A genetic approach for automatic AxC design exploration at RTL based on assertion mining and fault analysis

Alberto Bosio, Samuele Germiniani, Graziano Pravadelli, Marcello Traiola

▶ **To cite this version:**

Alberto Bosio, Samuele Germiniani, Graziano Pravadelli, Marcello Traiola. A genetic approach for automatic AxC design exploration at RTL based on assertion mining and fault analysis. IEEE Transactions on Emerging Topics in Computing, 2025, pp.1-15. ⟨10.1109/TETC.2025.3609050⟩. ⟨hal-05333873⟩

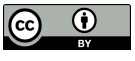
HAL Id: hal-05333873

https://inria.hal.science/hal-05333873v1

Submitted on 27 Oct 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY 4.0 - Attribution - International License

A genetic approach for automatic AxC design exploration at RTL based on assertion mining and fault analysis

Alberto Bosio*, Samuele Germiniani†, Graziano Pravadelli‡ and Marcello Traiola§
University of Verona, Department of Engineering for Innovation Medicine {† ‡}
University of Guglielmo Marconi {†}
Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270 {*}
University of Rennes, Inria, CNRS, IRISA, UMR6074 {§}
Email: *alberto.bosio@ec-lyon.fr, †samuele.germiniani@univr.it, †s.germiniani@unimarconi.it,
‡graziano.pravadelli@univr.it, §marcello.traiola@inria.fr

***Abstract*—In Approximate Computing (AxC), design exploration methods have been introduced to automatically identify approximation targets at the gate level. However, only some of them are applicable at Register Transfer Level (RTL); furthermore, the benefits of combining information from assertions and fault analysis have not been fully explored. This paper proposes a novel methodology for guiding AxC design exploration at RTL considering two approximation techniques: bit-width reduction and statement reduction. Then, it employs fault injection to mimic the approximation effect on the design under approximation. To guide the designer while assessing the approximation choices, assertions, which formally capture the behaviors implemented in the design, are dynamically generated from the RTL simulation traces. Then, the impact of fault injections on the truth values of the assertions is employed as a proxy for measuring the functional accuracy of the corresponding approximations. Based on this evaluation, a genetic algorithm is finally used to rank and cluster the approximation targets, thus providing the designer with an efficient and effective way to automatically analyze AxC variants in terms of the trade-off between accuracy and performance. The experiments carried out on state-of-the-art benchmarks show that the proposed approach represents a promising solution for the automation of AxC design exploration at RTL.**

***Index Terms*—Approximate computing, Fault analysis, Assertion-based verification, Assertion mining, nsga2, Design exploration, Testing, Simulation**

I. Introduction

AxC aims to achieve higher performances in terms of timing, power consumption, and design area with respect to the "classical" computing paradigm, at the cost of a degraded, but still acceptable, output accuracy [1], especially in application fields that have an inherent resiliency to errors, like, for example, audio/video digital signal processing, data analytics, machine learning, web search and wireless communications [2]–[6].

AxC can be applied at several abstraction levels of a given computing system: from circuit to algorithm [1], leading to a broad design exploration space that quickly became the bottleneck of the entire process. Indeed, the literature proposes many works to automatically trade-off between output accuracy and performance [7]. However, most of them cannot accurately identify resilient elements (e.g., HW components, HDL statements, etc.) of the design to be approximated. Consequently, exploring the design to look for approximation candidates may become a long and tedious procedure. In fact, existing approaches usually generate several approximate variants of the Design Under Exploration (DUE). Then, every variant is executed/simulated in order to determine the accuracy degradation [8], measured by employing specific metrics (e.g., similarity index, hamming distance, etc.) that depend on the target application.

Recently, the authors of [9] have presented a method to automatically identify parts of the DUE as approximation candidates that do not significantly compromise the design correctness. Then, they use assertions to evaluate the effect of approximations, i.e., logic formulas that capture the functional behaviors implemented in the design [10]. In this way, there is no need to employ a particular functional metric. In [11], assertions are evaluated on the approximated design for measuring how much the approximation alters the design functionality. However, in these papers, assertions themselves are not exploited to identify the statements of the DUE more suited for the approximation; thus, the AxC design exploration remains a long and challenging task. A first attempt to consider assertions for guiding the approximation is proposed in [12], where a metric is defined by analyzing the syntax tree of the assertions and the variable dependency graph of the DUE; however, this analysis does not consider the semantics of the assertions.

Conversely, in this paper, we semantically exploit assertions, together with fault injection and genetic algorithms, to guide the designer towards the identification of the DUE approximation variants that better fit the target accuracy. In particular, we consider two main approximation techniques, i.e., bit-width reduction and statement reduction; furthermore, we employ fault injection to mimic the effect of such approximations on the DUE. To guide the designer during the evaluation of the different approximation choices, a set of assertions, which formally capture the behaviors actually implemented in the DUE, is dynamically generated from the RTL simulation

traces by using an assertion miner. The assertions are then re-evaluated on the faulty (approximated) designs to analyze the variations of their truth values with respect to the original RTL implementation. Finally, these variations are exploited by a two-objective optimization procedure, which adopts a genetic algorithm to rank and cluster the approximation elements with respect to their impact on the functional behavior of the DUE. In this way, the effect of fault injections on the mined assertions is used as a proxy for estimating the functional accuracy of the corresponding approximations. This provides the designer with an efficient and effective way to automatically analyze AxC variants as a trade-off between accuracy and performance. Overall, the experimental analysis we conducted on four different benchmarks shows that the proposed methodology achieves the following main results.

- By analyzing the impact of fault injections on the assertions, the optimization procedure fastly ranks and clusters the elements (i.e., bits and statements) to be approximated together for any kind of design without requiring the use of a specific approximation metric, which generally depends on the target application.
- From this first result, the optimization procedure can further search for better approximation solutions by considering a user-defined specific error metric. This additional exploration is orders of magnitude faster than directly applying the optimization procedure with the specific error metrics without first analyzing the effect of faults on assertions.
- In terms of functional accuracy, each approximation cluster represents a non-dominated solution by considering the trade-off between cluster size and accuracy error. In addition, these clusters provide good results in terms of power/area/timing reduction.

The rest of the paper is organized as follows. Section II summarises the related work; Section III provides preliminary definitions necessary to understand the technical parts of the paper; Section IV reports our methodology; Sections V and VI detail the experimental results; finally, in Section VII we draw our conclusions.

II. Related work

As stated before, one of the most challenging problems in AxC is selecting the "portion" of the application to be approximated. The portion depends on the abstraction level: it can be an instruction of the source code or a statement in HDL mode. Existing methodologies allow the designer to explicitly select which lines or code blocks to approximate and how. For example, a programmer can annotate through pragmas that a given loop has to be approximated by applying the loop perforation technique [13]. Without annotations, each code line has to be considered as a potential approximation target, thus leading to virtually infinite possibilities of approximations.

At the hardware level, the ABACUS approach [14] works directly with RTL implementations (i.e., HDL code). It relies on a greedy algorithm to find a trade-off between accuracy and power consumption. Other approaches manually identify approximable sub-parts of circuits, mainly focusing on arithmetic components [15]–[17]. Furthermore, another class of promising approaches is the design of approximate systems through Evolutionary Algorithms (EAs) [18]. EAs are well-suited for approximate computing due to their proficiency in stochastic multi-objective optimization, in addition to their ability to manage various constraints. That is true because the problem of approximation can be framed as a multi-objective optimization problem, with accuracy, performance, and resource usage being conflicting design objectives. The outcome of the evolutionary process is a collection of solutions that demonstrate beneficial trade-offs among the main observed objectives. Various studies using EAs for approximation have been carried out, relying on different types of EAs, working at different abstraction levels (e.g., gate, RTL, SW), using different error calculation methods (e.g., simulation, statistical methods, BDD analysis, SAT solving, combinatorial analysis), and targeting different hardware platforms (e.g., processors, microcontrollers, FPGA, and ASIC) [8], [19]–[29]. Nevertheless, to the best of our knowledge, none of these takes advantage of assertions and fault injection to guide the designer through the AxC exploration. On the other hand, even if they aim at different goals, other approaches have been proposed for applying verification techniques to approximate computing. In [30], the authors present an AxC-based approach to achieve a fast and accurate enough repeated execution for security verification. In [31], the authors propose a dynamic verification methodology to assess the quality of the approximated circuit by exploiting mutations of test patterns and coverage information. However, none of the previous verification-oriented methodologies uses assertions to guide the design exploration. In [9], the authors introduce a verification-guided method to automatically identify program blocks suited for approximation while avoiding significant compromises on program correctness. The method identifies regions of code that are less influential for the computation of the program outputs and, therefore, more approximable. In particular, they generate a statement ranking based on whether an instruction affects a particular primary output of the designs. Assertions are then used to evaluate the impact of the approximation on the functional behavior of the final design. It is important to note that assertions are not directly used for statement identification (i.e., identify which portion of the code has to be approximated) as we do in this paper.

III. Preliminary definitions

We report hereafter the definitions of concepts used in the rest of the paper.

Definition 1: Given a finite sequence of time units $\langle t_1, ..., t_n \rangle$ and a set of variables $\{v^1, ..., v^m\}$, a **trace** is a sequence of tuples $(t_i, v_i^1, ..., v_i^m)$ such that v_i^j is the value assumed by variable v^j at time t_i.

Definition 2: An **assertion** is a Linear Temporal Logic (LTL) formula of the form $always(antecedent \rightarrow consequent)$.

We kindly refer the reader to [32] for a complete description of LTL operators and semantics according to the Property Specification Language (PSL).

Definition 3: Given a trace tr of length n and an assertion as, an **evaluation** of as with respect to tr is the sequence of

evaluation units $\langle e_1, ..., e_n \rangle$, where e_i is the truth value of the assertion at instant t_i.

