Journal of Systems Architecture 150 (2024) 103112

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

FPGA approximate logic synthesis through catalog-based AIG-rewriting technique

Mario Barbareschi[1,2], Salvatore Barone [*,1], Nicola Mazzocca[1,3], Alberto Moriconi[1]

Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Via Claudio 21, Naples, 80125, Italy

ARTICLE INFO

Keywords:
Automated design methodology
Approximate computing
Multi-objective optimization
Approximate logic synthesis
AIG rewriting
FPGA synthesis
Low area approximate circuits
Low power approximate computing circuits

ABSTRACT

Due to their run-time reconfigurability, short time-to-market, and lower prototype costs, FPGAs have become increasingly popular since their introduction. They found use in a wide variety of applications, including high-performance computing. However, when compared to ASICs, FPGAs offer lower performance, and they are power-hungry devices with low energy-efficiency. The emergence of Approximate Computing (AxC) represents a significant advancement in terms of enabling technology when applied to FPGA-based computing platforms. It has been effectively exploited in several application fields, achieving significant savings in energy and latency through a selective degradation of the output quality. Nevertheless, a generalized and systematic methodology for FPGA-based circuit design is still lacking. Indeed, most of the methods target ASIC-based systems, and, consequently, they offer minimal advantages or even an increase in resources when synthesized for FPGAs due to the architectural differences between the technologies.

In this paper, we attempt to address this shortcoming by introducing our method for designing combinational logic circuits. It is based on and-inverter graph rewriting and multi-objective optimization, aiming for optimal trade-offs between quality of results and hardware overhead. Extensive experimental campaigns empirically prove that both generic logic and arithmetic circuits benefit from this approach.

1. Introduction

Since their peculiar features, such as partial run-time reconfigurability, short time-to-market, and low prototyping costs, the popularity of Field Programmable Gate Array (FPGAs) has significantly increased since their first introduction. In facts, despite being low-performance, power-hungry and a lot less energy-efficient when compared to Application Specific Integrated Circuit (ASIC), FPGA is a common choice in a wide variety of applications, including high-performance computing, machine-learning, big-data analytics, and so forth. In this perspective, Approximate Computing (AxC) design paradigm offers a potential loophole to deal with FPGA overhead, since it allows for power and energy savings through a deliberated introduction of quality-degradation into intermediate computations, while keeping the final output quality within application constraints [1]. This is made possible by the inherent error-resiliency of many applications, due to self-healing properties of algorithms and computational patterns, redundant or noisy data, the existence of several equally correct outputs, or even perceptual limitations of end-users. Concerning hardware design, however, most of the research effort is devoted to obtaining performance improvements in ASIC-systems using gate or transistor-level approximation techniques. Conversely, the scientific literature includes only a restrained number of contributions pertaining to FPGA-based systems. Anyway, most of the discussed contributions rely upon either in-depth considerations and understanding of the underlying architecture of the target FPGA [2,3], or manual manipulation of Look-Up Tables (LUTs) [4], i.e, the main building block of the FPGA fabric.

Albeit being able to provide relevant results, the mentioned methodologies are neither scalable nor able to offer a significant approximate solution space, e.g., multiple trade-offs between the quality of result and hardware resource savings. Consequently, more systematic approaches have been proposed [5,6]. Nevertheless, as the authors themselves point out, the Achilles' heel of such approaches lies in the effort required to build accurate and faithful hardware resources predictors.

Bearing in mind the above, in this paper we resort to the And-Inverter Graph (AIG) rewriting technique, originally proposed in [7],

* Corresponding author.
E-mail addresses: mario.barbareschi@unina.it (M. Barbareschi), salvatore.barone@unina.it (S. Barone), nicola.mazzocca@unina.it (N. Mazzocca), alberto.moriconi@unina.it (A. Moriconi).

[1] Authors are listed in alphabetical order.
[2] Associate Member, IEEE.
[3] Member, IEEE.

https://doi.org/10.1016/j.sysarc.2024.103112
Received 19 October 2023; Received in revised form 5 February 2024; Accepted 13 March 2024
Available online 18 March 2024

but meant to be synthesized onto FPGA technology. In particular, we exploit the inherent affinity between k-feasible cut enumeration in AIGs and LUT mapping while employing Satisfiability-Modulo Theory (SMT)-based Exact Synthesis (ES) for AIGs to build approximate LUT configurations. The proposed approach makes advantage of a Multi-objective Optimization Problem (MOP) to select replacements leading to approximate configurations minimizing both error and AIG node-count, avoiding both machine-learning based predictors and high-fidelity estimators during the Design Space Exploration (DSE).

Resorting to the framework from [7], that is open-source and freely available at https://github.com/SalvatoreBarone/pyALS, we conducted a thorough experimental campaign to evaluate the approach against both generic-logic circuits and arithmetic circuits, while resorting to different error metrics.

Our original contribution can be summarized as follows: (i) we adapt the methodology from [7] to target the FPGA technology; (ii) we benchmark ALS based on AIG rewriting against several circuits from LGSynth and basic arithmetic circuits, executing more than 1200 different syntheses; (iii) through a thorough analysis, we deeply analyze how FPGA overhead, i.e., both the area and power consumption, correlates with AIG node-count, giving a detailed explanation based on the well-known power dissipation model for LUTs [8]; (iv) we check our proposal against the state-of-the-art, giving both qualitative and quantitative comparison; (v) we provide different case studies, targeting three different challenging hardware circuits, namely the Sobel edge-detector, Finite Impulse Response (FIR) and Convolutional Neural Network (CNN).

The remainder of this paper is organized as follows. Section 2 reviews relevant contributions from the scientific literature, while Section 3 first provides the reader with the required technical background, and then discusses in details the approximation methodology. Section 4 discusses the experimental setup and results, Section 5 discusses the effects of our approximation approach on the power dissipation model of LUTs, and Section 6 compares our approach with the state-of-the-art. Finally, before drawing conclusions in Section 8, in Section 7 we discuss several case-studies in which approximate circuits resulting from our method are exploited to reduce hardware requirements of larger and complex applications.

2. Related work

Performance of computing systems can be effectively enhanced by exploiting the AxC design paradigm [1] that trades off limited quality of results for performance gains or energy savings. AxC is applicable to a large variety of application fields, including signal, image and video processing [9,10], data analytics, machine learning [11–14], and even critical applications [15,16].

Anyway, most of the contributions focus on arithmetic circuits, since, usually, they are building blocks of larger complex applications [17,18].

Concerning hardware, the scientific literature distinguishes in timing and functional techniques [19]. While the former forces the circuit to operate on non-nominal voltage values or frequencies, the latter manipulates the logic being implemented. Starting from a reference implementation, logic can be altered, for instance, by reducing the precision of involved operands [20,21], exploiting the observability do not care set of nodes [22], by replacing near-identical pairs of signals [23], netlist transformations [24–26], abstract syntax tree transformations [27,28], and so forth. Furthermore, besides assuming a reference implementation as a starting point, the approach from [29] also allows generating approximate components from scratch through multi-objective Cartesian Genetic Programming (CGP).

All the above-mentioned approaches operate at the transistor or gate-netlist level, and allow for large silicon area and power savings while targeting ASIC. Nonetheless, due to the architectural differences between target technologies, savings achieved are far modest while targeting FPGA [5,30]. As a consequence, several works from the scientific literature proposed FPGA-based approaches for designing approximate components, especially arithmetic ones, exploiting the underlying architecture of the target FPGA.

Several contributions exploit the underlying architecture of fabrics to break the carry chain at one or multiple positions, while exploiting unused LUT inputs to predict the carry [3,31]. Authors of [4,30] exploits the structure of the 6-input lookup tables and carry chains of FPGAs, defining a methodology for recursively designing $n \times n$ multipliers based on $\frac{n}{2} \times \frac{n}{2}$ ones, optimized for FPGA-based systems. As drawbacks, rather than using the carry chain, the critical path delay and power consumption are reduced using additional LUTs to either compute or predict the carry for partial product summation.

In [6], the authors propose a methodology for designing application-specific approximate arithmetic operators for FPGA based systems. The methodology utilizes the 6-input LUTs and the associated carry chains to implement approximate operators while exploiting multi-objective Bayesian optimization to search for approximate multiplier that satisfy an application's accuracy and performance constraints. Furthermore, to cope with the large design space, various machine-learning models have been adopted to estimate the behavioral accuracy and corresponding performance gains of approximate circuits.

A different approach has been proposed in [5], that exploits machine-learning models to sift libraries of approximate components, which have been previously designed using ASIC-oriented approximation techniques. That methodology articulates in two distinct phases, i.e., the training of predictors for FPGA hardware requirements and the actual Pareto-front construction. It has been tested on circuits belonging to the EvoApprox8 library of approximate components [32], searching for those providing optimal trade-offs between error and FPGA overhead, specifically the power consumption. Nevertheless, the machine-learning based estimation approach of hardware resources is unfeasible if no library of approximate components is available, and, additionally, it demands high-fidelity estimators, that, as authors themselves observed, may be thoroughly cumbersome to achieve, even if a large library of components is available.

3. Catalog-based AIG-rewriting

As mentioned, we resort to the approach from [7], which is based on non-trivial local rewriting of AIGs and MOP and devised for ASIC technology. Hence, we provide the reader with essentials concerning these building blocks. In particular, we briefly introduce the AIG representation of digital circuits in Section 3.1, that is the circuit representation on which the approximation approach operates. Then, in Section 3.2 we provide full details concerning how k-feasible cuts within an AIG and their ES are exploited to introduce approximation. Finally, we discuss how to select approximate circuits minimizing both error induced by approximation and FPGA hardware requirements, i.e., how we perform DSE, in Section 3.4.

3.1. And-inverter graphs

An AIG [33] is a direct a-cyclic graph in which there are Primary Input (PI) nodes, which have no incoming edges, and logic-AND nodes, which have two incoming edges. Edges represent physical connections between nodes, and they can be marked as complemented or not.

Consider a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ and its set of input variables $\{x_1, \dots, x_n\}$. An AIG is formally defined as the set of nodes $\{x_{n+1}, \dots, x_{n+r}\}$ combined accordingly to Eq. (1), with r being the AIG size, i.e., its number of nodes, $s_{1i} < s_{2i} < i$ being indexes of the nodes and p_{1i}, p_{2i} being the polarity of the incoming edges of the ith node. Conventionally, the polarity of complemented edges is 0.

$$x_i = x_{s_{1i}}^{p_{1i}} \wedge x_{s_{2i}}^{p_{2i}} \qquad i \in [n+1, n+r] \tag{1}$$

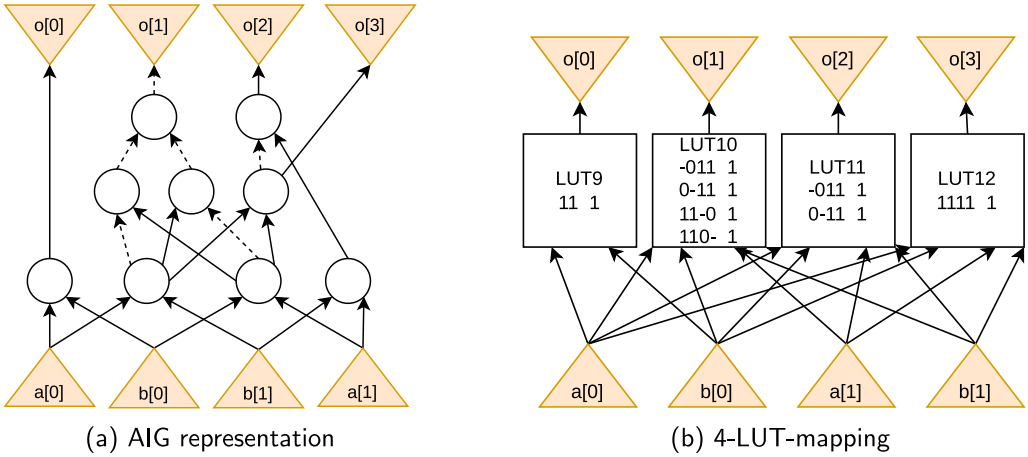


Fig. 1. AIG representation and 4-LUT-mapping of a 2-bits unsigned multiplier. Nodes implement the AND operation, solid line edges is the direct input, while dashed line edges represents inverted (i.e. complemented) input.

