Approximate Logic Synthesis via Iterative SMT-based Subcircuit Rewriting

Morteza Rezaalipour, Marco Biasion, Francesco Costa, Cristian Tirelli, Lorenzo Ferretti, Rodrigo Otoni, George A. Constantinides, and Laura Pozzi

***Abstract*—This paper presents a novel iterative approach to achieve effective and efficient approximate logic synthesis (ALS). The core idea is to perform circuit rewriting in a way that is both local, i.e., is applied piece-wise to selected subcircuits, and extensive, i.e., systematically explores the design space for good solutions. Concretely, we propose SubXPAT, a new Boolean rewriting framework which iteratively employs satisfiability modulo theories (SMT) solving to select and approximate key parts of a circuit. Selection aims at finding subcircuits that at the same time include a significant number of gates and can be efficiently approximated, which is done by searching for large convex subcircuits with a limited number of inputs and outputs. Approximation is guided by the use of a parametric template, structured as a sum of products, which allows for fine-grained control over the subcircuit characteristics. SubXPAT was implemented as an open-source tool and compared against other ALS tools implementing state-of-the-art techniques. Our experimental evaluation used a broad range of arithmetic circuits with different bit-widths and our results indicate that SubXPAT generates approximate circuits that are more area-efficient than those generated by state-of-the-art techniques in 72% of the cases.**

***Index Terms*—approximate computing, Boolean rewriting, local analysis, SMT solving.**

I. Introduction

APPROXIMATE computing is a new paradigm in computer design postulating that inexact hardware should be used whenever full accuracy is not required by a given application; hence, accuracy should be traded for improvements in metrics such as area and energy performance [1]. Approximate circuits can be employed in any error-tolerant application, including those for which an exact output cannot be distinguished from an inexact one, noisy data inherently prevents full accuracy, or no single golden answer exists.

Approximate logic synthesis (ALS) is the process of deriving an approximate circuit automatically, from a given exact circuit and a tolerated error threshold (ET). There exist several families of ALS methods, categorised in a recent survey [2] into those based on *netlist modification*, which generate the resulting circuit starting from an existing structure, e.g., by removal of gates or wires, and those based on *Boolean rewriting*, which generate the resulting circuit without reference to an original structure, e.g., by Boolean factorisation. Despite extensive work towards improving ALS techniques [3]–[8], approximation methods can still be further improved in terms of exploiting the flexibility allowed by the given ET.

With the aim of generating approximations that are better than those that can be obtained with current techniques we propose SubXPAT, a novel Boolean rewriting framework for the synthesis of approximate circuits that iteratively selects and approximates key subcircuits of the exact circuit. The core of SubXPAT is our algorithm for approXimate logic synthesis via a PArametrical Template, XPAT for short, which employs a satisfiability modulo theories (SMT) solver to shape a *parametrical template* into an optimised circuit, i.e., a circuit with a lower amount of gates, while remaining sound w.r.t. the given ET [9]. Concretely, the template we use is a sum of products and the role of the solver is to identify, for every circuit output, which products of which input literals must be included in the final synthesis. XPAT by itself, i.e., when it is applied to the whole circuit, yields highly optimised results, but is limited in its scalability, only being practical when applied to small circuits. To address this, the iterative infrastructure of SubXPAT is used to select subcircuits that are both small enough to be approximated by XPAT in a timely manner and large enough to lead to significant optimisations. Our approach targets the maximum error constraint, guaranteeing that a given error threshold is never exceeded by any input combination. This is a relevant metric in all scenarios in which large errors should be avoided, such as in image processing, with it also being adopted by several prior works [5], [7], [8].

To enable the practical use of SubXPAT we implemented it as an open-source tool. We performed an experimental evaluation using 23 arithmetic circuits and compared SubXPAT's results against those of the state-of-the-art ALS methods MUSCAT [7], MECALS [8], and EvoApprox [4]. Our results show that SubXPAT improves synthesised area w.r.t. the best state-of-the-art method in 192 of our 264 approximation targets, being thus the best approach in 72% of the cases.

To summarise, our contributions are the following:

(1) An SMT-based approximate logic synthesis algorithm,

Morteza Rezaalipour, Marco Biasion, Francesco Costa, Cristian Tirelli, and Laura Pozzi are all with the Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland (e-mail: morteza.rezaalipour@usi.ch; marco.biasion@usi.ch; francesco.costa@usi.ch; cristian.tirelli@usi.ch; laura.pozzi@usi.ch).

Lorenzo Ferretti is with the Silicon System AI group, Micron Technology, San Jose, USA (e-mail: lferretti@micron.com).

Rodrigo Otoni is with the Faculty of Science and Engineering, University of Groningen, Groningen, Netherlands (e-mail: r.b.otoni@rug.nl).

George A. Constantinides is with the Faculty of Engineering, Imperial College London, London, UK (e-mail: g.constantinides@imperial.ac.uk).

This work was partially supported by the Swiss National Science Foundation via grant 200020_188613.

Published in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD): https://doi.org/10.1109/TCAD.2025.3638267.

called XPAT, which synthesises approximate circuits via the shaping of a parametrical template into an optimised circuit (originally appeared in [9]).
(2) An iterative framework for approximate logic synthesis, called SUBXPAT, which enables iterative applications of XPAT to different subcircuits of the original circuit as a means of scaling the usage of XPAT to larger circuits.
(3) An open-source implementation of the SUBXPAT framework that allows for its usage in practice.
(4) An extensive evaluation showcasing both XPAT's ability to yield highly optimised circuits and the scalability gains achieved with SUBXPAT in relation to XPAT.

The remainder of this article is structured as follows. Section II discusses related work. Sections III, IV, and V respectively present XPAT, SUBXPAT, and iterative SUBXPAT. Section VI details the fine-tuning of our framework. Section VII reports our evaluation results. Finally, Section VIII presents our conclusions and future work.

II. State of the Art

Techniques for ALS can be categorised over two axes: approximation procedure and approximation target. The procedure can be either monolithic or iterative, depending if the approximation happens in one step or if a number of consecutive approximations is made. The target can be either the whole circuit or a subcircuit, with subcircuits with different characteristics being potentially chosen.

Monolithic approaches commonly target the whole circuit, since they perform approximation only once. Examples of this combination are MUSCAT [7] and CIRCUIT CARVING [5]. Due to having to perform a global analysis over the whole circuit, monolithic approaches have limited scalability, preventing their use in many practical scenarios. One positive aspect, however, is that having a view of the whole circuit at once can in principle allow for more powerful approximations.

Iterative approaches arose as a way to improve scalability, by means of performing local analysis over a subcircuit, allowing for ALS to be applied to significantly larger circuits. Examples of this combination are MECALS [8], GLP [10], and SASIMI [3], all of which select, at each iteration, a subcircuit consisting of a single gate. This restrictive selection aids in the aim of improving scalability, but at the expense of limiting approximation options. These approaches consider local approximation candidates (LACs) which consist, at each iteration, in simple modifications such as removing a gate or altering a wire. Existing iterative approaches are, thus, likely to underperform when compared to monolithic approaches, for the cases in which the monolithic approaches terminate in a reasonable amount of time.

In contrast to considering single modifications to a gate or a wire, the technique adopted by SUBXPAT considers, at each iteration, a powerful logic-rewriting LAC. Our technique goes beyond the state of the art by (1) considering multiple-output subcircuits for approximation, which are significantly more complex than single gates, and by (2) using an SMT-based rewriting guided by a parametrical circuit template.

Two other approaches of note are BLASYS [6], [11], [12] and EVOAPPROX [4], [13]. BLASYS also iteratively applies rewriting to subcircuits, in this case by means of logic factorisation, while EVOAPPROX employs genetic programming to achieve its approximations. Our evaluation (cf. Section VII) quantitatively shows that SUBXPAT outperforms the state of the art, corroborating the qualitative intuition presented here, i.e., that applying our powerful logic-rewriting yields better results than employing previously considered simple LACs, as done by MECALS, and that template-based rewriting can be more efficient than techniques such as genetic programming.

Lastly, note that techniques such as MUSCAT, MECALS, and EVOAPPROX are comparable with ours because they, like the present work, target maximum (worst-case) error. Hence, these techniques produce circuits that never exceed the given ET, for every input configuration. Another category of works was developed [14]–[18] to target, instead, average error metrics, such as error rate and mean absolute error – as such, these techniques are not directly comparable. Finally, it is also to be noted that in this work we bound the error of individual circuits, and that limiting the way that errors then propagate [19], [20] in whole network of such circuits (such as for example neural networks) is another complex and highly researched problem, which is outside the scope of this paper. For an extensive survey of ALS, and approximate computing more broadly, we refer the reader to [21], [22].

III. XPAT

We approach ALS through the lens of Boolean rewriting. This allows us to explore a large design space that is not restricted by the structure of the circuit that we aim to approximate, in principle enabling approximations that would otherwise not be achievable by considering structural methods alone. To guide the search over this large design space, we make use of a *parametrical template*. Concretely, XPAT's template consists of sums of products, with each sum dictating the value of one primary output and the parameters deciding which products of which primary inputs should be included in each sum. We chose this template because of its expressiveness, which guarantees us to be able to represent any Boolean function. The design space exploration is framed in terms of SMT formulas, whose solving leads to the setting of the template's parameters in such a way that the approximate circuit respects the given ET. The template-based exploration allows us to systematically partition the design space and to focus on sections of it which contain designs that are likely to lead to small synthesised areas, e.g., designs with a limited number of products. In the following we detail the template used, in Section III-A, the SMT problem and the SMT formulation we employed, in Section III-B and Section III-C, and how the design space is explored, in Section III-D.

