Circuit Carving: A Methodology for the Design of Approximate Hardware

Ilaria Scarabottolo, Giovanni Ansaloni and Laura Pozzi
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland
Email: {ilaria.scarabottolo},{giovanni.ansaloni},{laura.pozzi}@usi.ch

***Abstract*—Systems-on-Chip (SoCs) commonly couple low-power processors and dedicated hardware accelerators, which allow the execution of high-workload and/or timing-critical applications while relying on constrained resources. The functions performed by accelerators are often robust with respect to approximations that, when implemented in HW, can lead to circuits with tangibly lower area and power consumption. Research in approximate computing aims at developing effective strategies to explore the ensuing correctness/efficiency trade-off.**

In this context, we address the challenge of approximate circuit design in an innovative way, called here Circuit Carving, which consists in identifying the maximum portion of an exact circuit that can be discarded from it, or carved out, to derive an inexact version not exceeding an error threshold. We achieve this goal by proposing an algorithm based on binary tree exploration, bounded by conditions extracted from the circuit topology.

Our approach can be applied to any combinatorial circuit, without a-priori knowledge of its functionality. The proposed algorithm allows back-tracking in order to never be trapped in local minima, and identifies the exact influence of each circuit gate on the output correctness, resulting in inexact circuits with higher efficiency and accuracy with respect to state-of-the-art greedy strategies.

I. Introduction

The common practice for the design of digital Integrated Circuits (ICs) involves a compromise among three metrics: performance (throughput, latency), required resources (silicon area, energy/power envelope) and flexibility. Higher performance can then only be obtained by increasing the cost of an implementation, or by specialising it for a limited set of applications. Approximate computing challenges this scheme by adding a novel dimension to the IC design space: that of the expected correctness of the computation being performed at run-time.

At the foundation of the inexact computing paradigm is the observation that, for many applications, small deviations from correct outputs can be tolerated, either because many acceptable solutions exist (e.g.: in searching algorithms), or errors are not perceived by users (as in multimedia), or data is already corrupted to a degree in the acquisition stage (e.g.: in sensor networks). In all these scenarios, exact processing is unnecessary — even wasteful, from a resource usage viewpoint, if an inexact alternative requiring less transistors and/or energy can be envisioned.

A necessary step towards the design of inexact ICs is the development of combinational circuits with maximum performance and minimum cost for a given (small) loss in accuracy. Recently, inexact digital implementations of commonly-used arithmetic operators (such as adders and multipliers) have been proposed in literature (see Section II).

Similarly to [1], [2] and [3], we instead address this challenge from a different and more general viewpoint, by proposing a method that can be applied to any combinatorial circuit. We illustrate a framework called Circuit Carving that, having as input a generic combinatorial netlist, identifies its largest sub-circuit that can be discarded, *carved out* of the original one, without violating a user-specified error constraint. As opposed to the above-mentioned works, which are based on iterative simplifications on the input circuit, we cast the approximation problem as a binary search tree exploration which exploits accurate knowledge of the circuit gate-level error distribution. While this formulation has exponential complexity, we improve the scalability of our proposed method by implementing search-pruning criteria that exploit the circuit topology in order to effectively reduce exploration time.

Our framework leads to smaller and more energy-efficient inexact implementations with respect to alternatives based on greedy or heuristic algorithms. It results, on average, in an Energy-Delay-Area Product (EDAP) reduction of 33,4% compared to the strategy described in [2].

The paper contributions are summarised as follows:

- We detail a novel methodology, called Circuit Carving, for the design of inexact digital circuits, which explores the search space of the exact circuit subsets.
- We introduce pruning conditions that can effectively decrease the mentioned search space, and hence increase the framework scalability, by disregarding search space regions that cannot contain candidate solutions.
- We preliminarily identify each circuit gate influence on the output precision, guiding the exploration to accurate results.

II. Related work

Interest in approximate computing techniques has soared in recent years. Notable efforts in the field have been surveyed by Han et al. [4] and, more recently, by Mittal et al. [5].