An AIG is said to realize the Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ i.f.f. Eq. (2) holds.

$$f_i = x_{s_i}^{p_i} \qquad i \in [1, m],\ s_i \in [n+1, n+r] \tag{2}$$

A set $\{x_{i_1}, \ldots, x_{i_k}\}$ of nodes is said to be a *path* of length k i.f.f. it is an ordered sequence of interconnected nodes starting at a PI and ending at a Primary Output (PO). The longest path is called the *critical path*. Consider the set of paths ending in a node x_i, described by Eq. (3): the (i, L) pair – consisting of the root node x_i and the set of leaf nodes $L \subseteq \{x_1, \ldots, x_{n+s}\}$ – defines a *cut* i.f.f.

(a) $\forall p \in paths(i),\ p \cup L \neq \emptyset$ i.e., all paths to x_i contain at least a leaf node from L, and
(b) $\forall l \in L\ \exists p \in paths(i) : l \in p$ i.e., each leaf in L is at least within a path to x_i.

$$paths(i) = \bigcup_{j \in children(i),\, j \neq 0} \{paths(j),\ i\} \tag{3}$$

A cut is *k-feasible* if $|L| \leq k$, where $|L|$ is the cardinality of L.

The set of all k-feasible cuts having x_i as the root node is recursively defined by Eq. (4).

$$cuts_k(i) = \begin{cases} \emptyset & i = 0 \\ i & i \in [1, n] \\ cuts_k(s_{1i}) \oplus_k cuts_k(s_{2i}) & i \in [n+1, n+r] \end{cases} \tag{4}$$

The $\oplus_k$ operator in (4) is the saturating union over all the combinations of subsets extracted from two sets, defined by (5).

$$M_1 \oplus_k M_2 = \{m_1 \cup m_2 : |m_1 \cup m_2| \leq k,\ m_1 \in M_1, m_2 \in M_2\} \tag{5}$$

3.2. Generating approximate variants

The approximation technique from [7] cleverly exploits k-feasible cuts, from now on k-cuts, as the mean to introduce approximation. In essence, k-cuts within the AIG for a concerned circuit are enumerated using a partial-enumeration algorithm, since a complete enumeration is unfeasible for $k \geq 6$ [34]. Then, approximate variants for the circuit are generated by superseding carefully selected cuts with approximate ones of better performances. It is worth noticing that partial k-cut enumerations is, from a functional perspective, equivalent to k-LUT mapping; therefore, partial cut-enumeration algorithms are effectively adopted for FPGA synthesis [34].

Consider, for instance, the AIG in Fig. 1(a), which represents a 4-inputs-4-outputs Boolean function implementing the 2-bits unsigned integer multiplication, and concentrate on the 4-feasible cut having the $o[3]$ node as the root node, and PIs as leaf nodes, i.e., the LUT-12 of Fig. 1. Such LUT implements the $o[3] = a[0] \wedge a[1] \wedge b[0] \wedge b[1]$ Boolean function, for which the constant zero is a possible replacement.

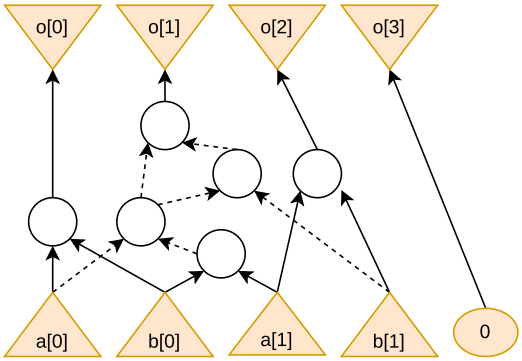


Fig. 2. Approximate configuration of the AIG of Fig. 1(a). This is obtained by replacing the $o[3] = a[0] \wedge a[1] \wedge b[0] \wedge b[1]$ Boolean function using the constant zero.

Then, by rewriting back the AIG, it will result in the Boolean function whose AIG is depicted in Fig. 2. The relationship between the size and the depth of AIGs and hardware requirements of the corresponding circuits suggests that whether the approximate circuit consists of fewer AIG nodes, then its hardware requirements will be lower than the original one, as suggested in [34].

To generate suitable replacements for k-cuts, authors of [7] exploit the ES. Indeed, ES could be effectively exploited to generate approximate variants for a given Boolean function f, by searching for a function $f' \neq f$ that requires less resources w.r.t. f yet it is acceptable according to some error metric. Consequently, for each unique k-cut resulting from the partial cut enumeration, the first step of the catalog-based AIG-rewriting approach consists of the generation of approximate k-cuts, i.e., k-cuts at increasing Hamming distance w.r.t. the original specification. In the following, we introduce the ES and the catalog-generation procedure.

3.3. Exact synthesis and approximate k-cuts generation

Essentially, the ES problem consists of finding a combinational circuit that realizes a given Boolean function specification, and that turns out optimal w.r.t. some cost criteria, which is usually the number of nodes and/or the circuit depth. So far, its computational complexity is

unknown, although the minimum circuit size problem, of which the ES is an instance, has been extensively studied and efficient algorithms for it are considered to be unlikely [35]. Nevertheless, solutions for the ES problem can be efficiently found by solving the decision problem in Eq. (6), which asks whether there exists an AIG of a given size r and a given maximum depth d that realizes a certain Boolean function f. In the following sections, we make use of the function *HasAIG(f, r, d)*: it either returns an AIG of size r (combined with a polarity p for the output node) that satisfies Eq. (6) if it exists, or *unsat* if such an AIG does not exist.

$$\exists\{x_{n+1},\dots,x_{n+r}\}, p\in[0,1] \ : \ \left(x_{n+r}^{p}=f\right)\wedge\left(l_{n+r}\leq d\right) \tag{6}$$

If the AIG depth is not considered, then (6) gets simplified as follows.

$$\exists\{x_{n+1},\dots,x_{n+r}\}, p\in[0,1] \ : \ \left(x_{n+r}^{p}=f\right) \tag{7}$$

Consider an n-inputs Boolean function $f:\mathbb{B}^n\rightarrow\mathbb{B}$. The ES problem formulation requires

(i) introducing $\{s_{1i}, s_{2i}\}$ indexes and $\{p_{1i}, p_{2i}\}$ Boolean variables, for $i\in[n+1, n+r]$;

(ii) enforcing constraint in Eq. (8), in order to both forbid cycles and to define an ordering between nodes;

$$s_{1i}<s_{2i}<i \qquad i\in[n+1,n+r] \tag{8}$$

(iii) encoding the logic-AND behavior of each node, as stated in Eq. (9);

$$b_i^{(t)}=a_{1i}^{(t)}\wedge a_{2i}^{(t)} \tag{9}$$

(iv) encoding PIs connection and enforcing values propagation through the AIG, using Eq. (10)

$$s_{ci}=j\Rightarrow a_{ci}^{(t)}=b_j^{(t)}\oplus\overline{p_{ci}} \qquad c=\{1,2\} \tag{10}$$

and, finally

(v) encoding the equivalence constraint and the actual function semantic (11).

$$b_{n+r}^{(t)}=\overline{p}\oplus f(t) \tag{11}$$

This formulation makes use of the explicit function representation – i.e., f is represented in terms of truth table values, for each of the possible 2^n input assignments. Moreover, in order to encode the behavior of the Boolean function for each input assignment, each node $i\in[n+1,n+r]$ is replicated once for each of the input vectors $t\in[0,\dots,2^n-1]$. The $a_{1i}^{(t)}$, $a_{2i}^{(t)}$ and $b_i^{(t)}$ variables represent, respectively, the value of input signals for the ith node and its output while the circuit input is set to the input vector $t\in[0,\dots,2^n-1]$. The $b_{n+r}^{(t)}$ node, which is the node having the largest index, is the root node of the AIG, i.e., the output node.

3.4. Design space exploration

As we mentioned, approximate variants for the circuit are generated by superseding carefully selected cuts with approximate ones of better performances. Hence, once the catalog is generated for each of the k-cuts – or k-LUT – within the circuit, the main challenge is to find replacements leading to Pareto-optimal trade-offs between quality of results and hardware requirements.

This requires coping with two major concerns: on one hand, the number of approximate variants and resulting approximate configurations grows quickly with the size of the concerned Boolean functions, and, on the other hand, preserving the quality of results while pursuing a reduction in hardware requirements are conflicting design goals.

As in [7], we cope with these exploiting MOP-based DSE, that is quite common in recent approaches from the scientific literature [36–39].

4. Evaluation

As mentioned, for evaluation purpose, we resorted to the tool from [7], that is released under GNU GPL3 open-source license[4].

Fig. 3 recaps and further details the work-flow. The tool takes the Hardware Description Language (HDL) implementation of the concerned circuit, and generates the corresponding AIG representation. Then, k-cuts enumeration is performed by leveraging fabric-independent k-LUT mapping, i.e, we do not consider any vendor-specific FPGA fabric at this stage. The catalog generation takes place as discussed in Section 3.2, and, as in [7], catalog entries are organized and stored in a database so that they can be subsequently reused[5].

The Archived Multi-Objective Simulated Annealing (AMOSA) heuristic [40] orchestrates the DSE: decision variables of the problem are k-LUT within the mapped circuit, and their domain is given by catalog entries. A LUT within the mapped circuit is randomly selected, and replaced using a suitable entry taken from the catalog; then, fitness-functions are evaluated to state the Pareto-dominance relationship between new candidate solutions and archived ones. While dominated candidate solution may be discarded by the heuristic, non-dominated ones are archived for further consideration. At the end of the DSE phase, the tool provides the HDL implementation for each non-dominated archived solutions, allowing for the final hardware implementations and measurement of the actual FPGA resource-requirement.

4.1. Experimental setup

For evaluation purpose, we considered a subset of the LGSynt91 generic-logic benchmark [41]. Furthermore, since they are building blocks for larger applications, such as Artificial Neural Networks (ANNs) [18,42–45], we also consider several arithmetic circuits, including various adders and multipliers. Specifically, we considered arithmetic circuits from the ArithsGen [46] and from the Arithmetic Module Generator [47,48] frameworks, as well as circuits from the EPFL Combinational Benchmark Suite [49].

We performed the ES of 4-feasible or 6-feasible cuts within the mentioned circuits, since a higher cardinality of AIG-cuts prohibitively increases the time needed to accomplish the ES [50]. The fitness-functions driving the optimization are error and hardware requirements, both to be minimized.

Pertaining to error metrics, as in [7], we resort to the Error Probability (EP) metric (12) when dealing with the LGSynth91, where the $[\![\cdot]\!]$ notation denotes the Iverson bracket (13).

$$e_{prob}(f,\hat{f})=\frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}[\![f(x)\neq\hat{f}(x)]\!] \tag{12}$$

$$[\![P]\!]=\begin{cases}1 \text{ if P is True}\\ 0 \text{ otherwise}\end{cases} \tag{13}$$

For what pertains to arithmetic circuits, we resort to the Absolute Worst-Case Error (AWCE) (14) as a metric for error assessment, assigning weights to POs according to their significance, i.e., the least significant bit is assigned a weight of $2^0=1$, the next one a weight of $2^1=2$, and so forth.

$$e_{awce}(f,\hat{f})=\max_{\forall x\in\mathbb{B}^n}|int(f(x))-int(\hat{f}(x))| \tag{14}$$

Regarding hardware-resource requirements, the number of FPGA LUTs, the power consumption and the maximum clock speed should be accurately measured for an accurate assessment. Anyway, this would require FPGA synthesis to be performed, leading the computational-time of DSE to be unsustainable. Therefore, as in [7], we resort to a

[4] Both the source-code and documentation are freely available at https://github.com/SalvatoreBarone/pyALS

[5] We made the database freely available at https://github.com/SalvatoreBarone/pyALS-lut-catalog

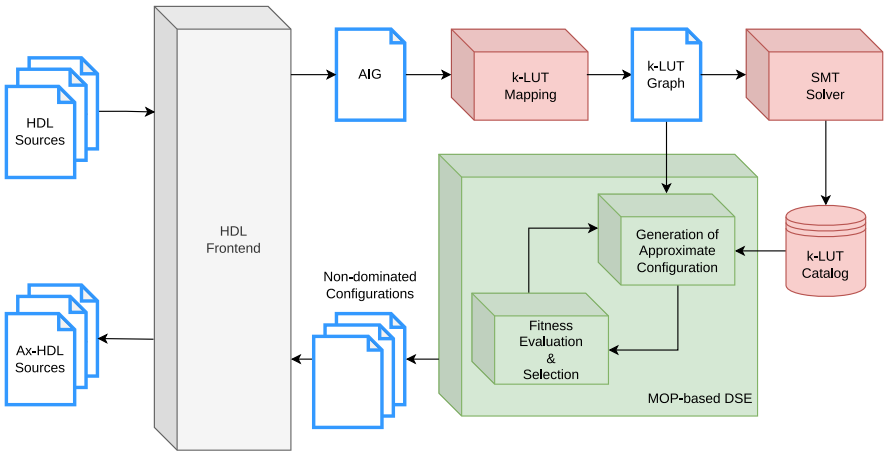


Fig. 3. Details of the workflow.

