IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 10, 2020, accepted July 22, 2020, date of publication July 29, 2020, date of current version August 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012673

Approximate Triple Modular Redundancy: A Survey

TOOBA ARIFEEN, ABDUS SAMI HASSAN, AND JEONG-A LEE, (Senior Member, IEEE)
Department of Computer Engineering, Chosun University, Gwangju 61452, South Korea

Corresponding author: Jeong-A Lee (jalee@chosun.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Education through the Basic Science Research Program under Grant 2019R1I1A3A01058887, and in part by the Korea Institute of Energy Technology Evaluation and Planning, and the Ministry of Trade, Industry and Energy, South Korea, under Grant 20184010201650.

ABSTRACT In recent years, approximate computing (AC) has attracted attention owing to its tradeoff between the exactness of computations and performance gains. AC has also been probed for the technique of Triple modular redundancy (TMR). TMR is a well-known fault masking methodology, with associated overheads, widely used in systems of different nature and at different levels. E.g.: layout-level, gate-level, HW-module level, software. At hardware level, through exploitation of AC the 200% area overhead problem due to triplication of the original modules in TMR can be reduced. By approximating the modules of TMR while ensuring that at least two of the approximate modules do not differ from the original module for every input vector, the facilitation of fault masking can lead to overhead reduction. Hence, approximate TMR (ATMR) aims to achieve cost-effective reliability. Nevertheless, due to the extensive search space, computational complexity, and principal fault masking function of ATMR, designing an ATMR is a challenging task. An ATMR technique must be scalable so that it can be easily adopted by circuits having large number of inputs and the extraction of ATMR modules remains computationally inexpensive. Compared with TMR, due to the inclusion of approximations, ATMR is more vulnerable to errors, and hence, the design technique must ensure awareness of input-criticality. To the best of the authors' knowledge, none of the existing survey articles on AC has reported on ATMR. Therefore, in this work, ATMR design techniques are thoroughly surveyed and qualitatively compared. Moreover, design considerations and challenges for designing ATMR are discussed.

INDEX TERMS Approximate computing, fault tolerance, single event upset, soft errors, ATMR, performance tradeoffs, TMR.

I. INTRODUCTION

Approximate computing (AC) is an emerging technology that allows the inexactness, and approximation of output where the required numerical exactness is governed by the error resilience threshold of the application under consideration. AC is motivated by the benefits offered through the utilization of error resilience of applications to yield sufficiently good results instead of the best ones. With the notion of ''provision of not more than necessary,'' AC probes and ensures for the accuracy requirements of applications while attaining performance benefits [1], [2], [6], [7].

The tradeoff between accuracy and optimization is the sole objective of AC. It aims to achieve performance benefits in terms of area, energy, power, and latency, at the expense of output quality. Particularly, noisy input, perceptual limitations, and data redundancy have made it easier to accept outputs of non-golden standard in applications where the accuracy desired is flexible. Such flexibility and tolerability of inexact (to an extent) outputs define the inherent error-resilient nature of a wide range of applications that form the in-demand realm of AC [1], [2], [6], [7].

To transform AC into a full-scale practical solution for the present-day limitations faced by the computing industry, diverse pioneering studies are being conducted, revealing potential challenges. Approximation strategies in the literature range from precision scaling to loop perforation to value approximation. Other popular approximation practices involve memoization, memory accesses, skipping tasks and multiple inexact program versions, inexact

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwei Gao.

or faulty hardware, voltage scaling, reduction of branch divergence in single-instruction multiple-data architectures, neural-network-based accelerators, and approximating neural networks. At the device level, static random-access memory (SRAM), dynamic random-access memory (DRAM), eDRAM embedded DRAM), and non-volatile memories have been approximated. At component level, AC has been explored for processor components, graphical processing units, and field-programmable gate arrays (FPGAs) [3]. Based on the implementation level, the work of [8], grouped up approximation computing techniques as: software, architecture, and hardware. Note that some of the approximation methods are implementable on more than one level. Circuit and hardware level techniques exploit voltage scaling and use inexact or fault hardware. Architectural level techniques focus on memory access skipping, use in neural networks, read/write memory approximations, data precision reduction and functional approximations. At software level, function skipping, functional approximations, memoization, loop perforation, read/write memory approximations, data precision reduction are adopted [8]. Existing literature mostly presents for approximate computing at the hardware level rather than software level.

However, challenges in the paradigm of AC are inevitable. Aggressive approximations can lead to output corruption, such as a non-terminable phase that is undetected by the assessment metric for the quality of the result. Uniform approximation is not suitable for applications with characteristic variance. For instance, each layer of deep neural networks can be approximated according to its characteristic variance. If the uniform approximation does not cover the worst characteristic variance, it can cause increased accuracy degradation. To control the influence of approximation on the quality of the result, an insight into program features and the response to fault injection is required, which can be further translated as approximable points through programming frameworks having tunable knobs. Output quality maintenance is essential for AC, necessitating the quality monitoring of the output. The acceptability of error rate and performance requirements must be satisfied, which necessitates a thorough analysis of how faults would internally affect the application under consideration for a given SER. The criteria of approximation and approximation zones may not have to be uniform for an application in order to satisfy the quality requirements. For instance, it is critical to express and distinguish between approximable and inapproximable regions in the design space at the hardware and software levels. This can be achieved through diverse means such as automation and output quality monitoring. Another desirable feature is AC supportive programming language support. It is essential to state that some applications are not well suited for approximations, such as cryptography and hard real-time applications, or inexact storage that does not decrease the amount of data operations [5], [7].

