A Parametrizable Template for Approximate Logic Synthesis

1st Morteza Rezaalipour
USI Università della Svizzera italiana
Lugano, Switzerland
morteza.rezaalipour@usi.ch

2nd Marco Biasion
USI Università della Svizzera italiana
Lugano, Switzerland
marco.biasion@usi.ch

3rd Ilaria Scarabottolo
USI Università della Svizzera italiana
Lugano, Switzerland
ilaria.scarabottolo@usi.ch

4th George A. Constantinides
Imperial College London
London, United Kingdom
g.constantinides@imperial.ac.uk

5th Laura Pozzi
USI Università della Svizzera italiana
Lugano, Switzerland
laura.pozzi@usi.ch

***Abstract*—This paper presents *XPAT*, a novel algorithm for the generation of approximate circuits which employs an SMT solver to shape the final resulting circuit on a given parametrizable template. The solver outlines which products of which input literals must be included in the final circuit in order to undergo a given error constraint. A miter is created containing the exact circuit description, the template, and a measure of the tolerated error, and by carefully tuning some template key parameters, such as limiting the number of literals per product, this algorithm is able to derive circuits that outperform the state of the art in terms of area. *XPAT* retrieved circuits with area smaller than those found by state of the art methods in 75% of the cases, and on average obtained 9.85% (up to 60.4% in some cases) improvement in area savings.**

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, and hence it should be traded for increased energy performance [1]. Approximate Logic Synthesis (ALS) is the process of deriving an approximate circuit, given an exact functionality and a tolerated error. There exist several families of ALS techniques, and a recent survey [2] distinguishes in particular between *Netlist modification*, where the resulting circuit is generated starting from an existing structure, for example by removal of gates or wires, and *Boolean rewriting*, where a circuit is created without reference to an original structure, for example by Boolean factorization.

This paper presents a novel Boolean rewriting algorithm, called *XPAT*, for the synthesis of approximate circuits which employs an SMT solver to shape the final result on a given parametrizable *template*. In particular, the template is a sum of products, and the solver outlines which products of which input literals must be included in the final synthesis for every circuit output. A miter is created containing the exact circuit description, the template, and a measure of the tolerated error between the exact and approximate output. By carefully tuning some template key parameters (for instance, the maximum number of literals that can appear in each of the products), this algorithm is able to synthesize circuits that outperform the state of the art in terms of area.

In the rest of the paper, Section II presents the most relevant works in the state of the art, Section III gives the motivation for this work, Section IV describes the proposed methodology, whose results are presented and discussed in Section V. Finally, Section VI concludes the paper.

II. State of the art

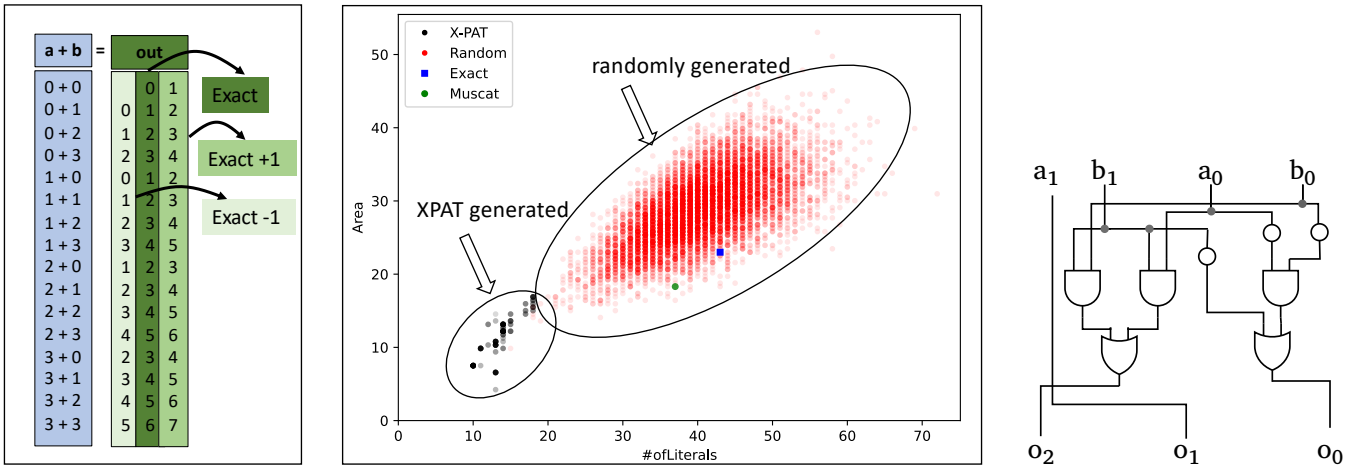
Several techniques have been proposed for ALS [2]. Some of them operate on a circuit netlist by removing some of its components – GLP [3] and Circuit Carving [4] by removing gates from a circuit, SASIMI [5] by modifying circuit wiring, EvoApproxLib using an evolutionary approach to create approximate variants [6]. Other techniques operate on the circuit truth table – BLASYS [7] employing matrix factorization for replacing subcuits, AIG-rewriting [8] enumerating cuts within a circuit and then replacing some with approximations.

