us as the numerical difference between the approximate ($S'[i]$) and exact ($S[i]$) output values generated by the approximate and EMs of the design for the ith input vector of X

$$\mathrm{ED}[i] = S'[i] - S[i]. \tag{1}$$

Having the set of ED[i] values, one may determine the distribution of the error at the output defined using the probability mass function (PMF). This function represents the frequency of error for each possible value, which can be used to obtain the desired error characteristics. Utilizing the set of ED[i] values, *mean error distance* (MED) is obtained from

$$\mathrm{MED} = \frac{1}{T} \times \sum_{i=0}^{T-1} \mathrm{ED}[i] \tag{2}$$

where T is the number of applied test vectors. If instead of the ED[i] values, the absolute of the ED[i] values are averaged, *mean absolute error distance* (MAED) is obtained.

Normalizing the ith ED with respect to the ith exact output value defines relative error distance (RED) as

$$\mathrm{RED}[i] = \frac{\mathrm{ED}[i]}{S[i]} \tag{3}$$

and by averaging the RED[i] values, one obtains *mean relative error distance* (MRED). Also, one may use the absolute value of the $|\mathrm{RED}[i]|$ to obtain *mean absolute relative error distance* (MARED).

For applications in the domain of image processing, *peak signal-to-noise ratio* (PSNR) could be of more interest as an accuracy metric for output quality. The metric, which is meaningful at the output of the whole circuit, is defined as

$$\mathrm{PSNR} = 10 \times \log_{10}\left(\frac{\max_{\mathrm{pixel}}^2}{\mathrm{MSED}}\right) \tag{4}$$

where $\max_{\mathrm{pixel}}$ is the maximum value for pixels of an image and MSED denotes the *mean-squared error distance* calculated as follows:

$$\mathrm{MSED} = \frac{1}{T} \times \sum_{i=0}^{T-1} \left(\mathrm{ED}^2[i]\right). \tag{5}$$

Rewriting (1) and (3), the approximate value of the ith output is determined using ED[i] or RED[i] as

$$S'[i] = S[i] + \mathrm{ED}[i] \tag{6}$$

$$S'[i] = (1 + \mathrm{RED}[i]) \times S[i]. \tag{7}$$

Obviously, depending on the sign of ED[i] (or RED[i]), the approximate value can be larger or smaller than the exact value. These equations will be used in Section IV when we explain the proposed error model.

III. Related Work

In this section, first, we review some of the relevant work in the area of error estimation techniques based on computational models at the DFG level. Next, the related works on the heuristic approaches used for approximate high-level designs are reviewed.

A. Error Estimation Techniques at the DFG Level

There are many works that take analytical approaches to model the error at the operation level (i.e., output error of an approximate arithmetic unit) (see e.g., [23] and [24]). Since these works normally do not deal with the error propagation when the module is used in an ADFG, they are not considered here. The proposed error analysis techniques at the DFG level may be categorized into two groups for estimating the PMF of the output error and estimating the output error metrics.

1) Estimation of PMF of the Output Error: The first category estimates the PMF of the error at the output of an ADFG from which the desired error metrics such as MED are obtained. In [25] and [26], modified interval arithmetic (IA) and affine arithmetic (AA) were used to represent the PMF of error at the output of an approximate module. A set of propagation rules were defined to obtain the PMFs of the exact, approximate, and error values at the output of an operation using the PMFs of the inputs, error at the inputs, and generated (intrinsic) error of the module. In [27], using the convolution operation, the PMF of the error at the output of a multibit operation simplified with approximate full adder units was defined. To estimate the PMF at the output of an ADFG, the error at the operation level was propagated in the DFG in a topological order.

A compiler-driven method with a set of heuristic rules to estimate the PMF of the error at the output of an ADFG was proposed in [28]. In this work, only unsigned adders were approximated and the PMF was calculated using convolution operations. In [29], the authors suggested a semi-analytical error model for obtaining the error PMF at the output of an ADFG using input statistics and the error tables storing the error generated by approximate modules with approximated LSBs. This model was used in the DSE framework of [19].

2) Estimation of Output Error Metrics: In the second category, the required error metrics are estimated without obtaining the full PMF of the error. In most cases, error metrics may provide enough information about the output quality and hence sufficing us to determine the full PMF of the error. This is true when evaluating the output quality in a DSE framework where different ADFGs are considered [30].

In the works of [10] and [14], the qualities of the explored designs in a DSE framework were estimated using pretrained supervised ML models (e.g., random forest). The models were DFG-dependent, requiring finding a separate ML model for each new DFG. In [31], based on the defined error sensitivity concept, the variance of the error at the output of an ADFG was modeled. The error model, then, was employed in an optimization problem which was solved using the ILP technique. SABER [7], which was an extension to the work of [31], focused on fine-grain modeling of the error variance in the case of truncation-based approximation.

In [32], to estimate the error metrics at the output of an approximate design, a LUT-based method was proposed. For each module in the library, two types of LUTs were generated. The first one contained the standard deviations (σ) of the output of a module based on the σ parameter of the operands (with normal distribution with mean $\mu = 0$), while the second one contained the error metrics generated by the approximate module itself based on the σ parameter of the operands. In the propagation of the error in an ADFG, first, the σ values of all the internal signals were obtained from the first LUTs through topological traversal of the nodes. Then, the intrinsic error metrics of all the nodes were obtained from the second LUTs. For each node, a regression model obtained the overall error metric of the module output using the intrinsic error metrics of an operation, as well as the operand errors.

In this work, as opposed to [32], a single precharacterization step is performed to obtain the overall error at the output of a module. More specifically, our proposed technique does not need to individually obtain the error generated from the approximate module itself and then use a separate model to calculate the final output error (e.g., a regression model in [32]). Also, in [32], only approximate adders were modeled.

B. DSE Algorithms for Optimization of Approximate Circuits

DSE algorithms may be used to find better approximate designs. For this purpose, a number of algorithms have been proposed in the literature, which may be categorized into two groups of local search and evolutionary approaches. Next, we review these works based on their categories.

1) Local Search Approaches: In the DSE algorithms of this group, in each iteration, the design space is expanded from one design point (called *a parent point*) to other points in the space. From the parent point, either one or multiple new design points in the design space are generated. Heuristic techniques are mostly used to choose this parent design point for the subsequent iteration.

ABACUS [9] used a linear ranking heuristic in each iteration to choose the parent design of the next iteration. The next parent design was chosen from the generated new designs of the current iteration obtained by applying some transformation operators to the current parent design. The autoAx tool in [10] employed ML models to choose a pseudo-PF (PPF) set of the design space by using a modified stochastic hill-climbing algorithm. In the end, simulation and synthesis tools were invoked on the designs in PPF to extract the final PF set of the design space among the PPF designs. The work in [11] proposed a method in which different approximate techniques were applied at different design abstraction levels. The design was approximated sequentially at each level (including software, HLS, RTL, and gate levels). Based on a linear cost function, a greedy method, which chose the parent design for the next iteration, was performed at each level.

Some works have considered the design space as a search tree where the tree nodes correspond to different approximate versions of the original design, with the original design located at the root of the tree and exploring the space via branch and bound [12], [13] or greedy [8], [14] algorithms.

2) Evolutionary Approaches: The second category of DSE algorithms for optimizing approximate circuits is based on expanding the design space from multiple design points in each iteration of the algorithm. In each iteration, in contrast to the algorithms of the first category, a set of parent design points in the design space (called *a population*) is used to generate a set of new approximate designs.

The ABACUS tool [9] was improved in [15] where instead of utilizing a single linear ranking heuristic to choose a single parent design for the next iteration, multiple parent designs were selected with the use of the linear method [9] and NSGA-II algorithm [21]. E-IDEA [16] was an extension to [12] with an evolutionary search engine to obtain the PF of the design space. The work in [17] and [18] used the NSGA-II algorithm [21] to solve the optimization problem with the two mutation and crossover operators to create the new population from the parent designs. In [19], initializing the PF pool with the original design, the heuristic applied all possible approximations separately to the ADFGs in the PF pool to generate new designs.

IV. Proposed Error Modeling Approach

In this section, we present our error modeling approach to estimate the error metrics at the output of a given ADFG. For any node in the ADFG, the output error is induced by two sources, which are the DOA of the node (if the node is mapped to an approximate module) and any errors in the input operands.

The estimation approach consists of two main steps. In the first step, which is performed once, the error at the operation level is precharacterized and stored in a set of LUTs. In the second step, which is invoked during the exploration of a DSE process, by propagating the errors of the operands and operators (using the precharacterization of the first step) from primary inputs (PIs) to the primary outputs (POs), the output accuracy of the ADFG is determined.

In this work, first, we use the MARED as the error metric to evaluate the output quality of an ADFG. Next, to overcome a possible MARED shortcoming in some cases, which will be described later, we also consider the MAED as the error metric to estimate the output quality.

For the quality evaluation of some image processing applications, other metrics such as PSNR are of more interest. Finding a technique to model the relationship between the errors at the output of the ADFG nodes and the application-level QoS metric is a complex task [33]. Not having this technique available to us at this stage, we assume that in the DSE process, an ADFG design with a smaller output MARED/MAED value would provide us with a design with a larger PSNR value. While the assumption is not necessarily valid for all the cases, our simulations show that using MARED/MAED as the quality metric with simulations in a DSE framework provides almost the same PF set (set of suggested designs) compared to the PFs obtained by using PSNR as the quality metric. The comparison of the two PF curves is presented in Fig. 8 (in Section VI-B) for the case of MARED. Note that the same holds for MAED as the quality metric in the exploration.

