Design Space Exploration of Approximation-Based Quadruple Modular Redundancy Circuits

Marcello Traiola, Jorge Echavarria, Alberto Bosio, Jurgen Teich, Ian O'Connor

Design Space Exploration of Approximation-Based Quadruple Modular Redundancy Circuits

Marcello Traiola[1], Jorge Echavarria[2], Alberto Bosio[1], Jürgen Teich[2] and Ian O'Connor[1]
[1]*University of Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270*, Ecully, France
[2]*Department of Computer Science, Friedrich-Alexander-Universität (FAU)*, Erlangen-Nürnberg, Germany
[1]{marcello.traiola, alberto.bosio, ian.oconnor}@ec-lyon.fr – [2]{jorge.echavarria, juergen.teich}@fau.de

***Abstract*—In the last decade, Approximate Computing (AxC) has been studied as a possible alternative computing paradigm. It has been used to reduce the overhead cost of conventional fault tolerant schemes, such as the Triple Modular Redundancy (TMR). One of the most recent propositions is the concept of Quadruple Approximate Modular Redundancy (QAMR). QAMR reduces the overhead cost w.r.t. conventional TMR structures, while guaranteeing the same fault-tolerance capability. In this paper, we propose a new approximation technique to realize the QAMR and we perform a Design Space Exploration (DSE) to find QAMR Pareto-optimal implementations. Moreover, we provide the design of a new majority voter for the proposed architecture. Experimental results show that it is possible to find QAMR variants achieving area and/or delay gains compared to the TMR counterpart, for 85.4% and 97% of the examined circuits for FPGA and ASIC technologies respectively.**

***Index Terms*—Fault tolerance; error correction; triple modular redundancy; TMR; approximate computing; quadruple approximate modular redundancy; QAMR; digital circuits; approximate computing**

I. Introduction

Electronic systems operating in harsh environments (e.g. radiative) face multiple physical phenomena which might lead to degraded performance or to errors [1]. The main effects in advanced nanometer electronics leading to permanent faults – hence to repeating failures – are aging and wear-out. Conversely, transient faults (soft errors) may be caused by energetic charged particles. Their propagation through the logic may ultimately cause a system malfunction. To mitigate the effects of these events on electronic circuits, several fault-tolerance approaches have been proposed. One of the most established fault-tolerant architectures is the Triple Modular Redundancy (TMR) [2]. Triplicating the circuit and performing a majority vote of the outputs of the three replicas ensures soft and hard error tolerance; this comes at the cost of 200% overhead and of extra area and delay of the majority voter.

Approximate Computing (AxC) is an increasingly established alternative computation paradigm that exploits the intrinsic resilience of some applications to achieve gains in terms of resources [3]. Indeed, for some applications, relaxing non-critical specifications and obtaining an inaccurate result does not necessarily impact the final outcome catastrophically. At the same time, this may provide disproportionate savings in terms of resources [4], [5]. AxC has been applied at different layers of computing systems, from hardware to software [4]. In this work, we focus on *Approximate Integrated Circuits (AxICs)*, stemming from AxC application to Integrated Circuits (ICs). In particular, we focus on *functional hardware approximation* that selectively changes the circuit functionality to reduce the area and/or delay. Functional hardware approximation approaches proposed in the literature can be grouped into three main categories [4]. 1) Ad-hoc approximation, necessary to alter very specific functionalities of the original circuit [6], [7]. Although efficient, ad-hoc approximation usually entails high design efforts. 2) Automatic synthesis methodologies for approximate circuits, which help to reduce the circuit cost while trying to minimize accuracy reduction [8], [9]. Such synthesis methodologies can help to manage the approximation of large circuits. 3) Hardware accelerated *Neural Networks (NNs)*, which are well known to accurately mimic logic functions while consuming often less than conventional circuits [10]. NNs can be efficiently accelerated by dedicated hardware.

AxC has also been applied to TMR to reduce the overhead stemming from the circuit triplication [11]–[15]. Rather than three precise replicas, three different AxICs are used to implement the Approximate Triple Modular Redundancy (ATMR). Unfortunately, while AxICs lead to lower TMR overhead, their error-masking capability is reduced. This makes the ATMR not a viable option for safety-critical scenarios. To overcome the above issue, the study in [16] proposed the concept of Quadruple Approximate Modular Redundancy (QAMR) to ensure the same fault tolerance properties as the TMR while still benefiting from approximation advantages. QAMR is not based on using three but rather four AxICs. Preliminary results reported in [16] proved the feasibility and the interest in the QAMR approach. However, so far, the proposed QAMR-oriented approximation method is based on realizing the AxICs by selectively removing outputs and the related logic cones driving them [16]. Although this approach allows using a conventional three-bit majority voter as in the TMR, the lack of an efficient four-bit voter prevents the use of other approximation approaches.

In this paper, we propose a novel QAMR approach. The main contributions of this work are:

1) a different approximation approach, based on *logic falsification* [17], to realize the four QAMR replicas;
2) the design of a new efficient four-bit majority voter enabling the new QAMR concept;
3) a Design Space Exploration (DSE) approach to find Pareto-

DOI: 10.1109/ICCAD51958.2021.9643561

optimal implementations w.r.t. area and delay, along with the analysis of achievable trade-offs in comparison to classic TMR and to a simple heuristic approach to determine optimized QAMR implementations, as proposed in [16].

The remainder of the paper is organized as follows. Section II reviews previous studies on AxC-based fault tolerance. Section III illustrates the proposed approach. Section IV details the DSE flow and shows the experimental results. Finally, Section V draws some conclusions.

II. Approximation-based Fault Tolerance

As already mentioned, several AxC-based proposals exist in the literature to reduce the overhead of TMR. This research area is generally known as Approximate TMR (ATMR) [11]. The ATMR approach employs three AxICs instead of three fully-precise replicas. For a given input, only one AxIC can give an incorrect answer. However, ATMR suffers from severe reliability limitations, unacceptable in safety-critical scenarios. Let us resort to an example to illustrate this issue. Let X be an input vector for the ATMR replicas. Let one of the three AxICs produce a wrong response – due to the approximation – while the other two produce a correct response. Let us imagine that a soft error occurs in one of the AxICs, thus modifying its output. If it occurs in the AxIC providing the approximate output, then the voter will still be able to produce the correct response, thanks to the two remaining AxICs. Conversely, if an AxIC providing the correct output to X experiences the error, there will be two incorrect responses, i.e., the approximate one and the faulty one. Thus, the voter may likely produce a wrong response. In summary, input vectors for which only two out of three AxICs compute correctly are not protected against faults. A possible alternative to mitigate the problem is to adopt a modified checker that computes the average of the results, as proposed in [18] for a software ATMR implementation. However, in this case the outputs are approximate (i.e., not precise). Finally, since designing fault tolerance architectures for safety-critical applications is a crucial task, realizing it by using AxC-based schemes entails some important challenges. For instance, to use the ATMR solution in safety-critical scenarios, unprotected input vectors must not be critical for the application. Unfortunately, such requirement may be impossible to satisfy, even for resilient applications.