The trade-off between the conflicting objectives of correctness and efficiency has been explored at widely different levels of the hardware/software stack. At the application level,

978-3-9819263-0-9/DATE18/©2018 EDAA

algorithmic [6] and compiler-driven [7] transformations have been proposed to lower workloads at run-time, while preserving as much as possible the quality of generated outputs. At the architectural level, under-supplied [8], [9] or time-starved [10] systems have been introduced, which operate in voltage or frequency regions outside their safe operational area, and adopt mechanisms to limit the impact of the ensuing hardware failures.

Most related to our work are circuit-level approximations, which aim at simplifying the gate-level implementations of boolean functions. The energy and area benefit of approximate circuits is two-fold: first, they can be implemented with fewer transistors. Second, they may disregard some inputs and outputs, tangibly reducing the amount of associated storage [2]. Entire functional units [11] can be constructed using approximate circuits as building blocks, realising complex functionalities such as neural networks [12], [13].

Many works in circuit-level approximations target specific arithmetic units (adders in Ye et al. [14], multipliers in Kulkarni et al. [15] and Rehman et al. [16]). Recently, a number of generic methodologies have been proposed, which can automatically approximate *any* combinatorial gate-level netlist. To this end, the authors of [17] express approximability in the form of don't cares, and simplify a circuit by using existing don't care optimisation techniques. The authors of [1], [2] and [3] perform an iterative greedy selection of the nets and gates which can be simplified out of a circuit, while the method described in Vasicek et al. [18] is based on an evolutionary heuristic. As opposed to these latter works, our framework allows the exploration of the entire design space derived from different approximation choices. Moreover, it provides the exploration algorithm with accurate knowledge of gate-level errors induced on the circuit output. Hence, it exposes better optimisation opportunities with respect to greedy alternatives, as highlighted by the comparison, presented in Section IV, with the Gate-Level Pruning (GLP) methodology introduced by Schlachter et al. [2].

Our method is based on a binary search tree exploration strategy, which has been successfully implemented on many optimisation problems in the IC design landscape, such as the identification of high-performance instruction set extensions [19] and system components [20] under architectural constraints. We explore its applicability, for the first time, to the inexact computing landscape.

III. Problem Formulation

The digital logic implementing a boolean function can be represented as a Direct Acyclic Graph (DAG) $G(N, E)$, where each node $n \in N$ represents a gate. An edge $e(a, b) \in E$ represents a connection from node a to node b, in that the output of node a is used by node b.

Nodes are annotated with weights $D(n)$, indicating the maximum difference visible at the circuit outputs, when setting the output of the node n to a constant value. The weight of an edge is equal to that of its destination node: $D(e(a, b) \in E) = D(b)$.

A cut $C(N_C, E_C)$, with $N_C \subset N,\ E_C \subset E$, identifies a subgraph of the original graph G. An example of cut is shown by the dashed line in Figure 1a. We define the set $O(C)$ of the outgoing edges of a cut as

$$O(C) = \{e(a_k, b_k) \in E_C \mid a_k \in N_c, b_k \notin N_c\}$$

If the outgoing edges of a cut are set to a constant value, 0 or 1, then the circuitry contained in the cut does not contribute to the circuit logic anymore, and the cut can therefore be suppressed. The graph $(G \setminus C)$ is, hence, a candidate inexact circuit.

The difference of a cut $D(C)$ is defined as the sum of the differences of its outgoing edges, and represents the maximal error introduced by removing from the exact circuit the gates corresponding to the nodes in C. In Figure 1a, cut C_1 has two outgoing edges, with weight 8 and 4 respectively, therefore $D(C_1) = 12$; the methodologies that can be employed to derive such weights are explained later on, in Section III-B; for now we shall assume these values known.

Circuit Carving (CC) aims at finding a cut $C_m(N_m, E_m)$ such that $D(C_m) <= T$, where T is a pre-defined error threshold, and

$$\forall C_i(N_i, E_i) \mid D(C_i) <= T \wedge i \neq m,\ |N_i| \leq |N_m|.$$

In other words, find the cut with the maximum number of gates whose removal from the exact circuit does not incur in a maximum error exceeding T.