A. Error Characterization at the Operation Level

In the first step of the proposed estimation technique, we precharacterize the output error of any arithmetic unit modules in the library (with a specific DOA value) considering their operand errors. Obviously, for an exact arithmetic unit whose operands are erroneous, there is still some error present at the output of the module caused by the error of input operands. The arithmetic units considered in this work include adders, subtractors, and multipliers.

To precharacterize the output error metric of a module, different combinations of the operand error metric values should be considered separately. Evidently, the output quality of a module is highly input dependent. In other words, for a given amount of operand errors, having different exact operand values can result in very different amount of output error. Different combinations of the exact operand interval ranges, therefore, should also be considered when doing output error characterization for a module.

As mentioned before, MARED is first used as the metric to characterize a module's output error. Our studies show that using MRED as the operand error metric leads to a more accurate error model. Therefore, we use this parameter as the metric for the operands of the module.

To obtain the output error metric of the module under a specified operand error values and interval ranges (R_j for operand j), first, the set of error-free operand test vectors (X) to the module is generated. When generating each vector of X, the value of each operand j is chosen randomly from the interval R_j. Using set X and specific error metric values for the operands, the set of erroneous/approximate operand values (X') to the module is generated. More specifically, for the ith vector of X', to generate the approximate value for the jth operand ($X'[i,j]$), we use the value for the jth operand in the ith vector of X ($X[i,j]$) and the MRED of operand $j(\text{MRED}_j)$ in

$$X'\big[i,j\big] = \big(1 + \text{MRED}_j\big) \times X\big[i,j\big]. \tag{8}$$

Unlike (7), where the corresponding RED value is used to obtain each approximate value from the exact one, in (8), the same MRED_j value is used for all exact values for operand j in X. Using a single positive (negative) MRED value, however, results in over (under) estimation of the X' values. To overcome this issue, two positive and negative MRED_j values, represented by ϵ_j^+ (≥ 0) and ϵ_j^- (<0) are defined as the specified error metric of the operand j. This means that the operand error metric ϵ_j^+ (ϵ_j^-) should be chosen by sweeping the range of $[0, \epsilon_{\max}^+]([\epsilon_{\min}^-, 0))$ where $\epsilon_{\max}^+$ ($\epsilon_{\min}^-$) is determined by the user. A larger value (absolute value) for $\epsilon_{\max}^+$ ($\epsilon_{\min}^-$) means the user needs to consider more values for the operand errors. Therefore, the two parameters should be chosen depending on the resource and runtime requirements. Obviously, adding more operand error values means more entries for the LUTs, which would increase the accuracy of the model while adding more time to the precharacterization step and increasing the size of the LUTs.

The operand errors are considered as random variables (RVs) with the average/mean value of ϵ_j^+ and ϵ_j^-. To generate each $X'[i, j]$ value, a random binary number h is generated. If h is zero, $X'[i,j]$ is obtained using e (instead of MRED$_j$) and $X[i,j]$ [in (8)] where e is generated by a random generator from a normal distribution ($\mu = \epsilon_j^+$ with a small standard deviation $\sigma = \sigma_j^+$). On the other hand, if h is one, $X'[i,j]$ is obtained from $X[i,j]$ and e where e, is randomly extracted from a normal distribution ($\mu = \epsilon_j^-$ and again a small $\sigma = \sigma_j^-$). Our results show that considering a small σ ($\sigma_j^{+/-} \cong 0.1 \times \epsilon_j^{+/-}$) for the operand errors of a DFG node is sufficient. This is because, based on our studies, the distribution of the positive/negative RED values at the output of the operation nodes of the ADFGs typically follows a normal distribution with a small standard deviation.

There may exist some approximate modules in the library where the output errors cannot be modeled by a single normal distribution. For these modules (called *mixed modules*), the error may be modeled as a set of Gaussian distributions, i.e., Gaussian mixture (as observed in [34]). If an ADFG node n is mapped to a mixed module for the subsequent DFG node (which has node n connected to its jth operand), the positive and negative operand errors cannot be sufficiently represented by a single ϵ_j^+ and ϵ_j^- value. In fact, a set of ϵ_j^+ and ϵ_j^- values should be used, where each element in the set represents the mean of one Gaussian component.

Applying the error-free operands (X), we denote the output of the EM of the module as S. The output of the actual module [which may be exact or approximate (EAM)] with the erroneous operands applied (X') is denoted by S'. From the set of S and S', the RED values for each input vector are obtained using (3). The positive (negative) RED values are averaged to create the positive (negative) MRED at the output denoted by ϵ_o^+ (ϵ_o^-). Moreover, the absolute values of RED are averaged to generate the MARED at the output denoted by $\epsilon_o^{||}$, which is used for the application-level quality estimation at the PO of the ADFG. As will be discussed in Section IV-B, the calculated ϵ_o^+ and ϵ_o^- values become the operand error metrics for a subsequent node in the ADFG. Additionally, the maximum of the positive ($\epsilon_o^{+,\max}$) and minimum of the negative ($\epsilon_o^{-,\min}$) RED values are generated for the next step (see Section IV-C). The error values for the output represented by the tuple of $\epsilon_o = (\epsilon_o^+, \epsilon_o^-, \epsilon_o^{||}, \epsilon_o^{+,\max}, \epsilon_o^{-,\min})$ are stored in an entry of the module LUT with $I = (R_0, R_1, \epsilon_0^+, \epsilon_0^-, \epsilon_1^+, \epsilon_1^-)$.

In this work, we partition the whole data value range into a collection of intervals (buckets) with fixed and variable sizes. We obtain the ϵ_o values for different combinations of the operand interval ranges (R_j for the operand j) when precharacterizing the arithmetic modules in the library for different operand error values. The former technique, which is called *the fixed* (F) interval method, divides the whole value range into intervals with a size of 256. The reason for choosing 256 as the size of the intervals is that the PIs to the DFGs of the error-resilient applications (which are, e.g., pixels of an image) in this work are 8-bit unsigned numbers.

An example of the intervals for the fixed interval method is shown in Fig. 1(a) for positive operand values. Considering positive values, in this method, for the fixed range of 256,

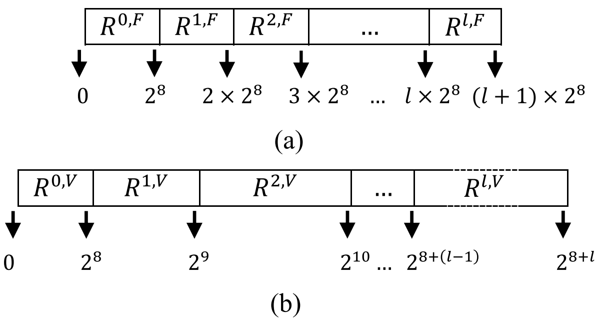

Fig. 1. Interval ranges for the (a) fixed and (b) variable (with $v\_w = 8$) interval methods for positive data values.

the fixed interval l (represented by $R^{l,F}$) covers the range of $[l \times 2^8, (l+1) \times 2^8 - 1]$. Similarly, the negative value ranges can be defined. To precharacterize a module with a dynamic range of $[0, 2^{16}-1]$ for the two operands, for each set of operand error values, $(2^{16}/2^8) \times (2^{16}/2^8) = 2^{16}$ interval combinations should be considered for the exact operand values. In this situation, which also happens in the case of a DFG with multiplier units, the precharacterization step becomes impractical. The reason is that the multiplication operation increases the range of the output and when the number of the multiplication stages increases (series multiplication), the output range and number of intervals and hence the number of combinations of the operand intervals of the following DFG nodes potentially follows geometric progression making the fixed interval method impractical (infeasible). We overcome this problem by a technique (namely, variable (V) interval method), where the whole value range is divided into intervals with variable sizes.

In this work, without loss of generality, we divide the whole range into variable intervals of $[V_1, V_2 - 1]$ where both V_1 and V_2 are considered as power of two numbers (as well as zero) and defined as below for positive value ranges for interval l as

$$\begin{cases} V_1 = 0, & V_2 = 2^{v\_w} \quad\quad l = 0 \\ V_1 = 2^{v\_w+(l-1)}, & V_2 = 2^{v\_w+l} \quad\quad l > 0 \end{cases} \tag{9}$$

where v_w is chosen empirically. Similarly, the negative value ranges can be defined. An example of the intervals for the variable interval method with $v\_w = 8$ for positive operand values is shown in Fig. 1(b) where interval l is represented by $R^{l,V}$. This way, to precharacterize a two-operand module with the dynamic range of $[0, 2^{16}-1]$ for the operands, $(16 - 8 + 1) \times (16 - 8 + 1) = 81$ interval combinations for the operands for each set of operand error values should be considered. Compared to 2^{16} intervals, which we had to consider in the case of the fixed interval method, this number is significantly smaller considerably improving the efficiency of the method. Note that lowering the value v_w leads to considering more intervals in the precharacterization step. Although this may increase the accuracy, it does increase the runtime of this step.

The use of variable sizes for the intervals decreases the accuracy compared to the case of the first method. Our results, however, show that the adverse impact on the performance of the proposed method is small (see Figs. 9 and 10).