Definition 4: An assertion **holds** in a trace if and only if its evaluation does not contain any evaluation unit whose value is false.

Definition 5: Given a trace tr, and an assertion as, a **contingency table** is a 3 × 3 matrix displaying the frequency distribution of $true$, $false$ and $unknown$ evaluation units of the antecedent with respect to the consequent of as in tr. A graphical representation of a contingency table is reported in Fig. 1, where ATCT corresponds, for example, to the number of instants in a trace where both the antecedent and the consequent evaluate to *true*.

	Cons. true	*Cons. false*	*Cons. unknown*
Ant. true	**ATCT**	**ATCF**	**ATCU**
Ant. false	**AFCT**	**AFCF**	**AFCU**
Ant. unknown	**AUCT**	**AUCF**	**AUCU**

Figure 1: Contingency table.

Definition 6: An **Augmented Contingency Table (ACT)** enriches each element of the contingency table with the set of time units $T = \{t_1, t_2, t_k\}$ in which the corresponding evaluations for antecedent and consequent occur in the trace. For example, the field ATCF is extended with the list of time units where the antecedent evaluates to *true* and the consequent to *false*. In the rest of the paper, we will refer to the enriched fields of an ACT by adding the aug subscript (e.g., $ATCF_{aug}$).

IV. Methodology

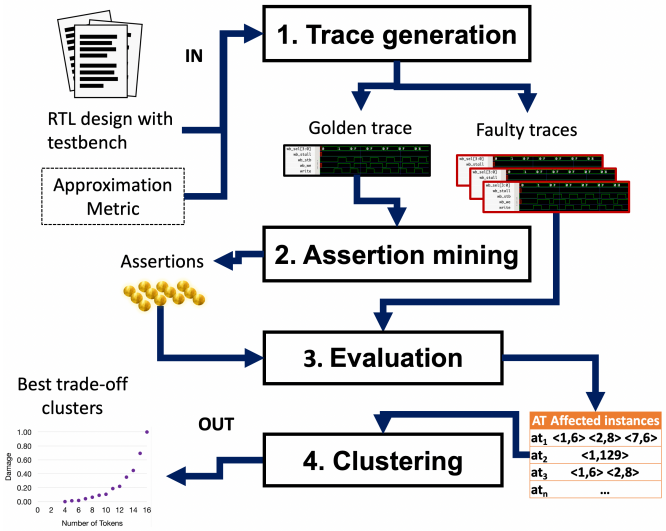


Figure 2: Overview of the methodology.

Our methodology is intended to provide the designer with an automatic way to explore AxC alternatives on RTL descriptions. Two approximation strategies have been considered:

- **Bit-width reduction:** fixing to a constant value one or more bits on a subset of design signals/registers;
- **Statement reduction:** removing one or more statements.

The methodology consists of the four sequential steps shown in Fig. 2. The input is the RTL description of the DUE together with a suitable test bench. Additionally, the user can provide (not mandatorily) an error metric (i.e., the Approximation metric). The output is a list of clusters of approximable DUE elements of increasing size; each cluster is a non-dominated solution representing a trade-off between the approximation error and the number of approximable elements in the cluster. In the rest of the paper, we refer to such elements with the term Approximation Tokens (ATs). In particular, according to the addressed AxC strategies, we consider the following kinds of ATs:

- **Statement token:** an instruction appearing in the RTL description of the DUE;
- **Bit token:** a bit of a signal/register occurrence appearing in a statement token.

Hereafter, we provide an overview of the four main steps of the methodology; a more detailed description is reported in the following sections.

1) **Trace generation:** in the first step of the methodology, we dynamically simulate the DUE and its approximations to generate a set of execution traces. First, we generate the golden trace by simulating the original implementation of the DUE. Then, for each AT, we generate a trace that reflects the effect of activating the corresponding approximation (i.e., bit-width reduction for bit tokens and statement reduction for statement tokens). These traces are obtained by simulating a faulty version of the DUE. In particular, stuck-at faults are injected in specific elements of the original implementation to exactly mimic the effect of approximating each AT.
2) **Assertion mining:** in the second step of the methodology, we employ an assertion miner to generate assertions holding on the golden trace. The assertions predicate on the inputs and outputs of the DUE and capture its golden behavior.
3) **Assertion evaluation:** In the third step of the methodology, we re-evaluate the assertions mined in step 2 on the faulty traces obtained in step 1. By comparing the variations on the ACT (def. 6) of each assertion evaluated on the golden and faulty traces, we automatically estimate the approximability of each AT in terms of how it affects the functionality of the original design. In this way, we provide the user with an assertion-based metric that we call "damage".
4) **Clustering:** As approximating only a single AT cannot provide significant benefits, in the last step, we provide a mechanism to cluster a set of ATs and estimate the effect of their simultaneous approximation. This is achieved by employing the Non-dominated Sorting Genetic Algorithm Non-dominated Sorting Genetic Algorithm II (NSGA2) to find non-dominated solutions that provide the best trade-off between the size of the clusters and the effect on the assertions of simultaneously approximating all the tokens in the cluster. At this point, while NSGA2 initially exploits our assertion-based metrics, if the user provides

an error metric as input, the search for non-dominated solutions is further improved by considering such a specific metric. Finally, the designer can choose the largest approximation cluster that better fits the tolerable error.

To simplify the detailed description of each step, in the following sections, we refer to the running example reported in Algorithm 1 containing the RTL implementation of an 8-bit adder. It takes as input two 8-bit unsigned integers (signals a and b) and returns their sum as output (out port).

Algorithm 1 Running example

```
1: module adder(a, b, clk, out)
2: input [7:0] a, b
3: input clk
4: output [8:0] out
5: reg [8:0] sum
6: always @(posedge clk)
7: begin
8:       sum = a + b
9: end
10: endmodule
```

A. Trace generation

In the first step of the methodology, we simulate the DUE to generate a set of execution traces. We assume that the test bench thoroughly stimulates the design, purposely covering all its functional behaviors. This is a desirable condition for any dynamic verification approach.

First, we generate the golden trace by simulating the original implementation. Then, for each target AT, we generate a trace reflecting the effect of its approximation according to either the bit-width reduction or the statement reduction strategy.

For each class of AT, we identify a fault model to mimic the effect of its approximation in the functional behavior of the DUE as follows:

- A bit token is approximated by injecting a stuck-at 0/1 on the target bit. Stuck-at X (i.e., unknown logic value) is not considered as the propagation of X values along the execution traces would prevent the evaluation of mined assertions in step 3.
- A statement token is approximated differently depending on the type of statement as follows:
  - assignment: the statement is removed; in case its left-hand side remains undefined, its value is assigned to 0 for bit-vectors and numeric types, stuck at 0/1 for a single bit/Boolean;
  - module instantiation: the statement is removed; in case the signals connected to the outputs of the module remain undefined, they are treated as in the case of assignments;
  - conditional statement: either the true or the false block is removed; in case any signal/register remains not assigned, it is treated as in the case of assignments.

In the running example, the statement $sum = a + b$ at line 8 is a statement token, while the third bit of signal a (i.e., $a[2]$) is considered a bit token. To inject a fault for the first bit token of signal a, a stuck-at 0 fault is inserted on the first bit of the signal (ex. a & 11111110). In this example, the only fault injected to approximate the only available statement token would consist of removing the instruction at line 8.

B. Assertion mining

In the second step of the methodology, we automatically mine assertions holding on the golden trace of the original DUE implementation. To extract them, we exploit the state-of-the-art and open-source assertion miner HARM [33], [34]. The miner is configured to automatically generate assertions on primary inputs and outputs of the DUE in the form $always(antecedent \rightarrow next[N](consequent))$, where N is the design depth, that is, the maximum number of clock cycles necessary to propagate the effects of primary inputs toward primary outputs. This is done to ensure that the mined assertions represent meaningful I/O relations. Of course, the mined assertions depend on the quality of the testbench; if the testbench exercises all the DUE behaviors, the mined assertions will formally capture them. Both the *antecedent* and the *consequent* of each assertion are instantiated by HARM following the form $prop_1$ && $prop_2$ && ... && $prop_k$. Each proposition $prop_i$ is in the form $c_l <= var_j <= c_r$, or $var_j == c$, or $var_j <= c$, or $var_j >= c$, where var_j is an input or an output of the DUE if located, respectively, in the *antecedent* or the *consequent*, while c_l, c_r and c are numeric constants. The propositions are automatically generated by the tool; therefore, the user only has to provide the inputs and outputs of the DUE. Propositions instantiating the antecedent (consequent) are generated from the input (output) signals of the DUE.

Finally, we rank the assertion set according to a score S, which is obtained for each assertion a by combining the following metrics:

- m_1: $Support(a) = ATCT/traceLength$;
- m_2: $Causality(a) = 1 - AFCT/traceLength$;

where, $ATCT$ and $AFCT$ are derived from the contingency table of a (see Def. 5).

The overall score S is, then, calculated for each assertion a, through the formula $\prod_{i=1}^{2} calibrate(sm_i(a)/sm_i(a_{max\_i}))$, where $sm_i(a)$ is the score of a by using the metric m_i, a_{max_i} is the assertion that yields the maximum score by using metric m_i, and $calibrate$ is a procedure that "calibrates" the input score by using the function $R(x) = 1/(1 + e^{(3.3-10.62x)})^2$. This function is a modified version of the Richards' curve that ranges from 0 to 1.

The intuition behind this ranking formula is that we want to **allow the simultaneous employment of two metrics in a single ranking procedure**. Furthermore, we want to give more importance to assertions presenting a higher score for both sorting metrics while penalizing assertions that score well only in one or none of them. After the ranking procedure is completed, we keep only the assertions whose score S is greater than 0.

Figure 3 shows some of the assertions mined for the running example and the corresponding values for $Support$, $Causality$ and S.