AC has been denoted as inexact computing and inaccurate computing where the quality metrics are mainly established based on the error features of the design and its performance. As AC involves outputs of non-golden standard for achieving efficiency optimization through quality configurability, error detection and correction techniques employ AC. Example of such applications are the recognition, mining, and synthesis (RMS) applications which utilize nonconventional input sources (e.g., sensors). For these applications, no specific golden output has to be computed as they deal with processing of noisy data and/or encompass a human interface having constrained perceptual aptitude. Through exploitation of the intrinsic error resilience of RMS applications, AC compromises on the numerical equivalence and trades off performance gains such as energy with computation quality such as accuracy [6]. The pre-computation architecture for decreasing the dynamic power dissipation of sequential circuits was among the initial applications of approximate logic circuits for error detection [9]. Presently, the collapse of Dennard's scaling and the rise of dark silicon are inevitable challenges in the modern computing industry [6], [10]. The processing power of computers is exhausted by the incessant increase in the data to be processed, and the dilemma of shrinking technology has added its intrinsic challenges. Computer systems are susceptible to soft errors caused by high-energy particles or larger type of radiations, leading to a bit flip in logic values [3], [4], [11]. Typically, these soft errors are rectified through a redundancy technique of fault masking. In triple modular redundancy (TMR), an original circuit is replicated three times, and the output of each replication is fed as an input to a majority voter that provides the final output [3], [4]. The triplication of the original modules leads to an area and related overhead of 200%. To overcome the overhead and attain performance benefits, the modules of TMR can be replaced with approximate versions such that the majority voter still produces the exact result. The process of developing TMR with approximate modules is referred to as approximate TMR (ATMR).

Numerous studies have focused on ATMR. Existing works on ATMR have mainly targeted FPGAs and application-specific integrated circuits (ASICs), where some strategies are FPGA-specific, and others are intended for ASIC but are FPGA-adoptable. FPGA-specific ATMR methods use binary decision diagram (BDD), line substitution, and testability measures. In contrast, Boolean factoring and prime implicant (PI) expansion/reduction have been used for the extraction of approximate circuits. The optimization of ATMR using Boolean factoring adopted the approach of full-ATMR and transistor-level structural reordering. In contrast, PI expansion/reduction exploited the list of literal-sorted candidates for optimization, and input vulnerability was explored as an optimization approach. Another approach for the extraction of approximation candidates for TMR is based on gate-level approximations, which were optimized through 1) multi-objective optimization, 2) testability and observability, and 3) parity analysis and binate gates. Cartesian genetic programming and dynamic probability estimation have also been used for obtaining approximate

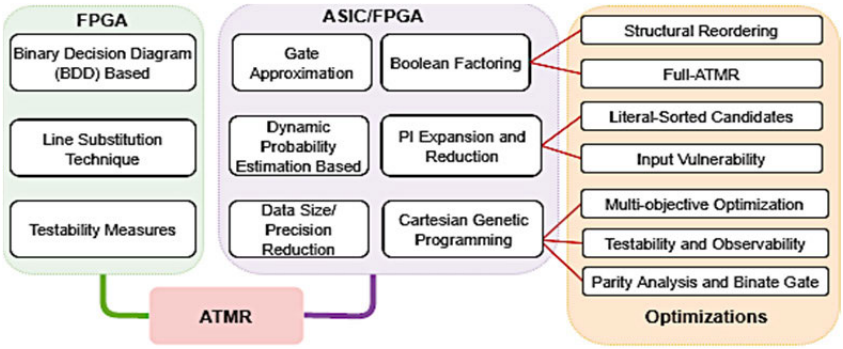


FIGURE 1. Classification of ATMR methods.

modules of ATMR. In other works, data size and precision reduction were used for intensity-based approximation. Thus, we present the classification of ATMR design techniques as shown in Fig. 1. As ATMR is a new concept and is sensitive to faults, a fault-tolerant voter for ATMR has also been proposed in the literature. In Table 4 present an insight on ATMR techniques while Table 3 illustrate the technique through example cases. The major contributions of the survey are as follows:

- The trends of ATMR design techniques are discussed. To the best of the authors' knowledge, this is the first attempt to review and compare ATMR techniques extensively.
- The ATMR methods are classified in accordance with the core approximation techniques.
- The ATMR methods are thoroughly surveyed and qualitatively compared in terms of unique features, potential benefits, and limitations. This comparison may aid researchers and engineers in selection of an appropriate ATMR technique based on their application requirements.