A. Exploration Algorithm

To find such cut, the algorithm explores a search tree representing all subsets of graph G. Each level n of the exploration represents the inclusion (1-branch) or exclusion (0-branch) of node n in a cut. Therefore, each cut $C(N_C, E_C)$ is represented by its corresponding path in the binary tree; Figure 1b shows the path corresponding to cut C_1 of Figure 1a.

Since this formulation has worst-case exponential complexity, pruning criteria are necessary to reduce the search space and obtain a more scalable framework.

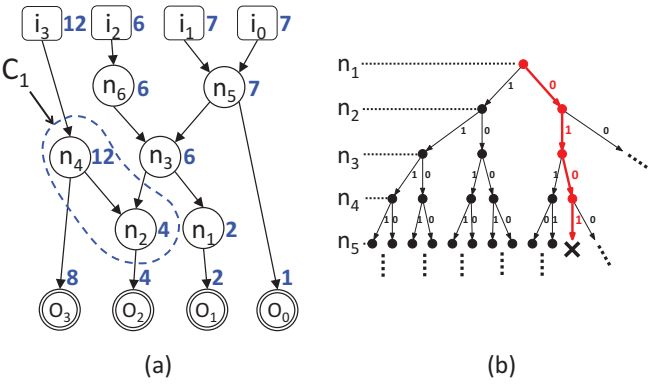


Fig. 1: a) Example of a graph G, and of a cut C_1 within it. Labels besides nodes represent their weight. The difference $D(C_1)$ of cut C_1 is 12 (sum of the differences of its outgoing edges). b) The binary search path corresponding to cut C_1, outlined in bold, red. If $D(C_1) > T$, the search in this path stops and the algorithm backtracks to the previous node.

We define valid any cut with a difference $D(C)$ not larger than T. Validity can be exploited as a pruning criterion when node exploration is performed in reverse topological order, i.e., $\forall e(u,v) \in E$, v is explored before u (as done in Figure 1a). In this way, $D(C)$ monotonically increases w.r.t. the exploration depth, because any outgoing edge of a cut cannot be recovered. If at some level i the difference overcomes the pre-defined error threshold T, the inclusion of further nodes in the cut cannot result in a valid candidate, and the search can be stopped (as in Figure 1b, considering an error threshold T smaller than 12). We call this the *validity* criterion and its effectiveness is quantified in Section IV-C.

The second pruning criterion is derived from the *closure* property of a cut. We define a cut to be closed if for $\forall n \in N$ of the original graph G:

$$\begin{cases} if\ \forall\ n_k \mid \exists\ e(n,n_k),\ n_k \in N_C \Rightarrow n \in N_C \\ if\ \forall\ n_k \mid \exists\ e(n_k,n),\ n_k \in N_C \Rightarrow n \in N_C \end{cases}$$

In other words, in a closed cut, if all children of a node n belong to the cut, n must also be in the cut. Likewise, if all parents of a node n are in the cut, n is in the cut as well. Figure 2a depicts an example of a cut (C_2) that is *not* closed.

A cut that is not closed *cannot* be the solution to the Circuit Carving problem, because there always exists a larger cut with the same difference. We indeed call such (larger) cut *closure*: formally, the closure $CL(C)$ of a cut is the minimal closed subgraph of G containing the cut: $N_C \subseteq N_{CL(C)}$, $E_C \subseteq E_{CL(C)}$. Figure 2a depicts the closure of cut C_2, $CL(C_2)$.