A. Parametrical Template

Consider a given circuit template with n primary inputs, $in_1, ..., in_n$, and m primary outputs, $out_1, ..., out_m$. The behaviour of each output out_i is formalised as follows:

$$out_i = \bigvee_{k \in \{1...K\}} Prod_k \tag{1}$$

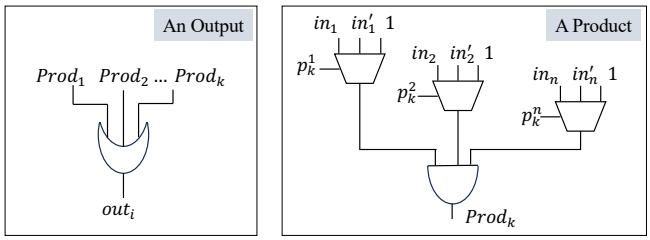

Fig. 1: Our 2-layer sum of products template. (a) A sum of K products. (b) A product of n primary inputs; parameter p_k^j determines how primary input in_j is to be handled.

where K is the number of products included in each sum, $Prod_k$ is a product of selected primary inputs, and $1 \leq i \leq m$, as illustrated in Figure 1a. Each product $Prod_k$ has n inputs, originating from n multiplexers, as illustrated in Figure 1b. Each multiplexer decides, based on parameter p_k^j, if primary input in_j is used as is, is negated, or is replaced by the constant 1, with $1 \leq j \leq n$; the usage of constant 1 indicates that the respective primary input is ignored by the product. Note that the multiplexers are not a part of the generated circuit, since the selected lines are chosen at compile time.

All p_k^j parameters are of 2-bit form, with the first bit indicating if in_j is used or ignored, and the second bit indicating the sign of in_j. Products can thus be described in terms of vectors of form $[p_k^1, ..., p_k^n]$. For example, having $in_1 = a$, $in_2 = b$, and $in_3 = c$ as primary inputs, the product described by $[11, 10, 01]$ would output ab', while the product described by $[01, 01, 10]$ would output c'.

With our template of sums of parametric products we can capture all circuit behaviours, except the output of constant 0. To address this, we can either add a further 1-bit parameter to each sum, to select whether the output of the sum or the constant 0 should be used for the respective primary output, or add another input to each p_k^j multiplexer parameter. We implemented both options and compared their performance, with our results shown Table I. Since the option of adding a further parameter to each sum is slightly faster on average, we decided to adopt it in this work.

Having m outputs, each defined as a sum of K products, with each product having n 2-bit parameters, and considering the parameter needed to express constant 0, the total number of parameters of our template is $m(2nK+1)$. This represents our design space, which we explore via SMT solving.

TABLE I: Performance comparison between different representations of constant 0; runtime is given in seconds.

Benchmark	Representation of constant 0	
	parameter to sums	input to multiplexers
4-bit adder	**15.64**	16.46
8-bit adder	**47.13**	48.76
16-bit adder	**205.53**	216.75
4-bit absolute difference	**3.48**	3.56
8-bit absolute difference	**83.44**	85.15
16-bit absolute difference	331.01	**319.69**
3-bit multiplier	**14.39**	14.85
6-bit multiplier	**130.97**	135.99
3-bit multiply-and-add	**23.98**	25.10
4-bit multiply-and-add	**80.81**	83.23
6-bit multiply-and-add	**144.85**	149.51

B. Background on Satisfiability Modulo Theories

The satisfiability modulo theories (SMT) problem involves deciding the satisfiability of a first-order logic formula within the scope of various first-order theories, such as arithmetic, bit-vectors, and arrays. SMT is a generalisation of the well known Boolean satisfiability (SAT) problem, whose added expressiveness allows for the representation of several real world problems. Concretely, an SMT formula F is satisfiable (SAT for short, not to be confused with Boolean satisfiability) if values can be assigned to its terms in such a way that it evaluates to *true*; if no such assignment is possible the formula is considered unsatisfiable (UNSAT for short).

As an example, consider Ex to be the following formula: $x = y + 1 \wedge y = z - 1 \wedge x \neq z \vee b$, where $x, y, z \in \mathbb{Z}$ and $b \in \mathbb{B}$. Ex is satisfiable with the assignment, also called a satisfying model, of $x = 1$, $y = 0$, $z = 1$, and $b = \top$; notice that $\vee$ binds stronger than $\wedge$, and that the arithmetic operators bind stronger than both $\vee$ and $\wedge$. The formula $Ex \wedge \neg b$, on the other hand, is unsatisfiable, since it inevitably leads to a logical contradiction and thus has no satisfying assignment.

Many efficient solvers exist to automate the solving of SMT formulas, most notably Microsoft's Z3 [23]. By wisely exploiting the power of modern solvers we can traverse large design spaces to find good approximate circuits, as described in the following sections. For further details on SMT, we refer the reader to [24].

C. SMT Formulation

Our formulation of the design space exploration is based on an error miter, which consists of the exact circuit, the parametrical template for approximation, two functions used for error estimation, and a query, as illustrated in Figure 2. Function $map : \mathbb{B}^m \rightarrow S$ does the mapping from the primary output of each circuit to an element of S, with S being a user-defined set containing interpretations of Boolean values. For example, with $m = 3$ and S instantiated to $\mathbb{Z}$ we can have $map(out_3, out_2, out_1) = 4{*}out_3 + 2{*}out_2 + 1{*}out_1$. Function $dist : S \times S \rightarrow Dist$ measures the distance, according to a desired metric and with $Dist$ being a totally ordered set, between the mapped primary outputs of the exact and approximate circuits. For example, with $Dist$ instantiated to $\mathbb{Z}$ we can have $dist(s_1, s_2) = |s_1 - s_2|$.

We aim to generate an approximate circuit that never exceeds the ET. To achieve this the template parameters must be instantiated in such a way that, for all possible primary input combinations, the distance between the exact and approximate primary outputs does not exceed the ET. Thus, we ask the SMT solver whether it is possible to assign values to the parameters in such a way that the approximate circuit always

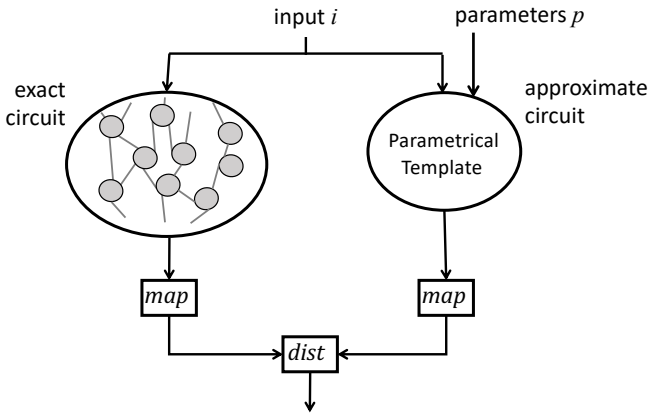

Fig. 2: Miter for XPAT.

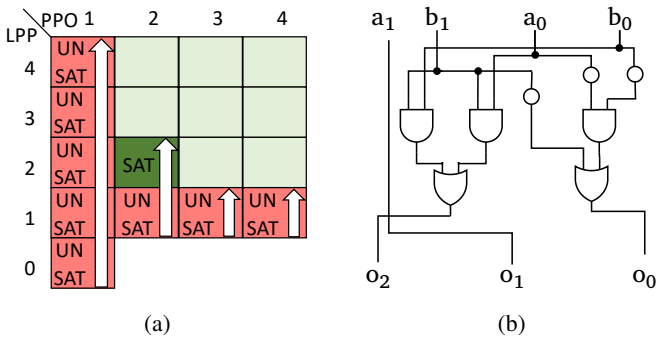

Fig. 3: (a) Design space exploration guided by LPP and PPO. In this example, cell (2,2) is found to be SAT and hence the eight cells dominated by it (in green shade) are not explored. Every other cell in the grid is explored. (b) Approximate circuit with an ET of 1 generated by XPAT from a 2-bit adder. The circuit corresponds to a model from cell (2,2).

respects the ET. Concretely, we ask whether the following formula is satisfiable:

$$\exists p \in P \,.\, \forall i \in \mathbb{B}^n \,.\, dist\begin{pmatrix} map(outExct(i)), \\ map(outAppx(i,p)) \end{pmatrix} \leq \text{ET} \quad (2)$$

where P is the set containing all possible values for the $m(2nK+1)$ parameters, and $outExct$ and $outAppx$ are functions respectively capturing the behaviour of the exact and the approximate circuits. A satisfiable result will be accompanied by an assignment to p that ensures that the approximate circuit always respects the ET. Note that this formulation provides a bound on maximum error, meaning that a satisfying model will guarantee to never exceed a given ET for every input combination. Also note that this formulation is orthogonal to maximum error metrics. For example, setting $dist(s_1, s_2)$ to be $|s_1 - s_2|$ will bound absolute error, and setting it to be $(s_1 - s_2)^2$ will bound square error.

D. Design Space Exploration

Simply asking the SMT solver for a satisfying assignment might lead us to a low quality solution, due to two reasons. First, the design space – whose size grows exponentially w.r.t. the number of template parameters – contains all possible configurations of the template, which enables numerous valid solutions. Second, the solver cannot differentiate between assignments of different quality, i.e., assignments that will lead to larger or smaller synthesised areas. To address these issues, we systematically limit the design space by restricting the number of literals per product (LPP) and products per output (PPO) allowed in the approximate circuit. By asking for a satisfying assignment that, in addition to respecting the given ET, also respects an upper bound on LPP and PPO, we can generate approximations that will utilise fewer gates and will thus lead to smaller synthesised areas; intuitively, we aim to consider sets $P_0 \subset P_1 \subset ... \subset P$ in our satisfiability query.

The additional constraints added might lead to a design space in which no valid solution exists. We must thus consider different pairs of LPP and PPO constraints, which we explore in a grid-like manner as illustrated in Figure 3. The starting point of the exploration has LPP = 0 and PPO = 1, with LPP being 0 symbolising that products are substituted by constants. In case the formula is unsatisfiable LPP is incremented and the solver is called again. If the LPP bound is reached, we increment PPO, set LPP to 1, and continue as before.