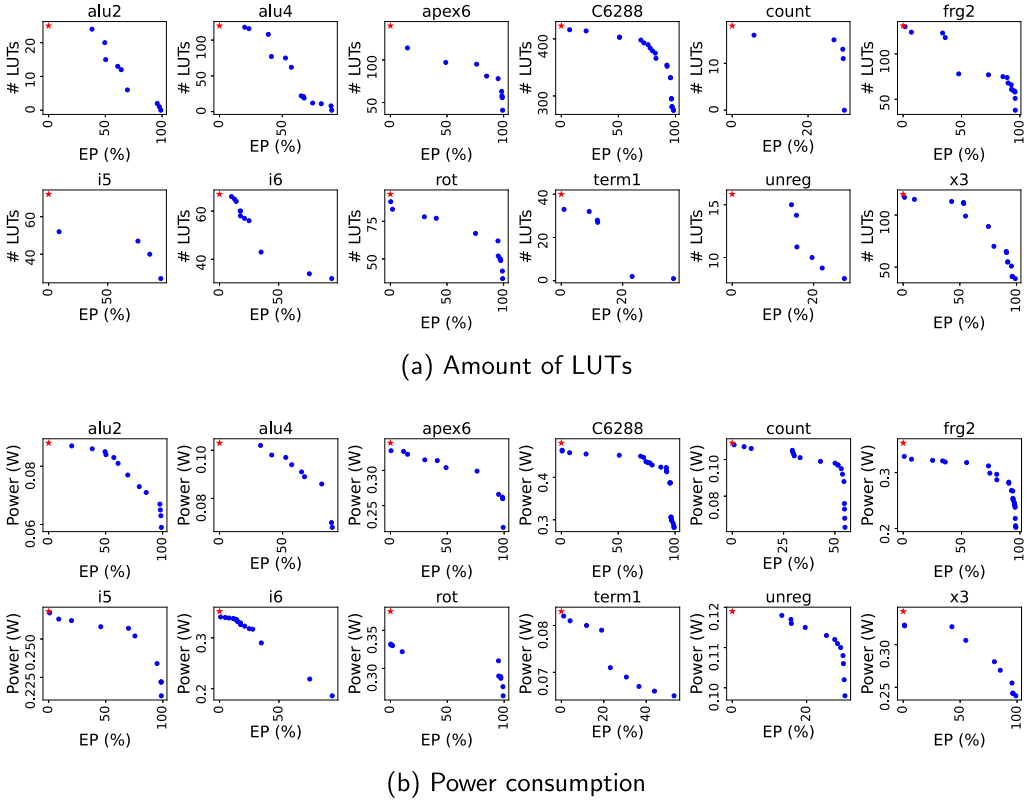


Fig. 4. FPGA resource requirements for generic-logic circuits from the LGSynth91 benchmark.

model-based estimation to drive the DSE. We estimate both the number of FPGA LUTs and latency from the AIG representation of circuits, since the correlation between hardware requirements and the number of nodes and depth of the AIGs has been empirically proven in [34].

Speaking of the AMOSA configuration, we address it on a per-circuit basis, resorting to advice from [7]. As a summary of a typical configuration, when generic-logic circuits from LGSynth91 are concerned, we let the cooling factor to vary in the [0.8, 0.9] range, the final temperature of the matter varying in the $[10^{-7}, 1]$ range, the hard and soft archive-size limit vary in the [50, 15] and [100, 300] ranges, respectively, the initial hill-climbing and annealing iterations in the [100, 350] and [250, 750] ranges, respectively. Having said that, depending on the circuit and the chosen configuration for the AMOSA heuristic, experiments involving such circuits take from a few minutes to several hours to complete.

For what pertains to arithmetic circuits, we observed that, as far as adders are concerned, the same configuration allows achieving good trade-offs between diversity of final non-dominated solutions and computational time. Such a configuration consists of a cooling factor varying in the [0.8, 0.95] range, the final temperature of the matter varying in the $[10^{-3}, 10^{-1}]$ range, hard and soft archive-size limits set to 100 and 250 elements, respectively, while the initial hill-climbing and annealing iterations are set to 200 and 350 respectively. On the other hand, when multipliers are concerned, the increased complexity of the problem – in terms of number of decision variables – requires more effort to be spent during DSE. We came up with the following configuration: the cooling factor is set to 0.95, the final temperature of the matter set to 10^{-7} degrees, the hard and archive-size limits set to 150 and 300 elements, respectively, initial hill-climbing and annealing iterations set to 550 and 750, respectively. As it is easy to foresee, the

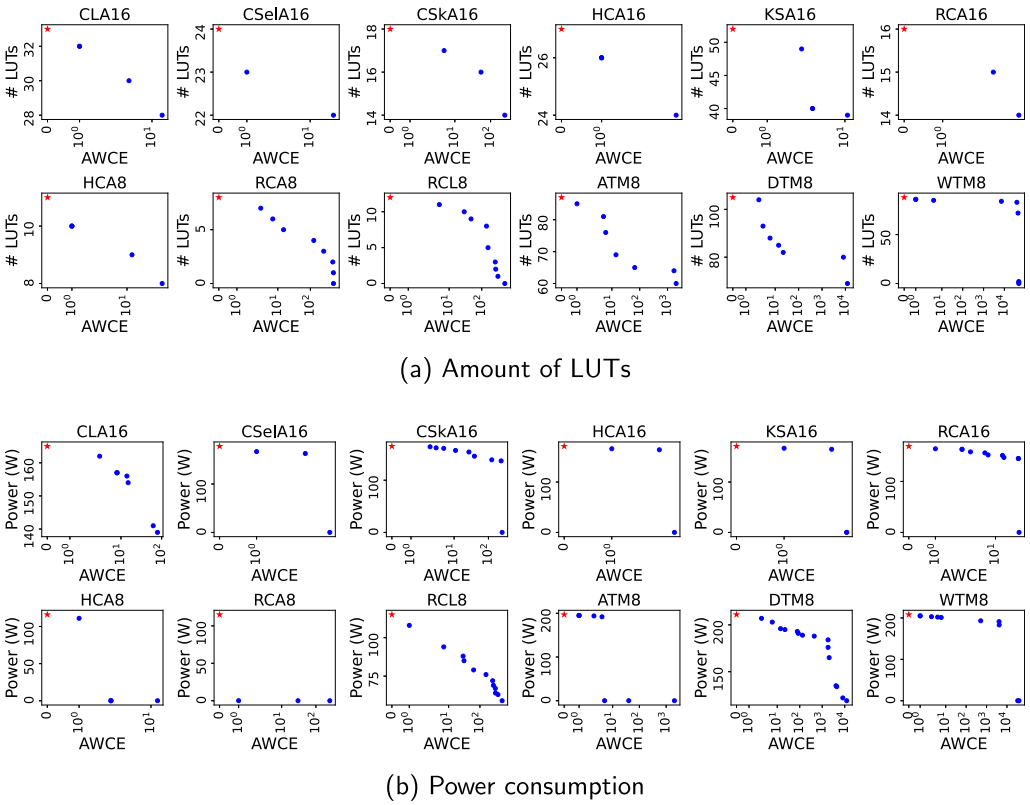


Fig. 5. FPGA resource requirements for arithmetic circuits. Kindly note that the x-axis is in semilogarithmic scale.

Table 1
Computational-time for circuits belonging to the LGSynth91 benchmark.

Circuit	alu2	alu4	apex6	C6288	count	frg2	i5	i6	rot	term1	unreg	x3
Comp.Time	16 m	7 m	6 m	9 m	4 m	23 m	13 s	11s	8 m	52 m	2 m	7 m

Table 2
Computational-time for circuits arithmetic circuits.

Circuit	CLA16	CSelA16	CSkA16	HCA16	KSA16	RCA16	HCA8	RCA8	RCL8	ATM8	DTM8	WTM8	sine
Comp.Time	≈20 h	≈20 h	≈26 h	≈25 h	≈20 h	≈21 h	≈13 h	≈12 h	≈13 h	≈66 h	≈60 h	≈62 h	≈210 h

more effort is spent during DSE, the more it lasts; consequently, while experimenting on adders typically requires a few hours, experiments involving multipliers may require a few days to complete.

Results, as well as computational time, are discussed in the following.

4.2. Experimental results

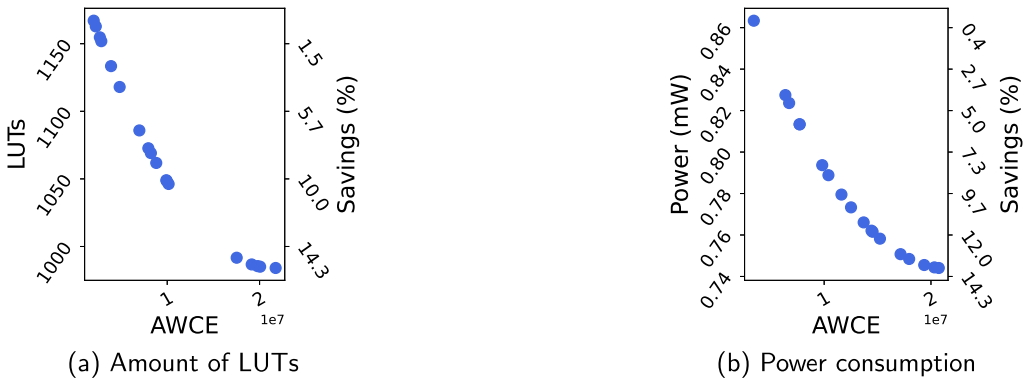
Tables 1 and 2 report the computational time that experiments tool to complete, respectively, for generic logic and arithmetic circuits.

At the end of the DSE, we targeted the Xilinx xc7a35ticsg324-1L Artix-7 FPGA to measure actual hardware requirements of resulting circuits. Synthesis was performed by Xilinx Vivado 2021.01 with default settings and disabling DSPs.

Fig. 4(a), Fig. 4(b) plot the number of required FPGA LUTs and power consumption against EP for circuits from the LGSynth91 benchmark. Figs. 5(a) and 5(b), instead, plots resource requirements, in terms of the number of required FPGA LUTs and power consumption against AWCE for various 8-bits arithmetic circuits, including array-tree multiplier (ATM), Dadda-tree multiplier (DTM), Wallace-tree multiplier (WTM), carry-skip adder (CSkA), ripple-carry adder (RCA) Han-Carlson adder (HCA) and carry-lookahead adder (CLA), which have been generated while resorting to the Arithmetic Circuit Generator for Hardware Accelerators (ArithsGen) [46]. Last, Figs. 6(a) and 6(b) plots resource requirements for the circuits from the EPFL Combinational Benchmark Suite [49].

Note that the x-axis for Fig. 5(a), Fig. 5(b), Fig. 6(a) and Fig. 6(b) is in semilogarithmic scale. The red star denotes the exact circuit, while the blue dots • denote approximate configurations resulting from DSE. There may be solutions that have the same error and gate-count (they are in a non-dominance relationship with each other) coming from the DSE that, when synthesized to FPGA still have the same error, of course, but they require different amount of LUTs to be implemented, or they consume different power. Hence, they are no more in non-dominance relationship; therefore, the dominated ones are actually discarded and not plotted. Hence, any point in Fig. 4(a) exhibiting the same error as in Fig. 4(b) may not refer to the same approximate configuration. This, of course, also applies to Fig. 5(a) and Fig. 5(b).

As for the area overhead, it is clear that approximate circuits require less FPGA LUTs on the error increasing since, as we select approximate solutions minimizing AIG node-count. In the same way, power consumption takes advantage of the approach as it decreases on the introduced error. But, as one can notice, Fig. 4(b) reports more non-dominated solutions than Fig. 4(a) as well as Fig. 5(b) and Fig. 5(a). This implies that given two different approximate configurations exhibiting the same error and requirements in terms of FPGA LUTs, one



(a) Amount of LUTs (b) Power consumption

Fig. 6. FPGA resource requirements for the *sin* circuit, from the EPFL Combinational Benchmark Suite [49]. Kindly note that the plots is in semilogarithmic scale.

of these has lower power consumption. This phenomenon can certainly also happen with ASIC technology, Anyway, at a first glance, it appears to be occurring very frequently, contrary to the experimental result reported in [7] based onto ASIC implementations.

This experimental observation led us to investigate how two different approximate implementations could exhibit two power consumption values against the same amount of FPGA LUTs, i.e., how approximate variants of a given circuit exhibit different values of dynamic power consumption despite requiring the same amount of LUTs. We provide an in-depth analysis of FPGA dynamic power consumption based on LUTs model in Section 5.