The concept of closure is crucial for reducing exploration: only closed cuts must be considered as candidates, since non-closed cuts are, by definition, sub-optimal. Therefore, the computation of closure $CL(C)$ can be exploited to prune the search. In fact, whenever the algorithm is exploring a non-closed cut C, at level n_i, it computes its closure and verifies whether the next node n_{i+1} in the search belongs to $CL(C)$. If it is so, it means that the current cut *must* be expanded to include node n_{i+1}, otherwise it could never represent a solution to the problem. Hence, the 0-branch at node n_{i+1} can be disregarded. An example of this criterion at work is shown in Figure 2b, where the 0-branches at nodes n_3 and n_6 are not explored. We call this the *closure* pruning criterion, and its effectiveness is quantified in Section IV-C.

The third and last pruning criterion considers the number of gates included in the maximum cut found so far. If, at some level i, the sum of the nodes still to be considered plus the nodes already included in the cut is less than the size of the best candidate already found, the algorithm avoids exploring any further. We call this the *residual gain* pruning criterion and its effectiveness is also quantified in Section IV-C.

The three aforementioned pruning criteria contribute to reducing the binary search tree exploration run-time by orders of magnitude, making the CC framework more scalable. However, for large circuits and large error values, the complexity of the exponential search may still result in unreasonable execution times. In these cases, an upper bound can be added on the number of recursive calls; results in Section IV-B demonstrate that, even when the number of recursions is bounded, our CC framework is tangibly more performant when compared to the state-of-the-art.

B. Labelling

A preliminary step to the binary search tree exploration consists in identifying and assigning weights $D(n)$ to all nodes $n \in N$ — we call this process labelling. Output nodes weights are set to their bit significance (as can be seen in Figure 1a: output o_0 has weight 1, o_1 has weight 2, o_2 has weight 4 and so on) and then weights are propagated upward. A conservative way to implement the upward propagation is to compute the weight of each gate having fanout more than one (such as n_3 in Figure 1a) as the sum of the weights of its outgoing edges. Thus, $D(n) = \sum_j n_j,\ \forall n_j \in N \mid \exists e(n,n_j)$. This represents an upper bound for gate weights, since it would be impossible, when removing a gate, to produce a difference on the output larger than the sum of the bits reachable from that gate.

However, the sum-labelling algorithm proves to be too conservative in many practical cases, as the simple example of Figure 3a shows: here, the output of gate A is connected to output 1 and to output 2 through a *not* gate. It is clear that, in this case, it would be impossible to obtain a difference of 3 by switching off gate A, regardless of the circuit primary inputs.

The exact values of differences $D(n)$ for single gates can be obtained through exhaustive simulations. To do so, each gate (node n_k) is set to a constant value (for the experiments, we employed value 0) and the circuit $G \setminus n_k$ is simulated for all its input values; its output is then compared with the exact output and $D(n)$ is set as the maximum obtained error. By deducing the exact difference values, a larger number of gates can be considered for inclusion in candidate cuts, when compared to using the conservative sum-labelling algorithm and, hence, the performance of the obtained inexact circuits is remarkably improved. On the other hand, our definition of *cut* difference $D(C)$ as the sum of its outgoing edges is still a conservative estimate. This is because, as in the example of Figure 3a, in many practical cases the removal of a cut cannot influence all its reachable outputs simultaneously.

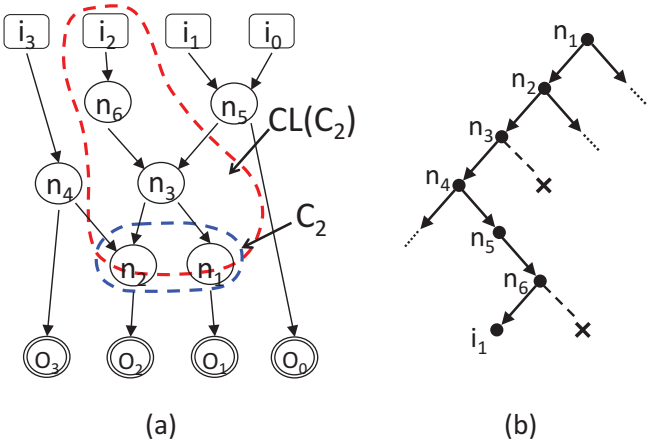
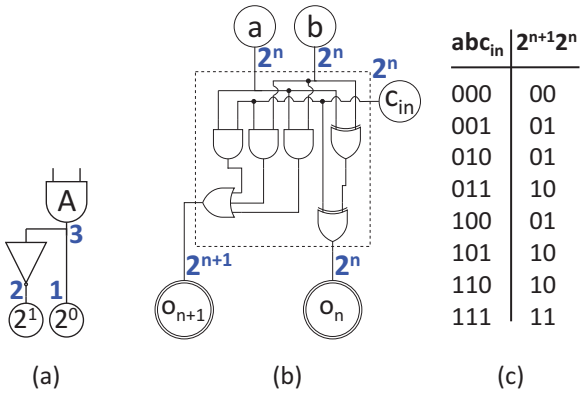