Note that, whenever a SAT is found, a whole section of the grid (indicated in green shade in Figure 3 and corresponding to values of LPP and PPO equal or higher than those of the current SAT cell) becomes dominated and hence needs not be explored. The process terminates once either a predefined number of satisfiable assignments is found, or both the LPP and the PPO bounds are reached.

We conclude this section with a note on guarantees: while this search is of course not guaranteed to find the optimal (e.g., smallest-area) approximation *after* synthesis for a given exact circuit and ET value (and this is also inherently true for comparable state-of-the-art methods, due to the unforeseen effects of the downstream synthesis process), it is guaranteed to find circuits requiring the smallest (i.e., non-dominated) product-literal pairs, within the bounds searched. We show experimentally that this method is able to yield high-quality circuits in practice (cf. Section VII).

IV. SubXPAT

Since XPAT requires finding a satisfiable assignment for a formula that quantifies over the whole circuit, it has limited scalability. To address this, SubXPAT approximates selected *subcircuits*, instead of the whole circuit, as done by XPAT. In the following we first detail the SMT formulation designed for SubXPAT, in Section IV-A, and then describe how subcircuit selection takes place, in Section IV-B and Section IV-C.

A. SMT Formulation for Local Approximation

Assuming that a subcircuit has been selected for approximation – and how this is done is described in the two following subsections – SubXPAT can then be used to perform a *local* approximation, i.e., an approximation that rewrites only that part of the exact circuit, and thus avoids the combinatorial explosion arising from applying XPAT to the whole circuit.

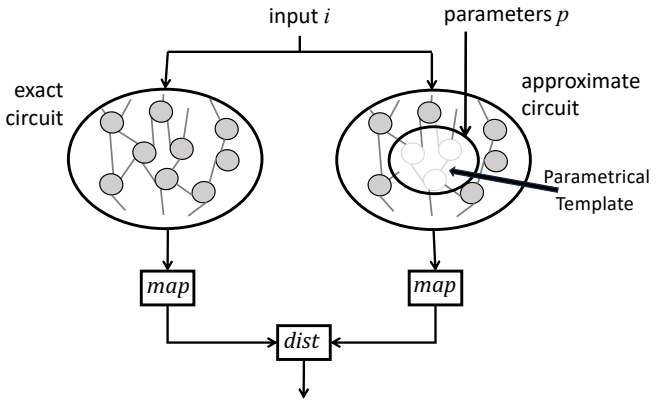

Fig. 4: Miter for SUBXPAT.

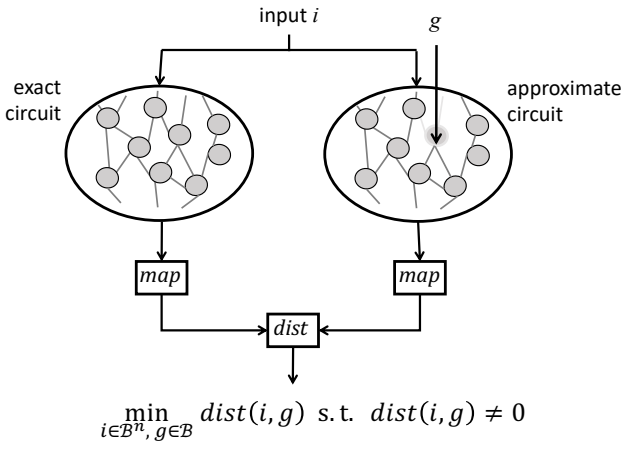

Fig. 6: Miter for $mnze_g$.

SUBXPAT's formulation is a generalisation of the formulation described in Section III-C. Similar to XPAT, a miter is used containing the exact circuit and a parametrical template for approximation. The crucial difference is that the gates not contained in the selected subcircuit remain untouched, as illustrated in Figure 4, which decreases the size of the design space and leads to a formula easier to solve. Concretely, function $outAppx$ is updated to only consider parameters for the template replacing the selected subcircuit, with the size of the set P being thus decreased. The size of the selected subcircuit dictates both the approximation's cost and potential benefits, with a selection containing all gates of the exact circuit making SUBXPAT subsume to XPAT.

B. Subcircuit Selection Criteria

A crucial question in subcircuit selection is: given a subcircuit and a tolerated ET, is an approximation of such subcircuit even *possible*? By this we mean: can the logic of such subcircuit be rewritten in a way that (1) its error is bounded by the ET, and (2) we do not need to resort to semantically-equivalent rewriting? The trivial case of semantically-equivalent rewriting, i.e., a rewriting in which the error is zero for all inputs, is always possible, but is not of interest since the aim of approximation is to *make use* of the tolerated ET to maximise

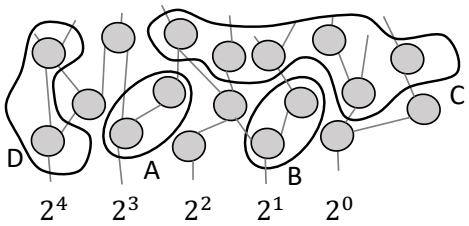

Fig. 5: Examples of subcircuits. Subcircuit A drives a single output, and modifying its logic will induce an error of 2^3. Modifying subcircuit B will induce an error of 2^1. Quantifying the potential error induced by modifying a multi-output subcircuit, such as C, is non-trivial. Subcircuit D is an example of non-convexity, since its inputs depend on its outputs.

area gains. In practice, we are interested in subcircuits that can be rewritten to make a non-zero error within the ET. Figure 5 illustrates this point. Subcircuit A influences only a single bit of the circuit, with significance 2^3. Assuming a tolerated error of 4, subcircuit A is not a good choice for approximation, because either it is rewritten in a semantically-equivalent way, and then the error made is 0, or it is rewritten in any other, non-semantically-equivalent, way, and the error made is 16. Hence, no real approximation is possible in this case. We call a subcircuit *feasible*, for a given ET, if a rewriting exists to induce an error larger than 0 and within ET. Subcircuit A is not feasible for an ET of 4, while subcircuit B is.

We are interested in formalising the concept of *feasibility*, in order to only select feasible subcircuits for approximation. To do so, we need to be able to quantify, or somehow bound, the effect that any perturbation of a gate has on the error at the output of a circuit. As we have seen, when a subcircuit only influences a single bit of the output, its effect on error is trivial to define. In the generic case, however, i.e., when a subcircuit has several outputs that have arbitrary downward cones leading to primary outputs, such as subcircuit C, estimating the error caused by a rewriting is non-trivial.

In order to define feasibility, we first define the quantity $mnze_g$ for every gate g in the circuit. This quantity represents the *minimum non-zero error* that can be induced by perturbing in any possible way the logic of *only* gate g. Formally, $mnze_g$ is defined as follows:

$$mnze_g = \min_{i\in\mathbb{B}^n, g\in\mathbb{B}} dist\begin{pmatrix} map(outExct(i)), \\ map(outAppx(i,g)) \end{pmatrix} \quad \text{s.t. } map(outExct(i)) \neq map(outAppx(i,g)) \tag{3}$$

The $mnze_g$ calculation determines the minimum non-zero distance between the output of the exact circuit, $map(outExct(i))$, and the output of an approximate circuit, $map(outAppx(i,g))$, where the latter is derived from the exact circuit by perturbing only gate g. This is illustrated in Figure 6; $dist(i,g) \neq 0$ is ensured by calculating the distance between different points. Note that the minimum is calculated over every input $i \in \mathbb{B}^n$ and every value of $g \in \mathbb{B}$.

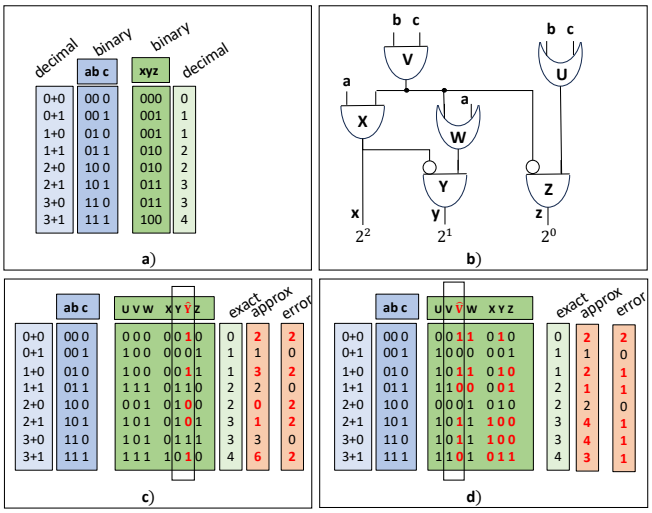

Fig. 7: Motivation for $mnze_g$ calculation. (a) Truth table for an adder of 2 plus 1 bits (3 inputs and 3 outputs), in both binary and decimal. (b) A circuit implementing such truth table. (c) Perturbing the output of gate Y in a specific way (called here $\hat{Y}$), causes an error of either 2 or 0 at truth table lines. Considering *every possible* perturbation, which can be done via the miter in Figure 6, returns a $mnze_Y$ of 2. (d) The effect of perturbing gate V propagates further through the circuit – a specific perturbation called $\hat{V}$ is shown here, for which the error at every truth table line is either 2, 1, or 0; the value of $mnze_V$, considering every possible perturbation, can be derived via the miter in Figure 6 to be 1.

We define a *gate* to be *feasible* w.r.t. a tolerated ET if $mnze_g \leq$ ET. Furthermore, we define a *subcircuit* to be *feasible* w.r.t. a tolerated ET if *all* its output nodes are feasible. The intuition for $mnze_g$ is given in Figure 7: the figure shows, for a toy example, the error that can be caused by a perturbation in different parts of a circuit. The intuition for calculating value $mnze_g$ is the following: if, at a given point in a circuit, the minimum non-zero error caused by *any* perturbation is larger than the allowed error, then there is no reason in choosing that particular location of the circuit for approximation. The calculation of $mnze_g$ for each gate is automated via the use of SMT optimisation [25], on the miter depicted in Figure 6.