C. Assertion evaluation

In the third step of the methodology, the generated assertions are re-evaluated on the faulty traces obtained by perturbing the original RTL description of the DUE while adopting the bit-width and the statement reduction strategies, as reported in Section IV-A. In particular, for each AT, the corresponding fault is injected, and the assertions are re-evaluated, generating a new ACT. It is worth noting that the set of time units in element $ATCF_{aug}$ is empty in the ACT of any assertion evaluated on the original implementation (as all assertions hold on the golden trace), while it is likely not empty for assertions affected by the presence of a fault. Consequently, we can observe variations in the composition of $ATCT_{aug}$ as well. The purpose of this procedure is then to analyze the effect of different approximation alternatives (represented by ATs) on the functional behaviors (captured by the assertions) of the design.

The procedure is implemented in the $evaluate$ function reported in Algorithm 2, whose behavior is detailed hereafter.

Algorithm 2

```
1:  function EVALUATE(A, gt, FT)
2:      GACT ← ∅
3:      diff ← ∅
4:      for all a ∈ A do
5:          GACT[a] ← genAugContingency(gt, a)
6:      for all ft ∈ FT do
7:          for all a ∈ A do
8:              fct ← genAugContingency(ft, a)
9:              diff_an ← annotate(a, GACT[a] − fct)
10:             diff[ft] = diff[ft] ∪ (diff_an)
11:     return diff
```

The function takes as inputs A, gt and FT, where A is the set of assertions mined in the previous step, gt is the golden trace, and FT is the set of all faulty traces generated in the first step of the methodology. Note that for each $ft \in FT$, there exists a corresponding unique approximation token at and a unique fault f. The function returns a dictionary $diff$, which stores, for each ft, a matrix of sets, where each set contains the differences between the ACT achieved on ft and gt for all assertions belonging to A.

First, we initialize variables $GACT$ and $diff$ (lines 2-3), where $GACT$ is intended to store the augmented contingency tables of the golden trace. Then, we iterate over the assertions in A such that the function genAugContingency(gt, a) evaluates the assertion a on the trace gt to retrieve the corresponding ACT (line 5).

N	Assertions	S	Causality	Support
1	G((**a** >= 58 && **a** <= 60) && (**b** >= 23 && **b** <= 63) |-> **out** >= 81 && **out** <= 126)	0.98	0.91	0.68
2	G((**a** >= 4 && **a** <= 63) && (**b** >= 61 && **b** <= 63) |-> **out** >= 65 && **out** <= 126)	0.96	0.64	1.00
...	...	...	...	...
38	G((**a** >= 28 && **a** <= 29) && (**b** >= 34 && **b** <= 35) |-> **out** >= 64 && **out** <= 73)	0.1	0.67	0.18

Figure 3: Assertions mined for the running example.

After that, for each couple $\langle ft, a\rangle \in FT \times A$, a "faulty" ACT is generated and stored in fct (line 8). ftc is then compared with the golden contingency table $GACT[a]$; this is done by returning the set difference between the augmented fields in the matrices (line 9). After that, each field in the matrix of differences is annotated using function $annotate$ (line 9). The $annotate$ function takes as input an ACT and an assertion a, for each field of the ACT of the form $\{t_1, t_2, ...t_k\}$ it annotates each evaluation instant with the corresponding assertion a, returning a set of the form $\{\langle a, t_1\rangle, \langle a, t_2\rangle, ..., \langle a, t_k\rangle\}$. The annotation procedure is necessary to keep track of the variations occurring for different assertions but at the same instant of the trace. The result of this operation is stored in $diff[ft]$ (line 10), which contains the impact of fault f for all the considered assertions. Finally, the differences stored in $diff$ are returned (line 11).

Since each matrix belonging to $diff$ contains 9 fields (3x3 matrix), there are several ways to estimate the impact of a fault f on the functional behavior of the design. In this work, we decided to rank each approximation token at by using the instances in which the corresponding fault f made the assertions in A fail. That is, each at is ranked by using the $ATCF_{aug}$ field (first row, second column) of the corresponding matrix $diff[ft]$. This choice is plausible because the higher the increment in the size of $ATCF_{aug}$, the worse the impact on the functional behavior, and as a consequence, the lower the approximability of the corresponding approximation token. In this context, we introduce the concept of damage caused by an AT, and more generally by a set of ATs, approximated on the DUE, as follows. The output of this step of the methodology is the damage caused by each single AT.

Definition 7: Given a set of k approximation tokens $ATS = \{at_1, at_2, ..., at_k\}$ corresponding to a set of faulty traces $FT = \{ft_1, ft_2, ..., ft_k\}$, the approximation **damage** of ATS is the number of unique couples $\langle a, t\rangle$ in the set $DIFF = \bigcup_{j=1}^{k} diff[ft_j].ATCF_{aug}$ where $diff[ft_j].ATCF_{aug}$ is the set of time instants of trace ft_j where the assertions fail (antecedent true implies consequent false).

To better understand the concept of damage, consider the example in Fig 4. The damage for the approximation tokens at_0 and at_1 is 3 and 2 as there are, respectively, 3 and 2 unique failing instances $\langle a, t\rangle$ in $DIFF$. However, the joint damage for the set containing both the ATs is 4, as they share the failing instance $\langle 23, 893\rangle$.

AT	$DIFF$	Damage
at_0	<23,893><103,776><65, 399>	3
at_1	<23,893><1,6>	2
$\{at_0, at_1\}$	<23,893><103,776><65, 399><1,6>	4

Figure 4: Example of damage.

D. Clustering

The output of the previous step provides an initial assessment of the approximability of the design for each AT. However, any meaningful approximation would require approximating multiple ATs at the same time. Therefore, in the

last step of the methodology, we apply a clustering algorithm to group ATs exposing a similar approximability to help the designer in the process of simultaneously approximating multiple ATs.

We employ NSGA2 to cluster the ATs. NSGA2 is a popular multi-objective optimization algorithm that uses genetic operators such as crossover and mutation to evolve a population of candidate solutions. We specialized NSGA2 to retrieve the *non-dominated set of solutions* containing the best trade-offs between the size and the damage of the cluster.

In this work, we postulate that the damage caused by approximating a cluster of ATs is a good estimator of the approximation effect on the functional accuracy of the DUE. Intuitively, ATs damaging the design in the same way (i.e., making the same assertions fail on the same instants of the trace) can be approximated together to produce a lower overall effect on the design functionality. As a consequence, the best clusters are those that contain many ATs with overlapping failing instances $\langle a, t\rangle$ in $DIFF$, as this reduces their damage.

Our clustering approach is divided into two main phases.

- Phase I: First, we run NSGA2 to quickly find an initial set of solutions that maximize the size of the clusters while minimizing the damage.
- Phase II: After that, if the user provided an error metric, we continue the NSGA2 search with the population so far returned, but with a new optimization objective, i.e., minimizing the user-defined error metrics while maximizing the size of the clusters. The second phase is slower and more computationally expensive than the first, as the error of the cluster is retrieved by re-simulating the design while simultaneously injecting all the faults corresponding to the ATs in the cluster.

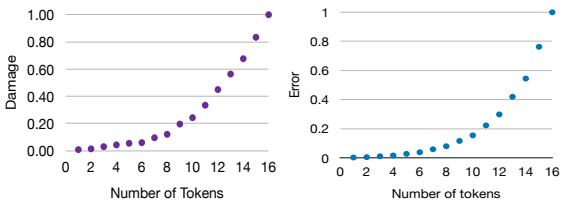
Thus, the purpose of Phase I is to quickly identify a non-dominated solution front by adopting an assertion-based metric, i.e., the damage, that does not require expensive re-simulations of the DUE. As shown in the experimental analysis, this generally achieves excellent results. Phase II aims to provide the user with the possibility of improving the initial solution by exploiting their preferred metrics. While this second search is slower than the first, its execution time is anyway orders of magnitude lower than running NSGA2 to minimize the user-define metric from the very beginning without starting from the population returned with the initial damage-based search.

We report below how we implemented each operator of the NSGA2 algorithm.

1) *Initialisation:* the algorithm starts by initialising a population of candidate solutions randomly. Our algorithm starts with a population of random clusters of size 1 because we want to gradually build bigger clusters by merging the best ones. The dimension of the population heavily affects the computational speed of the algorithm. We start with a population size equal to the total number of tokens, which is considered a good compromise between the dimensions of the search space and the computational load.
2) *Evaluation:* each solution in the population is evaluated and assigned to a fitness value based on how well it satisfies the objectives. In the first phase, our fitness value is the couple $\langle size, damage\rangle$ corresponding to the objectives we are trying to optimize; in the second phase, the fitness value is the couple $\langle size, user\_metrics\rangle$.
3) *Non-dominated sorting:* the solutions in the population are sorted into different non-dominated fronts. A solution is said to dominate another solution if it is better than the other solution in at least one objective and not worse in any other objective. The first front consists of solutions that are not dominated by any other solution; the second front consists of solutions that are not dominated by any solution in the first front, and so on.
4) *Tournament selection:* the next generation population is selected by combining the solutions from the different fronts based on their non-dominated rank and crowding distance. Solutions with a higher rank and a larger crowding distance are preferred. The crowding distance of each solution in a front is calculated based on its proximity to other solutions in the front. Solutions that are closer to others have a smaller crowding distance.
5) *Crossover:* the selected solutions are then subjected to crossover to create new offspring solutions. Each offspring is the cluster of tokens obtained by merging the tokens of the two parent clusters.
6) *Mutation:* in our implementation of NSGA2, we mutate the entire population instead of the single individual. In particular, for each individual Ind of the population, there are two options with 50% probability each:
   - We generate a new individual by adding a random token to Ind; this helps in recovering tokens that become extinct in previous iterations but that can be useful in the current population.
   - We generate a new random individual of the same size as Ind; this helps in eliminating individuals that perform worse than random choices, as they will most likely be discarded by the selection of the following iterations in favor of the random ones.
7) *Termination:* the algorithm terminates when a stopping criterion is met. State-of-the-art halting criteria include maximum time reached and convergence. Our algorithm halts when the last iteration improves the dominance of the last front for less than a user-defined percentage value. In this context, the **dominance** of the front is the percentage of the dominated surface with respect to the total search space. This calculation is possible because the boundaries of the search space can be inferred by using:
   - the empty cluster that does not produce any error;
   - the cluster containing all ATs, thus producing the maximum error.