Fig. 2: a) Cut C_2 is not closed, and cut $CL(C_2)$ represents its closure. b) Exploration path for cut C_2. Since $n_3 \in CL(C_2)$, branch 0 from n_3 in Figure 2b is not explored.



abc_{in}	$2^{n+1}2^n$
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

Fig. 3: a) Example of two-gate circuit portion with sum-labelling. b) A full-adder with exact labelling. c) Its truth table.

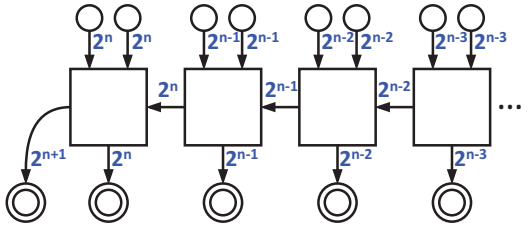


Fig. 4: A 4-bit ripple carry adder with labelling derived from weight-propagation.

Whenever the number of circuit inputs is too large to allow exhaustive gate-level simulations, two possible strategies can be adopted. The first one consists in running the simulations on a random subset of the circuit inputs, but this would invalidate the exactness of the labelling method. Otherwise, the graph topology can be exploited to infer, when possible, weight-propagation rules.

An example of the latter strategy application is the full-adder, depicted in Figure 3b. If we consider the last full-adder of a ripple carry chain (output weights, therefore, 2^{n+1} and 2^n), we can observe from its truth table in Figure 3c that by setting to a constant value one bit in $\{a, b, c_{in}\}$, while leaving the other two unchanged, the maximum obtainable difference is 2^n. Therefore, we can label the three inputs of the full adder with the weight of the less significant output bit, i.e., 2^n. Since this property holds for any of the previous full-adders of the carry chain, we can propagate it to derive the labelling for a ripple carry adder of arbitrary length (Figure 4).

In general, whenever it is possible to quantify the maximum difference on the output that can be produced by setting a gate (or a set of gates) to a constant value, a less conservative and, hence, more accurate labelling algorithm can be identified.

IV. Experimental evaluation

A. Experimental setup

We assessed the performance of our approach on three types of arithmetic circuits: combinational multipliers, ripple carry adders and carry lookahead adders. Circuit Carving can nonetheless be applied to any circuit, after its behavioural description has been translated to a gate-level netlist by a synthesis tool. We considered implementations with input bit-widths of 4 and 8 bits in the case of multipliers, and input bit-widths of 8 and 32 bits for adders.

Exact labelling of all gates was performed through exhaustive simulations of all input combinations, using the SIS framework [21], for the four benchmarks with smaller input number (4x4 multiplier, 8-bits ripple carry, 8-bits carry lookahead adder and 8x8 multiplier). For larger circuits, the topological considerations illustrated in Section III-B were adopted to propagate exact labelling values through the circuit graphs.

The binary search tree exploration algorithm detailed in Section III-A was implemented in C and run on the benchmark circuits under varying error constraints. As illustrated in Figure 5, the end-point of our approximation framework is a gate-level netlist, described in Verilog HDL, which can be readily verified with standard logic simulation tools, and synthesized to assess its area, delay and energy consumption. In the experimental evaluation presented in Section IV-B, hardware synthesis was performed employing Synopsys Design Compiler and a $40nm$ technology library. A critical path constraint of $1ns$ was imposed for the smaller circuits (in Figure 6, top row), while a $3ns$ constraint was used for the larger ones (bottom row of Figure 6).