Having defined feasibility, we now state our subcircuit selection criteria. We select subcircuits that are (1) feasible, (2) convex, (3) maximal in size, and (4) limited in their number of inputs and outputs. With feasibility already formalised, the rest of the criteria are described and motivated in the following.

Convexity: we must select subcircuits that are convex, in order to prevent the occurrence of cycles in the rewritten circuit. An example of non-convexity is subcircuit D in Figure 5, where one of the inputs depends on one of the outputs.

Maximality: in order to maximise approximation gain, we select subcircuits that are maximal in size – subject to the constraints given in the following point.

Limited I/O: The number of inputs and outputs of the subcircuit to be rewritten is directly proportional to search space complexity, i.e., to the number of parameters of the resulting template – as discussed in Section III-A. Hence, to improve scalability, we choose to search for subcircuits within a limited I/O count.

C. Subcircuit Selection Encoding

We now formalise the procedure for subcircuit selection, and detail the encoding that we use. We represent the circuit we want to approximate as a directed graph $G = (V, E, l)$, where V is the set of nodes abstracting the inputs, outputs, and gates in the circuit, $E \subseteq V \times V$ is the set of edges abstracting the circuit's wires, and $l : V \to \mathbb{N}^+$ is the partial function labelling vertices with their $mnze_g$; function l is partial to improve labelling efficiency (cf. Section VI-B). An SMT formula is used to encode the subgraph selection problem. For each node $v \in V$, a Boolean literal x_v is defined, whose value indicates if v is in the subgraph or not. The objective is to maximise the number of literals assigned to *true* while respecting selection constraints. Concretely, we ask the solver for a satisfying assignment to the following formula:

$$maximise\ |\{x_v \mid v \in V \wedge x_v = \top\}| \quad \text{s.t. } \phi \tag{4}$$

The selection constraints are encapsulated into ϕ as follows:

$$\phi = convex \wedge limitedIO \wedge allOutputsFeasible \tag{5}$$

To aid in the definition of ϕ, we first define constraints that capture if an edge is an input or an output to the subgraph. Concretely, for every edge $e \in E$ we define the following:

$$e^{in}_{s,d} = \neg x_s \wedge x_d \qquad e^{out}_{s,d} = x_s \wedge \neg x_d \tag{6}$$

where $e^{in}_{s,d}$ indicates that the source of the edge is not included in the subgraph while its destination is, and $e^{out}_{s,d}$ indicates the opposite scenario. We then define the sets of input and output nodes of the subgraph, I and O, as follows:

$$I = \{d \mid (s,d) \in E \wedge e^{in}_{s,d}\} \tag{7}$$

$$O = \{s \mid (s,d) \in E \wedge e^{out}_{s,d}\} \tag{8}$$

For a subgraph $H = (V_H, E_H, l_H)$ of G, we need to ensure convexity, limitation on the number of inputs and outputs, and feasibility of all outputs. Subgraph H is convex w.r.t. to G if, for every pair of nodes $s, d \in V_H$, all paths $\rho_G(s,d)$ are contained in E_H. A path in G between nodes s and d, denoted $\rho_G(s,d)$, is a sequence of edges $(s,w_1),(w_1,w_2),...,(w_n,d)$. To ensure convexity, we enforce that all paths leading to an input node must not come from V_H. Concretely:

$$convex = (\forall s \in V_H \,.\, ((\exists d \in I \,.\, \rho_G(s,d)) \implies \neg x_s)) \tag{9}$$

Given constants I_{max} and O_{max} dictating the maximum number of inputs and outputs desired, the limitation constraint is defined as follows:

$$limitedIO = (|I| \leq I_{max} \wedge |O| \leq O_{max}) \tag{10}$$

Lastly, the feasibility constraint is straightforward, as follows:

$$allOutputsFeasible = (\forall o \in O \,.\, l(o) \leq \text{ET}) \tag{11}$$

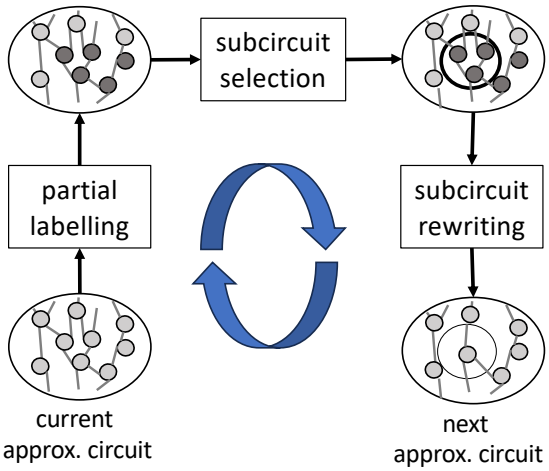

Fig. 8: Overview of iterative SUBXPAT; the exact circuit takes the place of the current approximate circuit initially.

By using SMT optimisation [25] to find a satisfying assignment to our maximisation formula, we can gather the subgraph with the maximum number of nodes that respects ϕ, i.e., that is convex, respects the I/O limit, and whose output nodes are all (individually) feasible.

V. ITERATIVE SUBXPAT

With SUBXPAT we are able to scale XPAT to larger circuits, making it more applicable in practice, but limit the potential gains by considering only one subcircuit of the exact circuit, as opposed to the whole exact circuit. We can, however, apply the rewriting process iteratively, at each step selecting and approximating a new subcircuit, as illustrated in Figure 8. This way we can improve scalability when compared to XPAT and apply its rewriting power to several parts of the circuit.

In pursuing iterative SUBXPAT, which we refer to simply as SUBXPAT, the main concern is which ET to use at each iteration, i.e. how much error to be *consumed* at each iteration. The naive approach is to consider the whole ET at each iteration. This, however, tends to yield poor results, because subcircuit selection can choose in the first iteration a subcircuit feeding primary outputs of high significance and thus severely hinder approximation possibilities in the following iterations. A better approach, which we choose to follow, is to start with a small fraction of the ET at the first iteration and monotonically increase it at each iteration. This effectively guides subcircuit selection to choose subcircuits feeding primary outputs of low, but increasing, significance, allowing for a higher percentage of the whole circuit to be approximated.

Following a monotonic approach enables many ET increase strategies. These can be, for instance, linear or exponential increases at each iteration, or a combination of both. SUBXPAT allows for fine-tuning of this and many other of its elements, as discussed in the next section.

VI. FINE-TUNING SUBXPAT

The SUBXPAT framework is very modular, accepting different strategies to accomplish its many steps. Here we discuss strategies relating to SMT encoding, in Section VI-A, gate labelling, in Section VI-B, subcircuit selection, in Section VI-C, and iterative ET increase, in Section VI-D.

A. SMT Encoding

How a problem is encoded into an SMT formula can have a significant effect on the solver's performance. Here we describe two encoding choices that led to order of magnitude performance improvement to two SUBXPAT steps.

While the structure of the double quantified formula defined in Section III-C is inherent to (SUB)XPAT, the encoding of some of its functions can be done in different ways. Functions map and $dist$ are defined over the sets S and $Dist$, which can be instantiated using different SMT theories. The more natural instantiation is to substitute both sets by $\mathbb{Z}$, yielding formulas over the theory of linear integer arithmetics (LIA). Although intuitive, this encoding requires the solver to engage in the expensive interplay between its SAT and theory engines, required by the conflict-driven clause learning with theory propagation approach followed by modern solvers [24]. Since we are dealing with circuits, an alternative is to substitute S and $Dist$ by $\mathbb{B}^m$, yielding instead formulas over the theory of fixed-size bit-vectors (BV). Since bit-vectors are very close to plain Boolean values, the solver will flatten, i.e., bit-blast, the formula and forward it to its SAT engine, leading to a computationally cheaper solving that does not need to engage with a theory engine.

The graph-based constraints described in Section IV-C can also be handled in different ways. The naive approach is to encode all nodes and edges as plain Boolean values. This, however, makes the solver need to infer the relations between the constraints during solving. Alternatively, abstract data types (ADTs) can be employed, which directly provide the solver the structural information inherent to the problem and prevent the need for solving inference.

B. Partial Labelling

As stated in Section IV-C, we label each gate with the minimum non-zero error that can be induced by modifying the logic of (only) said gate. An important aspect of SUBXPAT's labelling is that, to reduce computation time, we do not label all the circuit's gates, as this can become a bottleneck when applying SUBXPAT to large circuits. Instead, we only label a relevant subset of gates at each iteration. This is realised by first labelling the gates feeding the primary outputs, and then recursively labelling their predecessors *as long as their label is lower than the given ET*. Stopping when the ET is reached prevents wasting computation on labelling gates that will never be selected for approximation. Lastly, since the labelling of one node is independent of the labelling of other nodes, parallelising the process is a straightforward way of improving performance.

C. Subcircuit Selection Feasibility Criteria

While we found that having all outputs be feasible is a good subcircuit selection strategy in general, SUBXPAT allows for

different feasibility criteria. Requiring only a subset of outputs to be feasible – as opposed to all – is a straightforward relaxation of our chosen strategy. Conversely, having many outputs be jointly, rather than individually, feasible, i.e., the sum of their labels respects the ET, is a stricter approach. The flexibility given by this spectrum enables SUBXPAT to be customised for specific use cases if need be. Additionally, soft SMT constraints can potentially be used to establish optional selection objectives, e.g., to allow some outputs to be unfeasible but require their labels to be as low as possible, which enables further customisation based on user needs.