Recently, a novel approach referred to as QAMR was presented [16]. The QAMR is the first approximation-based fault-tolerant architecture *not sacrificing fault-masking capabilities*. Let us resort to Figure 1 to explain the QAMR approach. Let f be a Boolean function, whose input domain D can be split into four subsets $D1, D2, D3, D4$, as shown in Figure 1a. The conventional TMR approach, sketched in Figure 1b, uses three identical copies of the circuit implementing f and a voting scheme. This ensures fault tolerance when one of the three circuit replicas incurs some defective conditions. In such a case, the defective copy will produce incorrect outputs, for some inputs. Thanks to the other two correct copies, the majority voter can still provide the correct output. As sketched in Figure 1c, the QAMR employs four circuits suitably approximated to ensure the desired fault tolerance but achieve efficiency gains. Specifically, the four AxICs

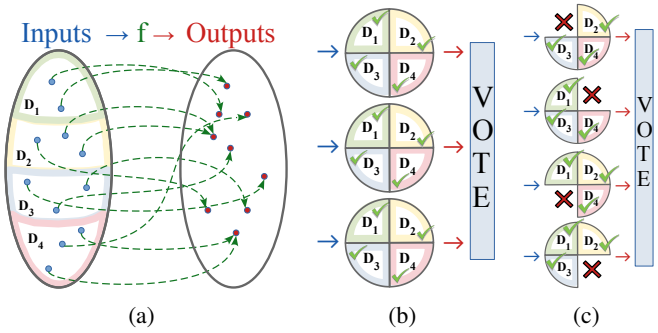

Fig. 1: (a) Boolean function's input domain split into four subsets (D1-D4); (b) the TMR uses three identical circuit copies: all the circuits produce accurate results for all the subsets (D1-D4), when no errors occur; (c) QAMR [16] uses four approximate circuits. As it can be seen, each input subset is covered three times, as in TMR.

should be approximated so that all the function domain subsets are covered three times – as in the TMR scheme. At the same time, using AxICs enables the opportunity to achieve reductions in cost [16]. The underlying insight is that a good AxC technique achieves higher reductions in circuit cost than it reduces the asserted fault tolerance properties. The approximation technique presented in [16] is based on a selective *removal of outputs*: a specific output is removed only from one replica and kept in the others. In this way, only three replicas are obtained for each output signal. Thus, a conventional majority voter can be used. Although innovative, the study in [16] provides only results obtained based on a random search-based exploration of the QAMR design space. Moreover, using four AxICs calls for concepts for four-bit majority voting. This hinders the utilization of different approximation approaches, which do not remove the circuit outputs. Finally, no automatic DSE strategies have been proposed for determining Pareto-optimal QAMR implementations. In this paper, we address these issues to discover the potential of QAMR to full extent.

III. Proposed QAMR Approach

In the following, we present a new QAMR-oriented approximation methodology, a new voting strategy, and we perform a thorough exploration of the QAMR design space. The approximations are applied to each replica individually to reduce its cost while *keeping all output signals*. This calls for a new *four-bit voting* approach. The straightforward method would be to put a selector circuitry between the four AxICs and the majority voter to prevent the wrong (approximate) response from propagating to the voter. However, there are two major drawbacks of this method: (i) the selector needs information on which replica delivers the wrong (approximate) response, at any time; (ii) the selector entails a big overhead that may undermine the approximation gains; we measured its overhead for ASIC technology (*FreePDK45 45nm* library [19]): +132% in area and +65% in delay, w.r.t. the conventional 3-bit majority voter. In the following, a new efficient voting strategy is proposed, based on the approximation choices made at design time. In the next subsection, we introduce the proposed approximation approach and, in Subsection III-B, we present the new voting strategy. Finally, in III-C, we illustrate the DSE and

present achieved trade-off solutions in terms of circuit cost and delay.

A. Approximation approach

To perform approximations in each of the AxICs, we resort to an approximate multi-level logic minimization methodology, namely *logic falsification* [17]. It refers to the process of modifying the on-set or alternatively the off-set of a given Boolean function, thereby introducing errors in its definition on purpose. The proposed QAMR approach produces approximated circuits satisfying the following conditions:

(i) for each given input subset (approximation subset), only one AxIC is allowed to produce an incorrect (i.e., approximate) response, whereas the other three AxICs must produce non-approximate responses;

(ii) for each given output, the four AxICs must provide the same fixed-by-design value, for their respective approximation subset. This condition enables the efficient design of the novel four-bit voting solution – as introduced in Subsection III-B.

To clarify, in Table I we report a simple example of approximation by logic falsification satisfying the above conditions. We firstly

TABLE I: Example of a 3-input and 2-output Boolean function with approximation applied by logic falsification (affected output values shown in red circles)

	Inputs			Original		$AxIC_0$		$AxIC_1$		$AxIC_2$		$AxIC_3$	
	A	B	C	out_0	out_1	out_0	out_1	out_0	out_1	out_0	out_1	out_0	out_1
$Subset_0$	0	0	0	1	0	(0)	(1)	1	0	1	0	1	0
	0	0	1	0	0	(0)	(1)	0	0	0	0	0	0
$Subset_1$	0	1	0	0	1	0	1	(0)	(1)	0	1	0	1
	0	1	1	1	1	1	1	(0)	(1)	1	1	1	1
$Subset_2$	1	0	0	0	0	0	0	0	0	(0)	(1)	0	0
	1	0	1	0	1	0	1	0	1	(0)	(1)	0	1
$Subset_3$	1	1	0	1	0	1	0	1	0	1	0	(0)	(1)
	1	1	1	1	0	1	0	1	0	1	0	(0)	(1)

Approximate output values chosen: $out_0 = 0$, $out_1 = 1$

partition the set of input-literal assignments into four subsets. For each subset, we apply logic falsification (i.e., we modify the output response) of only one of the replicas (as shown in red in Table I). The only constraint we put on the falsification of output values is that we require the different replicas to use the same falsification values for their input subset. Let us refer to the example circuit specified in Table I. Here, $AxIC_0$ provides out_0 = '0' and out_1 = '1' as response to all the inputs in $subset_0$ and a correct response to inputs in other subsets; the same goes for $AxIC_1$ which provides out_0 = '0' and out_1 = '1' as response to the inputs in $subset_1$ and a correct response to inputs in other subsets, and so on. In this way, we obtain four AxIC replicas with a corresponding QAMR structure providing the same robustness as the conventional (non-approximate) TMR while also enabling, thanks to the approximation, possible reductions in terms of resource costs, as it will be shown in Section IV.