   For the second phase, the user can also specify a maximum execution time.

Fig 5 reports the result of the above procedure with the running example. Fig 5 (1) shows the non-dominated solution front generated at the end of Phase I with the damage-based metrics, while Fig 5 (2) shows the front after Phase II, where the user-defined metrics was adopted. It is easily



(1) (2)

Figure 5: Non-dominated solution fronts by using the damage-based metrics (1) and the user-defined metrics (2) for the running example showed in Algorithm 1.

noticeable that the front in Fig 5 (1) is a good estimator of the front obtained with the user-defined metric in Fig 5 (2). In this running example, Phase II could not improve the solution front further. The achieved front correctly classifies the approximation tokens: as expected, all the clusters with a low error contain ATs affecting the least-significant bits of the signals in the statement at line 8 of Algorithm 1.

E. Scalability

In this section, we comment on the scalability of our approach. The complexity of the evaluation phase (Section IV-C) depends on the length $traceLength$ of the trace, the number $|A|$ of mined assertions, and the number $|AT|$ of approximation tokens. Formally, the temporal complexity of this procedure is then $O(|A| \times |AT| \times |traceLength|)$. The complexity of the clustering procedure (Section IV-D) depends on the number $|DIFF|$ of couples $\langle a, t \rangle$ in $DIFF$, the size $|P|$ of the population, and the number NI of iterations of $NSGA2$. Formally, the temporal complexity of this procedure is then $O(|NI| \times |P| \times |DIFF|)$, where the higher the value of $|AT|$, the higher the number of required iterations to achieve a satisfying result.

V. Experiments using bit-width and statement reduction

In this section, we focus on the evaluation of our methodology for the techniques of bit-width and statement reduction. The experiments have been carried out on a 3.5 GHz AMD Ryzen 3950x processor equipped with 32 GB of RAM (3600 MHz) and running Ubuntu 22.04 LTS. The tool and the benchmarks are freely available at [35].

We evaluated the proposed approach on the following RTL Verilog designs [36], [37]:

- BrentKung32b: 32 bit integer adder;
- InverseKin: inverse kinematics for a 2-joint arm;
- Sobel: edge detection filter for 1024x1024 pictures.
- ffnn: feed-forward neural network

With regard to the error metrics, we adopted the Structural Similarity (SSIM) index [38] for the Sobel filter and the classical Average Relative Error (ARE) for the other three designs.

The characteristics of the four designs are reported in Table I in terms of Area, Power, and Timing of the original precise (i.e., non-approximate) designs, the number of lines of code (Lines), the number of inputs/output (I/O), and the number of considered ATs (number of ATs) with respect to the adopted AxC technique, i.e., Bit-width Reduction (BR) or Statement Reduction (SR). For each benchmark, we considered a testbench generating a golden trace 1000 clock-cycle long. Furthermore, we used the top 1000 assertions generated by the HARM assertion miner. For the clustering procedure, phase 1, with the damage-based metrics, halted when the dominance increment was lower than 0.1%; for Phase II, with the user-defined metrics, we set a time limit of 10 minutes in addition to the previous condition.

Design	Precise (i.e. non-approximate)			Lines	I/O	Num of ATs	AxC technique
	Area (μm^2)	Power (mW)	Timing (ns)				
BrentKung32b	813.77	0.3263	1.15	735	4	656	BR
						564	SR
InverseKin	143001.81	30.59	2.44	226	6	773	BR
						65	SR
Sobel	1972.94	1.29	1.20	203	11	256	BR
						38	SR
ffnn	50491.99	27.49	0.10	430	7	2460	BR
						219	SR

Table I: Characteristics of the benchmarks.

We applied both the bit-width and the statement reduction approximation strategies on a set of ATs (i.e., bit tokens and statement tokens) by injecting faults as reported in Section IV-A. We then evaluated and clustered the ATs, as detailed in Sections IV-C and IV-D, by exploiting a set of assertions mined as described in Section IV-B. In this way, we obtained the non-dominated solution fronts for both the damage-based metrics and the user-based metrics. We finally evaluated the approximation effect of the cluster belonging to the non-dominated fronts in terms of functional accuracy and power/area/timing reduction. The goal is to show the effectiveness and efficiency of our approach in prompting the designer to simultaneously approximate a cluster of ATs that has a low impact on functional accuracy while guaranteeing a significant saving in terms of area and power.

A. Functional accuracy

In this section, we evaluate the effectiveness of the proposed approach from the point of view of the functional accuracy of the approximated designs. For each design and for each considered approximation technique, we report 2 charts in Figures 6, 7, 8 and 9 related to the analysis of the functional accuracy. The other 3 charts in the same figures are referred to the savings in terms of area/power/timing.

Charts (1) and (6) report the non-dominated solution fronts obtained by applying our methodology with the damage-based metrics (Phase I of Section IV-D) for the bit-width reduction and the statement reduction, respectively. Each point in these charts is a cluster, i.e., a set of ATs presenting the best trade-off between size and damage. In particular, on the x-axis, we report the number of tokens contained in each cluster C of the front. On the y-axis, we report the value of the damage obtained by simultaneously approximating all ATs included in C; the damage is normalized between 0 and 1 to improve readability.

Charts (2) and (7) refer instead to the functional accuracy of the approximated designs for bit-width reduction and statement reduction measured with the user-defined metrics. They

show three kinds of information:

- Violet points (Before push) refer to the values of the user-define metrics calculated for each cluster belonging to the non-dominated solution returned by Phase I of Section IV-D.
- Blue points (After push) refer to the values of the user-defined metrics calculated by further pushing the NSGA2 as described in Phase II of Section IV-D starting from the population corresponding to the non-dominated solution front returned by Phase I (i.e., from the violet points).
- Red points (Random) refer to the values of the user-defined metrics calculated on a set of clusters composed of randomly selected ATs, whose sizes exactly equal those of clusters corresponding to the blue points. The error plotted for each random cluster is generated by averaging the error of 100 random clusters for each size.

Overall, we can make three main observations, which are valid for all considered benchmarks and for both bit-width and statement reduction approximation techniques:

1) The trend of the violet points in Charts (2) and (7) is "similar"[1] to that showed in Charts (1) and (6), thus confirming that the non-dominated solutions generated by using the damage-based metrics are a good estimation of the non-dominated solutions obtainable by adopting the user-define metrics, i.e., the damage-based metrics is a very good proxy for the user-defined metrics.
2) Phase II of the clustering process barely improves the fronts provided by Phase I, showing the goodness of the initial estimation at the end of Phase I. This is, in particular, highlighted in Table II, where the dominance of Phase I and Phase II are compared.
3) The non-dominated solutions returned by our approach definitely perform better than the random selection, as the random case always produces a higher error for the same cluster dimension.

Design	Technique	Dominance after Phase I	Dominance after Phase II	Increment
BrentKung32b	BR	80%	84%	4%
	SR	74%	77%	3%
InverseKin	BR	82%	86%	4%
	SR	74%	76%	2%
Sobel	BR	39%	48%	9%
	SR	37%	40%	3%
ffnn	BR	65%	66%	1%
	SR	49%	49%	0%

Table II: Dominance achieved before and after pushing the front with Phase II of the clustering procedure.

B. Area, power consumption, and timing

We measured the effects on hardware attributes of applying the clusters of ATs obtained with the proposed approach. We synthesized the approximate designs and estimated relative area, power and timing compared to the original (Precise) design, as $1 - \frac{Precise - cluster_i}{Precise}$ (the *Original* design has value 1). We targeted the FreePDK45 [39] 45-nm standard cell technology library and resorted to the synthesis tool reports to extract area, power, and timing estimations. Charts (3)-(5) and (8)-(10) of Figures 6, 7 and 8 report the results, in terms of the trade-off between Error and relative area, power consumption, and timing. Note that, for the Sobel benchmark (Figure 8), on the y-axis, we show the SSIM to be consistent with other results. The blue points in the charts report the trade-offs for the non-dominated solutions obtained at the end of Phase II of the Clustering procedure; the red points represent the results for randomly chosen solutions. The dimension of the points follows the number of ATs in the corresponding cluster, as reported in the Figure legend. The precise version always has Relative Area/Power/Timing equal to 1 and Error equal to 0 (SSIM equal to 1, for Sobel), and it is represented as a green star.

First of all, we notice that the solutions achieved through the proposed approach (blue points) generally achieve better trade-offs for the three attributes compared to a random choice. Indeed, for the same error/SSIM (y-axis), blue points show generally reduced area/power/timing (x-axis) and, for the same area/power/timing (x-axis), blue points show generally reduced error/higher SSIM (y-axis). Moreover, we observe that an increasing cluster dimension (the number of tokens) corresponds to a reduced area/power/timing.

Overall, the results confirm that our approach generally provides a reliable guide to finding good trade-offs in terms of functional accuracy and area/power/timing savings.