4.3. Achieved savings while targeting different technologies

As discussed before, almost all the AxC approaches operate at the transistor or gate-netlist level, and allow for large silicon area and power savings while targeting ASIC. Nonetheless, due to the architectural differences between target technologies, savings achieved are far modest while targeting FPGA [5,6,30]. Though, since the previous Section empirically prove the AxC technique from [7] can be effectively adopted for both FPGA-based systems, besides ASIC ones, it is interesting to compare results the technique provides, varying the target implementation, in terms of silicon and power savings. Figs. 7 and 8 report such a comparison for generic logic circuits and arithmetic ones. The blue bullets • denotes savings induced in ASIC implementations, while the orange triangles ▲ denote savings for FPGA technology. Except in sporadic cases, the results shown in the figures mentioned above confirm what is known in the literature: applying the same approximation technique does not produce the same gain when considering different implementations. However, the gap is not much, especially for arithmetic circuits. And for some circuits the gain on FPGA is greater, opening the way for future investigations about which error metrics or properties a circuit must possess in order for it to be approximated with the same profit on both ASIC and FPGA.

5. The power dissipation model of LUTs

In this section, we aim to analyze the correlation existing between AIG node count and power consumption of approximate circuits. To this aim, two different strategies are possible. On one hand, we could rely on simulation-based tools that considering an FPGA configuration (namely bitstream), a specific FPGA device and a user-define workload (namely, a test-bench) can estimate power consumption by stimulating the input signals and executing a low-level device simulation. On the other hand, as we will discuss later, we could resort to a LUT model, that is generic and independent of a specific FPGA fabric, from which we could estimate the switching activity and get the power consumption avoiding costly simulation. Furthermore, instead of providing to each circuit a proper workload, we can consider each time a worst-case scenario.

5.1. Switching activity estimation

The power consumption for FPGA includes two terms, i.e., the static and the dynamic power. The former is the power from transistor leakage on all connected voltage rails and the circuits required for the FPGA to operate normally. It is highly operating-temperature dependent, yet a function of the manufacturing process and supply voltage, but it is not correlated to the design configured onto the device fabric. Conversely, the dynamic power is the power of the user design, due to the input data pattern and the design internal activity. This power is instantaneous and varies at each clock cycle, it depends on voltage levels, logic, and routing resources used (including I/O terminations, clock managers, and other circuits that need power when used).

Anyway, it can be roughly estimated as $P = V_{dd}^2 \cdot f_{clk} \cdot \sum_{n_i} C_i \cdot E_{sw}(n_i)$, where C_i is the capacitance of the node n_i, V_{dd} is the supply voltage, f_{clk} is the clock frequency, and $E_{sw}(n_i)$ is the average number of output transitions per time-unit $T = \frac{1}{f_{clf}}$ at node n_i, i.e., the SwA.

As it is easy to foresee, reducing the resource overhead required by a given circuit – i.e., the number of LUTs a circuit requires, as provided by our approach – implies a reduction in the dynamic power consumption. But experimental results, shown in the previous section, evidences that there is a further contribution to power savings, that clearly comes from SwA of involved LUTs reduction, being operating conditions – i.e., V_{dd}, f_{clk}, – out of the scope of our approach and left unchanged w.r.t. original circuit specification.

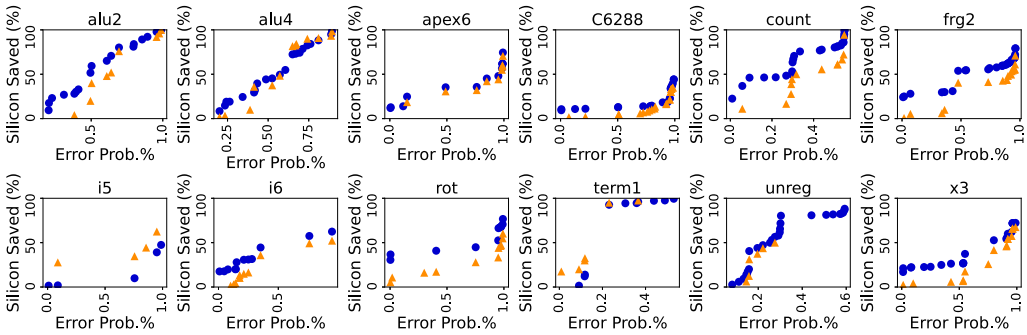
Estimating SwA, i.e., $E_{sw}(n_i)$, requires determining $p_0(n_i)$ and $p_1(n_i)$, i.e., the probability of the output signal at each node n_i being either logic-0 or logic-1, respectively. Under the zero-delay model, the $E_{sw}(n_i)$ can be estimated using Eq. (15) from [51]. Please, note that (15) also supposes that the n_i node can assume only the two mentioned complementary logic values; hence, $p_0(n_i) = 1 - p_1(n_i)$.

$$E_{sw}(n_i) = \frac{p_0(n_i) \cdot p_1(n_i)}{p_0(n_i) + p_1(n_i)} = p_0(n_i) - p_0(n_i)^2 \tag{15}$$

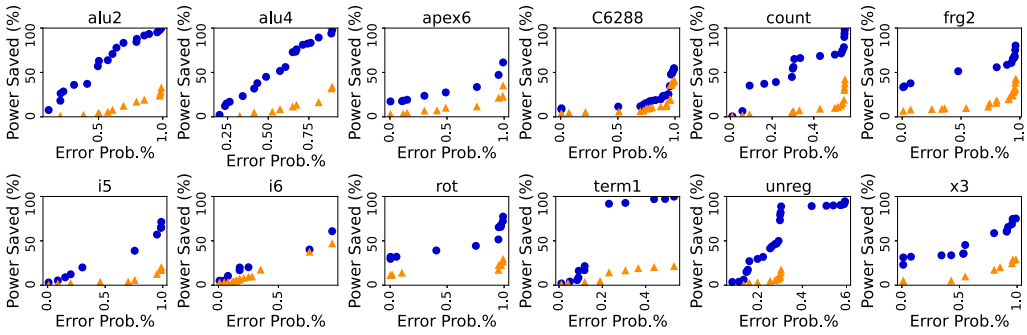
In the case of FPGA, determining $p_0(n_i)$ requires characterizing the activity of inputs and output signals, resorting, for instance, to a model such as the one from [8].

Essentially, a K-LUT takes K input signals $S = \{s_0, \ldots, s_{K-1}\}$ that allow selecting, as output, one between 2^K configuration bits from $X = \{x_0, \ldots, x_{2^K-1}\}$, as defined by the technology mapping $T_m(S, X)$.

Although the actual implementation may be quite different, the model from [8] states K-LUTs can be modeled as fully balanced binary trees of height $K-1$, 2^K-1 internal nodes corresponding to 2-to −1 multiplexers, and 2^K leaves representing configuration bits, as depicted in Fig. 9. For an accurate estimation of the E_{sw}, all the configuration bits, not just the one being selected as output, have to be considered, since even those that do not actually reach it partially propagate towards the output anyway contribute to the SwA, affecting the frequencies with which each configuration bit passes through multiplexers. Thus, the model computes the actual frequency with which the configuration bits

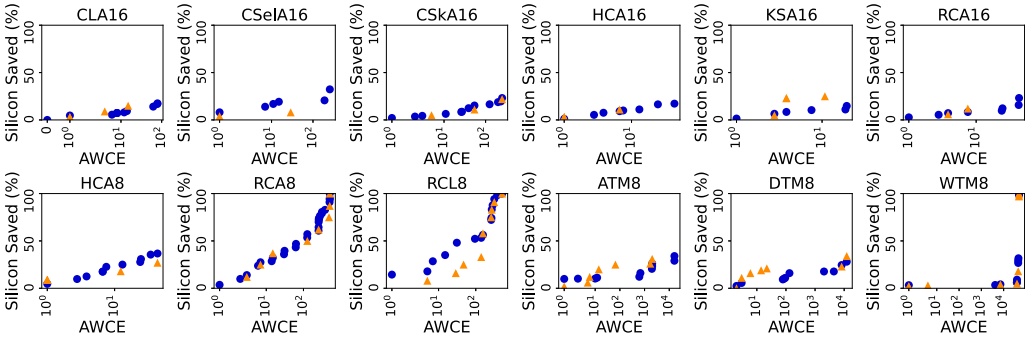


(a) Amount of saved silicon

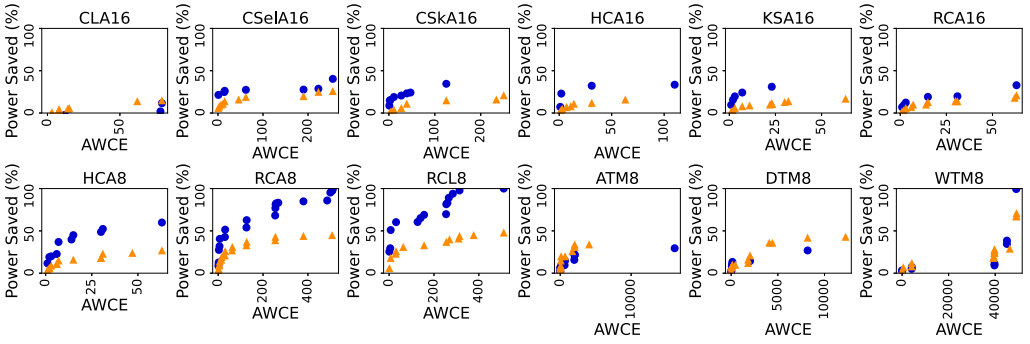


(b) Amount of saved power

Fig. 7. Comparison of savings provided with generic circuits while targeting either the ASIC or the FPGA technology. The blue bullets • denotes savings induced in ASIC implementations, while the orange triangles ▲ denote savings for FPGA technology.



(a) Amount of saved silicon



(b) Amount of saved power

Fig. 8. Comparison of savings provided with arithmetic circuits while targeting either the ASIC or the FPGA technology. The blue bullets • denotes savings induced in ASIC implementations, while the orange triangles ▲ denote savings for FPGA technology.

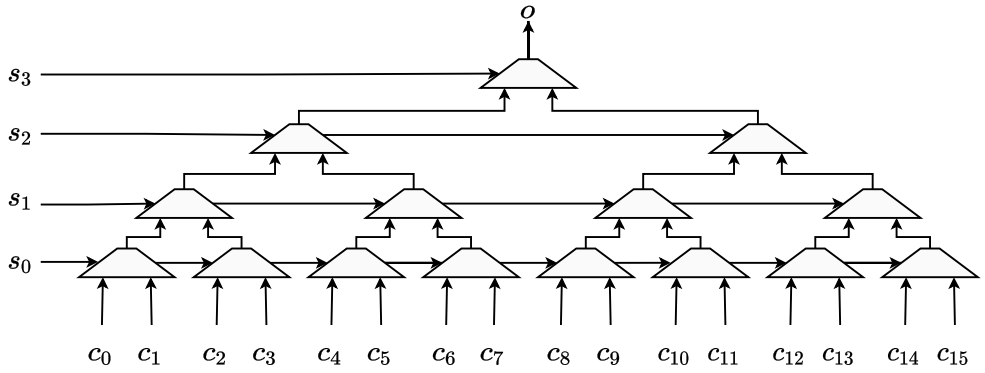


Fig. 9. Model of a 4-LUT as a fully balanced binary-tree of multiplexers.

traverse each node of the tree based on frequencies f_i with which the ith configuration bit is selected as output, as in (16).

$$F = \{f_i \in \mathbb{R} : f_i \in [0,1] \wedge \sum_{i=0}^{2^K-1} f_i = 1\} \tag{16}$$

The frequency whereby the configuration bits traverse each node of the tree can be computed using Eq. (17). The $f_{\ell,L}$ is the sum of the frequencies of the leaves on the left-hand side of the 2^ℓ trees rooted at level ℓ, while $f_{\ell,R}$ is the sum of the frequencies of the leaves on the right-hand side of the 2^ℓ trees rooted at level ℓ. Clearly, $f_{\ell,L} + f_{\ell,R} = 1$.

$$f_{\ell,L} = \sum_{j=0}^{2^\ell-1} \sum_{i=0}^{2^{K-\ell-1}-1} f_{j\cdot 2^{K-\ell}+i} \tag{17}$$

To compute $p_0(n_i)$, we resort to Eq. (18) from [8], that allows recursively computing $p_0(n_i)$ for each internal node of the tree based on the same but computed at child nodes, plus $f_{\ell,L}$ and $f_{\ell,R}$. In such an Equation, $L(n_i)$ and $R(n_i)$ respectively denote the left-hand-side and right-hand-side child of n_i, while $f_{\ell,L}$ and $f_{\ell,R}$ denote the left-hand side and right-hand side frequencies of the tree at level ℓ, as given by (17). Note the root-node is at level 0.