D. ET Increase

We discussed in Section V the fact that (1) a different ET value can be chosen at every iteration, and that (2) we choose to select monotonically increasing values. Still, different increase strategies can favour different types of circuits and of selection criteria. We opt for a simple linear increase based on fractions of the whole ET, as we have observed that this leads to the best results overall. In specific cases, however, a combination of large and small increases might lead to better results, since this allows SUBXPAT to utilise approximation potential leftover after a large step. Large and small steps can be either linear or exponential, with different combinations being worth exploring in different contexts.

VII. EVALUATION

We carried out an extensive evaluation of SUBXPAT and compared it against three state-of-the-art methods. We first detail our experimental setup, in Section VII-A, and then present our results related to XPAT, in Section VII-B, and SUBXPAT, in Section VII-C.

A. Experimental Setup

We implemented SUBXPAT in a tool of the same name. The tool[1] is open-source under the MIT licence and contains the whole machinery of SUBXPAT and different strategies for each of its steps. We used Z3 [23] v4.11.2 to solve SMT queries. For synthesis, we used YOSYS [26] v0.23+21 and the Nangate 45nm cell library. Besides providing off-the-shelf push-button support for ALS, SUBXPAT's modularity also allows users to implement custom strategies for each of its steps and to potentially use different solvers and synthesisers.

To evaluate SUBXPAT we used 23 Verilog specifications of arithmetic circuits, listed in Table II. We compared our approach, in terms of circuit area and tool runtime, to that of MUSCAT, MECALS, and EVOAPPROX. The EVOAPPROX library only has approximations for specific circuits[2], with the designs we considered being: unsigned 8-bit adders `04A`, `8AS`, `0H4`; unsigned 12-bit adders `2MB`, `04U`; unsigned 16-bit adders `0MH`, `0KC`; and unsigned 7-bit multipliers `0CA`, `013`. Our miters used $dist(s_1, s_2) = |s_1 - s_2|$, which implicitly consider all circuit inputs. LPP and PPO bounds were both set to 6 and the ET values ranged from 1/16 to 1/2 of the maximal circuit error. The experiments were done on a Linux machine with a 3.30 GHz Intel Core i9 and 256 GB of RAM.

[1]See https://github.com/LP-RG/subxpat.
[2]See https://github.com/ehw-fit/evoapproxlib.

TABLE II: Circuits used in SUBXPAT's evaluation.

Benchmark	Description	Number of gates
adder i8 o5	4-bit adder	52
adder i12 o7	6-bit adder	82
adder i16 o9	8-bit adder	115
adder i20 o11	10-bit adder	145
adder i24 o13	12-bit adder	182
adder i28 o15	14-bit adder	223
adder i32 o17	16-bit adder	263
absDiff i8 o4	4-bit absolute difference	96
absDiff i12 o6	6-bit absolute difference	167
absDiff i16 o8	8-bit absolute difference	191
absDiff i20 o10	10-bit absolute difference	270
absDiff i24 o12	12-bit absolute difference	427
absDiff i28 o14	14-bit absolute difference	441
absDiff i32 o16	16-bit absolute difference	505
mul i8 o8	4-bit multiplier	163
mul i10 o10	5-bit multiplier	283
mul i12 o12	6-bit multiplier	434
mul i14 o14	7-bit multiplier	618
mul i16 o16	8-bit multiplier	865
mAdd i9 o6	3-bit multiply-and-add	118
mAdd i12 o8	4-bit multiply-and-add	227
mAdd i15 o10	5-bit multiply-and-add	370
mAdd i18 o12	6-bit multiply-and-add	554

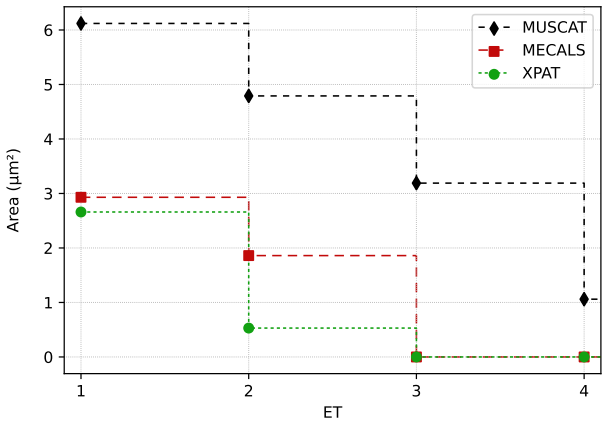

Fig. 9: Areas of approximations of a 2-bit adder.

B. XPAT Results

Our prior experiments [9] showed that XPAT yields good results for the benchmarks it can handle, but also showed that it has limited scalability. To highlight this point, we performed a study using adder circuits. Figure 9 shows the solutions found by XPAT, MECALS, and MUSCAT when trying to approximate a 2-bit adder with different ET values. As can be seen, XPAT yields better solutions than the compared tools. This, however, cannot be maintained for larger benchmarks, since XPAT's runtime increases exponentially with circuit size and quickly becomes intractable, as shown in Figure 10. The good solutions obtained with XPAT, coupled with its non-scalability, clearly motivate the need for SUBXPAT.

C. SubXPAT Results

To evaluate SUBXPAT we first performed five calibration studies to set its parameters, and then compared it against state-of-the-art methods. Calibration was done in terms of miter functions' instantiations, subcircuit feasibility, subcircuit

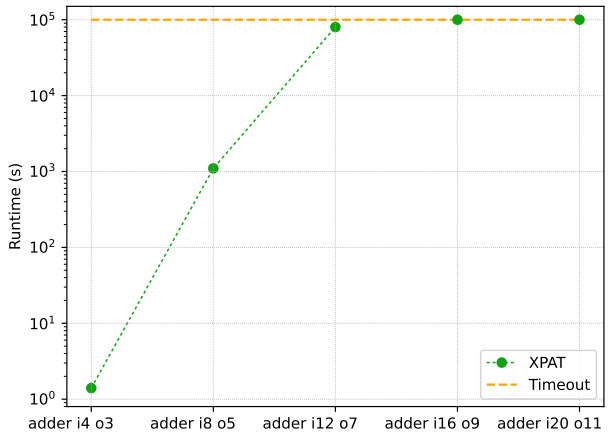

Fig. 10: XPAT runtime results for adders of different size.

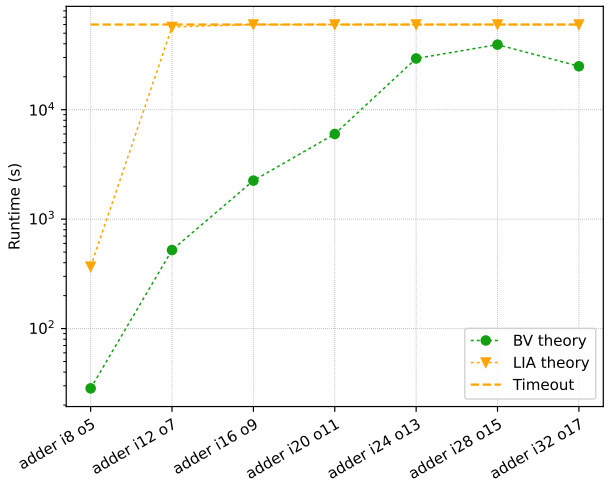

Fig. 11: SUBXPAT runtime results for adders of different size, using different SMT encodings.

dimensions, ET increase, and the resetting of constants, always using our adder benchmarks as will be explained shortly.

We used the Average Distance from Reference Set (ADRS) as a comparison metric in our experiments, inspired by its usage in the context of high-level synthesis [27], [28]. The ADRS measures the distance between the Pareto curve obtained by one particular method and the global Pareto curve, i.e., the curve obtained by considering all the results from the evaluated methods and retaining all non-dominated points. For each method, we collected the areas of the approximate circuits produced (or made available, in the case of EVOAPPROX) for certain ETs; while for MUSCAT, MECALS, and EVOAPPROX only the area of one approximation is reported per ET, for SUBXPAT the areas of all the intermediate designs are made available to the user. Having area and ET as our axes, the ADRS is formally defined as follows:

$$\text{ADRS}(\bar{P}, P) = \left[\frac{1}{|P|}\sum_{p\in P}\min_{\bar{p}\in\bar{P}}\delta(\bar{p}, p)\right] \times 100$$

$$\delta(\bar{p}, p) = \max\left\{0, \frac{A(\bar{p}) - A(p)}{A(p)}, \frac{E(\bar{p}) - E(p)}{E(p)}\right\}$$

where $\bar{P}$ and P are sets containing the points of the particular and global Pareto curves, and A and E are functions which extract the area and ET value of a given point. A low ADRS score is desirable, as it indicates that the curve of $\bar{P}$ is close to the curve of P.

1) Instantiation of Miter Functions: As discussed in Section VI-A, the miter functions *map* and *dist* can be instantiated using different SMT theories. We evaluated SUBXPAT's performance when using the LIA and BV theories, with the results obtained shown in Figure 11. As can be seen, the results corroborate our intuition that the BV theory leads to faster solving and (order of magnitude) faster SUBXPAT execution.

2) Subcircuit Feasibility Criteria: To evaluate the feasibility criteria, we considered the scenarios in which SUBXPAT selects subcircuits with at least one feasible output and subcircuits with all outputs feasible. The results obtained are summarised in Table III. As can be seen, our results indicate that requiring subcircuit outputs to be all individually feasible leads to better synthesised areas, as discussed in Section VI-C. Solutions for a 16-bit adder can be seen in Figure 12.

TABLE III: SUBXPAT results for two different feasibility criteria for subcircuit selection; runtime is given in minutes.