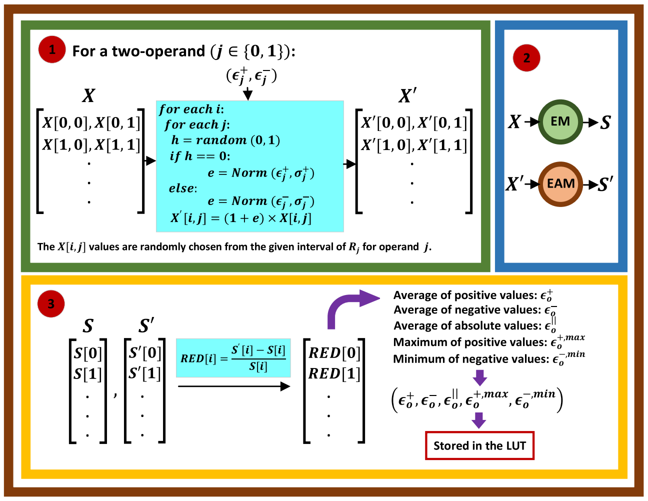
1
For a two-operand ($j \in \{0,1\}$):
$(\epsilon_j^+, \epsilon_j^-)$
X
$X[0,0], X[0,1]$
$X[1,0], X[1,1]$
for each i:
for each j:
$h = random\ (0,1)$
if $h == 0$:
$e = Norm\ (\epsilon_j^+, \sigma_j^+)$
else:
$e = Norm\ (\epsilon_j^-, \sigma_j^-)$
$X'[i,j] = (1+e) \times X[i,j]$
X'
$X'[0,0], X'[0,1]$
$X'[1,0], X'[1,1]$
The $X[i,j]$ values are randomly chosen from the given interval of R_j for operand j.
2
X
EM
S
X'
EAM
S'
3
S
$S[0]$
$S[1]$
S'
$S'[0]$
$S'[1]$
$RED[i] = \frac{S'[i] - S[i]}{S[i]}$
$RED[0]$
$RED[1]$
Average of positive values: ϵ_o^+
Average of negative values: ϵ_o^-
Average of absolute values: $\epsilon_o^{||}$
Maximum of positive values: $\epsilon_o^{+,max}$
Minimum of negative values: $\epsilon_o^{-,min}$
$(\epsilon_o^+, \epsilon_o^-, \epsilon_o^{||}, \epsilon_o^{+,max}, \epsilon_o^{-,min})$
Stored in the LUT

Fig. 2. Precharacterization step for a two-operand module from the library for a specified operand interval ranges and error values.

In addition, our simulations show that the estimated $\epsilon_o^{||}$ at the output of the 3 × 3 smoothing DFG (see Section IV-B for details) had 16 × higher error of modeling defined by MSED. The MSED was calculated with respect to the simulation values when the LUTs for the modules in the library were obtained with operands that were not bucketized compared to the fixed interval method.

In conclusion, in the precharacterization step of a two-operand module in the library, the output values of the tuple ϵ_o are obtained for different values in the tuple of I. Note that in I, R_j for each operand j is chosen from different intervals defined by fixed ($R^{l,F}$) or variable ($R^{l,V}$) interval methods. The results are stored in different entries of the module LUT along with I to be used in the next step (Section IV-B). Note that the tuple I for the two-operand modules with always one constant (i.e., error-free) operand becomes $I = (R_0, R_1, \epsilon_0^+, \epsilon_0^-)$. The size of the LUT depends on the number of different values considered for each parameter in I. For each module in the library, a separate LUT is required. For the LUT, the DOA of the module can be added as a parameter to I if the module provides different degrees of approximation. The pseudocode of the precharacterization step for a two-operand module under the specific operand errors and interval ranges are shown in Fig. 2. The proposed flow can also be applied to multioperand modules.

B. Propagation of the Error Within the ADFG

In this step, with the aid of LUTs generated from the previous step (obtained from fixed or variable interval methods), the error values are propagated through each ADFG to reach the ADFG POs. Let us explain the error propagation using the example shown in Fig. 3.

The DFG has three adder nodes implemented as a lower-part OR adder (LOA) [35] with four approximate LSBs (this design will be called *LOA4* in the remainder of this article). We denote the tuple of ϵ values $(\epsilon^+, \epsilon^-, \epsilon^{||}, \epsilon^{+,\max}, \epsilon^{-,\min})$ for operand j and output of each node n as $\epsilon_{j,n}$ and $\epsilon_{o,n}$, respectively. The value range of the operand j (output) of node n is denoted by $R_{j,n}$ ($R_{o,n}$).

Let us assume that in the precharacterization step, the LUTs were obtained by bucketing the whole data range of $[0, 2^{16}-1]$

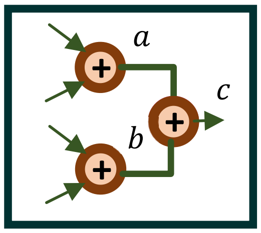

Fig. 3. ADFG with three approximate adder nodes.

into fixed intervals [see Fig. 1(a)]. The algorithm traverses the DFG nodes in the breadth-first approach [36], and thus, it starts from the DFG nodes, which have PIs as their operands (i.e., first-level DFG nodes).

Assuming that the PIs are error-free, ϵ_j^+ and ϵ_j^- for each operand j of the first level DFG nodes (adders a and b in Fig. 3) are zero ($\epsilon_{0,a}^{+/-} = \epsilon_{1,a}^{+/-} = \epsilon_{0,b}^{+/-} = \epsilon_{1,b}^{+/-} = 0$). In this work, we consider the PIs as 8-bit unsigned numbers, which are in the range of $[0, 255]$. Therefore, for the adders of a and b in the DFG of Fig. 3, $R_{0,a} = R_{1,a} = R_{0,b} = R_{1,b} = R^{0,F}$. This means that, the $\epsilon_{o,a}$ and $\epsilon_{o,b}$ values can be extracted from the LUT of LOA4. The $\epsilon_{o,a}$ and $\epsilon_{o,b}$ values constitute the operand errors of the adder c where $\epsilon_{0,c}^{+/-} = \epsilon_{o,a}^{+/-}$ and $\epsilon_{1,c}^{+/-} = \epsilon_{o,b}^{+/-}$. Furthermore, to extract the $\epsilon_{o,c}$ value from the LUT of LOA4, the intervals of the operands to c are required. Since the node $a(b)$ adds two numbers in the range of $[0, 255]$, the output is always within $[0, 510]$ (i.e., $R_{o,a} = R_{o,b} = [0, 510]$). Therefore, the exact operands to c, can be in either $R^{0,F}$ or $R^{1,F}$. This means that for the adder c, there are M combinations of input intervals with $M = 4$. All these combinations are considered and four separate values for $\epsilon_{o,c}$ are extracted from the corresponding LUT (each represented by $\epsilon_{o,c,m}$ with $0 \leq m < M$). The values of the final $\epsilon_{o,c}$ are obtained from

$$\epsilon_{o,c}^{+} = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\epsilon_{o,c,m}^{+}, \quad \epsilon_{o,c}^{-} = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\epsilon_{o,c,m}^{-}$$
$$\epsilon_{o,c}^{||} = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\epsilon_{o,c,m}^{||}$$
$$\epsilon_{o,c}^{+,\max} = \max_{0\leq m\leq M}\left(\epsilon_{o,c,m}^{+,\max}\right), \quad \epsilon_{o,c}^{-,\min} = \min_{0\leq m\leq M}\left(\epsilon_{o,c,m}^{-,\min}\right). \tag{10}$$

If the DFG has a larger depth, the ϵ values may be propagated in a similar way in the topological order (using a breadth-first search traversal algorithm [36]) of the DFG. The pseudocode for propagating the error values in a general ADFG is shown in Fig. 4. In Fig. 4, for each node n in the DFG (with N nodes), first, for each operand j, $\epsilon_o^{+/-}$ from the predecessor node p_j are set as the operand errors ($\epsilon_{j,n}^{+/-}$), where PIs are error free. The interval range of each operand ($R_{j,n}$) is also equal to the R_o of the predecessor node. The interval range of the operands needs to be translated to the interval ranges stored in the corresponding LUT (different for fixed and variable interval methods). Based on $\epsilon_{j,n}^{+/-}$ of the operands, for each combination m of interval values (the interval of operand j for the mth combination is denoted as $R_{j,n,m}$), I_m is defined and the $\epsilon_{o,n,m}$ values are extracted.