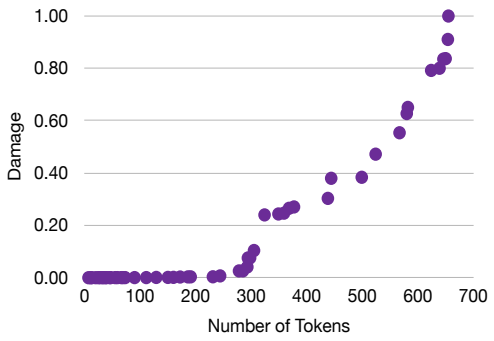
C. Execution time

In this section, we analyze the execution time of our approach. Columns in Table III report, for each benchmark (Design) and each approximation technique (Tech), the time required to fault simulate the DUE and generating the corresponding faulty traces per each fault mimicking each target AT (Time to simulate), the time to automatically generate the assertions (Time to mine), the time to evaluate the assertions on the faulty traces (Time to evaluate), the time to cluster the ATs with Phase I and Phase II of Section IV-D (Time to cluster Phase I/II), and the total time (Tot). Fault simulation is the most expensive step, as it depends on the number of considered faults (i.e., the number of ATs), while the clustering procedure by adopting the damage-based metrics (i.e., Phase I) is very fast. It is worth noting that Phase II is significantly slower than Phase I. Thus, given the limited improvement that Phase II guarantees w.r.t. Phase I, as shown by Table II, it is not strictly necessary to perform Phase II with the user-defined metrics. In addition, we want to emphasize that running NSGA2 would be orders of magnitude slower if it was asked to minimize the user-define metric without starting from the population returned by Phase I. In fact, reaching the front, represented by blue points in the charts, without applying Phase I required more than 8 hours for all benchmarks. This definitely highlights the efficiency of our damage-based approach.

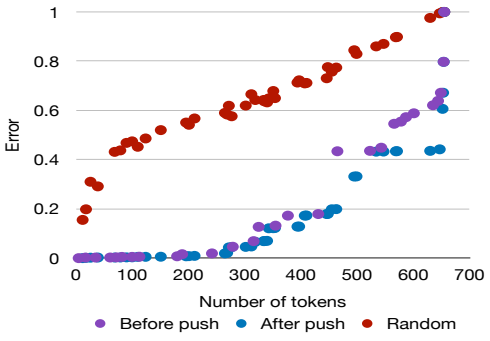
D. Correlation between damage and the real metric

In this section, we show the correlation and confidence between our "damage" metric and the real metric used to assess the error.

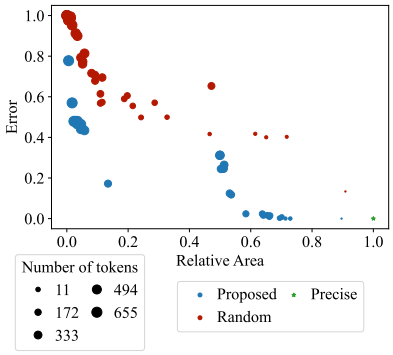
[1]Please note that for the Sobel benchmark, the higher the SSIM, the lower the error. Thus, we expect the trend of SSIM w.r.t. the damage to be specular.



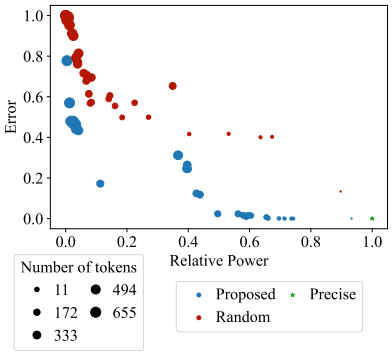
(1) BR for BrentKung32b
Size vs. Damage



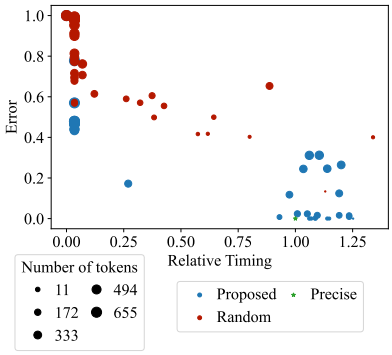
(2) BR for BrentKung32b
Size vs. Error



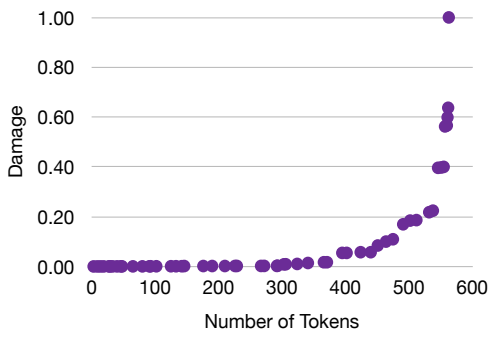
(3) BR for BrentKung32b
Area vs. Error



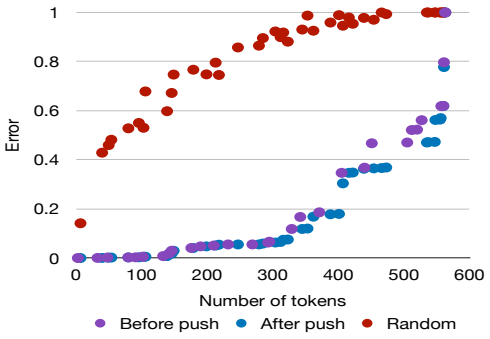
(4) BR for BrentKung32b
Power vs. Error



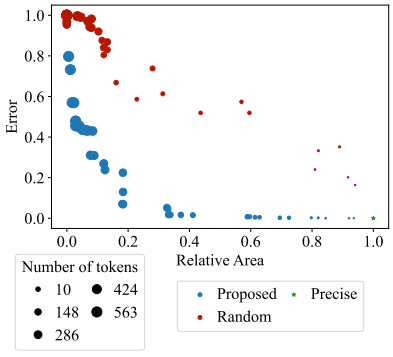
(5) BR for BrentKung32b
Timing vs. Error



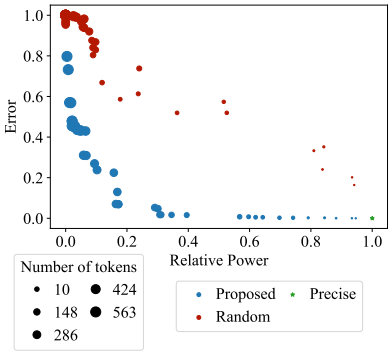
(6) SR for BrentKung32b
Size vs. Damage



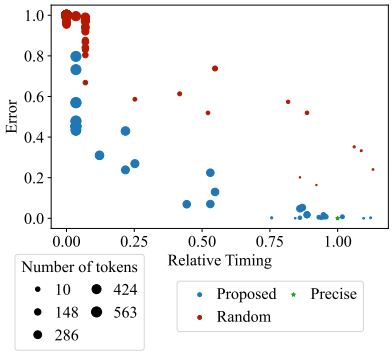
(7) SR for BrentKung32b
Size vs. Error



(8) SR for BrentKung32b
Area vs. Error

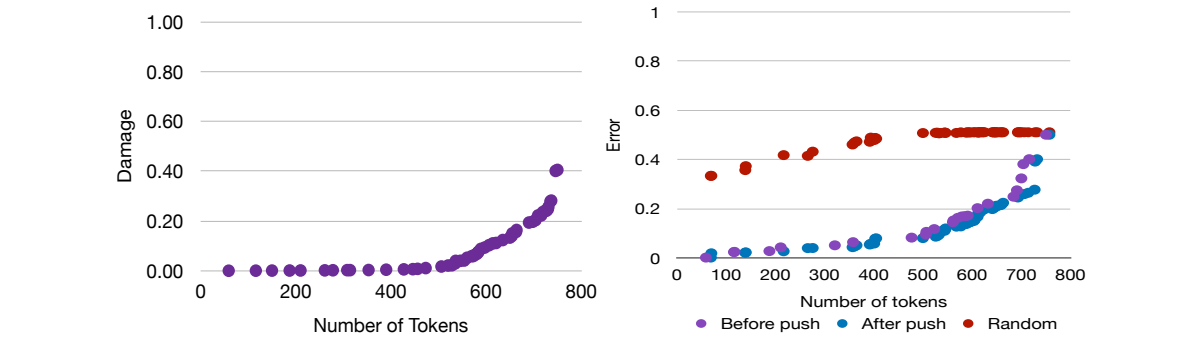


(9) SR for BrentKung32b
Power vs. Error



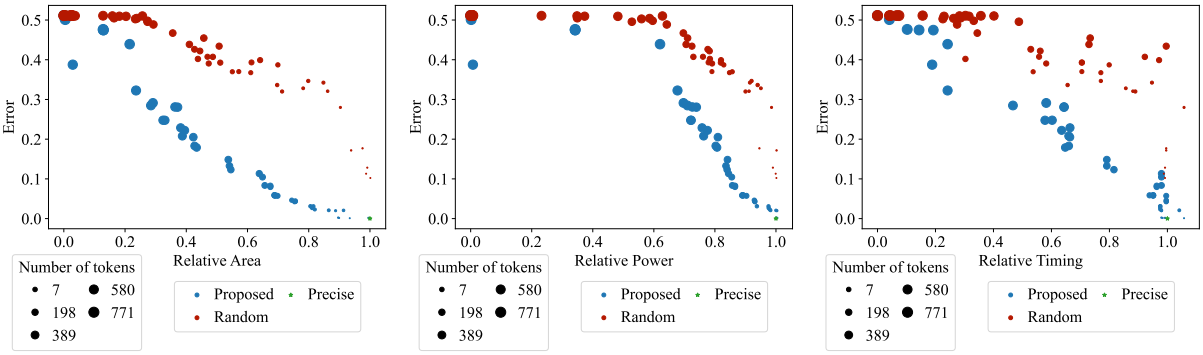
(10) SR for BrentKung32b
Timing vs. Error

Figure 6: Results for the BrentKung32b design. Charts 1-2 and 6-7 show the best trade-off clusters in terms of damage and error. Charts 3-5 and 8-10 show the best trade-off clusters in terms of area, power and timing.