$$p_0(n_i) = \begin{cases} p_0\left(L(n_i)\right) f_{\ell_i,L} + p_0\left(R(n_i)\right) f_{\ell_i,R}, \\ \qquad\qquad i \in \left[0, 2^K - 2\right] \\ \left(x_j = 1 \rightarrow 0\right) \wedge \left(x_j = 0 \rightarrow 1\right), \\ \quad i \in \left[2^K - 1, 2^{K+1} - 1\right], j \in [0, 2^K - 1] \end{cases} \tag{18}$$

Please note that the first (top) case applies to all nodes which index ranges in $\left[0, 2^K - 2\right]$, i.e., all nodes of the tree but leaves. For such nodes, $p_0(n_i)$ – i.e., the probability of the output signal at node n_i being logic-0 – is recursively computed as the sum of the probability of the output at left and right children nodes being logic-0 – viz. $p_0(L(n_i))$ and $p_0(R(n_i))$, respectively – each multiplied to the frequency whereby the configuration bits traverse each node of the tree until the considered child of n_i – i.e., $f_{\ell,L}$ and $f_{\ell,R}$, respectively, as computed by Eq. (17). Conversely, the second (bottom) case applies to nodes with index in $\left[2^K - 1, 2^{K+1} - 1\right]$, that are leaves of the tree, that matches configuration bits of the LUT. In this case, the probability of the output signal at node n_i simply matches configuration bits; hence, it coherently collapses to $p_0(n_i) = 0$ if the corresponding configuration bit is 1, else it equals to 1. Hence, $p_0(n_i)$ directly depends on the configuration bits and left-hand side and right-hand side frequencies values as computed by Eq. (17). The SwA for a whole K-LUT, given its configuration bits, F, can be computed using (19), that only contemplates internal nodes of the LUT.

$$E_{sw} = \sum_{i=0}^{2^K-2} p_0(n_i) - p_0(n_i)^2 \tag{19}$$

It is worth noticing that, there are more than one $T_m(S, X)$ that satisfies the proper mapping for a given boolean function and, depending on which one is selected during the synthesis, it exhibits different SwA. The selection of mapping solution with less SwA is due to the synthesis tool, namely Xilinx Vivado in our case, involving a signal reordering of the K-LUT. It goes that SwA optimization through signal reordering is out of our scope and not considered by the proposed approach.

5.2. Result discussing and final remarks

Despite expectations arising from experimental result, no direct proportionality relationship between the number of nodes and SwA is evident from the model discussed before. Furthermore, counterintuitive examples can be found.

Consider, for instance, the LUT configuration $X = \{0,0,0,0,1,1,0,0,1,0,1,0,1,1,1,1\}$, that requires 4 AIG nodes and, according to Eqs. (15) – (19), has 1.625 SwA: its approximation at Hamming distance 1 resulting from ES, i.e., $X = \{1,0,0,0,1,1,0,0,1,0,1,0,1,1,1,1\}$, exhibits 1.98 SwA despite requiring 3 AIG nodes. Besides, to the best of our knowledge, no correlation between the AIG node-count and SwA for LUTs has ever been identified in the scientific literature either.

Thus, to shed light on the source of the observed phenomenon, we exploited Eqs. (17)–(19) to characterize every catalog entry – i.e., every LUT specification – we collected during our experimental campaign both in terms of AIG node-count and SwA. To this end, we assume inputs are equally probable – i.e, $f_i = \frac{1}{2^K}$, $i = 1 \cdots 2^K - 1$ in (16) – and we take the signal reordering of inputs into account, by considering, for each of the catalog entries, the actual reordering that minimizes the SwA. The box-and-whisker plot in Fig. 10 reports this preliminary characterization. At first glance, a general decreasing trend in terms of SwA, as far as the AIG node-count decreases, can be observed. Consequently, during DSE, while the AMOSA is attempting to minimize the AIG node-count, it is very likely that it (accidentally) pursues also SwA minimization; hence, resulting approximate circuits exhibit a lower power dissipation.

Anyway, the SwA variation ranges overlap for adjacent AIG node values, and, furthermore, we can also observe that, although values of the SwA tend to be gathered around the median value, there are several outliers, even for those LUTs requiring a moderate number of AIG nodes, e.g., three or four nodes. Therefore, even though the SwA of a LUT is close to the median value, during the catalog-generation procedure it may produce configurations exhibiting higher SwA, despite requiring less nodes. This gives reason to the existence of counterintuitive examples, although they do not have significant impact on the overall approximate circuit.

Indeed, still concerning counterintuitive examples, the LUTs we collected during the experimental campaign underwent a further characterization, to highlight their common features, if any. Such a characterization drew attention to the AIG topology and truth-density – i.e., is the number of truth assignments over the input set. In particular, approximate LUTs that constitute counterintuitive examples exhibit

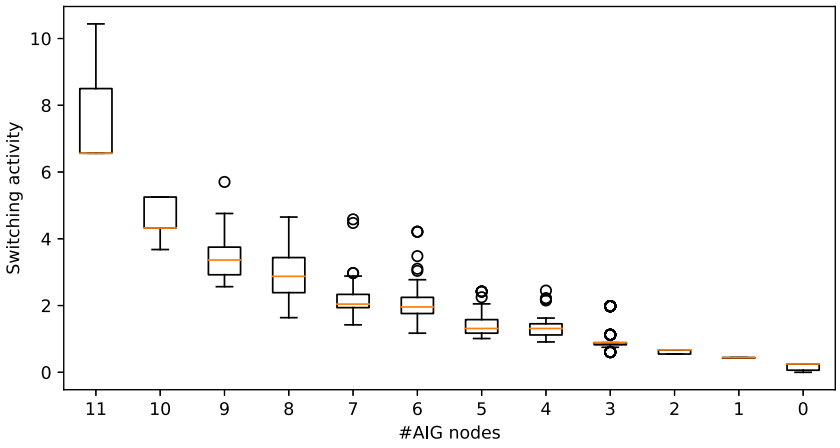


Fig. 10. SwA of catalog entries, varying the AIG node-count.

the following common features: (a) they exhibit complete functional dependency from all inputs; (b) they do not implement elementary Boolean functions, such as logic-AND or logic OR; (c) the exact specification from which they originate has truth-density is almost 50%; (d) their truth-density is almost 50%; (e) the number of nodes of the corresponding AIG is very close to the theoretical minimum required by (a).

Additionally, we observed that in case (c) is true, to reach the termination criterion, the catalog-generation procedure is very likely to reorganize the truth table until resulting in propagating an input signal (meant that the approximate specification has 50% of truth density). Hence, during intermediate stages of the mentioned procedure, the truth-density fluctuates around to 50%, without ever being equal to it. Under these circumstances, the signal reordering of inputs is ineffective in reducing the SwA, since either the truth assignment is insusceptible to reordering or any reordering equals the original specification.

6. Comparison with previous works

In Section 2 we discussed several contributions from the scientific literature pertaining to the design of FPGA-based approximate computing systems, emphasizing the most recent contributions. In the following, we compare our approach with the state of the art, both qualitatively and quantitatively.

One of the most interesting approach is that from [5], that, roughly, sifts through a library of approximate components (specifically, the EvoapproxLib [52], that has been designed while targeting ASIC) looking for those circuits providing optimal error/resources trade-offs for FPGA. Machine-learning models are exploited to predict FPGA requirements, avoiding FPGA synthesis. From a methodological perspective, when compared to the mentioned approach, our methodology allows designing FPGA-based approximate components from scratch. This means it does not require a library of ASIC-based approximate components, from which those providing optimal trade-offs between error and FPGA resource requirements have to be sifted. Furthermore, our method does not require high-fidelity predictors, since it exploits the node-count and depth of AIGs to estimate resource requirements during the DSE. In order to get a fair comparison of circuits resulting from our method against those from [5], we considered the exact (non-approximate) multiplier and adder from the ApproxFPGAs library, which are freely available at https://github.com/ehw-fit/approx-fpgas as starting point for our approach. In particular, we considered 8-bits and 12-bits operators, and once our workflow was completed, we synthesized resulting circuits while targeting a Xilinx xc7a35ticsg324-1L Artix-7 FPGA. For an unbiased comparison, we also resynthesized the circuits from the ApproxFPGAs library on the same device. Figs. 11 and 12 report results both in terms of FPGA LUT and power consumption, respectively. Red crosses × denote results from [5], while blue dots • denote results from our method. As the reader can observe, results from our method are competitive with the state-of-the-art, since the Pareto-fronts in Figs. 11 and 12 are barely distinguishable.

Table 3
Computational-time required to design 8-bits unsigned-integer arithmetic circuits.

Circuit	mul8u	add8u	mul12u	add12u
Our method	≈67 h	≈13 h	≈72 h	≈15 h
ApproxFPGAs [5]	≈192 h	≈24 h	≈34 h	≈22 h

It is interesting to note, as far as power is concerned, that the approximate circuits resulting from our approach are equally good compared with those obtained from other state-of-the-art methods even though they may be more onerous in terms of LUTs, as is evident in Figs. 11(c) and 11(d), which report LUTs for 8-bits adder and multipliers, respectively, and Figs. 12(c) and 12(d), which, instead, report power consumption.

Pertaining to computational time, Table 3 compares our methodologies against [5]. The first row of that Table reports the computational time needed to perform our workflow as a whole – including the ES, the DSE and the final rewriting and hardware synthesis – while resorting to the algorithm configuration discussed in Section 4.1, performed on a 16-cores/32-threads AMD Ryzen 9550. The second row of the table reports the computational time required by the ApproxFPGAs as reported in [5], while running on a 16-cores/32-threads Intel Xeon E5 CPU. These times include the time required for synthesizing the dataset, training and evaluating the models, and re-synthesizing the Pareto optimal circuits. Please note that, for the method in [5], the larger the circuit, the smaller the search space, due to the reduced number of candidate circuits in the library; hence, circuit size and computational time do not correlate. Since we do not need any model to be trained to estimate hardware requirements of candidate approximate circuits, and since we can perform faithful estimations from AIG nodes, when compared with [5], our method requires significantly less computational time to provide approximate circuits.

The approach from [6] is one of the latest contributions to the scientific literature. Basically, it encodes circuits as binary strings. Given a circuit requiring N LUTs for partial product generation, the method from [6] encodes each of the variant through a binary string of length N, and allows generating 2^N different approximate variants. A "0" at any location in the N-bits string disables corresponding LUT, which means that LUT will not contribute to producing the intermediate results. Concerning fitness functions driving the DSE, the authors of [6] resort to the product between the power-delay-product and the number of LUTs as a measure for hardware resource requirements, while the

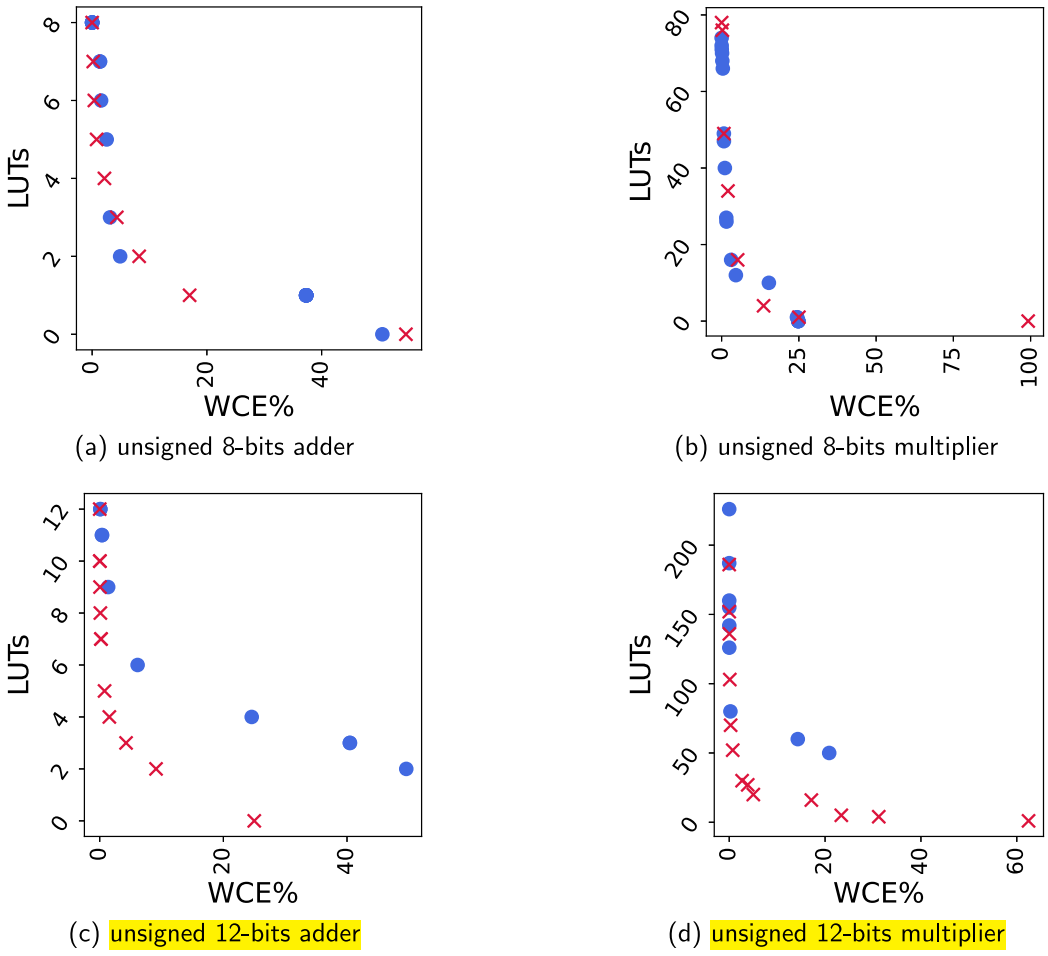


Fig. 11. Comparison with [5] in terms of FPGA LUT. Red crosses × denote results from [5], while blue dots • denote results from our method.