B. Performance of inexact circuits

The trade-offs between performance and maximum tolerable error, for the inexact implementations designed with our framework Circuit Carving (CC), can be observed in Figure 6. The figure of merit on the vertical axis is the Energy-Delay-Area Product (EDAP), normalized with respect to the corresponding exact circuit. The horizontal axis, *error magnitude*, indicates the error constraint, as a percentage of the maximum output of the circuit.

For comparison, we implemented the methodology described in Schlachter et al. [2], called Gate-Level Pruning (GLP); Figure 6 hence reports the results of the circuits approximated using GLP, alongside with those approximated using CC. GLP is based on a greedy selection of nets iteratively simplified from an exact circuit, chosen in an order given by the (conservative) sum-labelling algorithm, and then simulated — for a subset of combinations of the inputs — to check for correctness. In practice, the GLP method corresponds to choosing *a single path* in the search space, determined by sum-labelling, as opposed to an entire exploration with backtracking, coupled with awareness of the exact weights, which is performed instead by Circuit Carving. This makes

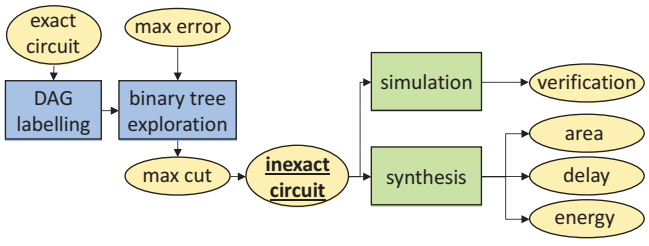


Fig. 5: Block scheme of the implemented framework.