Note that, depending on how we choose a) the input subsets and b) the values for the approximate outputs, we obtain different outcomes in terms of final area and delay after circuit synthesis. To explore this vast search space systematically, a DSE (discussed in Subsection III-C) is applied to find optimal circuits. In particular, we are interested in finding solutions close to Pareto-optimality in terms of circuit cost and delay and compare these trade-offs with that of the classical TMR version. First, we introduce the new voting strategy needed.

B. A sorting-network-based voting approach

As described in Subsection III-A, we generate four AxICs, such that any given one may provide approximate responses only when the other three provide non-approximate responses. To be able to correctly and efficiently perform a majority vote, we propose a voting approach based on the well-known binary Sorting Networks (SortNets) [20]. SortNets are made of comparators and wires. A wire carries a binary value through the network. A comparator connects two wires and sorts the two values carried by the wires. Figure 2a shows the symbolic representation of a comparator;

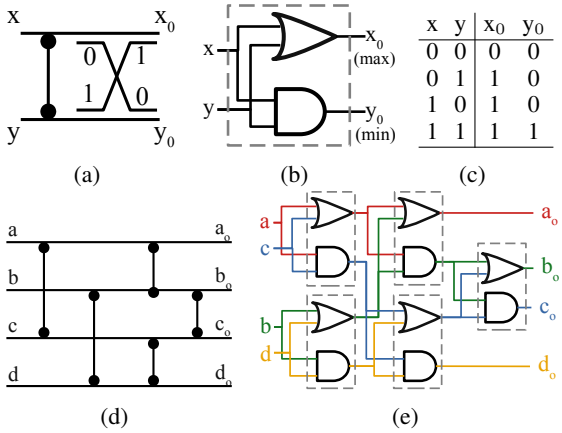

Fig. 2: (a) comparator symbol: two signals are sorted when connected; (b) logic-gate implementation of the comparator and (c) corresponding truth table. (d) Four-wire sorting network representation and (e) corresponding gate-level implementation.

Figure 2b represents the corresponding logic-gate implementation, and Figure 2c the related truth table. As shown in the figure, the comparator employs a *logic OR* and a *logic AND* to perform the binary *max* and *min* functions, respectively. The function of the network is to produce a vector of sorted signals, by arranging the comparators accordingly. In our context, the above network is applied to sort the four outputs produced by the four AxICs. In Figure 2d, a corresponding four-wire sorting network representation is shown and in Figure 2e the corresponding logic-gate implementation. This leads to a clear separation of the *zero* ('0') values from the *one* ('1') values, at the bottom and at the top of the SortNet output, respectively. Let us resort to an example to show the merit of the idea. Figures 3a and 3b depict two simple examples where one AxIC produces a wrong value, due to the approximation. Let the correct response of an output signal be '0' but let one of the replicas produce a '1' due to the approximation (Figure 3a), the SortNet arranges the values so that the wrong response ends up at the top. Conversely, in case the correct response for an output signal is '1' (Figure 3b) but one of the replicas produces a '0' due to the approximation, the SortNet arranges the values so that the wrong response ends up at the bottom. In this way, the three

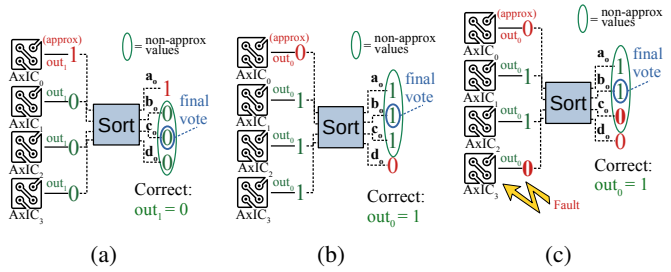

Fig. 3: (a) and (b) four-wire SortNet working principle (see Table I, input '000'). (c) Example showing the correctness of the SortNet-based voting strategy under occurrence of a fault.

non-approximate values wrapped in the green ovals in Figure 3 can be voted. As final vote signal we use the middle value in the ovals, i.e., c_o in Figure 3a and b_o in Figure 3b. Indeed, we resort to the mid-value select approach [21] stating that "sorting three signals – one incorrect and two correct – leads a correct one to lay between the other two". This is shown in the example in Figure 3c: even when a fault strikes an AxIC producing a non-approximate response and turns it into a wrong one, the SortNet-based voter is still able to deliver the correct response, i.e., $b_o = 1$ in the example. To suitably embed the voting structures depicted in Figure 3 into the QAMR architecture, we choose – *at design time* and for each output – the suitable variant of the SortNet among those in Figures 3a and 3b. Indeed, thanks to the *second approximation condition* (Sec. III-A), we guarantee – by design – that the AxIC replicas produce the same approximate response for each given output (see Table I).

Finally, since we do not need all the four output values of the SortNet, we can optimize the implementation. This leads to the logic-gate implementations shown in Figures 4a and 4b. The

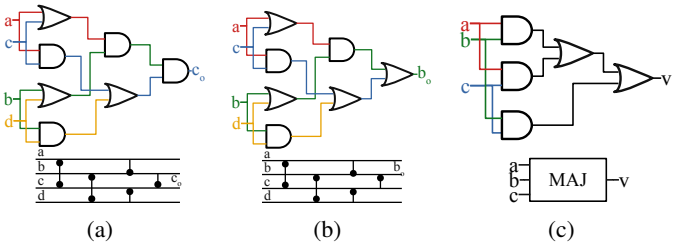

Fig. 4: (a) and (b) sorting-network-based voter implementations (see Figures 3a and 3b); (c) conventional majority voter

former (4a), has to be used to perform a vote for approximate outputs equal to '1' (i.e., Figure 3a and out_1 in Table I). The latter (4b), has to be used to perform a vote for approximate outputs equal to '0' (i.e., Figure 3b and out_0 in Table I). For comparison, in Figure 4c, we report the conventional 3-bit majority voter logic-gate implementation. The proposed SortNet-based voter entails a moderate overhead in terms of logic gates (7 instead of 5) and still has three logic levels, thus it has no impact on the circuit delay, compared to the 3-bit counterpart.