Note that if the operand error values, I_m is not present in the LUT, the error values should be rounded to the nearest

```
Input: The ADFG with $N$ nodes
       The set of LUTs generated from the previous step
Output: $\epsilon_o^{||}$ at the PO of the ADFG
1.  Begin
2.    for $n$ from 0 to $N-1$ (nodes numbered in breadth-first order)
3.      $p_j$ = predecessor of $n$ for operand $j \in \{0,1\}$
4.      $\epsilon_{0,n}^{+/-} = \epsilon_{o,p_0}^{+/-}$ and $\epsilon_{1,n}^{+/-} = \epsilon_{o,p_1}^{+/-}$ (consider error-free PIs)
5.      if normalization is required
6.        Normalize $\epsilon_{0,n}^{+/-}$ and/or $\epsilon_{1,n}^{+/-}$
7.      end if
8.      $R_{0,n} = R_{o,p_0}$ and $R_{1,n} = R_{o,p_1}$
9.      Translate $R_{0,n}$ and $R_{1,n}$ into $M$ interval combinations of $R_{0,n,m}$
        and $R_{1,n,m}$ values (depending on (F) or (V) interval methods
        stored in the corresponding LUT)
10.     for $m$ from 0 to $M-1$
11.       Using $I_m = (R_{0,n,m}, R_{1,n,m}, \epsilon_{0,n}^+, \epsilon_{0,n}^-, \epsilon_{1,n}^+, \epsilon_{1,n}^-)$ for LUT,
          extract $\epsilon_{o,n,m} = \left(\epsilon_{o,n,m}^+, \epsilon_{o,n,m}^-, \epsilon_{o,n,m}^{||}, \epsilon_{o,n,m}^{+,max}, \epsilon_{o,n,m}^{-,min}\right)$
          ($I_m = (R_{0,n,m}, R_{1,n,m}, \epsilon_{0,n}^+, \epsilon_{0,n}^-)$ if one operand of $n$ is
          always constant)
12.       if $I_m$ not present in the LUT
13.         Round $I_m$ elements to the nearest values stored in LUT
14.         Use the new $I_m$ to extract $\epsilon_{o,n,m}$ from LUT
15.       end if
16.     end for
17.     $\epsilon_{o,n}^+ = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\epsilon_{o,n,m}^+$ , $\epsilon_{o,n}^- = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\epsilon_{o,n,m}^-$
        $\epsilon_{o,n}^{||} = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\epsilon_{o,n,m}^{||}$
        $\epsilon_{o,n}^{+,max} = \max_{0\le m<M}\left(\epsilon_{o,n,m}^{+,max}\right)$, $\epsilon_{o,n}^{-,min} = \min_{0\le m<M}\left(\epsilon_{o,n,m}^{-,min}\right)$
18.     Determine $R_{o,n}$ based on the operation of $n$, $R_{0,n}$ and $R_{1,n}$
19.   end for
20.   return $\epsilon_o^{||}$ of the ADFG PO
21. End
```

Fig. 4. Pseudocode of propagating the error values in an ADFG.

values stored. The ϵ_o at the output of the node $n(\epsilon_{o,n})$ is estimated from $\epsilon_{o,n,m}$ values [using (10)]. The interval range of the exact output value of the node $n(R_{o,n})$ is also determined for subsequent nodes.

C. Shortcomings of the *MARED* Metric

In this section, the shortcoming of estimating MARED at the output of an ADFG is discussed, and two possible solutions are presented.

Consider the case of an approximate subtractor (or an approximate adder with operands that have different signs). When obtaining the output quality in the precharacterization step, the *RED* values may become (much) larger than the value "1" (>100%) for some input vectors. In these cases, the ED of the module will be larger than the exact value of the subtraction (e.g., when the values of the operands are close to each other, and both have the same sign). An example of the histograms of RED at the output of an approximate LOA4 subtractor and adder is shown in Fig. 5. Error-free operands of both modules are in the interval $[0, 255]$. For the subtractor, the large output RED values make the value of MARED also large.

If we consider this large positive and negative MRED as the operand errors for a subsequent node in an ADFG, the

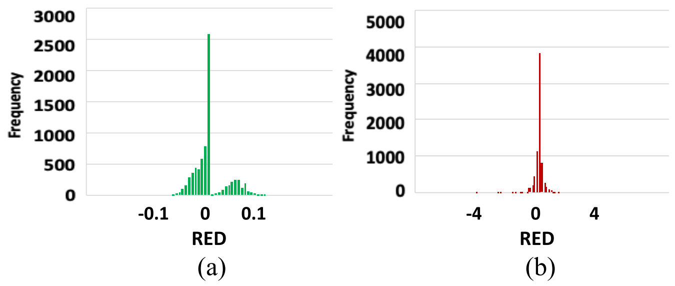

Fig. 5. Histogram of an approximate (a) adder and a (b) subtractor.

error will accumulate and become unrealistically large as it traverses the ADFG, especially for larger values of the exact operands. As the first solution, we consider normalizing the ϵ_j^+ (and/or ϵ_j^-) value with respect to $\epsilon_j^{+,\max}$ (and/or $\epsilon_j^{-,\min}$) as

$$\epsilon_j^+ = \frac{\epsilon_j^+}{\epsilon_j^{+,\max}}, \quad \epsilon_j^- = \frac{\epsilon_j^-}{|\epsilon_j^{-,\min}|}. \tag{11}$$

Therefore, when estimating MARED at the output of an ADFG, the said normalization technique must be applied when needed (line 5 in Fig. 4). While this solution solves the error accumulation problem, it decreases the accuracy of the estimated error at the output of the ADFG.

As the next solution, which controls the increase in the inaccuracy of error estimation of the ADFG output, we suggest changing the output error metric from MARED to MAED. To incorporate this change, we have to modify some parts of the two steps of Sections IV-A and IV-B as will be explained. To distinguish these two different techniques, we denote the technique that estimates the MARED at the output of an ADFG as MR and the other one that uses MAED as MD. For the MD method, in the precharacterization of a specific module in the library under given operand error values and interval ranges, for the error of each operand j, three error metrics are required (compared to two for the MR method). These include the positive and negative MED values and standard deviation σ of the ED values for operand j (σ_j).

We represent the positive (negative) MED values for operand j with δ_j^+ (δ_j^-) to distinguish it from ϵ_j^+ (ϵ_j^-). In the case of MR, due to the small variation of relative errors, a small σ was used. The reason that the σ values for the operand errors are required in the case of MD is that the scattering of the ED values across the MED is larger compared to that for MRED. Therefore, a single δ_o^+ (δ_o^-) value is not sufficient to estimate the positive (negative) ED values at the module output when the errors are propagated across the ADFG. Using a similar approach to the one explained in Section IV-A, after generating the set X, we obtain each $X'[i,j]$ from $X[i,j]$ and the operand errors. More specifically, if the generated h is zero, a RV e with a normal distribution ($\mu = \delta_j^+$ and $\sigma = \sigma_j$) is extracted and used in

$$X'[i,j] = X[i,j] + e. \tag{12}$$

If h is one, $X'[i,j]$ is obtained from (12) where e is randomly extracted from a normal distribution ($\mu = \delta_j^-$ and $\sigma = \sigma_j$). After obtaining S and S', which are the output values from EM and EAM, respectively, the ED values for each input

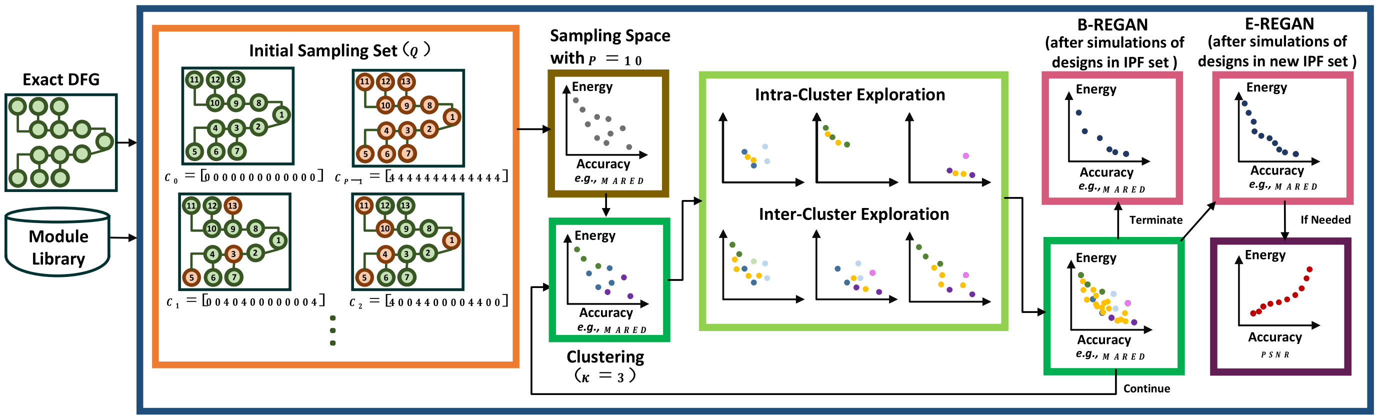
Exact DFG
Module Library
Initial Sampling Set (Q)
$C_0 = [0000000000000]$
$C_{P-1} = [4444444444444]$
$C_1 = [0040400000004]$
$C_2 = [4004400004400]$
Sampling Space with P = 10
Energy
Accuracy e.g., MARED
Clustering (κ = 3)
Intra-Cluster Exploration
Inter-Cluster Exploration
B-REGAN (after simulations of designs in IPF set)
E-REGAN (after simulations of designs in new IPF set)
Terminate
If Needed
Continue
Accuracy PSNR

Fig. 6. REGAN steps to extract the PF of the design space.

vector are obtained using (1). The positive (negative) ED values are averaged to create the positive (negative) MED output value denoted by δ_o^+ (δ_o^-). The maximum of the σ values of the positive and negative ED values is also obtained and denoted by σ_o. Also, to generate the MAED value of the module output, the absolute values of the ED values are averaged. It is denoted by $\delta_o^{||}$ and used for the application-level quality estimation at the PO of the ADFG. Error values for the output, which will be stored in one entry of the module LUT for $I = (R_0, R_1, \delta_0^+, \delta_0^-, \sigma_0, \delta_1^+, \delta_1^-, \sigma_1)$ are represented by the tuple of $\delta_o = (\delta_o^+, \delta_o^-, \sigma_o, \delta_o^{||})$. For a two-operand module with always one constant operand, I is considered as $(R_0, R_1, \delta_0^+, \delta_0^-, \sigma_0)$.