Benchmark	ADRS per criteria		Runtime per criteria	
	one feasible	all feasible	one feasible	all feasible
adder i8 o5	**0.00**	0.30	2.06	**0.47**
adder i12 o7	0.27	**0.00**	**3.77**	8.70
adder i16 o9	0.15	**0.01**	54.52	**37.48**
adder i20 o11	0.04	**0.01**	224.42	**99.95**
adder i24 o13	0.05	**0.03**	546.22	**489.57**
adder i28 o15	0.12	**0.00**	**269.94**	655.20
adder i32 o17	0.09	**0.02**	575.32	**415.77**
Average	0.10	**0.05**	**239.47**	243.88

3) Subcircuit Dimensions: To assess the effect of subcircuit dimensions in solution quality and in runtime, we evaluated our tool with different limits on the number of subcircuit inputs and outputs. We considered input/output (I/O) values of 2/1, 3/2, and 6/3. The results obtained are summarised in Table IV.

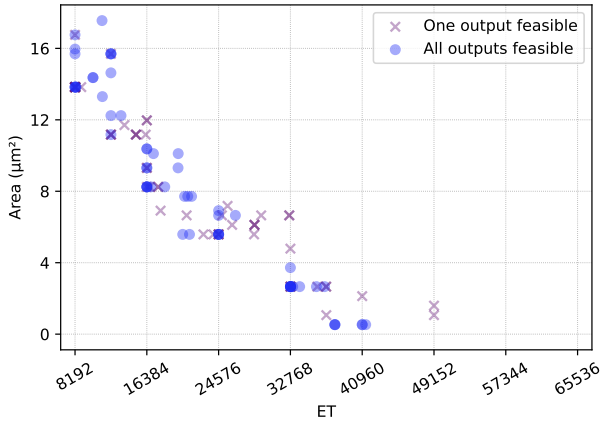

Fig. 12: Areas of SUBXPAT approximations of a 16-bit adder obtained with different criteria for subcircuit selection.

TABLE IV: SUBXPAT results for three different I/O limits for subcircuit selection; runtime is given in minutes.

Benchmark	ADRS per I/O			Runtime per I/O		
	2/1	3/2	6/3	2-1	3-2	6-3
adder i8 o5	0.17	0.15	**0.15**	**0.56**	1.04	3.09
adder i12 o7	1.70	0.27	**0.00**	**1.31**	2.09	13.83
adder i16 o9	0.53	0.09	**0.03**	**2.26**	3.94	233.55
adder i20 o11	0.37	0.04	**0.00**	**3.92**	7.33	35.64
adder i24 o13	0.57	0.43	**0.00**	8.57	**8.35**	102.46
adder i28 o15	0.44	0.12	**0.02**	**9.62**	11.33	160.80
adder i32 o17	1.00	**0.10**	0.17	**15.45**	16.38	61.38
Average	0.68	0.17	**0.05**	**5.96**	7.21	87.25

TABLE V: SUBXPAT results for two different ET increase strategies; runtime is given in minutes.

Benchmark	ADRS per strategy		Runtime per strategy	
	exponential	linear	exponential	linear
adder i8 o5	0.25	**0.05**	0.86	**0.47**
adder i12 o7	1.79	**0.00**	**8.24**	8.70
adder i16 o9	2.58	**0.00**	**35.49**	37.48
adder i20 o11	0.42	**0.00**	**8.71**	99.95
adder i24 o13	0.94	**0.00**	**49.98**	489.57
adder i28 o15	1.50	**0.00**	**209.54**	655.20
adder i32 o17	5.56	**0.00**	1176.24	**415.77**
Average	1.86	**0.01**	**212.72**	243.88

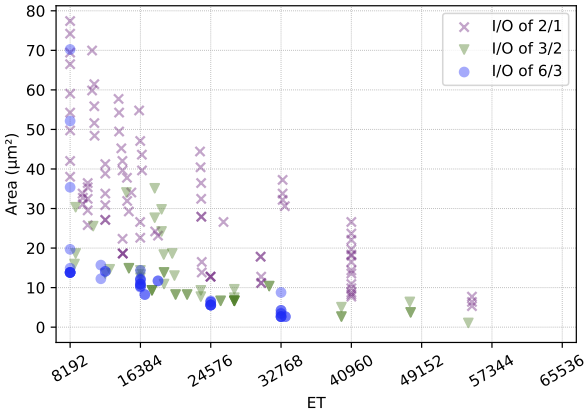

Fig. 13: Areas of SUBXPAT approximations of a 16-bit adder obtained with different I/O limits for subcircuit selection.

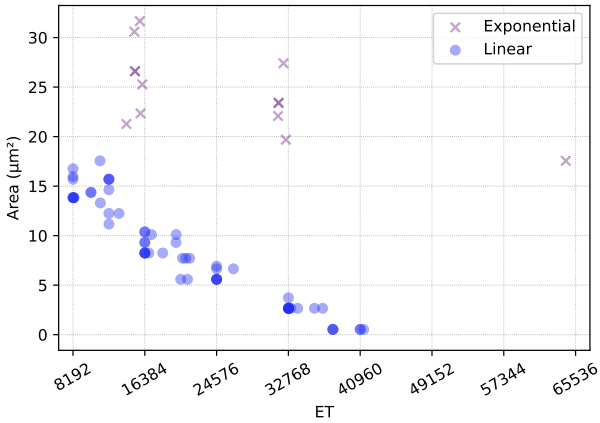

Fig. 14: Areas of SUBXPAT approximations of a 16-bit adder obtained with different ET increase strategies.

As can be seen, our results indicate that larger I/O limits lead to both better approximations and higher (yet tractable) runtimes. Solutions for a 16-bit adder can be seen in Figure 13.

4) ET Increase Strategies: To evaluate our choice of ET increase strategy (cf. Section VI-D), we compared the strategies of exponential increase and of linear increase. The results obtained, summarised in Table V, indicate that a linear ET increase leads to better synthesised areas. Solutions for a 16-bit adder can be seen in Figure 14.

5) Resetting of Constants: During the execution of its many iterations, SUBXPAT can set some primary outputs to be constant values. By default, this is a final decision and cannot be changed in following iterations. However, to enable greater flexibility, we consider allowing the resetting of such constants at each iteration, i.e., primary outputs previously set to 1 could be changed to 0 and vice versa. To assess the effect of this change, we evaluated our tool in two modes: one in which resetting is never allowed and one in which it is always allowed. The results obtained are summarised in Table VI. As can be seen, our results indicate that allowing for constants to be reset improves solution quality at the cost of increased runtime. Solutions for a 16-bit adder can be seen in Figure 15. Note that we achieve the resetting of constants seamlessly and automatically, by having the appropriate primary outputs be free variables in our SMT encoding; allowing the solver the flexibility to reset constants expands the design space to include potentially better approximations, with the exploration of the added space adding tractable runtime overhead.

6) Comparison Against the State of the Art: Having SUBXPAT configured to (1) use the BV theory, (2) have all outputs feasible as a subcircuit selection criteria, (3) use 6/3 as the I/O limit for subcircuit selection, (4) increase the ET linearly at each iteration, and (5) have constant resetting enabled, we now compare it against the state of the art. The results obtained are summarised in Table VIII. As can be seen, SUBXPAT yields better quality solutions in most cases, sometimes having an ADRS score of less than half than that of the other methods. In terms of runtime, SUBXPAT stands between MECALS and MUSCAT on average; runtimes for EVOAPPROX are not available[2]. For SUBXPAT, the average subcircuit size, in terms of subgraph nodes, was 18.06, and the average number of iterations was 16.61, as shown in Table VII. The solutions of all methods for eight of our benchmarks can be seen in Figure 17. Finally, and to summarize the results: out of the total 264 points collectively found in the global Pareto Curves of all benchmarks, 72% of these points (specifically, 192 out of 264) correspond to designs generated by SUBXPAT.

7) Effect on Circuit Delay and Power: We also evaluated how each method performed w.r.t. delay and power. While our focus is on area gains, this ideally should not come at a great cost in other metrics. We found that, overall, SUBXPAT yields approximations with delay and power similar to those obtained from the compared state-of-the-art methods. To illustrate, the area∗delay∗power product for the solutions shown for the 16-

TABLE VI: SUBXPAT results for two constant resetting modes; runtime is given in minutes.

Benchmark	ADRS per mode		Runtime per mode	
	Never	Always	Never	Always
adder i8 o5	0.17	**0.08**	1.92	**0.29**
adder i12 o7	0.22	**0.00**	9.88	**8.17**
adder i16 o9	0.41	**0.01**	**16.46**	36.73
adder i20 o11	0.04	**0.01**	**31.19**	99.07
adder i24 o13	0.05	**0.04**	**44.03**	488.68
adder i28 o15	0.06	**0.00**	**185.40**	653.83
adder i32 o17	0.04	**0.00**	**62.80**	413.83
Average	0.14	**0.02**	**50.24**	242.94

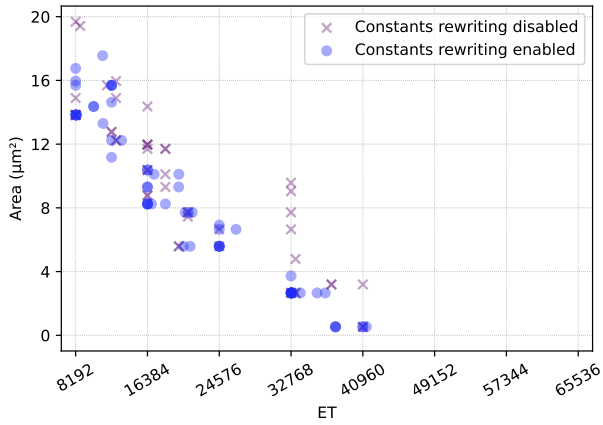

Fig. 15: Areas of SUBXPAT approximations of a 16-bit adder obtained with different constant resetting modes.

bit adder in Figure 17 (top right plot) can be seen in Figure 16.