C. Design space exploration

To explore the front of Pareto-optimal QAMR variants, we perform a DSE. We want to obtain optimized QAMR variants and compare these w.r.t. area cost and delay trade-offs with the TMR version. We model the DSE as a Multi-objective Optimization Problem (MOP). Basically, a MOP consists of a set of *objective functions* to be either minimized or maximized subject to a set of constraints. Since different objectives often represent conflicting objectives, the DSE goal is to seek for a set of equally good solutions being close to the *Pareto front*. Given two solutions $x, y : x \neq y$, x is said to *dominate* y iff x is better or equally good in all objectives than y and at least better in one objective. If a solution is not dominated by any others, it is called a *Pareto-optimal* solution.

In the case of NP-hard MOPs, exact resolution algorithms turn out to be too computationally expensive. Therefore, usually they are not applicable when the search space is very large. Consequently, we resort to a Multi-objective evolutionary algorithm (MOEA) heuristic to produce an approximation of the Pareto front in a rather reasonable time. These are largely used in the literature to find Pareto fronts for MOPs [22]–[24]. MOEAs operate on a set of *individuals*, called *population*, that evolves and, eventually, converges to a set of Pareto-optimal solutions. Each individual is represented as a *chromosome*, i.e., a data structure encoding the search space. During the evolution process, new offspring is generated either through or in combination of *mutation* and *crossover*. A *crossover* takes two parent chromosomes to produce a new chromosome.

Applied to our problem of QAMR synthesis, we consider its final area and delay as the two objectives, both to be minimized. Indeed, since it is well-known that power consumption and area are non-competing objectives (i.e., reducing area generally leads to a reduced power consumption), we chose to optimize area and delay. These two objectives are well known to be competing. An individual of the MOEA corresponds to a QAMR implementation, i.e., the attributes of the four approximate replicas and the voter. In particular, we resorted to NSGA-II [23] and used binary encoding for two chromosomes introduced to encode the search space: the first chromosome encodes the choice of which two input literals shall be used for selecting the partition of the input domain of the given Boolean function into four subsets, whilst the second chromosome is used to select the values of the falsified outputs (as shown in red in Table I). In this way, by mutating the chromosomes, the MOEA makes the QAMR implementations evolve towards the Pareto front. The next section describes our DSE flow in more details.

IV. QAMR DSE Flow and Experimental Results

In this section, we firstly describe the adopted DSE flow, then we show and comment our experimental setup and results, and compare them with the state of the art.

A. DSE flow

Figure 5 shows the flow of the proposed DSE. It is based on the utilization of a MOEA (i.e., NSGA-II [23]) to explore the different possible approximation opportunities. As a point

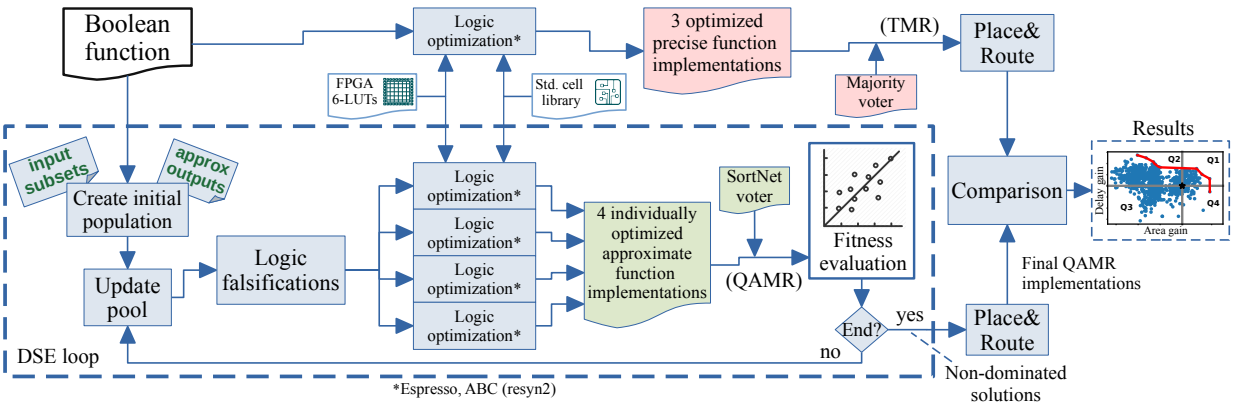

Fig. 5: MOEA-based DSE flow.

of reference, also for the original Boolean function, a logic optimization is applied to create an optimized *precise* (i.e., non-approximate) function implementation. Three identical replicas will form the TMR reference implementation along with a conventional majority voter. For this design point and also each explored approximate QAMR implementation as proposed by the MOEA, Espresso [25] and ABC resyn2 [26] are used to perform different logic optimizations[1] oriented to the final implementation in 6-LookUp Tables (LUTs) Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) technologies. Considering the DSE, a first population of solutions (i.e., multiple different input subsets and approximate output values) is generated randomly. According to its chromosome, the logic falsification operation is applied to the Boolean function for each individual in the population, as described in Subsection III-A, to obtain four approximate Boolean functions (see Table I for an example). The subsequent logic optimization creates four optimized approximate functions; these will form the QAMR along with the SortNet-based voter. Finally, the fitness functions are evaluated, i.e., the costs in terms of area and delay of each explored QAMR is evaluated w.r.t. the target technology. For the ASIC target, the circuit area cost is determined by the number of logic gates, for the FPGA as the number of used LUTs in the respective synthesis reports. For both the technologies, the critical path delay is obtained from the synthesis reports. Finally, the chosen MOEA is elitistic: at each time, it keeps an archive of so-far non-dominated individuals (i.e., QAMR versions providing improvements in terms of area and/or delay). The dominance-free archive is also updated from population to population. The process iterates until a stop condition is reached, hence obtaining a final population of QAMR solutions on the Pareto front or at least close to it. Last, each solution in the final dominance-free archive is Place&Routed according to the target technology (i.e., FPGA or ASIC). The obtained QAMR implementations are then compared with the reference TMR and the results are reported.

[1]For a fair comparison, we applied exactly the same logic optimizations to both QAMR and TMR.