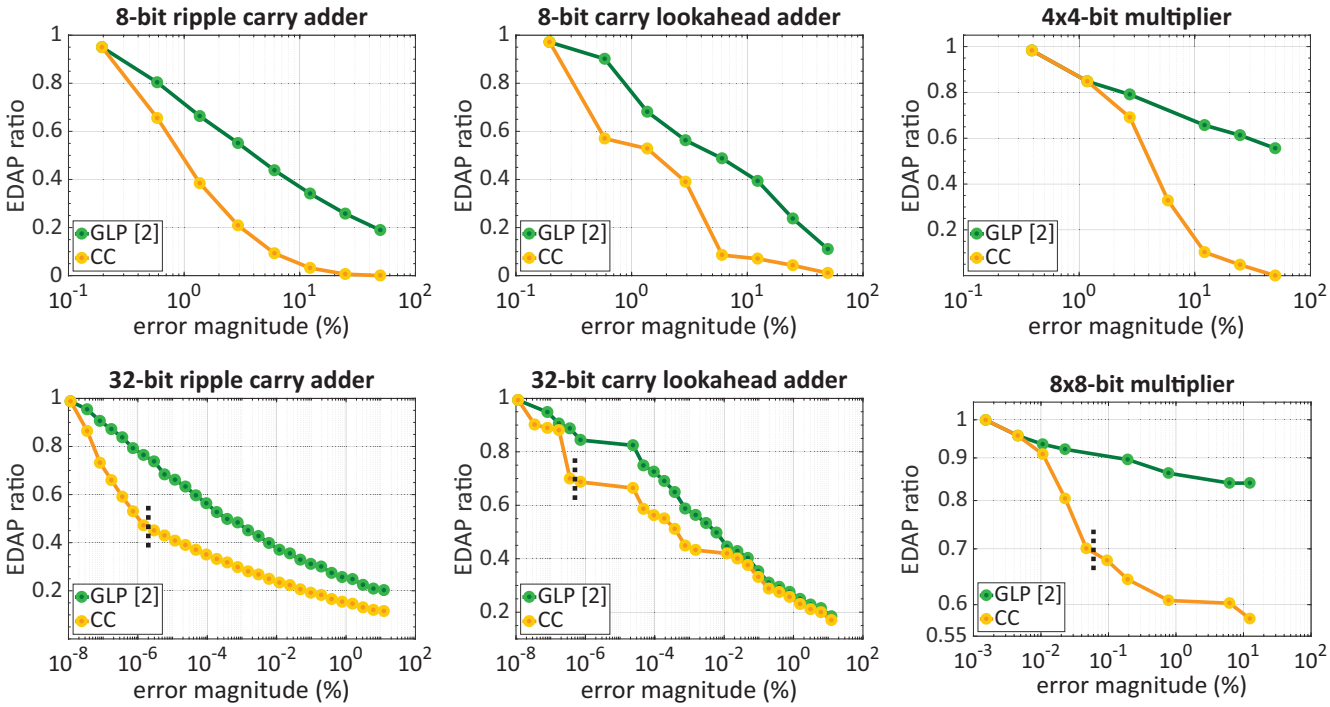


Fig. 6: Energy-Delay-Area Product (EDAP) of the inexact circuits derived from our methodology and from the greedy approach of Schlachter et al. [2], varying the error constraint. EDAP values are normalized with respect to the exact implementations.

CC more computationally expensive, but also more effective, in terms of finding larger sub-circuits to be eliminated from the exact circuit.

In fact, our framework consistently achieves better results across all benchmarks, i.e.: instances having smaller EDAPs for the same error constraint. As an example, CC reduces the EDAP of an 8x8-bit multiplier by 40% when an error magnitude of 1% (i.e., an error of 1% of the maximum output) is tolerable, while GLP only leads to a ∼10% EDAP reduction.

Note that, for the benchmarks processed in the top three graphs of Figure 6, our binary search tree exploration algorithm scaled for all possible error values. However, for the larger benchmarks processed in the bottom three graphs, while exploration did terminate for small error values, it did not terminate — within a threshold of one hour of computation — for larger error values. The divide is shown by the vertical, dashed line in the graphs. For the points to the right of the vertical dashed line, where complete exploration was not performed, we simply reported the best result achieved at the time of termination. It can be seen that the CC methodology, even when stopped short of finishing the exploration, still finds larger subgraphs to be removed, resulting in approximate circuits with lower EDAP, when compared to GLP.

C. Effectiveness of Search Space Pruning

To quantify the effectiveness of the three search space pruning criteria detailed in Section III-A (*validity*, *closure* and *residual gain*), we ran our exploration algorithm on the 8-bit ripple carry adder benchmark, each time turning off some of the criteria, and we plotted the number of calls required to complete exploration in each case. Note that, since the criteria are exact, when they are turned off the solution found at the end of exploration is always the same — but at the cost of more recursive calls being performed. Results are shown in Figure 7, which reports the number of recursive calls (capped at 10^9) required when different combinations of the three criteria were used.

The data shows that exploration does not terminate within 10^9 calls when the *validity* condition is not considered. Compared to *validity* alone and for an error magnitude of 5%, the combination of *validity* and *closure* results in a reduction of the search space of ∼10X, while applying *validity* and *residual gain* reduces it by ∼30X. When all three pruning criteria are used, a search space reduction of ∼600X is achieved. For larger errors, the difference between curves increases even more, to several orders of magnitude, and finally, for extremely large error values (which correspond to removing almost the whole circuit) exploration correctly converges extremely fast when the *residual gain* criterion is turned on.

V. Conclusion

We herein introduce a novel methodology, based on exact gate-level errors determination and binary search tree exploration, to design inexact combinational circuits. Our approach evaluates, as approximation candidates, entire circuit subgraphs, instead of iteratively selecting individual nets, as the state-of-the-art does. This wide exploration scope, along with the insights provided by accurate labelling, allows identification of highly efficient solutions (area- and energy-wise) under an error constraint.