(1) BR for InverseKin
Size vs. Damage

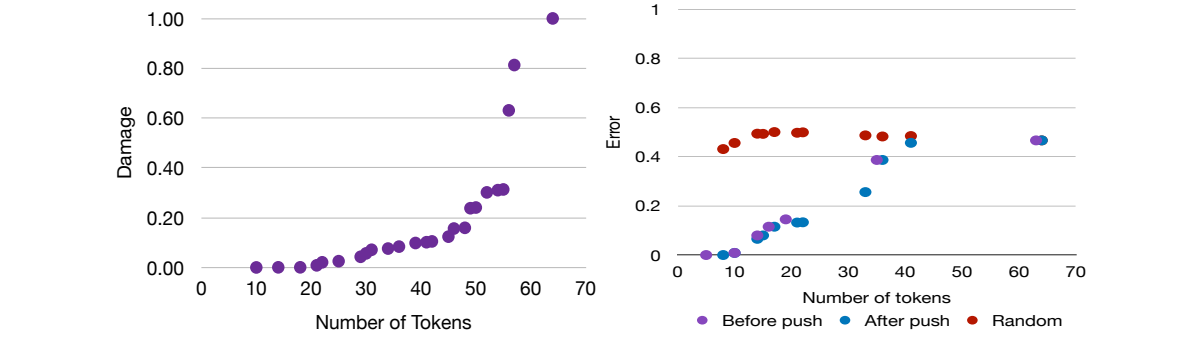
(2) BR for InverseKin
Size vs. Error



(3) BR for InverseKin
Area vs. Error

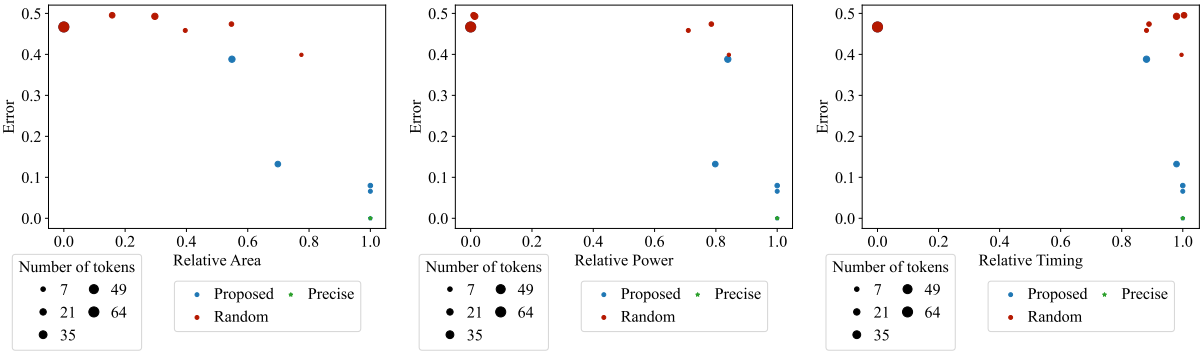
(4) BR for InverseKin
Power vs. Error

(5) BR for InverseKin
Timing vs. Error



(6) SR for InverseKin
Size vs. Damage

(7) SR for InverseKin
Size vs. Error

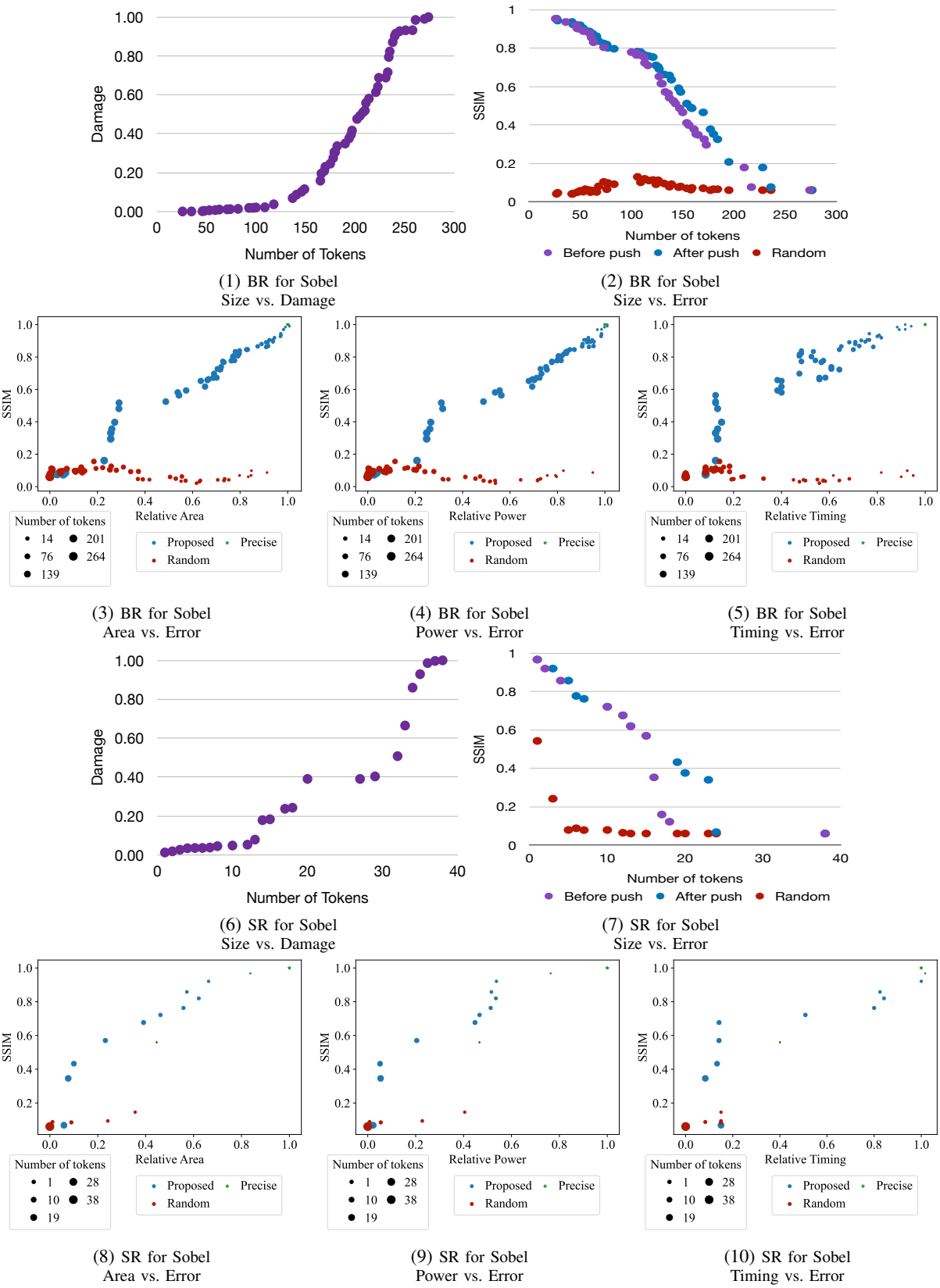


(8) SR for InverseKin
Area vs. Error

(9) SR for InverseKin
Power vs. Error

(10) SR for InverseKin
Timing vs. Error

Figure 7: Results for the Inverse Kinematic design. Charts 1-2 and 6-7 show the best trade-off clusters in terms of damage and error. Charts 3-5 and 8-10 show the best trade-off clusters in terms of area, power and timing.