The size of the design space can be calculated as follows:

$$\binom{n}{2} \cdot 2^m = \frac{n! \cdot 2^m}{2! \cdot (n-2)!} \tag{1}$$

In Equation 1, n is the number of function inputs and m the number of function outputs. In particular, $\binom{n}{2}$ corresponds to the number possible choices of which two input literals shall be used for selecting the partition of the input domain of the given Boolean function into four subsets; 2^m is the number of possible combinations of the values of the falsified outputs.

B. Experimental Setup

We evaluated our approach on 41 generic combinational circuits from the publicly available LGSynth'91 benchmark suite [27], ranging in complexity from $n = 4$ to 128 inputs and $m = 5$ to 66 outputs. For each circuit, we applied the flow described in Figure 5. The genetic algorithm parameters were set as follows: the mutation probabilities of the chromosomes choosing the input subsets and fixing the output values were set to n^{-1} and 0.5, respectively, where n is the number of input literals; both chromosomes had a crossover probability of 0.6; both the population size and the number of epochs were set to 256. After the DSE, to evaluate the obtained final set of non-dominated solutions for FPGA and ASIC target technologies, we applied a Place&Route phase for each non-dominated solution (see Fig. 5). For the FPGA target, we used *Xilinx Vivado* and targeted the Xilinx Zynq-7020 FPGA; for the ASIC technology we used *Genus Synthesis Solution* from Cadence and targeted the *FreePDK45 45nm* technology library [19]. For both, we used standard synthesis commands without further optimizations. To obtain the final area/delay results, we resorted to the tool reports. As point of comparison to analyze the QAMR gains, we applied the same process to the reference TMR. Moreover, we synthesized, in both technologies, the proposed SortNet-based voter to compare it to the conventional majority voter.

C. Experimental Results

In Table II, we present the results of the synthesis for the proposed new SortNet-based voter and compare it to the conventional majority voter in terms of area and delay.

TABLE II: Synthesis results for the proposed SortNet-based voter and comparison with the conventional majority voter

	Attribute	SortNet-based voter	Majority voter
FPGA	Area (LUTs)	1	1
	Delay (ns)	5.229	5.229
ASIC	Area (μm^2)	16.425	11.732
	Delay (ns)	0.138	0.138

Concerning FPGA technology, both voters occupy a single LUT, since both implement a simple logic function. Consequently, also the delay is the same. We report the maximum delay for the slowest process corner allowed by Xilinx.

Concerning ASIC technology, the SortNet voter has the same delay as the conventional majority voter and a moderate area overhead (+40%). However, considering that the voter is usually smaller than the circuits to vote for, the introduced overhead is likely to be negligible compared to the gains achieved thanks to the approximation. In the following, reported results take into account the whole architecture, i.e., AxICs and voter.

As already mentioned, the results provided by the MOEA is a set of Pareto-optimal QAMR variants. The results reported are expressed as relative gains for both area and delay, according to the following formula:

$$\text{gain}\ (\%) = \frac{(\text{TMR value} - \text{QAMR value})}{\text{TMR value}} \cdot 100 \qquad (2)$$

We organized the results on a Cartesian plane where x-axis represents the *area gain* w.r.t. TMR and the y-axis the *delay gain* w.r.t. TMR. The reference TMR version lays in the origin of the Cartesian axes. The QAMR variants can be positioned in different quadrants of the Cartesian plane. Specifically:

- if a variant lies in the *Q1* quadrant ($x > 0 \wedge y > 0$), it achieves gains in terms of **both area and delay**;
- if a variant lies in the *Q2* quadrant ($x \leq 0 \wedge y > 0$), it achieves gains only in terms of delay;
- if a variant lies in the *Q4* quadrant ($x > 0 \wedge y \leq 0$), it achieves gains only in terms of area;
- if a variant lies in the *Q3* quadrant ($x \leq 0 \wedge y \leq 0$), it does not achieve gains, at all.

To give a detailed example, in Figure 6, we report the final population for the *SEQ* circuit mapped to ASIC, along with the relative Pareto front. The figure highlights that for a lot of QAMR

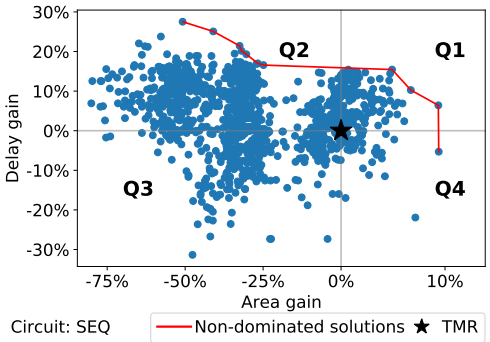

Fig. 6: Example of QAMR population and related MOEA-generated Pareto front, for the SEQ circuit [27]