AWCE (14) or the Mean Absolute Error (MAE) (20) have been adopted to measure the error.

$$e_{\text{mae}}(f, \hat{f}) = \frac{1}{2^n} \sum_{x \in \mathbb{B}^n} |f(x) - \hat{f}(x)| \tag{20}$$

From the qualitative perspective, the grain with which approximation can be introduced in circuits while exploiting our approach is finer when compared to that in [6]. In fact, as we observed in the previous section, results indicates that some replacements produce a reduction in power consumption, even though they did not result in any decrease in the number of LUTs. As a result, a more gradual and controlled degradation of the quality of results can be produced while lowering the hardware overhead. Indeed, the only case our approach leads to LUT suppression happens when the replacement being selected from the catalog either propagates one of the input signals, or even a constant signal. The only downside of having such a fine degree with which it is possible to act on the circuits is, as is easy to imagine, a generalized increase in the size of the search space.

In order to quantitatively compare the methods, we performed our method while performing the DSE to minimize the AIG node count as well as the error entailed by approximation. The latter is measured either as AWCE or MAE. For a fair comparison, we measured the actual FPGA requirements by synthesizing resulting circuits while targeting the xc7vx330t device of the Virtex-7 family, as done in [6].

In Figs. 13 and 14 we report a comparison of results from the mentioned methods in terms of FPGA resources (measured either as number of LUT or power consumption) and error (which is measured in terms of AWCE or MAE). In the mentioned figures, red crosses × denote results from [6], while blue dots • denote results from our method. As the reader can observe, such results confirm our previous statement: our method allows more gradual and controlled degradation of the quality of results while lowering the hardware overhead. Indeed, results from our method exhibit lower power consumption w.r.t. those from [6], albeit the latter produces circuits requiring less LUTs.

As for the computational time, once again we can safely claim our approach allows faster design, as it does not involve any model to be trained to estimate hardware requirements of candidate approximate circuits. Indeed, since we can perform faithful estimations from AIG nodes, when compared with [6], our method requires significantly less computational time to provide approximate circuits: while the accuracy and PPA estimation for a single approximate 8-bits multiplier consumes nearly 3.55 min of processing time with the method in [6], our method allows evaluating hundreds of variants in the same amount of time, since assessing the error on the whole 2^{16} possible inputs for an 8-bit multiplier only a few seconds, and the time required to count the number of AIG nodes is paltry.

7. Case studies

In this Section, we discuss several case-studies in which approximate circuits resulting from our method are exploited to reduce hardware requirements of larger and complex applications. In particular, we discuss the implementation of a Sobel edge detector in Section 7.1, a FIR filter in Section 7.2, and, finally, we discuss convolutional neural networks in Section 7.3.

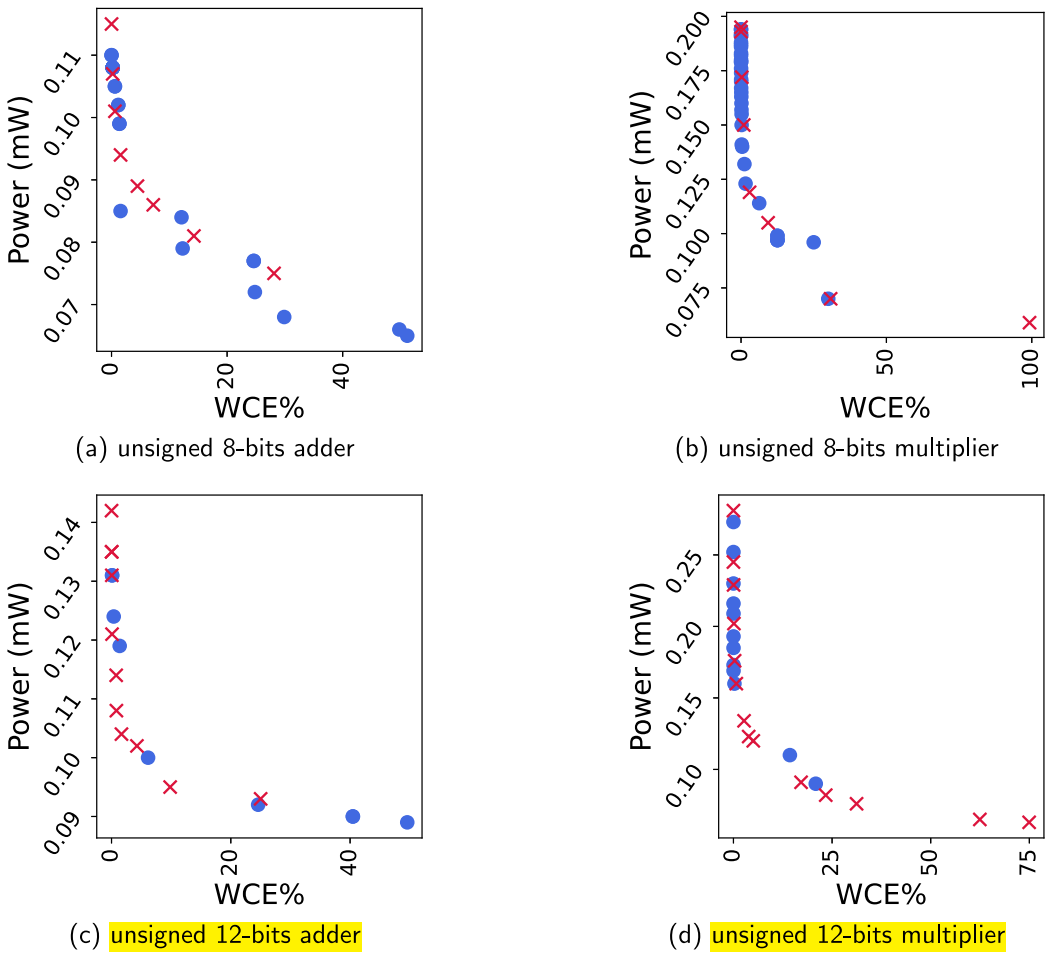


Fig. 12. Comparison with [5] in terms of power consumption. Red crosses × denote results from [5], while blue dots • denote results from our method.

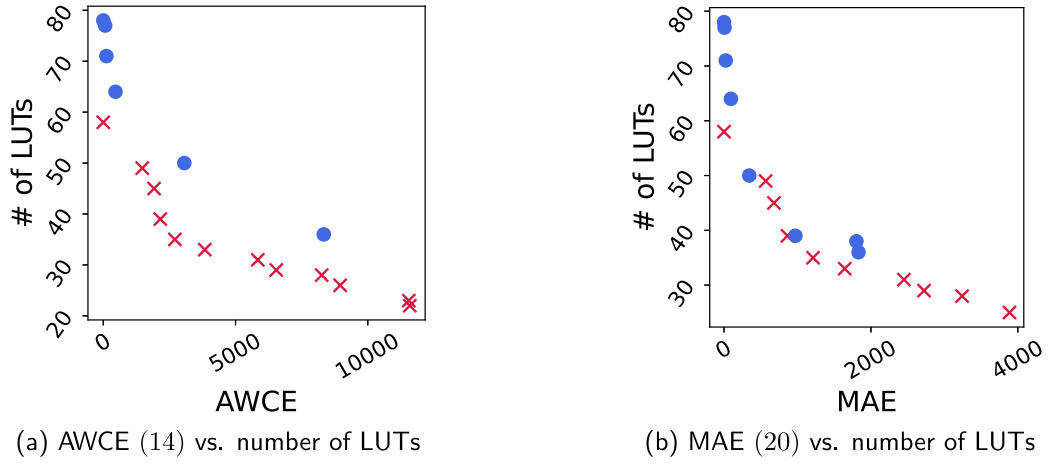
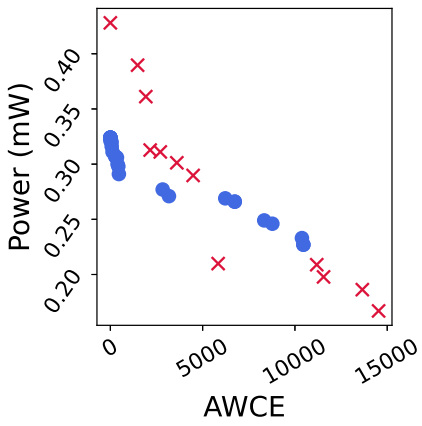


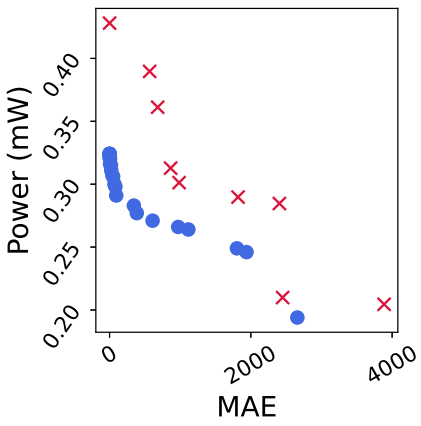
Fig. 13. Comparison with [6] in terms of LUTs. Red crosses × denote results from [6], while blue dots • denote results from our method.

7.1. The Sobel edge-detector

One of the major fields of application for AxC is image-processing, since, due to perceptual limitations of human eyes, imperceptible reduction of image quality can lead to important savings. One quite common image-processing application is the Sobel edge-detector; therefore, we provide a case-study concerning the design of a hardware accelerator for the aforementioned application, exploiting approximate components resulting from our approach. We only approximate adders, since, at the hardware level, multiplications by two can be implemented as a left shift, which is just wiring requiring no resources. Specifically, we adopted an approximate implementation of a 16-bits Kogge-Stone adder that exhibits AWCE of 32, costs 46 LUTs when synthesized on a Xilinx xc7a35ticsg324-1L Artix-7 FPGA (6 LUTs, or 11% less than its exact counterpart), and consumes 155 mW (26 mW, or 16% less than its exact counterpart) on the same device. We target the above-mentioned FPGA while performing synthesis of both the exact and the approximate implementations of the Sobel edge-detector, to measure their actual



(a) AWCE (14) vs. power consumption



(b) MAE (20) vs. power consumption

Fig. 14. Comparison with [6] in terms of power consumption. Red crosses × denote results from [6], while blue dots • denote results from our method.

(a) Visual test with Pepper

(b) Visual test with House

Fig. 15. Visual test. Original images are on the left; while the output of the exact implementation of the edge-detector is reported in the center. Images resulting from the approximate edge-detector are on the right. From top to bottom, the right-most images exhibit PSNR of 28.9 and 28.5.

hardware requirements. The exact implementation of the mentioned edge-detector requires 435 LUTs and consumes 743 mW, while its approximate implementation requires 349 LUTs and 620 mW, providing savings up to 19% and 16% for LUTs and power consumption, respectively.

Furthermore, we also report a comparison between images computed through the use of the exact and the approximate implementations in Fig. 15: images computed using the approximate implementation are reported on the right of Fig. 15, and exhibit a PSNR in the [28.5, 28.9] range w.r.t. reference images, i.e., those reported at the center of Fig. 15. Kindly note that the higher the PSNR the higher the quality of the images.