In the error propagation step within the ADFG, similar to the procedure explained in Section IV-B, for each node n in the DFG, the $\delta_o(R_o)$ values from the predecessor nodes are set as the operand errors (intervals) as $\delta_{j,n}^{+/-}$ and $\sigma_{j,n}$ ($R_{j,n}$) for each operand j. Based on the $\delta_{j,n}^{+/-}$ and $\sigma_{j,n}$ values of the operands and the combination of all possible M interval values, M separate $\delta_{o,n,m}$ values are extracted from the LUT and the δ_o value at the output of $n(\delta_{o,n})$ is estimated using

$$\delta_{o,n}^+ = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\delta_{o,n,m}^+, \quad \delta_{o,n}^- = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\delta_{o,n,m}^-$$
$$\sigma_{o,n} = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\sigma_{o,n,m}, \quad \delta_{o,n}^{||} = \left(\frac{1}{M}\right)\sum_{m=0}^{M-1}\delta_{o,n,m}^{||}. \tag{13}$$

Note that in (13), the $\sigma_{o,n}$ value is approximated. The interval of the output of n (i.e., $R_{o,n}$) is also obtained for the subsequent nodes.

V. Proposed DSE Framework

In the final step, the proposed error model is integrated in the quality evaluation step of an ADFG exploration framework. In this work, without loss of generality, we adopt and modify the DSE framework of [20], named EGAN, such that when a new ADFG is visited, the quality of the design is obtained by the error model instead of a simulator. We name the modified EGAN framework as *rapid EGAN* (REGAN).

The REGAN framework belongs to the category of evolutionary approaches for the DSE algorithms (explained in Section III-B2). It automatically extracts approximate circuits with their corresponding configurations (i.e., the DOA of the modules mapped to the DFG nodes) that are located on the PF of the trade-off space of accuracy versus energy consumption. By using the proposed error model, REGAN achieves a significant reduction in the number of explored ADFGs compared to that of the EGAN approach while achieving reasonable results.

Key steps of the REGAN framework, which is depicted in Fig. 6, start with a random initial set of P ADFGs (denoted as Q), including the fully exact and fully ADFGs. The fully exact design refers to the original DFG, whereas the fully approximate circuit refers to the ADFG, where all DFG nodes are replaced by the approximate arithmetic unit (of compatible operation type) with the highest DOA from the library. Based on the chosen DOAs for the modules to be assigned to the DFG nodes, the configuration of an ADFG in REGAN is represented by an N-word vector (denoted as C) of integer numbers, where N is equal to the number of nodes in the DFG, which are amenable to approximation (i.e., adder/subtractor and multiplier nodes in this work). Each element in the vector represents the DOA (e.g., the number of approximate LSBs in the LOA adder) of the corresponding DFG node. For instance, for the input DFG in Fig. 6 with 13 nodes, the DOA of all the DFG nodes can take values from the set of $D = \{0, 4\}$, where a DFG node with the degree of zero (four) is represented by the green (red) color.

For the ADFGs in Q, the corresponding energy consumption and the output error (accuracy) of each ADFG are determined, and the sampling space (the space containing the design points) is generated. The energy consumption of a given ADFG is obtained using the precomputed energy consumption values of the arithmetic units in the library. Ignoring the scheduling step and resource sharing in the HLS process, the energy costs of the arithmetic modules assigned to the ADFG nodes are accumulated to estimate the total energy consumption of the design.

To obtain the output accuracy of an ADFG in REGAN, we use the error models explained in Section IV, where the exploration process uses MARED (or MAED) as the output error metric. Using different combinations of fixed/variable interval and MR/MD methods, we end up with four different error models that can be incorporated into REGAN.

Subsequent steps of REGAN are iterative. First, the design points in the initial sampling space are clustered based on their similarities into κ clusters by employing the k-means clustering algorithm [37]. Three parameters are considered

```
Input: $C_H$ and $C_L$: Input configurations with $N$ elements
       $\zeta$ : Percentage of total generated candidates to choose
       $D$: Set of DOA values for the modules in the library
Output: $G$: Set of selected neighboring configurations
1.  Begin
2.      $G = \emptyset$
3.      $allCandidates = \emptyset$
4.      for $n$ from 0 to $N-1$
5.          $C_{L,limit} = C_L[n]$
6.          while $C_{L,limit} < C_H[n]$
7.              update $C_{L,limit}$ based on the values in $D$
8.              $C'_L = C_L$ with $C'_L[n] = C_{L,limit}$
9.              Add $C'_L$ to $allCandidates$
10.         end while
11.     end for
12.     $G$ = Randomly choose $\zeta$ percent of $allCandidates$
13.     Determine energy and error values of elements in $G$
14.     return $G$
15. End
```

Fig. 7. Pseudocode of generating a set of neighbors from the two configurations of C_H and C_L with N elements.

for determining the degree of similarity between the ADFGs consisting of the integer representation (C), the output error, and the energy cost (hamming distance between the vectors where each vector is constituted from the three corresponding parameters).

Next, in each iteration, a set of new ADFGs is generated. To generate a new set of approximate designs, two different existing parent design points with the configuration vectors of C_L and C_H (where C_L has lower output error and higher energy consumption than C_H) are considered. The expansion process aims to generate a set of configurations (considered as the set of neighboring configurations and denoted as G) with lower energy consumption compared to C_L while having higher energy consumption compared to C_H. The pseudocode for this process is shown in Fig. 7.

Next, ζ percentage ($0 < \zeta \leq 100\%$) of all possible new configurations is selected as the set of neighboring configurations G to be evaluated and added to the design space. The parameter ζ is tuned in a way that a good trade-off between the framework runtime and the accuracy of the final PF obtained can be achieved. The energy consumption of each ADFG in G is estimated by accumulating the energy costs of the arithmetic modules assigned to the ADFG nodes while the proposed error models are used to estimate the quality of the output of the ADFG. With the use of the aforesaid method, the expansion of the design space in REGAN has two steps (using the pseudocode of Fig. 7).

1) In the first step, called *intracluster exploration*, each cluster is expanded separately, where every combination of two configurations, which are located on the local PF of that cluster, are considered for generating additional configurations. The amount of the expansion is determined based on parameter ζ_1 ($\zeta = \zeta_1$ in Fig. 7). The local PF of a cluster includes the design points that dominate all other points in that cluster.
2) While the intracluster exploration step helps in exploring the unobserved configurations, it may lead to getting trapped in the saddle point of the design space. Thus, in the next step, named the *intercluster exploration* step, the union of two different clusters is expanded. The combinations of each two configurations (i.e., each of the two configurations is chosen separately from each cluster in the union) on the local PF of the union cluster are considered for generating new configurations. In this step, ζ_2 percent ($\zeta = \zeta_2$ in Fig. 7) of all the possible generated configurations are added to the design space.

In REGAN, the total number of iterations to terminate the exploration process is considered similar to that of EGAN [20]. If the heuristic is not terminated, the set of all design points so far (both dominated and nondominated) is considered as the initial sampling set for the subsequent iteration.

At the end of the last iteration of the exploration, the PF of all the points in the design space (including the initial set and the newly generated points) is extracted and denoted as the initial PF (IPF). Since the output quality of the designs on the IPF was obtained via the error models (which is not as accurate as the simulation approach), the designs on the IPF set are simulated to extract more accurate (final) PF of the trade-off space of MARED/MAED versus energy consumption. The REGAN DSE with this extracting final PF method is called *Basic REGAN* (B-REGAN).

In the proposed error model, since precomputed LUTs are used to estimate the error values at the node outputs, it is possible that many ADFGs visited in the exploration have the same error values for their outputs. The reason is that LUTs are characterized with discrete values, and if, in the propagation step, the input operand errors of an ADFG node (which are obtained from the output error of the corresponding predecessor nodes in the ADFG) are not present in the corresponding LUT, the operand error values would be rounded to the nearest existing values stored in the LUT (line 12 in Fig. 4). This means that, by using the error model (instead of the simulation approach), we may dismiss some good ADFGs that would have been in the final PF set of simulations were performed. To address this issue, a modification to B-REGAN is needed.

Once the exploration is completed and the IPF set is obtained, some extra designs are selected and added to the IPF set. To generate these extra points, for each energy consumption value, λ percent of all the explored designs with similar energy consumption having the lowest estimated MARED/MAED values are selected. This way, a certain percentage of the higher quality explored designs for each energy consumption value are added to the IPF set. Designs on the new IPF set are simulated, and the final PF of the trade-off space of MARED/MAED versus energy consumption is obtained from all the simulated designs. When this method of extracting the final PF is used in REGAN, the DSE approach is called *Enhanced REGAN* (E-REGAN), which generates more design points than the PF extracted by B-REGAN.