8) Limitations: While we shown that SUBXPAT is able to generate lower area circuits than state-of-the-art methods, scalability remains its limitation. To quantify this: while MECALS reports the ability to process a 128-bit adder within 24 hours, SubXPAT scalability within 24 hours stops at a 20-bit adder. Improving on this limitation will be the subject of future work.

VIII. CONCLUSIONS

We proposed SUBXPAT, a novel iterative approach for ALS that is effective and efficient. It employs an SMT solver to search for good approximations of an exact circuit – operating iteratively on subcircuits of the original circuit and achieving significant area savings while maintaining efficiency. Our evaluation, conducted over a broad range of arithmetic circuits, indicates that SUBXPAT can yield better quality approximations than state-of-the-art methods, for circuits of size similar to the ones used. While limited in size, these circuits are useful in practice: for example, 8-bit approximated multipliers are extensively used in low-power neural networks [29], [30].

Avenues for future work include investigating (1) the use of alternative parametrical templates, e.g., templates with more than two layers of logic, (2) the formulation of the design space exploration in new ways, e.g., via quantified Boolean formulas (QBF), and (3) improving the runtime of local approximations, e.g., by attempting to avoid the use of double quantification in a sound manner. Such improvements could aid SUBXPAT to be able handle larger benchmarks, such as those in the ISCAS'85 [31] and EPFL [32] benchmark suites. Furthermore, a complementary line of research is the investigation of alternative error metrics, e.g., mean error as opposed to worst-case error, still in the context of template-based Boolean rewriting.

TABLE VII: SUBXPAT's average subcircuit size (in number of gates) and number of iterations, per benchmark and overall.

Benchmark	Avg. subcircuit size	Avg. number of iterations
adder i8 o5	8.61	9.12
adder i12 o7	9.88	12.50
adder i16 o9	12.08	13.88
adder i20 o11	14.02	15.88
adder i24 o13	16.84	12.50
adder i28 o15	17.18	15.50
adder i32 o17	18.00	15.88
absDiff i8 o4	10.39	7.38
absDiff i12 o6	14.74	13.00
absDiff i16 o8	18.51	18.38
absDiff i20 o10	19.89	16.50
absDiff i24 o12	21.65	21.75
absDiff i28 o14	21.02	21.12
absDiff i32 o16	23.74	21.62
mul i8 o8	11.88	16.75
mul i10 o10	19.79	18.00
mul i12 o12	23.77	19.38
mul i14 o14	25.11	20.25
mul i16 o16	31.14	21.00
mAdd i9 o6	12.76	14.88
mAdd i12 o8	17.64	19.62
mAdd i15 o10	21.79	19.00
mAdd i18 o12	25.67	18.25
Overall average	18.09	16.61

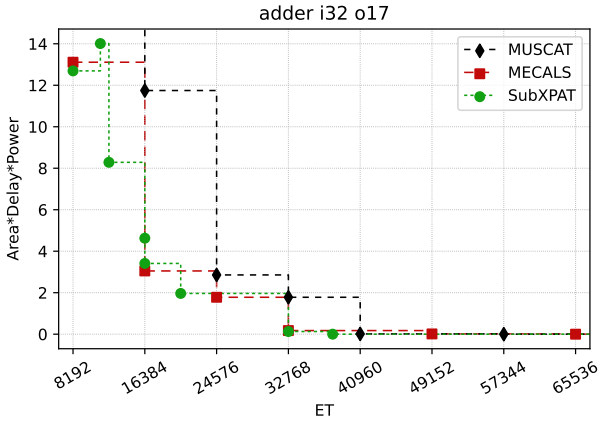

Fig. 16: Area∗delay∗power of adder i32 o17 approximations.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. Kim, "Approximate Computing: A Survey," *IEEE Design and Test*, vol. 33, no. 1, pp. 8–22, 2016.

[2] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate Logic Synthesis: A Survey," *Proc. of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

[3] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits," in *Proc. of the Design, Automation and Test in Europe Conference*, 2013, pp. 1367–1372.

[4] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and

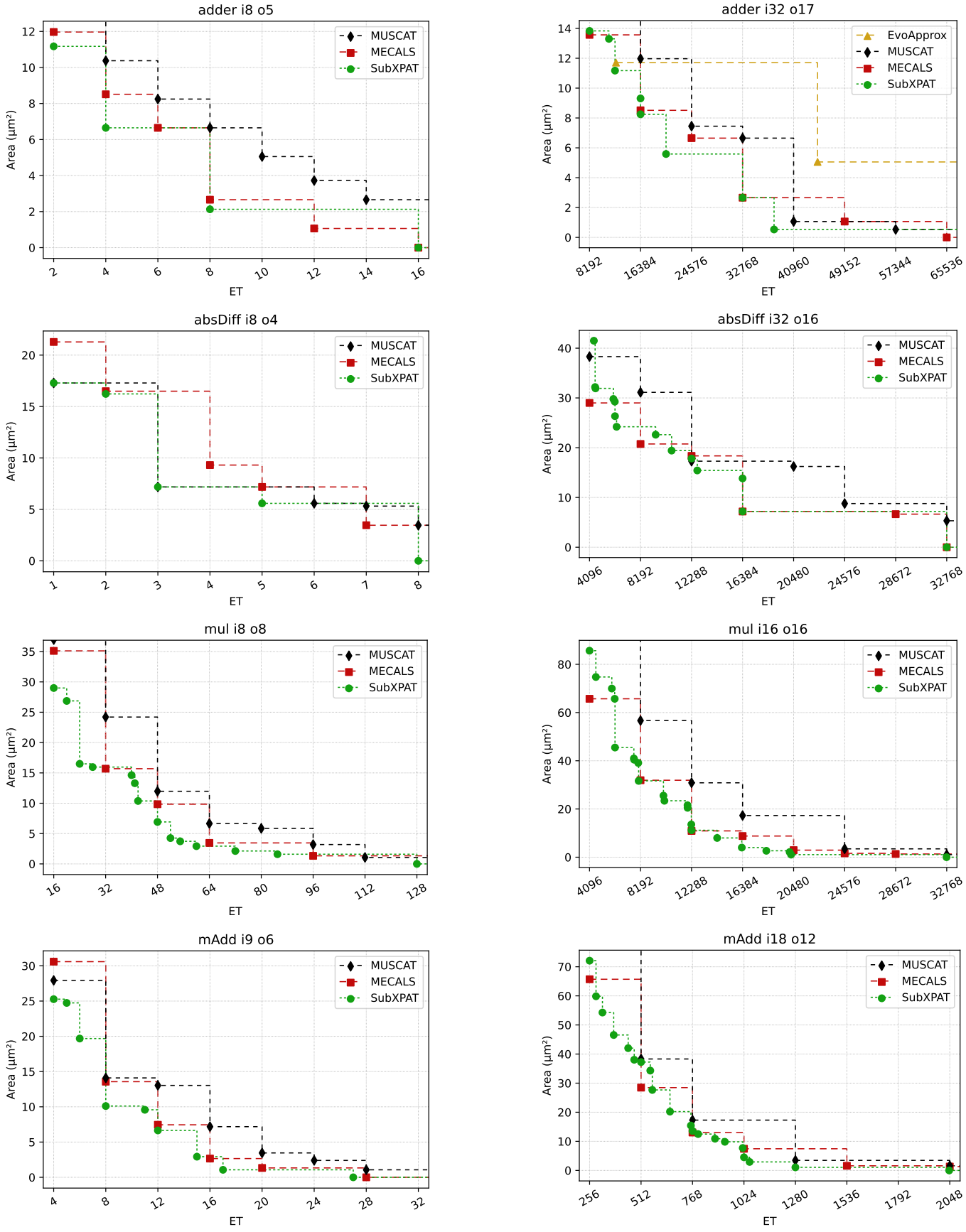

Fig. 17: Areas of SubXPAT, MUSCAT, MECALS, and EvoApprox approximations of eight benchmarks, two of each type.

TABLE VIII: Comparison between SUBXPAT and three state-of-the-art methods; runtime is given in minutes.

Benchmark	ADRS per method				Runtime per method			
	SUBXPAT	MUSCAT	MECALS	EVOAPPROX	SUBXPAT	MUSCAT	MECALS	EVOAPPROX
adder i8 o5	**0.07**	0.61	0.12	-	0.47	2.12	**0.02**	-
adder i12 o7	**0.02**	0.31	0.37	-	8.7	55.53	**0.11**	-
adder i16 o9	**0.00**	0.21	0.13	0.32	37.48	827.33	**0.71**	N/A
adder i20 o11	**0.00**	0.25	0.08	-	99.95	830.69	**0.89**	-
adder i24 o13	**0.02**	0.23	0.11	0.30	489.57	138.04	**2.13**	N/A
adder i28 o15	**0.00**	0.27	0.14	-	655.2	138.83	**4.34**	-
adder i32 o17	**0.00**	0.42	0.13	1.33	415.77	728.32	**31.52**	N/A
absDiff i8 o4	**0.02**	0.07	0.14	-	1.86	40.95	**0.03**	-
absDiff i12 o6	**0.03**	0.20	0.11	-	7.43	597.67	**0.23**	-
absDiff i16 o8	**0.04**	0.24	0.12	-	84.61	831.47	**2.38**	-
absDiff i20 o10	**0.07**	0.18	0.11	-	137.51	832.53	**6.57**	-
absDiff i24 o12	**0.05**	0.20	0.08	-	438.07	141.38	**15.45**	-
absDiff i28 o14	**0.04**	0.37	0.17	-	475.25	142.98	**29.64**	-
absDiff i32 o16	**0.03**	0.22	0.05	-	1130.27	143.54	**67.51**	-
mul i8 o8	**0.02**	0.33	0.18	-	25.77	831.14	**1.03**	-
mul i10 o10	**0.02**	0.26	0.18	-	158.74	832.9	**14.07**	-
mul i12 o12	**0.03**	0.48	0.15	-	375.28	835.96	**46.88**	-
mul i14 o14	**0.03**	0.30	0.07	1.96	990.48	**146.62**	289.83	N/A
mul i16 o16	**0.01**	0.42	0.12	-	2704.7	**155.34**	621.63	-
mAdd i9 o6	**0.01**	0.29	0.17	-	14.05	728.23	**0.12**	-
mAdd i12 o8	**0.01**	0.33	0.26	-	129.51	832.31	**4.65**	-
mAdd i15 o10	**0.01**	0.38	0.17	-	308.83	140.02	**19.57**	-
mAdd i18 o12	**0.01**	0.34	0.15	-	615.53	144.22	**71.31**	-
Average	**0.02**	0.30	0.14	0.98	404.57	439.05	**53.51**	N/A

Benchmarking of Approximation Methods," in *Proc. of the Design, Automation and Test in Europe Conference*, 2017, pp. 258–261.