In a recent work [9], Witschen *et al.* propose an innovative methodology for ALS called MUSCAT, where each circuit edge is annotated with a cut-point that, if cut, removes such edge. By formulating ALS as a satisfiability problem, they search for *minimal unsatisfiable subsets*, which in turn correspond to approximate circuits within the error threshold. The authors showcase the superiority of this approach to a number of state-of-the-art techniques, including AIG-rewriting [8] and EvoApproxLib [6].

The experiments in Section V compare our strategy to MUSCAT and BLASYS, and demonstrate how our strategy outperforms the state of the art in terms of area of the approximate circuits – with some limitations in scalability, discussed in the same section.

III. Motivation

Consider the truth table for an exact 2-bit adder – shown in Figure 1a) and visualized using integer values. If we are willing to tolerate an error of at most 1, how many truth tables exist satisfying this constraint? Since for each input



(a) (b) (c)

Fig. 1: a) There are $3^{15} * 2$ = about 22 million distinct truth tables for approximate 2-bit adders which do not exceed an error of 1. b) In red: randomly generated 50K of such 28M TTs, then synthesised, and plotted area vs. total number of literals of a minimized sum of products. In black, the circuits generated by *XPAT*. c) The circuit returned by *XPAT*.

combination we can have either the exact output or its value plus one or minus one, there are $3^{15} * 2$ = 28,697,814 distinct satisfying truth tables. An ideal ALS methodology is one that outputs a circuit corresponding to the truth table which, out of the 28 million possible ones, occupies the smallest amount of area once synthesized. We generated 50,000 random truth tables among the satisfying 28 million, synthesized them, and plotted them (area vs. the total number of literals in their minimised sum-of-products). Figure 1b) shows the results. As can be observed: 1) the area after (multi-level) synthesis is in clear correlation with the number of literals of a minimised 2-level circuit. 2) The area of the circuits generated by *XPAT* is remarkably small, and smaller than the best circuit returned by MUSCAT. Figure 1c) reports the circuit that *XPAT* returned. In the following, we explain how.

IV. Methodology

XPAT generates an approximate circuit by selecting a set of parameters in a template. The template considered in this work is a sum of products, and the parameters select which products of which literal should drive each output of the circuit. We first explain the SMT formulation we use to reach this goal, and then detail the design of the chosen parametrical template.

A. SMT Formulation

Figure 2 illustrates the miter employed in our formulation. Consider a circuit (the exact circuit, on the left of the figure) with n primary inputs and m primary outputs. In order to define an error between an approximate and an exact circuit, we define two functions, w and d. Function w denotes the mapping from Boolean values to word-level values (in the case of our experiments, generating unsigned integers). The second function, d, is a difference measure between those values (in our experiments it is set to $|w_1 - w_2|$, for absolute error).

The miter also contains an approximate circuit (shown in the right part of Figure 2) which in our methodology is expressed as a *parametrical template* driven by a set of parameters p. The next subsection will explain the template in detail – for now, suffice to say that the choice of the parameter values completely specifies the functionality of the approximate circuit. Given, then, a parameter combination p expressing an approximate circuit, and given an input combination i, function $d(i, p)$ expresses the error made by approximate circuit p for input combination i.