For smaller designs, the proposed methodology scales, resulting in optimal inexact implementations. Larger designs

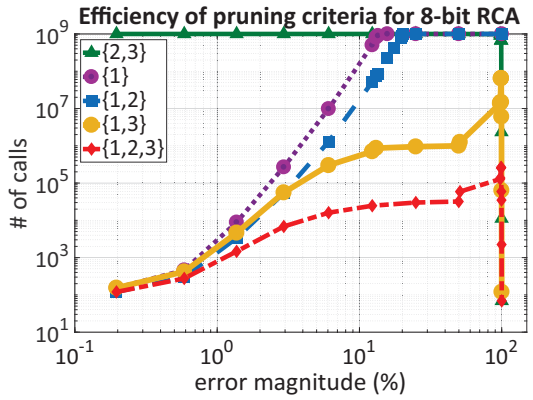


Fig. 7: Number of recursive calls in the binary search tree exploration algorithm, when different combinations of pruning criteria are applied: 1) *validity*, 2) *closure* and 3) *residual gain*.

were also tackled, adopting topological considerations to extend the labelling phase, and simply limiting the number of recursive calls performed by the exploration algorithm. As future work, we plan to further study these two aspects, by 1) devising a more generic and scalable labelling algorithm, and 2) developing a new exploration algorithm with lower average-case complexity and hence improved scalability with respect to the present one, which represents a first attempt at devising an exact solution.

The illustrated framework can be easily integrated in standard EDA tool-chains, allowing the use of approximate components in complex hardware blocks. A comprehensive study of the resulting performance and accuracy is a natural complement to this paper.

Acknowledgments

This work has been partially supported by the E4Bio (grant no. 200021-159853) and the MagicISEs (grant no. 200021-156397) projects evaluated by the Swiss NSF and by the MyPreHealth (grant no. 16073) project founded by the Hasler Stiftung.

References

[1] 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*, Mar. 2013, pp. 1367–1372.

[2] 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, no. 5, pp. 1694–1702, Feb. 2017.

[3] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Mar. 2014, p. 364.

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE European Test Symposium (ETS)*, May 2013, pp. 1–6.

[5] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, May 2016.

[6] A. Sinha, A. Wang, and A. P. Chandrakasan, "Algorithmic transforms for efficient energy scalable computation," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Jul. 2000, pp. 31–36.

[7] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, Sep. 2011, pp. 124–134.

[8] S. S. Basu, L. G. Duch, R. Braojos Lopez, G. Ansaloni, L. Pozzi, and D. Atienza Alonso, "An inexact ultra-low power bio-signal processing architecture with lightweight error recovery," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 2017.

[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *SIGPLAN Notices*, vol. 47, no. 4, Mar. 2012, pp. 301–312.

[10] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *MICRO 36: Proceedings of the 36th Annual International Symposium on Microarchitecture*, Dec. 2003, pp. 7–18.

[11] K. Nepal, Y. Li, R. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Mar. 2014, p. 361.

[12] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: energy-efficient neuromorphic systems using approximate computing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Aug. 2014, pp. 27–32.

[13] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2016, pp. 1–8.

[14] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of the International Conference on Computer Aided Design*, Nov. 2013, pp. 48–54.

[15] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proceedings of the 24th International Conference on VLSI Design*, Jan. 2011, pp. 346–351.

[16] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proceedings of the International Conference on Computer Aided Design*, Nov. 2016, pp. 1–8.

[17] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *Proceedings of the 49th Design Automation Conference*, Jun. 2012, pp. 796–801.

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

[19] L. Pozzi, K. Atasu, and P. Ienne, "Exact and approximate algorithms for the extension of embedded processor instruction sets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1209–29, Jul. 2006.

[20] W. Sun, M. J. Wirthlin, and S. Neuendorffer, "FPGA pipeline synthesis design exploration using module selection and resource sharing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 254–265, Jan. 2007.

[21] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Sis: A system for sequential circuit synthesis," EECS Department, University of California, Berkeley, Tech. Rep., 1992.