[5] I. Scarabottolo, G. Ansaloni, and L. Pozzi, "Circuit Carving: A Methodology for the Design of Approximate Hardware," in *Proc. of the Design, Automation and Test in Europe Conference*, 2018, pp. 545–550.

[6] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization," in *Proc. of the Design Automation Conference*, 2018, pp. 1–6.

[7] L. Witschen, T. Wiersema, M. Artmann, and M. Platzner, "MUSCAT: MUS-based Circuit Approximation Technique," in *Proc. of the Design, Automation and Test in Europe Conference*, 2022, pp. 172–177.

[8] C. Meng, J. Sun, Y. Mai, and W. Qian, "MECALS: A Maximum Error Checking Technique for Approximate Logic Synthesis," in *Proc. of the Design, Automation and Test in Europe Conference*, 2023, pp. 1–6.

[9] M. Rezaalipour, M. Biasion, I. Scarabottolo, G. A. Constantinides, and L. Pozzi, "A Parametrizable Template for Approximate Logic Synthesis," in *Proc. of the IEEE International Conference on Dependable Systems and Networks Workshops*, 2023, pp. 175–178.

[10] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and Applications of Approximate Circuits by Gate-Level Pruning," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.

[11] S. Hashemi and S. Reda, "Generalized Matrix Factorization Techniques for Approximate Logic Synthesis," in *Proc. of the Design, Automation and Test in Europe Conference*, 2019, pp. 1289–1292.

[12] J. Ma, S. Hashemi, and S. Reda, "Approximate Logic Synthesis Using Boolean Matrix Factorization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 15–28, 2022.

[13] 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. of the Design Automation Conference*, 2019, pp. 1–6.

[14] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate Logic Synthesis by Resubstitution with Approximate Care Set," in *Proc. of the Design Automation Conference*, 2020, pp. 1–6.

[15] G. Liu and Z. Zhang, "Statistically Certified Approximate Logic Synthesis," in *Proc. of the International Conference on Computer Aided Design*, 2017, pp. 344–351.

[16] C. Meng, Z. Zhou, Y. Yao, S. Huang, Y. Chen, and W. Qian, "HEDALS: Highly Efficient Delay-Driven Approximate Logic Synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3491–3504, 2023.

[17] C. Meng, H. Wang, Y. Mai, W. Qian, and G. D. Micheli, "VACSEM: Verifying Average Errors in Approximate Circuits Using Simulation-Enhanced Model Counting," in *Proc. of the Design, Automation and Test in Europe Conference*, 2024, pp. 1–6.

[18] C. Meng, A. Mishchenko, W. Qian, and G. D. Micheli, "Efficient Resubstitution-Based Approximate Logic Synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 6, pp. 2040–2053, 2025.

[19] G. Ansaloni, I. Scarabottolo, and L. Pozzi, "Judiciously Spreading Approximation Among Arithmetic Components with Top-Down Inexact Hardware Design," in *Proc. of the International Symposium on Applied Reconfigurable Computing*, 2020, pp. 14–29.

[20] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 83:1–83:36, 2023.

[21] V. Leon, M. A. Hanif, G. Armeniakos, X. Jiao, M. Shafique, K. Pekmestzi, and D. Soudris, "Approximate Computing Survey, Part I: Terminology and Software & Hardware Approximation Techniques," *ACM Computing Surveys*, vol. 57, no. 7, pp. 1–36, 2025.

[22] ——, "Approximate Computing Survey, Part II: Application-Specific & Architectural Approximation Techniques and Applications," *ACM Computing Surveys*, vol. 57, no. 7, pp. 1–36, 2025.

[23] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proc. of the International Conference on Tools and Algorithms for Construction and Analysis of Systems*, 2008, pp. 337–340.

[24] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View*. Springer Berlin, Heidelberg, 2016.

[25] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "vZ - An Optimizing SMT Solver," in *Proc. of the International Conference on Tools and Algorithms for Construction and Analysis of Systems*, 2015, pp. 194–199.

[26] C. Wolf, "Yosys Open SYnthesis Suite," https://yosyshq.net/yosys.

[27] B. C. Schäfer and Z. Wang, "High-Level Synthesis Design Space Exploration: Past, Present, and Future," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2628–2639, 2020.

[28] L. Ferretti, J. Kwon, G. Ansaloni, G. D. Guglielmo, L. P. Carloni, and L. Pozzi, "Leveraging Prior Knowledge for Effective Design-Space Exploration in High-Level Synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3736–3747, 2020.

[29] M. S. Kim, A. A. D. Barrio, H. J. Kim, and N. Bagherzadeh, "The Effects of Approximate Multiplication on Convolutional Neural Networks," *IEEE Trans. on Emerging Topics in Computing*, vol. 10, no. 2, pp. 904–916, 2022.

[30] C. Meng, W. Burleson, W. Qian, and G. De Micheli, "Gradient Approximation of Approximate Multipliers for High-Accuracy Deep Neural Network Retraining," in *Proc. of the Design, Automation and Test in Europe Conference*, 2025, pp. 1–7.
[31] J. P. Hayes, M. C. Hansen, and H. Yalcin, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design and Test of Computers*, vol. 16, no. 03, pp. 72–80, 1999.
[32] L. Amaru, P.-E. Gaillardon, and G. D. Micheli, "The EPFL Combinational Benchmark Suite," *Proc. of the International Workshop on Logic and Synthesis*, pp. 1–5, 2015.

Morteza Rezaalipour received his MSc degree in computer architecture from the K. N. Toosi University of Technology, Iran, in 2019, and his PhD degree in informatics from the Università della Svizzera italiana, Switzerland, in 2025. He is a team leader at MicroLab in the Department of Computer Engineering at K. N. Toosi University of Technology. His research interests include approximate logic synthesis, low-power design, and memory systems.

Marco Biasion received his BSc degree in informatics from the Università della Svizzera italiana, Switzerland, in 2023, where he is currently pursuing a MSc degree in informatics with specialisation in computer systems. His research interests include approximate logic synthesis, reconfigurable architectures, and hardware optimisation.

Francesco Costa received his BSc degree in informatics from the Università della Svizzera italiana, Switzerland, in 2023, and his MSc degree in computational science from the same institution, in 2025. His research interests include computer simulations, computational biology, and bioinformatics.

Cristian Tirelli received his MSc degree in computer engineering from the University of Cassino and Southern Lazio, Italy, in 2020. He is currently pursuing a PhD degree in informatics at the Università della Svizzera italiana, Switzerland. He was awarded an SNSF Mobility Grant that allowed him to stay one year (2022/23) as a visiting researcher at the University of California at Los Angeles, USA. His research interests include compiler methods, hardware accelerators, and reconfigurable architectures.

Lorenzo Ferretti received his PhD degree in informatics from the Università della Svizzera italiana, Switzerland, in 2020, where he then held a position as a Postdoctoral Researcher until 2021. After being awarded an SNSF Postdoc Mobility Fellowship he then joined the University of California at Los Angeles, USA. Currently, he is a Principal AI Architect in the Silicon System AI group at Micron Technology. His research interests include high-level synthesis, graph-learning applied to hardware optimisation, and generative models applied to test generation.

Dr. Ferretti serves as a reviewer for different IEEE journals and conferences and is currently part of the technical program committee of the Design, Automation and Test in Europe Conference.

Rodrigo Otoni received his PhD degree in informatics from the Università della Svizzera italiana, Switzerland, in 2023, where he then held a position as a Postdoctoral Researcher until 2025. Currently, he is an Assistant Professor at the University of Groningen, Netherlands. His background is in formal methods and his research at present focuses on automated reasoning in the contexts of verification, synthesis, and certification.

Dr. Otoni recurrently serves as a reviewer for journals and conferences such as ACM TOSEM, ACM TOPS, FMCAD, TACAS, and CAV. He was recently awarded a grants from the Hasler Foundation and the SNSF for his independent research.

George A. Constantinides received his PhD degree in electrical and electronic engineering from Imperial College London, UK, in 2001. Since 2002, he has been with the faculty of Imperial College London, where he is currently a Professor of Digital Computation and the Head of the Circuits and Systems Research Group.

Prof. Constantinides serves on several program committees and has published over 200 research papers in peer-refereed journals and international conferences. He was the General Chair of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays in 2015 and is a Fellow of the British Computer Society.

Laura Pozzi received her PhD degree in computer engineering from Politecnico di Milano, Italy, in 2000. She is currently a Professor with the Faculty of Informatics, Università della Svizzera italiana, Switzerland. She was a Postdoctoral Researcher with the École Polytechnique Fédérale de Lausanne, Switzerland; a Research Engineer with STMicroelectronics, USA; and an Industrial Visitor with the University of California at Berkeley, USA. Her current research interests include automating embedded processor customization, high-performance compiler techniques, innovative reconfigurable fabrics, high-level synthesis design space exploration, and approximate computing.

Prof. Pozzi serves as an Associate Editor for the IEEE Transactions on Computer-Aided Design and for IEEE Design and Test.