TABLE III: Experimental results organized in categories

Circuit	Inputs	Outputs	Target	Quadrant*	Runtime (hours)
5xp1	7	10	ASIC	Q1	9.04
			FPGA	Q2	8.69
alu2	10	6	ASIC	Q1	10.69
			FPGA	Q2	10.22
alu4	14	8	ASIC	Q2	35.24
			FPGA	Q1	33.96
apex1	45	45	ASIC	Q1	28.74
			FPGA	Q2	29.90
apex3	54	50	ASIC	Q2	30.30
			FPGA	Q2	31.22
apex4	9	19	ASIC	Q1	39.17
			FPGA	Q2	39.54
b12	15	9	ASIC	Q2	9.09
			FPGA	Q2	8.72
b9	41	21	ASIC	Q2	10.47
			FPGA	Q3	10.47
c8	28	18	ASIC	Q2	9.22
			FPGA	Q2	8.64
cc	21	20	ASIC	Q2	8.66
			FPGA	Q2	8.33
cht	47	36	ASIC	Q2	9.16
			FPGA	Q2	8.83
clip	9	5	ASIC	Q2	10.22
			FPGA	Q1	9.69
cm138a	6	8	ASIC	Q2	8.07
			FPGA	Q4	7.28
cm162a	14	5	ASIC	Q2	8.28
			FPGA	Q2	8.04
cm163a	16	5	ASIC	Q1	8.28
			FPGA	Q2	7.91
cm42a	4	10	ASIC	Q2	8.18
			FPGA	Q3	7.75
count	35	16	ASIC	Q2	10.20
			FPGA	Q1	9.82
cu	14	11	ASIC	Q2	8.32
			FPGA	Q1	7.79
dalu	75	16	ASIC	Q2	8.09
			FPGA	Q2 or Q4	55.10
decod	5	16	ASIC	Q2	12.17
			FPGA	Q3	7.55
duke2	22	29	ASIC	Q2	13.86
			FPGA	Q1	10.84
e64	65	65	ASIC	Q2	106.35
			FPGA	Q1	12.54
ex4	128	28	ASIC	Q2	14.89
			FPGA	Q2	31.56
ex5	63	63	ASIC	Q2	16.53
			FPGA	Q1	15.29
example2	85	66	ASIC	Q2	8.49
			FPGA	Q2	14.59
i1	25	16	ASIC	Q2	28.16
			FPGA	Q3	7.78
k2	45	45	ASIC	Q1	9.15
			FPGA	Q1	23.16
lal	26	19	ASIC	Q2	8.44
			FPGA	Q2	8.31
misex1	8	7	ASIC	Q1	8.95
			FPGA	Q1	7.95
misex2	25	18	ASIC	Q2	33.40
			FPGA	Q1	8.15
misex3	14	14	ASIC	Q1	8.66
			FPGA	Q1	34.24
pcle	19	9	ASIC	Q2	9.17
			FPGA	Q2	7.78
pcler8	27	17	ASIC	Q2	8.27
			FPGA	Q2 or Q4	7.21
pm1	16	13	ASIC	Q2	8.78
			FPGA	Q2	7.22
sct	19	15	ASIC	Q2	28.05
			FPGA	Q2	7.60
seq	41	35	ASIC	Q1	8.16
			FPGA	Q2 or Q4	28.10
tcon	17	16	ASIC	Q3	10.23
			FPGA	Q3	7.22
term1	34	10	ASIC	Q2	9.02
			FPGA	Q1	11.46
unreg	36	16	ASIC	Q2	9.83
			FPGA	Q3	7.69
vg2	25	8	ASIC	Q2	6.54
			FPGA	Q1	9.03
x2	10	7	ASIC	Q2	64.84
			FPGA	Q1	7.32
Min.	4	5	–	–	6.54
Max.	128	66	–	–	106.35
Avg.	30	21	–	–	15.8

*The circuit has non-dominated variants in the reported quadrant

Summary		FPGA		ASIC	
Circuits with non-dominated variants	in Q1	34.1%	(14)	22.0%	(9)
	in Q2	41.5%	(17)	75.6%	(31)
	in Q4	2.4%	(1)	0.0%	(0)
	in Q2 or Q4	7.3%	(3)	0.0%	(0)
	in Q3	14.6%	(6)	2.4%	(1)
Max Area Gain		39.75%		9.4%	
Max Delay Gain		14.6%		30.32%	
Circuits with gaining variants (i.e., Q1, Q2, Q4):		**85.4%**		**97.6%**	

variants (i.e., individuals in the population) we achieved a gain in **both area and delay** (quadrant Q1), or only in time (quadrant Q2), or only in area (quadrant Q4); for some variants, the DSE also achieved no gains at all (quadrant Q3). More importantly, the DSE allowed us to find Pareto-optimal solutions (highlighted in red), including even solutions achieving a gain in terms of both area and delay (Q1) or at least in terms of one of them (Q2, Q4).

We grouped the results into 5 categories:

1) circuits presenting **at least one** Pareto-optimal solution in the *Q1* quadrant;
2) circuits (not in Q1) presenting at least one Pareto-optimal solution in the *Q2* quadrant;
3) circuits (not in Q1) presenting at least one Pareto-optimal solution in the *Q4* quadrant;
4) circuits (not in Q1) presenting Pareto-optimal solutions either in the *Q2* or in the *Q4* quadrants; and
5) circuits presenting all Pareto-optimal solutions in the *Q3* quadrant.

In Table III, we report the results of the DSE in terms of number of circuits in each category, for FPGA and ASIC technologies.

For FPGA technology, we were able to find, for 34.1% of the circuits, QAMR variants achieving gains in terms of **both area and delay**; for 41.5% of the circuits, we found variants achieving gains in terms of delay; 2.4% of the circuits achieved gains in terms of area and other 7.3% in terms of either area or delay. Only for six circuits (15%) we did not find any variants achieving a gain. For two of them the QAMR versions are equivalent to the TMR (they are located in the origin of the Cartesian plane). The maximum achieved gains were 39.75% in area and 14.6% in delay.

Concerning ASIC technology, for 22% of the circuits we found variants achieving gains in terms of **both area and delay**; for 75.6% of the circuits we found variants achieving gains in terms of delay. Only for one circuit (2%) the DSE did not produce any variants achieving gains. The maximum achieved gains were 30.32% in delay and 9.4% in area.

In general, the approach always succeeded in finding QAMR variants in Q1 for large circuits: their non-approximate versions had on average $\approx$ 125 LUTs and $\approx$ 7.21 ns of delay for FPGA, and $\approx$ 1797.2 μm^2 area and $\approx$ 0.843 ns of delay for ASIC. Conversely, circuits with QAMR versions in Q3 occurred only in case of very small test circuits: their non-approximate versions counted on average $\approx$ 11 LUTs and $\approx$ 5.52 ns of delay for FPGA, and $\approx$ 41.298 μm^2 area and $\approx$ 0.061 ns of delay for ASIC. For example, for the *tcon* circuit, we did not manage to find any TMR-dominating QAMR solution, neither for FPGA nor for the ASIC targets: its non-approximate version just occupies 4 LUTs and has 5.255 ns delay when mapped to FPGA, and occupies 41.298 μm^2 and has 0.061 ns delay when mapped to ASIC. Finally, the runtime needed to complete the MOEA-based DSE spans from 6.5 hours to 106.35 hours, with an average of 15.8 hours.

D. Comparing and extending previous results on QAMR

In this subsection, we report another comparison of the approach presented in this paper with the state-of-the-art QAMR approach as reported in [16], being based on the *output removal* approximation. However, in [16], no technique to co-optimize area and delay was proposed and no optimizer (such as NSGA-II) was employed to select the best subset of outputs to remove from each replica. Rather, the work was based on random search. Therefore, to extend and realize a fair comparison, we replicated the technique described in [16], i.e., *output removal*, and performed a MOEA-based DSE on a reduced subset of the experimental circuits and Place&Routed the resulting non-dominated solutions to the 45nm ASIC technology, as in [16]. In this DSE, the MOEA's task was to select the best subset of outputs to remove from each replica.