(1) BR for Sobel Size vs. Damage

(2) BR for Sobel Size vs. Error

(3) BR for Sobel Area vs. Error

(4) BR for Sobel Power vs. Error

(5) BR for Sobel Timing vs. Error

(6) SR for Sobel Size vs. Damage

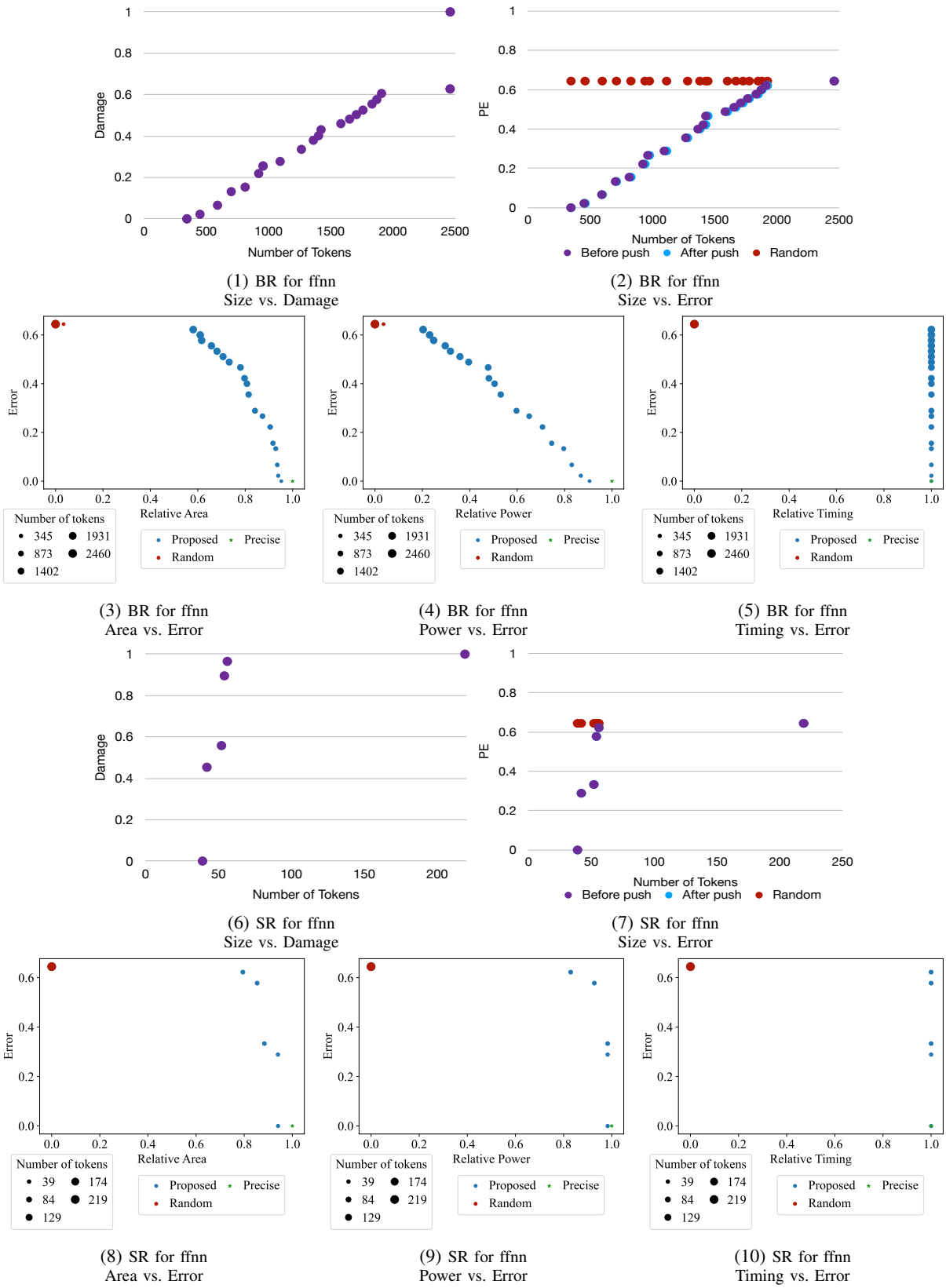
(7) SR for Sobel Size vs. Error

(8) SR for Sobel Area vs. Error

(9) SR for Sobel Power vs. Error

(10) SR for Sobel Timing vs. Error

Figure 8: Results for the Sobel design. Charts 1-2 and 6-7 show the best trade-off clusters in terms of damage and error. Charts 3-5 and 8-10 show the best trade-off clusters in terms of area, power and timing.



(1) BR for ffnn Size vs. Damage

(2) BR for ffnn Size vs. Error

(3) BR for ffnn Area vs. Error

(4) BR for ffnn Power vs. Error

(5) BR for ffnn Timing vs. Error

(6) SR for ffnn Size vs. Damage

(7) SR for ffnn Size vs. Error

(8) SR for ffnn Area vs. Error

(9) SR for ffnn Power vs. Error

(10) SR for ffnn Timing vs. Error

Figure 9: Results for the ffnn design. Charts 1-2 and 6-7 show the best trade-off clusters in terms of damage and error. Charts 3-5 and 8-10 show the best trade-off clusters in terms of area, power and timing.

Design	Technique	Time to simulate	Time to mine	Time to evaluate	Time to cluster Phase I	Time to cluster Phase II	Tot
BrentKung32b	BR	12m 48s	1m 55s	2m 28s	0m 15s	6m 11s	23m 37s
	SR	11m 12s		2m 10s	0m 22s	6m 3s	19m 47s
InverseKin	BR	9m 36s	2m 31s	8m 58s	0m 15s	10m 0s	31m 20s
	SR	7m 28s		0m 39s	0m 12s	1m 46s	10m 5s
Sobel	BR	10m 8s	3m 2s	0m 22s	0m 5s	7m 51s	21m 28s
	SR	2m 8s		2m 20s	0m 7s	10m 0s	14m 35s
ffnn	BR	14m 57s	41s	18s	60s	9m 41s	26m 36s
	SR	1m 21s		2s	30s	43s	3m 16s

Table III: Execution time of the proposed methodology.

Benchmark - Technique	**Pearson Corr.**	**P-value**	**Spear. Corr.**	**P-value**	**R squared**
Sobel BR	0.87	< 0.001	0.96	< 0.001	0.76
Sobel SR	0.78	< 0.001	0.87	< 0.001	0.61
Invk2j BR	0.81	< 0.001	0.84	< 0.001	0.66
Invk2j SR	0.65	< 0.001	0.47	< 0.001	0.42
BKung32b BR	0.52	< 0.001	0.83	< 0.001	0.48
BKung32b SR	0.69	< 0.001	0.73	< 0.001	0.58
ffnn BR	0.93	< 0.001	0.99	< 0.001	0.87
ffnn SR	0.99	< 0.001	0.99	< 0.001	0.99

Table IV: Correlation coefficients for the reported benchmarks and techniques.

Table IV provides various correlation coefficients and their associated p-values for different approximation techniques and designs.

- **Pearson Correlation Coefficient (Pearson Corr.)** measures the linear relationship between two variables. Values closer to +1 or -1 indicate a stronger linear relationship. Significant values (p-value < 0.05) across all techniques suggest reliable linear relationships.
- **Spearman Correlation Coefficient (Spearman Corr.)** measures the rank-order relationship between two variables. All techniques show significant Spearman correlations, suggesting strong rank-based associations.
- **P-value** indicates the probability of obtaining a correlation coefficient at least as extreme as the one observed under the assumption that there's no correlation. A p-value < 0.05 is typically considered statistically significant. The table shows significant results for all Pearson and Spearman correlations.
- **R-squared** represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s). Higher values indicate a better fit of the model to the data. The R-squared values vary, suggesting different strengths of model fits across techniques and benchmarks.

We derive the following additional observations.

- *Inversek2j BR* has notably high correlations across Pearson and Spearman, along with the highest R-squared value, indicating a very strong and reliable relationship in this section.
- Techniques like *Sobel SR* and *BrentKung32b SR* show moderate Pearson correlations but high Spearman correlations, indicating that while the relationship may not be linear, a strong monotonic relationship exists.
- For the ffnn techniques: both *ffnn BR* and *ffnn SR* stand out with exceptionally high correlation values and R-squared scores, especially *ffnn SR*, which achieves near-perfect correlations (Pearson, Spearman and R-squared of 0.99). This indicates that the ffnn techniques perform exceptionally well in terms of both linear and rank-based associations. The near-perfect R-squared values imply that these techniques capture almost all the variance in the data, making them highly reliable for predictive tasks.
- Comparison between SR and BR variants: across techniques, *SR* and *BR* variants show varying levels of performance. For instance, *Sobel SR* and *Inversek2j SR* have lower Pearson and R-squared values compared to their BR counterparts, indicating that while SR versions may capture rank-order relationships (high Spearman correlations), they might not explain as much of the variance as BR versions in a linear sense.
- Moderate performing techniques: techniques such as *Sobel SR* and *BrentKung32b SR* exhibit lower Pearson correlations yet retain significant Spearman correlations, indicating a strong monotonic relationship but a weaker linear association. These results suggest that for these techniques, the relationship between variables may not be strictly linear, possibly due to non-linear transformations inherent in the methods or data interactions.

It is important to note that the metrics reported in table IV are calculated using the entire population (which always contains at least 100 samples/individuals) of the NSGA2 and not just the Pareto front; this ensures statistical relevance even when the number of points on the front is low.

The table indicates statistically significant relationships between the variables measured across different techniques, with varying degrees of linear and rank-order relationships. R-squared values suggest that some models fit the data better than others, which might reflect the nature of the data or the suitability of the linear model to capture the relationships in certain techniques.

VI. Comparison with other methods

We compared our methodology (i.e., DEA) with four other approaches from the state of the art. Those approaches are named the Monte Carlo Tree Search (MTCS), Learning-based fast Design space exploration framework for Approximate circuit synthesis (LDAX) [40], Static Checkpoint (SC), and Dynamic Checkpoint (DC) [37] for which the source codes are available at [41]. For the comparison, we modified the DEA approach to handle the approximation technique called "component substitution", which is the only one used by the approaches mentioned above. This technique aims to replace a given component with an approximate version of it. In this context, an AT consists of replacing an exact component

with an approximate one. Table V reports the results from the comparison. We used four benchmarks available at [37]. Each benchmark is implemented as a data flow composed of a network of components (i.e., multipliers and adders) described in HDL at the gate level. The complexity of the benchmark is determined by the number of replaceable components as described below:

- *ffnn* : feed-forward neural network (i.e., the same used in Section V), 197 components;
- *imsharp*: Image sharpening, 49 components;
- *gauss_blur*: Gaussian blur, 17 components;
- *fft*: fast Fourier transform filter, 120 components.

The technical experimental setup for DEA is the same as in Section V. For the other approaches, we used their source code [37] to run the experiments. For all approaches, we used the same Error metric and threshold as defined in [37] (reported at the bottom of table V). We used the same set of approximate components as [37], which are part of Evoapproxlib [42]. The obtained approximate benchmark has been synthesized by using FreePDK45 [39] 45-nm standard cell technology library and resorted to the synthesis tool reports to extract are and power consumption as reported in the table. Columns 3 to 7 display the absolute error (which is always equal or better than the threshold) from the methodology applied, the percentage of component reduction, the reduction in area and power, and the time taken to execute the entire process, respectively.

From the table, it can be noticed that the DEA methodology always provided the highest percentage of component reduction (column 4), as well as the lowest absolute error (column 3) w.r.t. the other methods. This is because the DEA approach aims to maximize the number of replaced components and minimize the induced error.

On the other hand, there is no direct relation between the percentage of component reduction and area and power. The reason is that not all the available approximate components (i.e., multipliers and adders) have the same "size". For example, in the case of the *gauss_blur*, the DC, MCTS and DEA give 100% of component reduction. However, they do not provide the same set of approximate components, thus leading to differences in the obtained area and power overhead. It can be noticed that this variance impacts all the methodologies.

Finally, we can see that the DEA method provides the best area and power reduction (while inducing the lowest error) for the two complex benchmarks (ffnn and fft), requiring the lowest computational time to provide a solution.