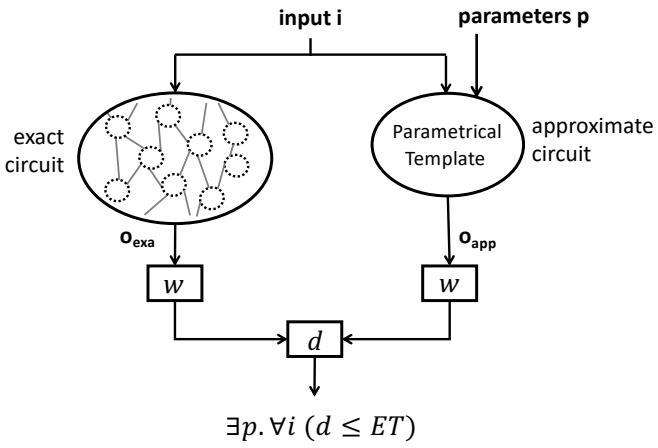


Fig. 2: The miter for the presented ALS algorithm. The approximate circuit functionality is determined by a set of parameters p, chosen by the solver. The distance d between the two outputs should be no higher than a given threshold ET for every possible input combination i

If our aim is to generate an approximate circuit that never exceeds a given error threshold ET, then, for this circuit to be valid, the distance between exact output and approximate output must be smaller than threshold *for every* input combi-

nation. The question we pose to the SMT solver is whether an assignment to the parameters exists such that, for every input combination, the resulting error is always below the threshold: $\exists p \forall i : d(i,p) <= ET$

B. Template definition

The parametrizable template for the approximate circuit is expressed as a sum of products. Given m primary outputs $o_1, ..., o_m$, each output is set to:

$o_i = \bigvee_{k=1}^{K} P_k$

where K is the (limited) number of products included in each output sum. Figure 3a) depicts this structure. Figure 3b) then shows how each product P_k is encoded in our miter: A number n of multiplexers (one per primary input) are *and*-ed together; the selector of each multiplexer is a 2-bit parameter p_j which chooses the desired option for input j out of three possibilities: the input itself, its negation, or value 1. If value 1 is passed on, the input does not appear in the final product.

More specifically, the first bit of p_j is set to true if the corresponding primary input is included in the product, while the second bit indicates its sign. If the first bit is set to false the corresponding input is not considered and the output is 1. In this case, we force the second bit to be true in order to avoid symmetries.

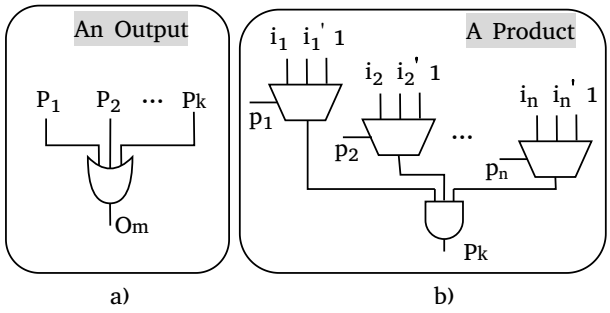


Fig. 3: a) Structure of an output o_m as a sum of K products. b) Structure a product: a 2-bit parameter p_i is used to select whether the product includes input i, or its negation, or excludes it by passing value 1. Vectors $[p_1, ..., p_n]$ hence represent all possible products of at most n literals.

Product P_k is then represented by a $2n$-bit vector, $P_k = [p_0, ..., p_n]$, with n the number of primary inputs. For instance, given $PI = \{a, b, c\}$ a set of primary inputs, product $P_0 = [(1,1),(1,0),(0,1)]$ translates to ab'; product $P_1 = [(0,1),(0,1),(1,0)]$ translates to c'.

The final result is, then, a set of parameters expressing a sum of products for each of the circuit primary outputs. As a very last note, since a constant zero in output cannot be expressed by the template detailed above, a further 1-bit parameter is added in order to select whether an output is set to constant 0, or is set to the sum of products detailed above. The total number of bits needed to define the template for a circuit of n inputs and m outputs, with K products per sum, is, then, $m(2nK + 1)$.