Figure 7 reports in the Cartesian planes the experimental results: the green squares (■) represent the non-dominated solutions obtained from the DSE when using the *logic falsification* approximation, as presented in this paper; the orange circles (●) represent the non-dominated solutions obtained from the DSE when using the *output removal* approximation; and the blue triangles (▲) represent the results reported in [16], i.e., the non-dominated solutions obtained from random search when using the *output removal* approximation. Finally, the reference TMR version is depicted as a black star (★) in the origin of the axes.

As expected, the proposed MOEA-based DSE allowed to find solutions dominating those obtained using the simple random search used in [16]. Indeed, for the *output removal* approximation, the non-dominated solutions found with the DSE (●) dominate the solutions found with the random exploration (▲).

On the other hand, when comparing the outcomes of the two DSE, i.e., performed by using *logic falsification* (■) and *output removal* (●) respectively, it is not possible to identify a clear dominance of one front of the other front. To suitably measure and illustrate this condition, let us resort to the *Coverage of two sets* metric, proposed in [28]. Let *A* and *B* be two sets of non-dominated solutions for a MOP. The function $\mathcal{C}$ maps the pair $(A,\ B)$ to the interval $[0, 1]$:

$$\mathcal{C}(A, B) := \frac{|\{\forall \beta \in B;\ \exists\, \alpha \in A : \alpha \succeq \beta\}|}{|B|} \tag{3}$$

where the expression α *covers* β ($\alpha \succeq \beta$) means that the solution α *dominates* the solution β or they are *the same* solution. The value $\mathcal{C}(A, B) = 1$ means that all points in *B* are dominated or equal to at least one point in *A*. Conversely, the value $\mathcal{C}(A, B) = 0$ represents the situation where no points in *B* are dominated or equal to any point in *A*. When using this metric, both $\mathcal{C}(A, B)$ and $\mathcal{C}(B, A)$ have to be considered, as they are not necessarily equal. In Table IV, we report the values of the above described $\mathcal{C}$ metric applied to the outcomes of the two DSE (i.e., with *logic falsification* (■) and *output removal* (●)), for the circuits in Figure 7. While for some circuits – such as *clip* and *seq* – solutions obtained using logic falsification clearly cover those obtained with

TABLE IV: *Coverage of two sets* metric evaluation over the DSE results when using *logic falsification (LF)* and *output removal (OR)*, for the circuits in Figure 7.

Circuit	5xp1	alu2	clip	cm42a	count	decod	pcle	pm1	seq
$\mathcal{C}$ (LF, OR)	0.00	0.50	1.00	0.00	0.00	0.00	0.00	0.00	1.00
$\mathcal{C}$ (OR, LF)	1.00	0.29	0.00	0.00	0.85	1.00	0.19	1.00	0.00

LF: *Logic Falsification* (■), OR: *Output Removal* (●)
$\mathcal{C}(A, B)$: *coverage of two sets* metric [28]

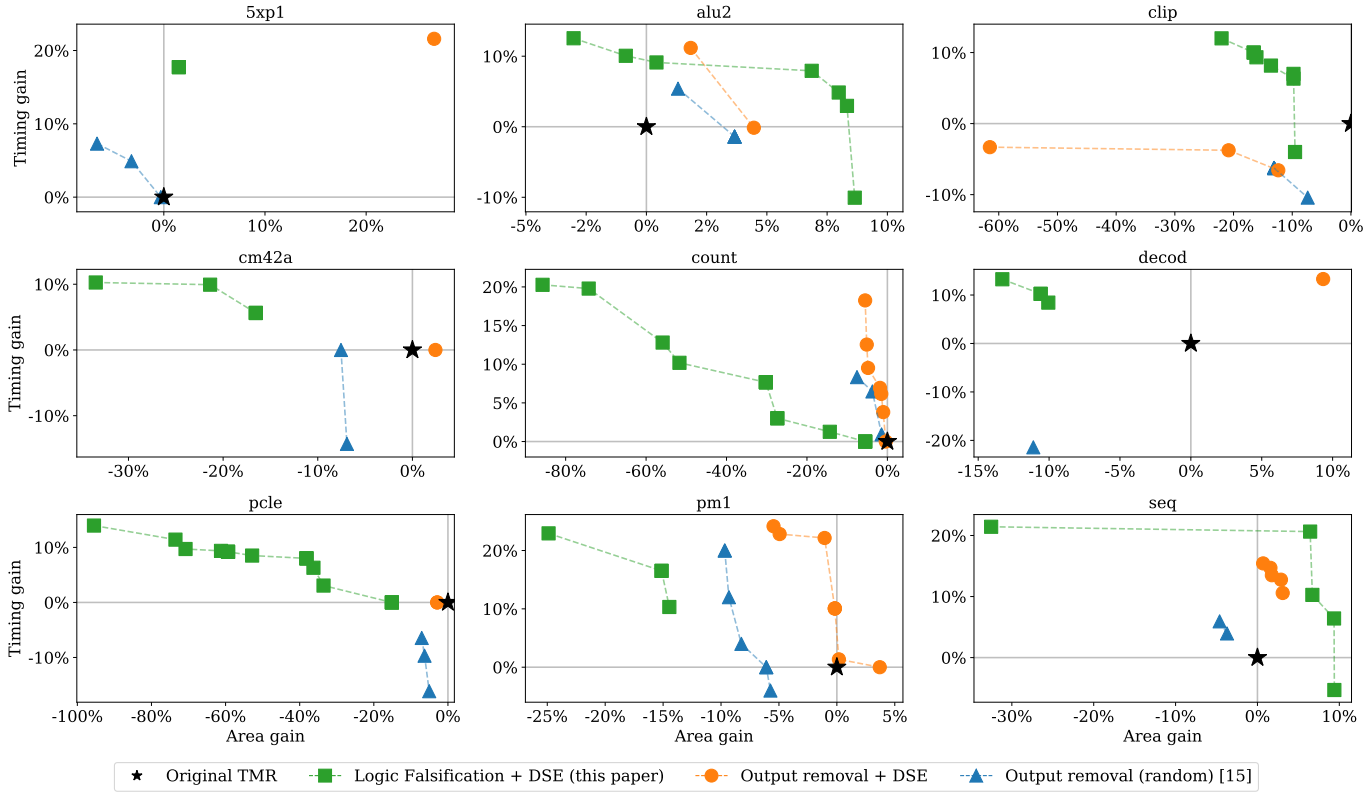

Fig. 7: Comparison with the study in [16]. Pareto fronts for selected circuits, obtained as follows: (i) by using the systematic DSE of logic falsifications as proposed in this paper (depicted as ■), (ii) by applying the DSE – as proposed in this paper – to the *output removal* approach from [16] (depicted as ●), and (iii) by using the *output removal* approach *randomly*, i.e., the results reported in [16], (depicted as ▲). The results are expressed in terms of area and delay percentage gain.