As mentioned before, for image processing applications, PSNR is of high interest. The final PF set extracted using B-REGAN (or E-REGAN), represents the designs, which are located on the PF curve of the trade-off space of MARED/MAED and energy consumption. To extract the PF curve of the trade-off between PSNR and energy consumption, the simulation engine is invoked on the PF obtained by

TABLE I

Number of DFG Nodes, Number of Addition (or Subtraction) and Multiplication Operations, Data-Path Dynamic Range and QoS Metric in Each Benchmark

Benchmarks	|Nodes|	|Add|	|Multiply|	Dynamic Range	QoS metric
Smoothing (3×3)	8	8	0	$[-2^{11}, 2^{11}-1]$	$PSNR$
Sobel	13	13	0	$[-2^{10}, 2^{10}-1]$	$PSNR$
Smoothing (5×5)	29	24	5	$[-2^{15}, 2^{15}-1]$	$PSNR$

B-REGAN (or E-REGAN) to obtain the corresponding PSNR values.

VI. Results and Discussion

A. Simulation Setup and Benchmarks

To assess the efficacy of the proposed method, the benchmarks from the image processing domain, including the image smoothing filter (3 × 3 and 5 × 5 kernels) and Sobel edge detector were considered. Table I shows the number of DFG nodes, addition/subtraction and multiplication operations, data-path dynamic range, and QoS metric for these benchmarks in the REGAN framework. We implemented the framework in Python and employed Synopsys VCS tool as the simulation engine.

For implementing the ADFGs, the libraries of approximate adders and multipliers comprised of LOA adders [35] and TOSAM multipliers [38] were employed. To realize different accuracy levels of the arithmetic units, in LOA, a different number of approximate LSBs were replaced with OR gates, while in TOSAM, a different number of LSBs were kept in for implementing the ADFG of the benchmarks.

In the case of 3 × 3 smoothing and Sobel, we assumed that the employed adder could support $|W|$ approximation degrees where $|W| = 3$ (exact, LOA with 2 (LOA2) and 4 (LOA4) approximate bits) and $|W| = 4$ (exact, LOA2, LOA4 and LOA with 6 (LOA6) approximate bits). For the 5 × 5 smoothing benchmark, the adder and multiplier were assumed to have $|W|$ approximation degrees where $|W| = 2$ (one exact and one approximate implementation) and $|W| = 3$ (one exact and two approximate implementations). In the case of $|W| = 2$ ($|W| = 3$), for the approximate adder, LOA4 (LOA2 and LOA4) was considered. In the case of the approximate TOSAM multiplier, for the multiplicand operand, 8 (8 and 12) LSBs were kept, while for the multiplier operand, 4 (4 and 6) LSBs were used. The multiplier with x/y LSBs for the multiplier/multiplicand operand is denoted by TOSAM $x-y$. The operation principles of the approximation in TOSAM are based on truncating both the operands regardless of their errors.

The REGAN framework was implemented with the parameters of $P = 10$, $\kappa = 3$, $\zeta_1 = \zeta_2 = 100\%$ [20]. The inputs to the considered benchmarks were images obtained from the USC-SIPI image database [39]. To evaluate the output accuracies of the image processing applications for REGAN, the output image of the EM of the application was considered as the golden output and $PSNR$ was considered as the accuracy metric.

As was explained in Sections IV-A and IV-C, the error LUTs of the arithmetic units in the library could be obtained through fixed (F) or variable (V) intervals and the MR or MD techniques.

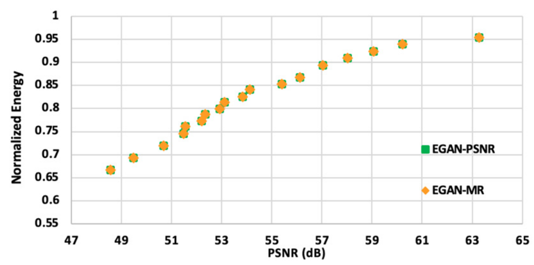

Fig. 8. Extracted PF by EGAN [20] for 3 × 3 smoothing benchmark with PSNR (EGAN-PSNR) and MARED (EGAN-MR) metrics used in the exploration.

The combination provided us with four different error models that can be incorporated in the REGAN framework. We denote these four techniques by F-MR, V-MR, F-MD, and V-MD, where the notation is self-explanatory. Also, the REGAN framework itself was implemented as B-REGAN or E-REGAN (as was explained in Section V). Also, the parameter λ, which controlled the extra designs added in E-REGAN, was considered as 50% for all the benchmarks.

In order for the fixed interval method to be practical (due to the runtime complexity), based on our simulations, the dynamic range of the DFG should be considered within $[-2^{11}, 2^{11}-1]$. The LUTs, therefore, obtained with the fixed interval method could not be used for the 5 × 5 smoothing benchmark due to the larger dynamic range of this application. Thus, in this case, the variable interval method should be used. For the variable interval method, the dynamic range of $[-2^{15}, 2^{15}-1]$ was considered.

In the precharacterization step for the MR method, the value of ±10% was considered for $\epsilon^{+/-}_{\max/\min}$ when sweeping the operand error values with a step of 1%. For each operand error value, a small $\sigma^{+/-} \cong 0.1 \times \epsilon^{+/-}$ was considered.

For the MD method, to avoid large precharacterization step runtime, the δ and σ values were chosen as power of two numbers. The maximum absolute value of 16 was considered for both δ and σ. In addition, we assumed that the σ value for each operand error could take values larger or equal to its corresponding δ absolute value. The values for the operand errors in the precharacterization step for both MR and MD methods were chosen empirically based on the benchmarks. Obviously, adding more different operand error values to the LUTs would increase the accuracy of the model while adding more time to the pre-characterization step and increasing the size of the LUTs.

B. Results

In this section, first, we show that the MARED of the output of an ADFG can be used as the accuracy metric in the exploration framework for obtaining the PF in the trade-off space of PSNR versus energy dissipation in image processing applications.

In Fig. 8, for the 3 × 3 smoothing benchmark with $|W| = 3$, the PF extracted by EGAN [20] utilizing the simulated PSNR values (EGAN-PSNR) is represented by green squares. The orange diamonds depict the PF obtained by using the simulated $MARED$ values (EGAN-MR) in the exploration. This means that in the latter, the designs located on the PF of the trade-off between MARED and energy

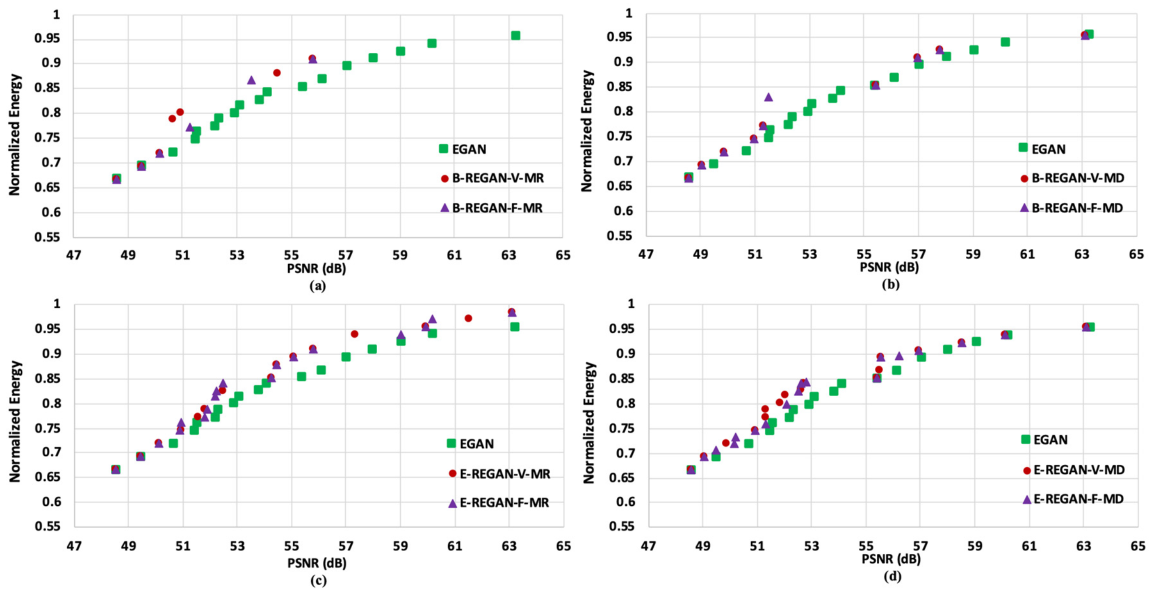

Fig. 9. PFs extracted by EGAN [20] and REGAN for 3 × 3 smoothing benchmark with $|W| = 3$ for the B-REGAN framework with (a) MR and (b) MD method, and E-REGAN framework for the (c) MR and (d) MD method.