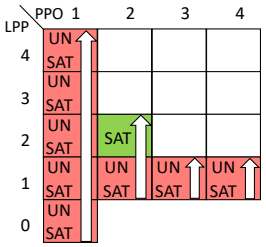


Fig. 4: The number of maximum products per output (*PPO*) and maximum literals per products (*LPP*) are input parameters to miter generation. The space of the possible combinations of these two values is then explored, via iterative solver calls, with the aim of minimizing the resulting approximate circuit area.

C. Limiting the number of literals of the resulting circuits

To search small-area solutions first, we can ask the solver whether a circuit exists exactly meeting a given specification, i.e. not exceeding a given pair of values: the number of allowed products per output (*PPO*), and the number of allowed literals per product (*LPP*). For the 2-bit adder and the constraint ET=1 of the motivational example, the smallest satisfying pair (*PPO*, *LPP*) is (2,2), corresponding to the circuit shown in Figure 1c).

In order to find such pair(s), we explore the space of these values in a column-wise manner, as illustrated in Figure 4. Starting with $PPO = 1$ and $LPP = 0$ (zero literals per product corresponds to each output being driven by a constant), a miter is generated accordingly, and the SMT solver is invoked. If a satisfying set of parameters does not exist, *LPP* is increased, a new miter is generated, and the SMT solver is again invoked. The search proceeds column-wise, until either the end of the column is reached or until a SAT is returned (green cell in the figure). Then, *PPO* is increased, and the next column is explored. White cells in the figure are dominated and hence need not be explored.

V. Experimental evaluation

Experimental Setup. To assess the efficiency of *XPAT*, we have applied it to a number of arithmetic circuits (adder, absolute difference, multiplier, multiply-add) with up to 12 bits in input, and we have compared it with the MUSCAT [9] and BLASYS [7] ALS algorithms. Area numbers were obtained by synthesizing the circuits retrieved by *XPAT*, MUSCAT and BLASYS using yosys and library gscl45nm. Experiments are carried out on a Linux Machine with a 3.30GHz Intel Core i9 CPU and 256GBs of RAM.

Area Results. Figure 5 shows the area of the circuits generated by the three methods, for error thresholds (denoted by ET) ranging from $1/8$ up to $1/2$ of the maximum error. The shaded blue shows the areas where *XPAT* could not scale, i.e. could not find a valid circuit within two hours. The area retrieved are smaller than those found by the state of the art methods in 75% of the cases.

Scalability and Limitations. While the method outperforms state of the art for the benchmarks used in this paper, runtimes increase beyond hours for larger benchmarks. This