output removal ($\mathcal{C}\,(\mathrm{LF},\mathrm{OR}) = 1$), for other circuits – such as *5xp1*, *decod*, and *pm1* – the solutions obtained with output removal cover those obtained with logic falsification ($\mathcal{C}\,(\mathrm{OR},\mathrm{LF}) = 1$). For the remaining circuits, both the approaches are able to identify non-dominated solutions not covered by those of the other method. In particular, for the *count* and *pcle* circuits, using the *output removal* approach, it was possible to find some solutions covering the ones found by *logic falsification* (85% and 19%, respectively), and others equally good. For *alu2*, the solutions found using *logic falsification* covered 50% of those found with *output removal*; at the same time, the solutions found with *output removal* covered 29% of those found with *logic falsification*. Finally, for the *cm42a* circuit, the solutions found by both approaches show both methods to be equally good, as mutually, the Pareto front of one did not cover any point of the Pareto front of the other one.

V. Conclusion

In the context of error-tolerant applications, various studies proposed to apply approximate computing (AxC) to relax reliability constraints and achieve efficiency gains. Unfortunately, in most advanced safety-critical computing systems, sacrificing reliability may result in endangering human lives. Recently, the first AxC-based fault-tolerant approach not sacrificing the reliability requirements has been presented, i.e., the *Quadruple Approximate Modular Redundancy (QAMR)* [16]. It uses four AxICs to reduce the standard TMR overhead without sacrificing fault tolerance capabilities. So far, QAMR-oriented circuits approximation was based on *output removal*, i.e., selectively removing output cones. The main problem preventing the use of other approximations was the lack of an efficient four-bit voter. In this paper we proposed to use another approximation approach, i.e., based on *logic falsification*, to explore QAMR architectures. To do so, we proposed a new sorting-network-based four-bit majority voter and adapted it to the new proposed approach. Finally, we performed an extended Design Space Exploration (DSE), based on a Multi-objective evolutionary algorithm (MOEA), to find Pareto-optimal QAMR configurations for both FPGA and ASIC technology implementations.

Results showed that it is possible to find QAMR variants achieving gains compared to the TMR counterpart for 85.4% and 97% of the examined circuits, for FPGA and ASIC technologies, respectively. In particular, for FPGAs, QAMR variants superior in terms of both area and delay were found for ≈34% of the circuits and in terms of either of them for ≈51% of the circuits; for ASICs, QAMR variants superior in terms of both area and delay were found for ≈22% of the circuits and in terms of timing for ≈75% of the circuits. Moreover, we applied the proposed MOEA-based DSE also to the *output-removal*-based approximation approach proposed in [16] and showed that the DSE delivers non-dominated solutions also w.r.t. the random search used in [16].

ACKNOWLEDGMENT

This work has been partially funded by the projects "IDEX Lyon OdeLe". It has been partially supported also by the German Science Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant TE163/22-1.

REFERENCES

[1] K. Weide-Zaage and M. Chrzanowska-Jeske, "Semiconductor devices in harsh conditions," *CRC Press*, 2016.

[2] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[3] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–9.

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

[5] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, Mar. 2016.

[6] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC Design Automation Conference 2012*, 2012, pp. 820–825.

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

[8] A. Raha, S. Venkataramani, V. Raghunathan, and A. Raghunathan, "Quality configurable reduce-and-rank for energy efficient approximate computing," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 665–670.

[9] V. Mrazek, Z. Vasicek, and R. Hrbacek, "Role of circuit representation in evolutionary design of energy-efficient approximate circuits," *IET Computers Digital Techniques*, vol. 12, no. 4, pp. 139–149, 2018.

[10] S. Eldridge, F. Raudies, D. Zou, and A. Joshi, "Neural network-based accelerators for transcendental function approximation," in *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*, ser. GLSVLSI '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 169–174.

[11] I. A. C. Gomes, M. Martins, A. Reis, and F. L. Kastensmidt, "Using only redundant modules with approximate logic to reduce drastically area overhead in tmr," in *2015 16th Latin-American Test Symposium (LATS)*, 2015, pp. 1–6.

[12] B. D. Sierawski, B. L. Bhuva, and L. W. Massengill, "Reducing soft error rate in logic circuits through approximate logic functions," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3417–3421, 2006.

[13] A. Sánchez-Clemente, L. Entrena, M. García-Valderas, and C. López-Ongil, "Logic masking for set mitigation using approximate logic circuits," in *2012 IEEE 18th International On-Line Testing Symposium (IOLTS)*, 2012, pp. 176–181.

[14] I. A. Gomes, M. G. Martins, A. I. Reis, and F. L. Kastensmidt, "Exploring the use of approximate tmr to mask transient faults in logic with low area overhead," *Microelectronics Reliability*, vol. 55, no. 9, pp. 2072 – 2076, 2015, proceedings of the 26th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[15] T. Arifeen, A. S. Hassan, and J. A. Lee, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139 851–139 867, 2020.

[16] B. Deveautour, M. Traiola, A. Virazel, and 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.

[17] J. Echavarria, S. Wildermann, and J. Teich, "Design space exploration of multi-output logic function approximations," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018.

[18] G. Rodrigues, J. Fonseca, F. Kastensmidt, V. Pouget, A. Bosio, and S. Hamdioui, "Approximate tmr based on successive approximation and loop perforation in microprocessors," *Microelectronics Reliability*, vol. 100-101, p. 113385, 2019.

[19] "Freepdk45," https://www.eda.ncsu.edu/wiki/FreePDK45:Contents, accessed: February 2, 2022.

[20] D. G. O'connor and R. J. Nelson, "Sorting system with n-line sorting switch (US3029413A)," Patent.

[21] B. W. Johnson, *Design & Analysis of Fault Tolerant Digital Systems*. USA: Addison-Wesley Longman Publishing Co., Inc., 1988.

[22] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[24] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithms: Analyzing the state-of-the-art," *Evolutionary computation*, vol. 8, no. 2, pp. 125–147, 2000.

[25] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for pla optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 727–750, 1987.

[26] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification." [Online]. Available: http://www.eecs.berkeley.edu/∼alanmi/abc/

[27] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0, technical report," 1991. [Online]. Available: https://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.7z

[28] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.