Design	Method	Error Abs.	Comp. Red.	Area Red.	Power Red.	Exec. Time
ffnn*	LDAX	4%	10%	2%	8%	36m
	SC	6%	35%	21%	36%	20m
	DC	4%	10%	4%	11%	21m
	MCTS	6%	28%	16%	25%	74m
	DEA	2%	65%	44%	70%	5m
imsharp†	LDAX	33db	18%	8%	8%	17m
	SC	30db	14%	9%	19%	41m
	DC	100db	4%	2%	1%	18m
	MCTS	31db	45%	23%	20%	123m
	DEA	100db	45%	14%	44%	15m
gauss_blur‡	LDAX	0db	0%	0%	0%	17m
	SC	30db	18%	15%	21%	12m
	DC	30db	100%	25%	36%	21m
	MCTS	32db	100%	19%	28%	28m
	DEA	100db	100%	0%	0%	9m
fft§	LDAX	3.57%	5%	0%	0%	17m
	SC	3.56%	49%	14%	0%	36m
	DC	2.16%	7%	3%	0%	15m
	MCTS	2.66%	35%	12%	0%	90m
	DEA	0.58%	99%	44%	8%	10m

Used metrics (threshold) [37]: *PE (10%), †‡PSNR (30db), §MREP (5%)

Table V: State-of-the-art comparison

VII. Conclusions

In this paper, we presented a novel solution leveraging assertion mining, fault injection and genetic algorithms to identify the elements (signal/register bits or statements) to be approximated into RTL descriptions through the bit-width and the statement reduction strategies. Approximation alternatives are implemented by fault injection. Their impact on the functional accuracy of the DUE is then estimated accordingly to a damage-based metric that evaluates the effect of the approximation on a set of assertions automatically mined from the original DUE implementation to capture its functionality. A genetic algorithm-based clustering procedure is then proposed to guide the designer in the identification of the best clusters of elements to be approximated. Experimental results show that our methodology, for bit-with reduction and statement reduction, and compared to other state-of-the-art approaches, is able to provide the best trade-offs between the size of the approximation cluster and functional accuracy, guaranteeing significant savings in terms of area/power/timing.

VIII. Acknowledgment

This work has been partially supported by the INdAM GNCS, and it was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (PNRR - Missione 4 Componente 2, Investimento 1.5 - D.D. 1058 23/06/2022, ECS-00000043).

References

[1] A. Bosio, D. Menard, and O. Sentieys, Eds., *Approximate computing techniques*, 1st ed. Cham, Switzerland: Springer Nature, Jun. 2022.

[2] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, vol. 1, no. 2, 2015.

[3] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. of IEEE ETS 2013*, 2013.

[4] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. of ACM/IEE DAC 2013*, 2013.

[5] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. of ACM/IEEE DAC 2015*, 2015.

[6] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.

[7] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[8] S. Barone, M. Traiola, M. Barbareschi, and A. Bosio, "Multi-objective application-driven approximate design method," *IEEE Access*, vol. 9, pp. 86 975–86 993, 2021.

[9] S. Mitra, M. Das, A. Banerjee, K. Datta, and T.-Y. Ho, "A verification guided approach for selective program transformations for approximate computing," in *Proc. of IEEE ATS 2016*, 2016.

[10] H. Foster, D. Lacey, and A. Krolnik, *Assertion-Based Design*, 2nd ed. USA: Kluwer Academic Publishers, 2003.

[11] S. Mitra, M. K. Gupta, S. Misailovic, and S. Bagchi, "Phase-aware optimization in approximate computing," 2017.

[12] A. Bosio, M. Bragaglio, S. Germiniani, S. Mori, G. Pravadelli, and M. Traiola, "Assertion-aware approximate computing design exploration on behavioral models," in *Proc. of IEEE LATS 2022*, 2022.

[13] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. of ACM ESEC/FSE 2011*, 2011.

[14] K. Nepal, Y. Li, R. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proc. of ACM/IEEE DATE 2014*, 2014.

[15] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[16] W. Liu, T. Cao, P. Yin, Y. Zhu, C. Wang, E. E. Swartzlander, and F. Lombardi, "Design and analysis of approximate redundant binary multipliers," *IEEE Transactions on Computers*, vol. 68, no. 6, pp. 804–819, 2018.

[17] V. Mrazek, L. Sekanina, and Z. Vasicek, "Libraries of approximate circuits: Automated design and application in cnn accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 406–418, 2020.

[18] L. Sekanina, "Evolutionary Algorithms in Approximate Computing: A Survey," *Journal of Integrated Circuits and Systems*, vol. 16, no. 2, pp. 1–12, Aug. 2021.

[19] A. Lotfi, A. Rahimi, A. Yazdanbakhsh, H. Esmaeilzadeh, and R. K. Gupta, "Grater: An approximation workflow for exploiting data-level parallelism in fpga acceleration," in *Proc. of IEEE/ACM DATE 2016*, 2016.

[20] I. Albandes, A. Serrano-Cases, A. Sánchez-Clemente, M. Martins, A. Martínez-Álvarez, S. Cuenca-Asensi, and F. L. Kastensmidt, "Improving approximate-tmr using multi-objective optimization genetic algorithm," in *Proc. of IEEE LATS 2018*, 2018.

[21] I. Alouani, H. Ahangari, O. Ozturk, and S. Niar, "A novel heterogeneous approximate multiplier for low power and high performance," *IEEE Embedded Systems Letters*, vol. 10, no. 2, pp. 45–48, 2018.

[22] D. Ma, R. Thapa, X. Wang, X. Jiao, and C. Hao, "Workload-aware approximate computing configuration," in *Proc. of IEEE/ACM DATE 2021*, 2021.

[23] M. Barbareschi, S. Barone, and N. Mazzocca, "Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study," *Knowledge and Information Systems*, vol. 63, no. 6, pp. 1577–1596, Jun. 2021.

[24] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[25] A. J. Sanchez-Clemente, L. Entrena, R. Hrbacek, and L. Sekanina, "Error mitigation using approximate logic circuits: A comparison of probabilistic and evolutionary approaches," *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1871–1883, 2016.

[26] Z. Vasicek and L. Sekanina, "Search-based synthesis of approximate circuits implemented into fpgas," in *Proc. of FPL 2016*, 2016.

[27] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. of IEEE/ACM DATE 2017*, 2017, pp. 258–261.

[28] M. Traiola, J. Echavarria, A. Bosio, J. Teich, and I. O'Connor, "Design space exploration of approximation-based quadruple modular redundancy circuits," in *Proc. of IEEE/ACM ICCAD 2021*, 2021.

[29] M. Barbareschi, S. Barone, A. Bosio, J. Han, and M. Traiola, "A genetic-algorithm-based approach to the design of DCT hardware accelerators," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 18, no. 3, pp. 1–25, Jul. 2022.

[30] X. F. M. Ye and S. Wei, "Runtime hardware security verification using approximate computing: A case study on video motion detection," in *Proc. of IEEE AsianHOST 2019*, 2019.

[31] Y. M. K. Yoshisue and T. Ishihara, "Dynamic verification of approximate computing circuits using coverage-based grey-box fuzzing," in *Proc. of IEEE IOLTS 2021*, 2021.

[32] "Standard for property specification language (PSL)," *IEC 62531:2012(E) (IEEE Std 1850-2010)*, pp. 1–184, 2012.

[33] S. Germiniani and G. Pravadelli, "Harm: A hint-based assertion miner," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4277–4288, 2022.

[34] "Harm assertion miner," 2023. [Online]. Available: https://github.com/SamueleGerminiani/harm

[35] "Dea," 2024. [Online]. Available: https://github.com/SamueleGerminiani/harm/tree/main/src/dea

[36] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2017.

[37] M. Awais, H. G. Mohammadi, and M. Platzner, "Deepapprox: Rapid deep learning based design space exploration of approximate circuits via check-pointing," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024, pp. 88–93.

[38] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[39] "FreePDK45 — NC State EDA — eda.ncsu.edu," https://eda.ncsu.edu/freepdk/freepdk45/, [Accessed 31-12-2024].

[40] M. Awais, H. Ghasemzadeh Mohammadi, and M. Platzner, "Ldax: A learning-based fast design space exploration framework for approximate circuit synthesis," in *Proceedings of the 2021 Great Lakes Symposium on VLSI*, ser. GLSVLSI '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 27–32. [Online]. Available: https://doi.org/10.1145/3453688.3461506

[41] "Deep approx tools," https://git.uni-paderborn.de/axmcts/public/deepapprox.

[42] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 258–261.

Alberto Bosio received his MSc (03) and PhD (06) in Computer Engineering at the Politecnico di Torino (Italy). He is now a Full Professor at Ecole Centrale de Lyon, Institue of Nanotechnology (France). His research activities are related to the design and test of advanced digital circuits and systems. He is a member of the IEEE.

Samuele Germiniani Samuele Germiniani received the PhD degree in Computer Science from the University of Verona, Italy, in 2023, where he is currently a post-doc research fellow at the Department of Engineering for Innovation Medicine. His main research interests are related to semi-formal verification and embedded security of cyberphysical systems. He is a member of the IEEE.

Marcello Traiola received a Ph.D. in Computer Engineering in 2019 from the University of Montpellier, France, and a Laurea degree (MSc) in Computer Engineering in 2016 from the University of Naples Federico II, Italy. Currently, he is a tenured research scientist with the Inria Research Institute at the IRISA laboratory in Rennes, France, on the TARAN research team. He researches emerging computing paradigms (approximate computing, in-memory computing) with an interest in hardware design, testing, and reliability. He is an IEEE member.

Graziano Pravadelli PhD in computer science, IEEE senior member, IFIP 10.5 WG chair, is full professor of information processing systems at the Department of the Engineering for Innovation Medicine at the University of Verona (Italy) since 2018. In 2007 he co-founded EDALab s.r.l., an SME working on the design of IoT-based monitoring systems. His main interests focus on system-level modeling, simulation and semi-formal verification of embedded systems, as well as on their application to develop virtual coaching and telemedicine platforms for people with special needs.