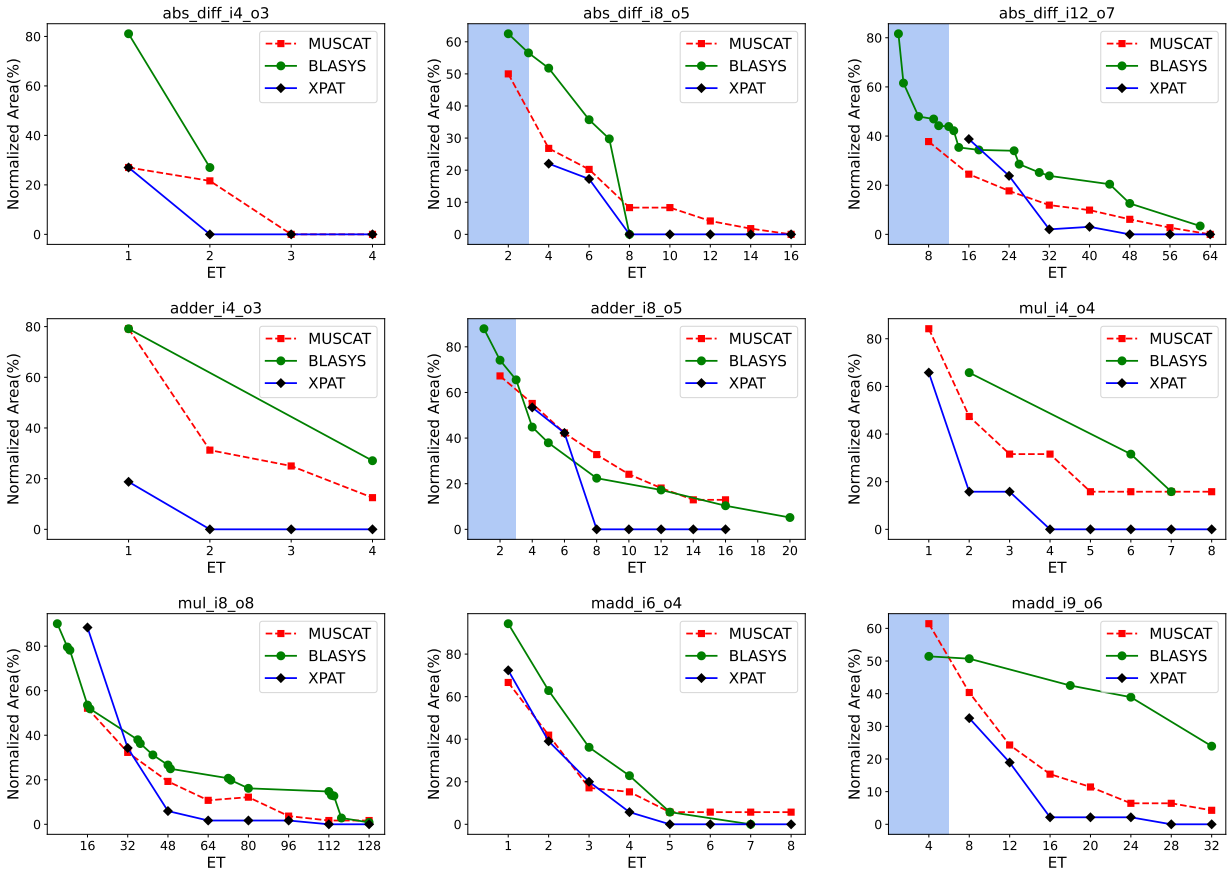


Fig. 5: Comparison of areas obtained by our proposed approach against MUSCAT [9] and BLASYS [7] (the letters "i" and "o" stand for the number of primary inputs and outputs of the circuits, respectively.).

limitation might be overcome by considering a multi-level template, as opposed to the two-level one considered here, or by applying XPAT to subparts of the circuit, iteratively, as opposed to the whole. This is the future direction that we are going to explore.

VI. Conclusion

We have presented an innovative ALS method called *XPAT* which generates approximate circuits via the use of a parametrizable template. An SMT solver finds a combination of parameters that shape the template so that a certain error threshold is met. The approximate circuit is thus built bottom-up and from scratch, along the guide of the template, as opposed to by removing or modifying parts of an existing circuit as many previous works do. *XPAT* is shown to improve on the state of the art, especially for large errors. Where it could scale, *XPAT* retrieved circuits smaller than those found by MUSCAT and by BLASYS in 75% of the cases. On average it obtained 9.85% improvement in area savings (up to 60.4%).

References

[1] Q. Xu, T. Mytkowicz, and N. Kim, "Approximate computing: A survey," *IEEE Design and Test*, vol. 33, pp. 8–22, Jan. 2016.

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

[3] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and applications of approximate circuits by gate-level pruning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 1694–1702, Feb. 2017.

[4] I. Scarabottolo, G. Ansaloni, and L. Pozzi, "Circuit Carving: A methodology for the design of approximate hardware," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 545–550, Mar. 2018.

[5] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 1367–1372, Mar. 2013.

[6] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 1–6, 2017.

[7] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate logic synthesis using boolean matrix factorization," in *Proceedings of the 55th Design Automation Conference*, pp. 55:1–55:6, June 2018.

[8] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *Proceedings of the International Conference on Computer Aided Design*, pp. 1–8, Nov. 2016.

[9] L. Witschen, T. Wiersema, M. Artmann, and M. Platzner, "Muscat: Mus-based circuit approximation technique," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 172–177, 2022.