7.2. The FIR filter

In this case study, we target one of the most common signal processing applications, which is the FIR filter, whose general definition is reported in (21). There, x_j is the jth sample of the input signal, b_i is the ith coefficient of the signal, and y_n is the nth sample of the output signal. Here we will replace the multiplication in (21) using an approximate multiplier to save FPGA LUTs and power.

$$y_n = \sum_{i=0}^{N} b_i \cdot x_{n-i} \tag{21}$$

We consider two different FIR filters: they both have a 500 Hz cut-off frequency, 60 dB attenuation in the stop band, and use fixed-point arithmetic. The first is a low-pass filter that adopts the Q1.7 fixed-point arithmetic, while the second is a high-pass filter using the Q1.15 representation. Please note that fixed-point arithmetic can be performed using regular signed integer circuitry. In order to evaluate our approach, we assess the impact of approximation by measuring the PSNR between the output signals produced by the non-approximate and approximate FIRs.

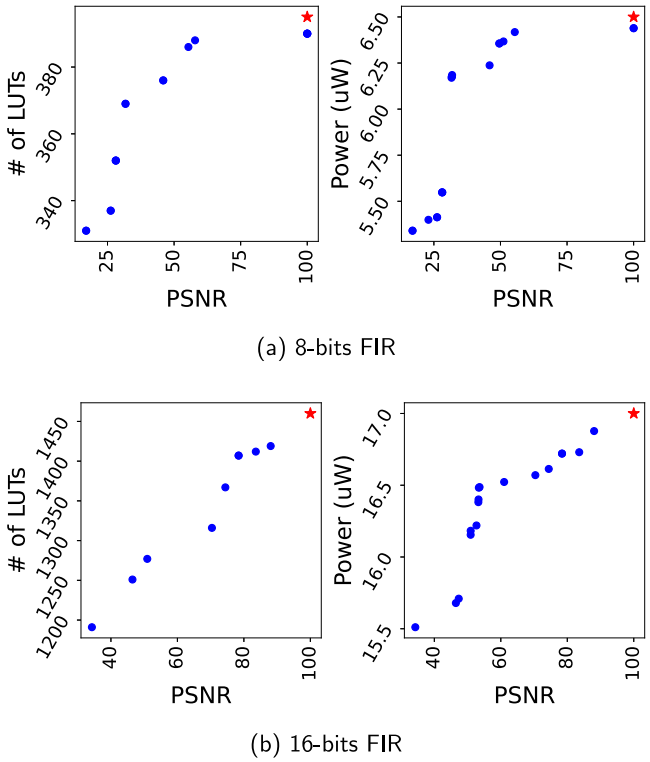


Fig. 16. PSNR and hardware resources for 8-bits and 16-bits FIRs while using approximate multipliers. The red star ⋆ denotes the reference (non-approximate) implementation, while the blue bullets • denotes filters being implemented using approximate multipliers.

Specifically, we adopted the 8-bits and 16-bits Wallace tree multiplier as starting point, and, after the DSE, we synthesized the design targeting a Xilinx xc7a35ticsg324-1L Artix-7 FPGA, collecting hardware requirements. We, then, performed simulation in order to compute the PSNR while processing a saw-tooth signal. Figs. 16(a) and 16(b) report results for the 8-bits and 16-bits FIRs, respectively. The red star ⋆ denotes the reference (non-approximate) implementation, while the blue bullets • denotes filters being implemented using approximate multipliers. As the reader can observe, approximate multipliers resulting from our method allow achieving significant savings while itroducing only a restrained amount of error in the final application.

7.3. Convolutional neural network

In this case study, we target CNNs, a class of ANNs most commonly applied to analyzing visual imagery [53,54]. The computational model of artificial neurons is inspired by its biological counterpart, but ANNs are organized in distinct *layers*, which define the network's depth, rather than being modeled as an amorphous blob of connected neurons. Neurons belonging to adjacent layers can be either fully or partially connected, while there is no connection between neurons within the same layer. Various types of layers with various topologies have been defined over the years. For instance, in Fully-Connected Layers (FCLs), neurons of the layer are connected to all the neurons of the preceding layer, while in Convolutional Layers (CLs), neurons in a layer are connected only to a small region of the previous layer, and they perform computations that are not just a function of the inputs. Indeed, as reported in Eq. (22), the output of each of the neurons is a non-linear function f of the weighted-sum involving learned weights w_i and inputs x_i, plus a bias b.

$$y = f\left(\sum_{i}^{n} w_i \cdot x_i + b\right) \tag{22}$$

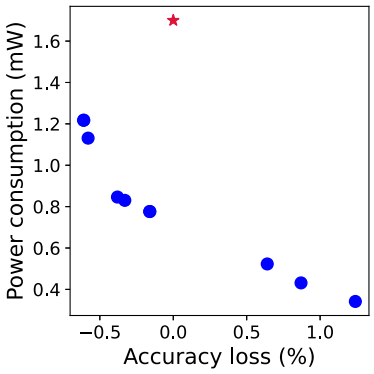


Fig. 17. Accuracy loss and hardware resources for LeNet-5 while using approximate multipliers. The red star ⋆ denotes the reference (non-approximate) implementation, while the blue bullets • denotes CNN being implemented using approximate multipliers.

Concerning approximation, it is typically introduced by replacing exact multiplications within neurons with approximate ones [43]. Hence, we aim at designing a multiplier to perform the $w_i \cdot x_i$ part of Eq. (22), targeting convolutional layers of the CNN with the goal of simultaneously reducing the hardware requirements and the impact of approximation on the classification accuracy loss during the inference phase. We resort to the approach from [42], which replaces accurate convolution layers within ANNs using approximate ones. Specifically, for each convolution layer, a suitable approximate multiplier is selected, while the overall classification error – i.e., accuracy loss – and energy consumption are simultaneously minimized through a MOP. During the DSE, we estimate the power-consumed by the approximate CNN as the weighted sum of power-consumption of each multiplier of the net, respectively, as done in [42].

The network architecture we consider in our case study is LeNet5 [55], trained to classify images from the Modified National Institute of Standards and Technology (MNIST) benchmark [56], which consists of 70000 28 × 28 grayscale images of handwritten digits, of which 60000 are for training purpose and 10000 testing. The network has been quantized using 8-bit signed integer arithmetic, and it exhibits 99.07% accuracy.

Results are reported in Fig. 17. Once again, we denote the reference (non-approximate) implementation of the CNN using the red star ⋆, while the blue bullets • denotes CNN being implemented using approximate multipliers. Please, note we only report the power consumption, since area requirements strictly depends on the architecture of the tensor processing unit. Power, instead, largely depends on the number of multiplications being performed. Since the negligible accuracy loss and the savings we can observe in Fig. 17, we can reiterate that our method is able to provide efficient implementations for approximate multipliers suitable for FPGA synthesis.

8. Conclusion

In this paper, we proposed the AIG rewriting technique [7] to provide approximate synthesis of logic circuit on FPGA technology, exploiting similarities between k-feasible cut enumeration and LUT-mapping for FPGA technology. We compute approximate LUTs while resorting to SMT-based ES for AIGs, and we go through multi-objective optimization to select LUTs-replacements leading to approximate configurations minimizing both error and circuit area. We underwent the methodology to a thorough experimental campaign involving both generic-logic circuits and arithmetic circuits, and experimental evidence proved our approach allows achieving significant savings both in terms of silicon area and power consumption. Furthermore, findings we discussed in Section 5 are worth being considered to further improve the effectiveness of the proposed approach when the FPGA is the

target technology. Last, we compared our approach with state-of-the-art proving solutions resulting from the former are competitive with those from the latter.

CRediT authorship contribution statement

Mario Barbareschi: Writing – review & editing, Writing – original draft, Methodology, Investigation. **Salvatore Barone:** Writing – review & editing, Writing – original draft, Software, Methodology. **Nicola Mazzocca:** Supervision. **Alberto Moriconi:** Writing – review & editing, Writing – original draft, Software, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Q. Xu, T. Mytkowicz, N.S. Kim, Approximate computing: A survey, IEEE Design Test 33 (1) (2016) 8–22, http://dx.doi.org/10.1109/MDAT.2015.2505723, Conference Name: IEEE Design Test.

[2] J. Echavarria, S. Wildermann, A. Becher, J. Teich, D. Ziener, FAU: Fast and error-optimized approximate adder units on LUT-based FPGAs, in: 2016 International Conference on Field-Programmable Technology (FPT), 2016, pp. 213–216, http://dx.doi.org/10.1109/FPT.2016.7929536.

[3] B.S. Prabakaran, S. Rehman, M.A. Hanif, S. Ullah, G. Mazaheri, A. Kumar, M. Shafique, DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems, in: 2018 Design, Automation Test in Europe Conference Exhibition (DATE), 2018, pp. 917–920, http://dx.doi.org/10.23919/DATE.2018.8342140, ISSN: 1558-1101.

[4] S. Ullah, S.S. Murthy, A. Kumar, *SMApproxlib*: library of FPGA-based approximate multipliers, in: Proceedings of the 55th Annual Design Automation Conference, ACM, San Francisco California, 2018, pp. 1–6, http://dx.doi.org/10.1145/3195970.3196115, URL https://dl.acm.org/doi/10.1145/3195970.3196115.

[5] B.S. Prabakaran, V. Mrazek, Z. Vasicek, L. Sekanina, M. Shafique, ApproxFPGAs: Embracing ASIC-based approximate arithmetic components for FPGA-based systems, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1–6, http://dx.doi.org/10.1109/DAC18072.2020.9218533, ISSN: 0738-100X.

[6] S. Ullah, S.S. Sahoo, N. Ahmed, D. Chaudhury, A. Kumar, AppAxO: Designing application-specific approximate operators for FPGA-based embedded systems, ACM Trans. Embed. Comput. Syst. (2022) 3513262, http://dx.doi.org/10.1145/3513262, URL https://dl.acm.org/doi/10.1145/3513262.

[7] M. Barbareschi, S. Barone, N. Mazzocca, A. Moriconi, A catalog-based AIG-rewriting approach to the design of approximate components, IEEE Trans. Emerg. Top. Comput. (2022) http://dx.doi.org/10.1109/TETC.2022.3170502.

[8] M.J. Alexander, Power optimization for FPGA look-up tables, in: Proceedings of the 1997 International Symposium on Physical Design - ISPD '97, ACM Press, Napa Valley, California, United States, 1997, pp. 156–162, http://dx.doi.org/10.1145/267665.267707, URL http://portal.acm.org/citation.cfm?doid=267665.267707.

[9] H.A. Almurib, T.N. Kumar, F. Lombardi, Approximate DCT image compression using inexact computing, IEEE Trans. Comput. 67 (2) (2018) 149–159, http://dx.doi.org/10.1109/TC.2017.2731770.

[10] A. Mercat, J. Bonnot, M. Pelcat, K. Desnos, W. Hamidouche, D. Menard, Smart search space reduction for approximate computing: A low energy HEVC encoder case study, J. Syst. Archit. 80 (2017) 56–67, http://dx.doi.org/10.1016/j.sysarc.2017.09.003, URL https://www.sciencedirect.com/science/article/pii/S1383762117300371.

[11] V.K. Chippa, S.T. Chakradhar, K. Roy, A. Raghunathan, Analysis and characterization of inherent application resilience for approximate computing, in: Proceedings of the 50th Annual Design Automation Conference on - DAC '13, ACM Press, Austin, Texas, 2013, p. 1, http://dx.doi.org/10.1145/2463209.2488873, URL http://dl.acm.org/citation.cfm?doid=2463209.2488873.

[12] V.K. Chippa, D. Mohapatra, K. Roy, S.T. Chakradhar, A. Raghunathan, Scalable effort hardware design, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 22 (9) (2014) 2004–2016, http://dx.doi.org/10.1109/TVLSI.2013.2276759.

[13] S. Venkataramani, S.T. Chakradhar, K. Roy, A. Raghunathan, Approximate computing and the quest for computing efficiency, in: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015, pp. 1–6, http://dx.doi.org/10.1145/2744769.2744904, ISSN: 0738-100X.

[14] G. Li, X. Ma, Q. Yu, L. Liu, H. Liu, X. Wang, CoAxNN: Optimizing on-device deep learning with conditional approximate neural networks, J. Syst. Archit. 143 (2023) 102978, http://dx.doi.org/10.1016/j.sysarc.2023.102978, URL https://www.sciencedirect.com/science/article/pii/S1383762123001571.

[15] B. Deveautour, M. Traiola, A. Virazel, P. Girard, QAMR: an approximation-based fully reliable TMR alternative for area overhead reduction, in: 2020 IEEE European Test Symposium (ETS), 2020, pp. 1–6, http://dx.doi.org/10.1109/ETS48528.2020.9131574, ISSN: 1558-1780.