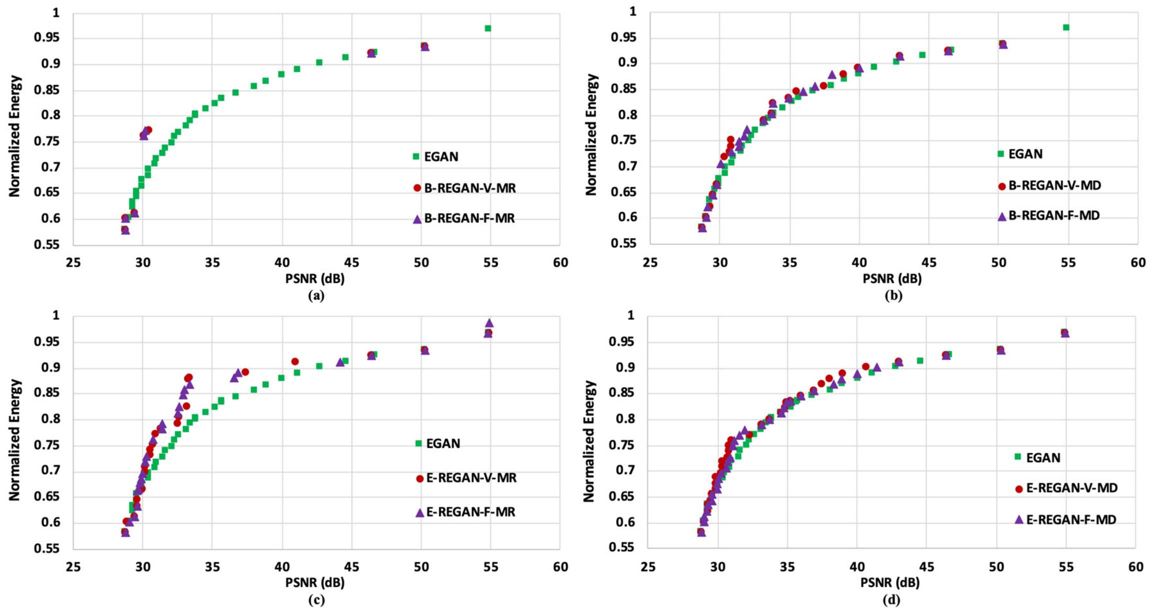

Fig. 10. PFs extracted by EGAN [20] and REGAN for Sobel benchmark with $|W| = 3$ for the B-REGAN framework with (a) MR and (b) MD method, and E-REGAN framework for the (c) MR and (d) MD method.

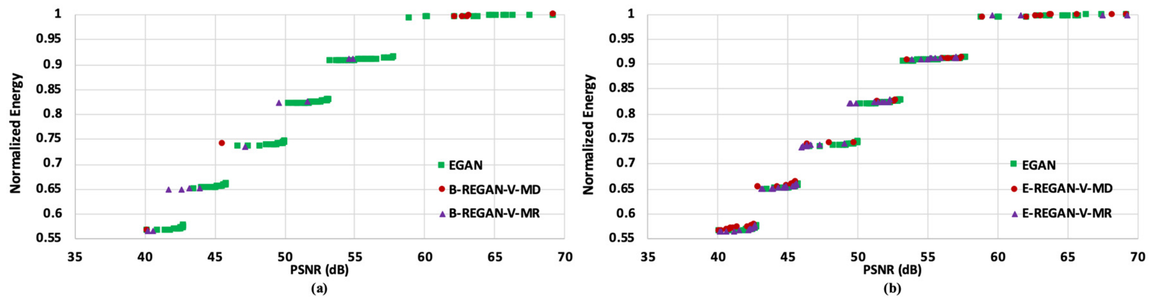

Fig. 11. PFs extracted by EGAN [20] and REGAN for 5 × 5 smoothing benchmark with $|W| = 2$ for the variable interval method for the (a) B-REGAN and (b) E-REGAN frameworks.

consumption were again simulated to determine the PSNR values for these configurations. It can be seen that the PFs are almost identical, and the orange diamonds cover all the green squares.

Next, we present results for evaluating the efficacy of the proposed REGAN framework to efficiently extract the PF curve representing the trade-off between PSNR and energy consumption for the benchmarks of Table I. The PF

TABLE II

NUMBER OF DOA VALUES FOR EACH DFG NODE ($|W|$), DIFFERENT DSE FRAMEWORKS, NUMBER OF SIMULATIONS, (|SIM|), EXECUTION RUNTIME (|TIME|), RUNTIME SPEEDUP WITH RESPECT TO EGAN (SPEEDUP) AND THE NUMBER OF ADFGS ON THE EXTRACTED PF FOR THE BENCHMARKS (|PF|)

Benchmarks	$\|W\|$	DSE		|Sim|	|Time|	Speedup	|PF|
Smoothing (3×3)	3	EGAN		470	30	-	19
		B-REGAN	F-MR	27	1.3	23×	6
			V-MR	26	1.2	24×	7
			F-MD	34	1.5	20×	11
			V-MD	34	1.4	21×	10
		E-REGAN	F-MR	135	8.9	3×	19
			V-MR	130	9.1	3×	16
			F-MD	102	5.9	5×	19
			V-MD	100	5.8	5×	21
	4	EGAN		1104	70	-	26
		B-REGAN	F-MR	40	3.6	20×	14
			V-MR	34	2.8	25×	16
			F-MD	42	2.7	26×	16
			V-MD	34	3.2	22×	16
		E-REGAN	F-MR	320	28	2×	22
			V-MR	306	28	2×	21
			F-MD	293	23	3×	29
			V-MD	308	25	3×	27
Sobel	3	EGAN		1288	93	-	36
		B-REGAN	F-MR	19	0.95	98×	7
			V-MR	19	0.88	106×	7
			F-MD	35	2.7	34×	22
			V-MD	34	2.2	43×	20
		E-REGAN	F-MR	140	13	7×	27
			V-MR	183	18	5×	24
			F-MD	142	13	7×	35
			V-MD	118	9.7	10×	35
	4	EGAN		7081	527	-	58
		B-REGAN	F-MR	20	1.7	304×	6
			V-MR	20	1.9	273×	6
			F-MD	33	6	89×	18
			V-MD	31	5	118×	16
		E-REGAN	F-MR	429	46	11×	32
			V-MR	497	53	10×	31
			F-MD	795	90	6×	50
			V-MD	623	63	8×	50
Smoothing (5×5)	2	EGAN		5267	830	-	117
		B-REGAN	V-MR	34	6.8	125×	12
			V-MD	24	3.2	250×	9
		E-REGAN	V-MR	690	133	6×	66
			V-MD	509	87	10×	56
	3	EGAN		20000	3000	-	197
		B-REGAN	V-MR	28	4.4	725×	13
			V-MD	21	2.6	1200×	7
		E-REGAN	V-MR	1312	230	13×	96
			V-MD	980	159	20×	56

curves generated by the different frameworks are depicted in Figs. 9–11 for the 3 × 3 smoothing, Sobel (with $|W| = 3$), and 5 × 5 smoothing (with $|W| = 2$) benchmarks, respectively.

For all the benchmarks, the PF obtained by exhaustive exploration of the design space is impractical and the PF generated by EGAN was considered as the reference set, which is depicted by green squares in Figs. 9–11. As was also observed in [20], EGAN is able to generate reasonable PFs of the design space when the exhaustive exploration of the design space is impractical.

In Figs. 9(a) and 10(a), the PFs generated by the B-REGAN framework for the MR method and in Figs. 9(b) and 10(b) for the MD method for both the fixed (F) and variable (V) interval methods are represented by purple triangles and red dots, respectively for the 3 × 3 smoothing and Sobel benchmarks. The extracted PFs by the E-REGAN framework for the MR method is depicted in Figs. 9(c) and 10(c) while Figs. 9(d) and 10(d) show the results for the MD method for both the cases of the fixed and variable interval methods by purple triangles and red dots, respectively.

For the 5 × 5 smoothing benchmark, since the dynamic range of the DFG is outside $[-2^{11}, 2^{11} - 1]$, the fixed interval method LUTs could not be used. Hence, in Fig. 11(a), the extracted PFs by B-REGAN and in Fig. 11(b), the extracted PFs by E-REGAN for the MR and MD methods are represented only for the variable interval method by purple triangles and red dots, respectively.

It can be deducted from Figs. 9–11, compared to B-REGAN, E-REGAN generates closer PFs to the PFs generated by EGAN. This is due to the higher number of simulations in the case of E-REGAN, compared to that of B-REGAN, increasing the ability of extracting better PF curves. In the case of the 3 × 3 Smoothing benchmark, in both B-REGAN and E-REGAN, for both the MR and MD models, the fixed interval method generates slightly better PFs compared to those of the variable interval method. More specifically, the average of the Euclidian distances between each point on the PF obtained by EGAN and the closest point on the PF curve obtained by E-REGAN (for MD and fixed interval method) shows 14% reduction compared to E-REGAN when the variable interval method is employed.

For the Sobel benchmark, however, in the case of the MR models, for the B-REGAN, the variable method generates better PFs (on average, 5% smaller Euclidian distance) compared to that of the fixed interval method. This may happen because of the normalization operation explained in Section IV-C.

For the 3 × 3 smoothing and Sobel benchmarks, the MD method generates slightly better results (on average, 23% and 68% smaller Euclidian distances) compared to those of the MR method for both B-REGAN and E-REGAN frameworks. For the 5 × 5 smoothing benchmark, in B-REGAN, the MR method works better (on average, 47% smaller Euclidian distance) while in E-REGAN, the MD method outperforms (on average, 24% smaller Euclidian distance). This happens because of the randomness in the REGAN framework, as well as the small number of simulated designs compared to the design space of the 5 × 5 smoothing benchmark.

Table II reports the number of DOA values for each DFG node ($|W|$), different DSE frameworks (B-REGAN, E-REGAN, F-MR, V-MR, F-MD, and V-MD), the number of simulations (|Sim|), execution runtime (|Time|) in minutes, runtime speedup with respect to EGAN (Speedup) and the number of ADFGs on the extracted PF for the benchmarks (|PF|). The runtime speedup is calculated from the ratio of EGAN runtime with respect to the runtime of the REGAN framework.

As was expected, in Table II, the number of simulations for the B-REGAN is smaller than that of E-REGAN for all the benchmarks, which results in higher speedup for B-REGAN. On the other hand, E-REGAN finds larger number of designs on the Pareto curve compared to those of B-REGAN.

For different benchmarks, increasing the size of the design space (as a result of increasing $|W|$) had different impacts on the runtime speedup of the REGAN framework compared to EGAN. For instance, in the Sobel (5 × 5 smoothing) benchmark, on average, for the case of $|W| = 4$ ($|W| = 3$), the speed up of REGAN with respect to EGAN is 2.6× (5×) of the case of $|W| = 3$ ($|W| = 2$). For the 3 × 3 smoothing (which has a smaller design space compared to those of the other two benchmarks), the average speed up is almost similar in the case of $|W| = 3$ and $|W| = 4$. Observing the speedup values in Table II, the REGAN framework provides on average 92 × speed improvement compared to the case of purely simulation-based EGAN framework.

It should be mentioned that the low quality of the obtained PF curves generated by REGAN are acceptable when considering the large runtime speedups. Considering a smaller speedup in the framework can lead to improvements in the quality of the obtained PF if it is required. This can be obtained by for example increasing the λ parameter in E-REGAN. In the case of 5 × 5 smoothing (V-MD), increasing λ by 20% results in increasing the size of the PF set by 14%, with only 6% increase in the runtime.

VII. Conclusion

In this work, we proposed an efficient technique to estimate the error at the output of an approximate data-flow graph (ADFG). The technique, which relied on bucketizing the input dynamic range into intervals, was integrated in a DSE framework to determine better ADFGs faster. The realization of ADFG was based on replacing some of the exact arithmetic nodes by their approximate counterparts. The error estimation technique employed a precharacterization step, which was performed on the arithmetic units in the library to generate a set of LUTs. The LUTs stored the output error of the modules based on the operand errors and interval ranges. From the values extracted from the LUTs, the error was propagated for ADFGs to estimate the output error.

The effect of utilizing four different error models, including the combinations of using two different input bucketizing techniques along with two different error metrics integrated in two modified DSE frameworks were investigated. The efficacies of the proposed methods in the framework were assessed using three image processing benchmarks. The results showed that all methods generated the corresponding PFs with a very good accuracy compared to those obtained by the simulation approach with an average of 98% runtime reduction.

References

[1] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, and D. K. Gracio, "The changing paradigm of data-intensive computing," *Computer*, vol. 42, no. 1, pp. 26–34, Jan. 2009, doi: 10.1109/MC.2009.26.

[2] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surv.*, vol. 55, no. 4, pp. 1–36, Apr. 2023.

[3] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 1–34, Aug. 2017, doi: 10.1145/3094124.

[4] H. Afzali-Kusha, O. Akbari, M. Kamal, and M. Pedram, "Energy and reliability improvement of voltage-based, clustered, coarse-grain reconfigurable architectures by employing quality-aware mapping," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 480–493, Sep. 2018, doi: 10.1109/JETCAS.2018.2856838.

[5] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2013, pp. 1–6, doi: 10.7873/DATE.2013.280.

[6] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012, pp. 796–801, doi: 10.1145/2228360.2228504.

[7] D. Sengupta, F. S. Snigdha, J. Hu, and S. S. Sapatnekar, "SABER: Selection of approximate bits for the design of error tolerant circuits," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6, doi: 10.1145/3061639.3062314.

[8] L. Witschen, H. G. Mohammadi, M. Artmann, and M. Platzner, "Jump Search: A fast technique for the synthesis of approximate circuits," in *Proc. Great Lakes Symp. VLSI*, May 2019, pp. 153–158, doi: 10.1145/3299874.3317998.

[9] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6, doi: 10.7873/DATE.2014.374.

[10] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "AutoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6, doi: 10.1145/3316781.3317781.

[11] S. Xu and B. C. Schafer, "Exposing approximate computing optimizations at different levels: From behavioral to gate-level," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 11, pp. 3077–3088, Nov. 2017, doi: 10.1109/TVLSI.2017.2735299.

[12] M. Barbareschi, F. Iannucci, and A. Mazzeo, "Automatic design space exploration of approximate algorithms for big data applications," in *Proc. 30th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Mar. 2016, pp. 40–45, doi: 10.1109/WAINA.2016.172.

[13] M. Barbareschi, F. Iannucci, and A. Mazzeo, "A pruning technique for B&B based design exploration of approximate computing variants," in *Proc. IEEE Comput. Soc. Annu. Symp. (ISVLSI)*, Jul. 2016, pp. 707–712, doi: 10.1109/ISVLSI.2016.110.

[14] M. Awais, H. G. Mohammadi, and M. Platzner, "LDAX: A learning-based fast design space exploration framework for approximate circuit synthesis," in *Proc. GLSVLSI*, 2021, pp. 27–32.

[15] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 18–30, Jan. 2019, doi: 10.1109/TETC.2016.2598283.

[16] S. Barone, M. Traiola, M. Barbareschi, and A. Bosio, "Multi-objective application-driven approximate design method," *IEEE Access*, vol. 9, pp. 86975–86993, 2021, doi: 10.1109/ACCESS.2021.3087858.

[17] F. Vaverka, R. Hrbacek, and L. Sekanina, "Evolving component library for approximate high level synthesis," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8, doi: 10.1109/SSCI.2016.7850168.

[18] M. Barbareschi, S. Barone, A. Bosio, J. Han, and M. Traiola, "A genetic-algorithm-based approach to the design of DCT hardware accelerators," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, pp. 1–25, Jul. 2022, doi: 10.1145/3501772.

[19] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 187–192, doi: 10.23919/DATE.2017.7926980.

[20] M. Vaeztourshizi, M. Kamal, and M. Pedram, "EGAN: A framework for exploring the accuracy vs. energy efficiency trade-off in hardware implementation of error resilient applications," in *Proc. 21st Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2020, pp. 438–443, doi: 10.1109/ISQED48828.2020.9137041.

[21] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[22] M. Vaeztourshizi and M. Pedram, "An efficient error estimation technique for pruning approximate data-flow graphs in design space exploration," in *Proc. 23rd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2022, pp. 102–107, doi: 10.1109/ISQED54688.2022.9806280.

[23] 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 Trans. Comput.*, vol. 68, no. 1, pp. 21–38, Jan. 2019, doi: 10.1109/TC.2018.2859960.

[24] D. Sengupta and S. S. Sapatnekar, "FEMTO: Fast error analysis in multipliers through topological traversal," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 294–299, doi: 10.1109/ICCAD.2015.7372583.

[25] J. Huang, J. Lach, and G. Robins, "A methodology for energy-quality tradeoff using imprecise hardware," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012, pp. 504–509, doi: 10.1145/2228360.2228450.

[26] J. Huang, J. Lach, and G. Robins, "Analytic error modeling for imprecise arithmetic circuits," in *Proc. SELSE*, 2011, pp. 64–69.

[27] D. Sengupta, F. S. Snigdha, J. Hu, and S. S. Sapatnekar, "An analytical approach for error PMF characterization in approximate circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 1, pp. 70–83, Jan. 2019, doi: 10.1109/TCAD.2018.2803626.

[28] J. Castro-Godínez, S. Esser, M. Shafique, S. Pagani, and J. Henkel, "Compiler-driven error analysis for designing approximate accelerators," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1027–1032, doi: 10.23919/DATE.2018.8342163.

[29] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer, "Statistical quality modeling of approximate hardware," in *Proc. 17th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2016, pp. 163–168, doi: 10.1109/ISQED.2016.7479194.

[30] C. Dharmaraj, V. Vasudevan, and N. Chandrachoodan, "Optimization of signal processing applications using parameterized error models for approximate adders," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 2, pp. 1–25, Mar. 2021, doi: 10.1145/3430509.

[31] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6, doi: 10.1145/2744769.2744863.

[32] W.-T.-J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 47–53, doi: 10.1109/ICCD.2013.6657024.

[33] C. Liu, J. Han, and F. Lombardi, "An analytical framework for evaluating the error characteristics of approximate adders," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1268–1281, May 2015, doi: 10.1109/TC.2014.2317180.

[34] A. Ghasemazar and M. Lis, "Gaussian mixture error estimation for approximate circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 302–305.

[35] 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 Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010, doi: 10.1109/TCSI.2009.2027626.

[36] R. Sedgewick and K. Wayne, *Algorithms*. Reading, MA, USA: Addison-Wesley, 2011.

[37] C. Elkan, "Using the triangle inequality to accelerate K-means," in *Proc. 20th Int. Conf. Mach. Learn.*, 2003, pp. 147–153.

[38] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1161–1173, May 2019, doi: 10.1109/TVLSI.2018.2890712.

[39] (2022). *The USC-SIPI Image Database*. [Online]. Available: http://sipi.usc.edu/database

Marzieh Vaeztourshizi received the B.S. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2014 and the M.S. degree from the University of Southern California, Los Angeles, CA, USA, in 2018, where she is currently working toward the Ph.D. degree in electrical engineering.

Her research interests include approximate computing, low power designs, and design space exploration of approximate circuits.

Massoud Pedram (Fellow, IEEE) received the B.S. degree in electrical engineering from the California Institute of Technology, Pasadena, CA, USA, in 1986, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 1989 and 1991, respectively.

In 1991, he joined the Ming Hsieh Department of Electrical Engineering, University of Southern California (USC), Los Angeles, CA, USA.