[16] M. Traiola, J. Echavarria, A. Bosio, J. Teich, I. O'Connor, Design space exploration of approximation-based quadruple modular redundancy circuits, in: 2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD), 2021, pp. 1–9, http://dx.doi.org/10.1109/ICCAD51958.2021.9643561, ISSN: 1558-2434.

[17] W. Liu, Q. Liao, F. Qiao, W. Xia, C. Wang, F. Lombardi, Approximate designs for fast Fourier transform (FFT) with application to speech recognition, IEEE Trans. Circuits Syst. I. Regul. Pap. 66 (12) (2019) 4727–4739, http://dx.doi.org/10.1109/TCSI.2019.2933321.

[18] Z.-G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, J. Henkel, Weight-oriented approximation for energy-efficient neural network inference accelerators, IEEE Trans. Circuits Syst. I. Regul. Pap. 67 (12) (2020) 4670–4683, http://dx.doi.org/10.1109/TCSI.2020.3019460.

[19] A. Ranjan, S. Venkataramani, S. Jain, Y. Kim, S.G. Ramasubramanian, A. Raha, K. Roy, A. Raghunathan, Automatic synthesis techniques for approximate circuits, in: S. Reda, M. Shafique (Eds.), Approximate Circuits: Methodologies and CAD, Springer International Publishing, Cham, 2019, pp. 123–140, http://dx.doi.org/10.1007/978-3-319-99322-5_6.

[20] T. Yeh, P. Faloutsos, M. Ercegovac, S. Patel, G. Reinman, The art of deception: Adaptive precision reduction for area efficient physics acceleration, in: 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), 2007, pp. 394–406, http://dx.doi.org/10.1109/MICRO.2007.9, ISSN: 2379-3155.

[21] M. Traiola, A. Savino, S. Di Carlo, Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications, Microelectron. Reliabil. 102 (2019) 113309, http://dx.doi.org/10.1016/j.microrel.2019.06.002, URL https://linkinghub.elsevier.com/retrieve/pii/S0026271418309442.

[22] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, A. Raghunathan, SALSA: Systematic logic synthesis of approximate circuits, in: DAC Design Automation Conference 2012, 2012, pp. 796–801, http://dx.doi.org/10.1145/2228360.2228504, ISSN: 0738-100X.

[23] S. Venkataramani, K. Roy, A. Raghunathan, Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits, in: 2013 Design, Automation Test in Europe Conference Exhibition (DATE), 2013, pp. 1367–1372, http://dx.doi.org/10.7873/DATE.2013.280, ISSN: 1530-1591.

[24] J. Castro-Godínez, H. Barrantes-García, M. Shafique, J. Henkel, AxLS: A framework for approximate logic synthesis based on netlist transformations, IEEE Trans. Circuits Syst. II 68 (8) (2021) 2845–2849, http://dx.doi.org/10.1109/TCSII.2021.3068757.

[25] E. Zacharelos, I. Nunziata, G. Saggese, A.G. Strollo, E. Napoli, Approximate recursive multipliers using low power building blocks, IEEE Trans. Emerg. Top. Comput. 10 (3) (2022) 1315–1330, http://dx.doi.org/10.1109/TETC.2022.3186240.

[26] H. Waris, C. Wang, W. Liu, J. Han, F. Lombardi, Hybrid partial product-based high-performance approximate recursive multipliers, IEEE Trans. Emerg. Top. Comput. 10 (1) (2022) 507–513, http://dx.doi.org/10.1109/TETC.2020.3013977.

[27] K. Nepal, Y. Li, R.I. Bahar, S. Reda, ABACUS: A technique for automated behavioral synthesis of approximate computing circuits, in: 2014 Design, Automation Test in Europe Conference Exhibition (DATE), 2014, pp. 1–6, http://dx.doi.org/10.7873/DATE.2014.374, ISSN: 1558-1101.

[28] K. Nepal, S. Hashemi, H. Tann, R.I. Bahar, S. Reda, Automated high-level generation of low-power approximate computing circuits, IEEE Trans. Emerg. Top. Comput. 7 (1) (2019) 18–30, http://dx.doi.org/10.1109/TETC.2016.2598283.

[29] L. Sekanina, Z. Vasicek, Approximate circuit design by means of evolvable hardware, in: 2013 IEEE International Conference on Evolvable Systems (ICES), 2013, pp. 21–28, http://dx.doi.org/10.1109/ICES.2013.6613278.

[30] S. Ullah, S. Rehman, M. Shafique, A. Kumar, High-performance accurate and approximate multipliers for FPGA-based hardware accelerators, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 41 (2) (2022) 211–224, http://dx.doi.org/10.1109/TCAD.2021.3056337.

[31] W. Ahmad, B. Ayrancioglu, I. Hamzaoglu, Low error efficient approximate adders for FPGAs, IEEE Access 9 (2021) 117232–117243, http://dx.doi.org/10.1109/ACCESS.2021.3107370, Conference Name: IEEE Access.

[32] V. Mrazek, R. Hrbacek, Z. Vasicek, L. Sekanina, EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, 2017, pp. 258–261, http://dx.doi.org/10.23919/DATE.2017.7926993, ISSN: 1558-1101.

[33] A. Mishchenko, S. Chatterjee, R. Brayton, DAG-aware AIG rewriting: a fresh look at combinational logic synthesis, in: 2006 43rd ACM/IEEE Design Automation Conference, 2006, pp. 532–535, http://dx.doi.org/10.1145/1146909.1147048, ISSN: 0738-100X.

[34] A. Mishchenko, S. Cho, S. Chatterjee, R. Brayton, Combinational and sequential mapping with priority cuts, in: 2007 IEEE/ACM International Conference on Computer-Aided Design, 2007, pp. 354–361, http://dx.doi.org/10.1109/ICCAD.2007.4397290, ISSN: 1558-2434.

[35] C.D. Murray, R.R. Williams, On the (Non) NP-hardness of computing circuit complexity, Theory Comput. 13 (1) (2017) 1–22, http://dx.doi.org/10.4086/toc.2017.v013a004, URL http://www.theoryofcomputing.org/articles/v013a004.

[36] L. Sekanina, Z. Vasicek, V. Mrazek, Automated search-based functional approximation for digital circuits, in: S. Reda, M. Shafique (Eds.), Approximate Circuits, Springer International Publishing, Cham, 2019, pp. 175–203, http://dx.doi.org/10.1007/978-3-319-99322-5_9, URL http://link.springer.com/10.1007/978-3-319-99322-5_9.

[37] S. Barone, M. Traiola, M. Barbareschi, A. Bosio, Multi-objective application-driven approximate design method, IEEE Access 9 (2021) 86975–86993, http://dx.doi.org/10.1109/ACCESS.2021.3087858.

[38] M. Barbareschi, S. Barone, N. Mazzocca, Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study, Knowl. Inf. Syst. (2021) 1–20, http://dx.doi.org/10.1007/s10115-021-01565-5, URL https://link.springer.com/article/10.1007/s10115-021-01565-5.

[39] M. Barbareschi, S. Barone, A. Bosio, J. Han, M. Traiola, A genetic-algorithm-based approach to the design of DCT hardware accelerators, ACM J. Emerg. Technol. Comput. Syst. 18 (3) (2022) 1–25, http://dx.doi.org/10.1145/3501772, URL https://dl.acm.org/doi/10.1145/3501772.

[40] S. Bandyopadhyay, S. Saha, U. Maulik, K. Deb, A simulated annealing-based multiobjective optimization algorithm: AMOSA, IEEE Trans. Evol. Comput. 12 (3) (2008) 269–283, http://dx.doi.org/10.1109/TEVC.2007.900837, Conference Name: IEEE Transactions on Evolutionary Computation.

[41] S. Yang, Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0, Citeseer, 1991.

[42] V. Mrazek, Z. Vasicek, L. Sekanina, M.A. Hanif, M. Shafique, ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining, in: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019, pp. 1–8, http://dx.doi.org/10.1109/ICCAD45719.2019.8942068, URL http://arxiv.org/abs/1907.07229, arXiv:1907.07229.

[43] M.S. Ansari, V. Mrazek, B.F. Cockburn, L. Sekanina, Z. Vasicek, J. Han, Improving the accuracy and hardware efficiency of neural networks using approximate multipliers, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 28 (2) (2020) 317–328, http://dx.doi.org/10.1109/TVLSI.2019.2940943.

[44] M.H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, Special session: Approximation and fault resiliency of DNN accelerators, in: 2023 IEEE 41st VLSI Test Symposium (VTS), IEEE, 2023, pp. 1–10.

[45] M.S. Kim, A.A. Del Barrio, H. Kim, N. Bagherzadeh, The effects of approximate multiplication on convolutional neural networks, IEEE Trans. Emerg. Top. Comput. 10 (2) (2022) 904–916, http://dx.doi.org/10.1109/TETC.2021.3050989.

[46] J. Klhufek, V. Mrazek, ArithsGen: Arithmetic circuit generator for hardware accelerators, in: 2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2022, pp. 44–47, http://dx.doi.org/10.1109/DDECS54261.2022.9770152, ISSN: 2473-2117.

[47] N. Homma, T. Aoki, Arithmetic module generator, 2022, URL https://www.ecsis.riec.tohoku.ac.jp/topics/amg/.

[48] R. Ueno, N. Homma, T. Aoki, Automatic generation system for multiple-valued galois-field parallel multipliers, IEICE Trans. Inf. Syst. E100.D (8) (2017) 1603–1610, http://dx.doi.org/10.1587/transinf.2016LOP0010, URL https://www.jstage.jst.go.jp/article/transinf/E100.D/8/E100.D_2016LOP0010/_article.

[49] L. Amaru, P.-E. Gaillardon, G.D. Micheli, The EPFL combinational benchmark suite, in: Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS), Mountain View, California, USA, 2015, URL http://infoscience.epfl.ch/record/207551.

[50] M. Soeken, L.G. Amarù, P.-E. Gaillardon, G. De Micheli, Exact synthesis of majority-inverter graphs and its applications, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (11) (2017) 1842–1855, http://dx.doi.org/10.1109/TCAD.2017.2664059.

[51] M. Pedram, Power minimization in IC design Principles and applications, ACM Trans. Des. Autom. Electron. Syst. 1 (1) (1996) 54.

[52] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, J. Han, Scalable construction of approximate multipliers with formally guaranteed worst case error, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 26 (11) (2018) 2572–2576, http://dx.doi.org/10.1109/TVLSI.2018.2856362.

[53] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127, http://dx.doi.org/10.1561/2200000006.

[54] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117, http://dx.doi.org/10.1016/j.neunet.2014.09.003, https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135.

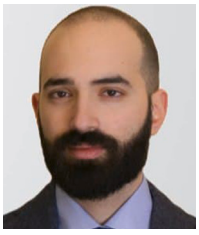
[55] Backpropagation applied to handwritten zip code recognition, Neural Comput. 1 (4) (1989) 541–551, http://dx.doi.org/10.1162/neco.1989.1.4.541.

[56] Y. LeCun, C. Cortes, C. Burges, MNIST Handwritten digit database, 1998, URL http://yann.lecun.com/exdb/mnist/.

Mario Barbareschi is a Tenured Assistant Professor of Computer Systems at the Department of Electrical Engineering and Information Technologies of the University of Naples Federico II. He received the Ph.D. in Computer and Automation Engineering in 2015 from the University of Naples Federico II. His research interests include Hardware Security and Trust, Approximate Computing, emerging technologies, and embedded systems. He has authored more than 70 peer-reviewed papers published in leading journals and international conferences.

Salvatore Barone received the Ph.D. in Information Technologies and Electrical Engineering in 2022, and the Master Degree in Computer Engineering cum laude in 2018, both from the University of Naples Federico II, Italy, where he is an Assistant Professor. His research interests include Safety Critical Systems, Railway Systems, Approximate Computing and Embedded Systems based on the FPGA technology.

Nicola Mazzocca is full professor of Computer Systems at the Department of Electrical Engineering and Information Technologies of the University of Naples Federico II. Since 1994, he has held numerous university courses and in professional training activities on different topics, including, high-performance systems, distributed systems, embedded systems, security, and reliability. His research activities concern: computer architecture, distributed systems, high-performance systems, and safety-critical applications. He is author of more than 250 papers on international journals, books, and conference proceedings of congresses.



Alberto Moriconi received the Master Degree in Computer Engineering cum laude in 2019, from the University of Naples Federico II, Italy, where he is currently a Ph.D. student. His research interests include Approximate Computing, Safety Critical Systems, Railway Systems, and Embedded Systems based on the FPGA technology.