The rest of this paper is organized as follows. Section II describes the typical concepts associated with ATMR. In Section III, issues that must be considered by ATMR design techniques are highlighted. In Section IV, existing ATMR methodologies are reviewed. Section V provides a qualitative comparison among reviewed ATMR schemes. Finally, this paper is concluded in Section VI.

II. PRELIMINARIES

The common concepts related to ATMR design techniques are detailed in this section.

A. ATMR AND FULL ATMR

The concept of replacing the exact modules of a TMR with approximate modules is referred to as ATMR. However, for maintaining the fault masking ability of TMR, the working principle of ATMR has been stipulated as follows [11]:

"*Only one of the approximate modules can differ from the original circuit at each input vector scenario, allowing the majority voter to still select two match outputs out of three for any input vector.*"

TABLE 1. Approximate TMR: An example.

Input			Original circuit output [G]	Modules forming ATMR			Voter output [ATMR output]
A	B	C		[G]	Approximate Circuit 1 [f1]	Approximate Circuit 2 [f2]	
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	**0**	1	1
0	1	1	0	0	0	**1**	0
1	0	0	0	0	0	**1**	0
1	0	1	1	1	**0**	1	1
1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	1

Table 1 presents an ATMR using G, f1, and f2. G is the output of a given circuit, where G= A'B' + A'C' + AC. It can be seen that G and f1 produces same outputs except for the input vectors A'BC' and AB'C (bold values) where G produced 1 and while f1 produced 0 (highlighted in yellow). Similarly, G and f2 produces sane outputs except for the input vectors A'BC and AB'C' where G produced 0 and while f2 produced 1. Hence, f1and f2 are approximate module for G. When G, f1, f2 forms an ATMR, the voter ensures that majority value is propagated to enable the output to be same as G (highlighted in green).

B. FULL-ATMR

When all the three exact modules of TMR are replaced with approximate modules, a fuller form of ATMR is obtained, which is called full-ATMR (FATMR) [11], [12]. Table 2 presents FATMR for G= BC' + AB'. FATMR consists of f1=A, f2= C' + AB', and f3=A'BC'. Here, f3 is the output of the third approximate module. The approximate module f1 differs from G at A'BC' and ABC (bold values) by producing complemented values (highlighted in yellow), f2 differs from G at A'B'C' by producing 1 (highlighted in yellow) instead of 0, and f3 differs from G at AB'C', AB'C, ABC' by producing 0 (highlighted in yellow) instead of 1. the voter which enables output to be same as G (highlighted in green). Note that in contrast with TMR, if an error arises at A'BC' of f2 then f1 and f2 will propagate an erroneous output through voter. In case of TMR, such an error would not

TABLE 2. FATMR: An example.

Input			Original circuit output [G]	Modules Forming FATMR			Voter output [FATMR output]
A	B	C		Approximate circuit 1 [f1]	Approximate circuit 2 [f2]	Approximate circuit 3 [f3]	
0	0	0	0	0	1	0	**0**
0	0	1	0	0	0	0	0
0	1	0	1	**0**	1	1	**1**
0	1	1	0	0	0	0	0
1	0	0	1	1	1	0	1
1	0	1	1	1	**1**	0	**1**
1	1	0	1	1	1	0	1
1	1	1	0	1	0	0	**0**

have propagated to the output as all modules are exact copy of G.

C. (UN)PROTECTED INPUT VECTORS

In context of ATMR, the probable input patterns for the circuit G under consideration are input vectors. Protected input vectors are input for which approximate module output does not contradict that from G. Unprotected input vectors are input vectors of G for which the approximate module differs from G. For example, in Table 1, 010, 011, 100, and 101 are unprotected vectors. In Table 2, 000, 010, 100, 101, 110, and 111 are unprotected vectors, and the remaining input vectors are protected. The ATMR voter ensures same output as G for unprotected vectors. The percentage of acceptable unprotected vectors is an application-defined phenomenon [9].

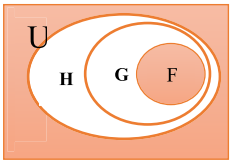


FIGURE 2. Relationship among F, H and G.

D. OVER/UNDER APPROXIMATION

The approximate logic functions diverge from each other with respect to minterms in Hamming code distance. As one of the three modules is allowed to contradict the exact output for each input vector, the voter still chooses two outputs out of the three for the input vectors. This working principle is upheld through the principle: $F \subseteq G \subseteq H$ as shown in Fig. 2, where G is the exact module and F and G are the approximate modules such that minterm of G should be the minterm of H, and minterm of F should be the minterm of G. This condition guarantees that H leads to 0 when G leads to 0, and F leads to 1 when G leads to 1. The approximate functions H and F are referred to as over-approximate/0-approximation and under-approximate/1-approximation functions, respectively [13], [14].

E. PI AND PI EXPANSION/REDUCTION

For a Boolean function, the cover of one or more minterms of the sum of products is referred to as an implicant and if they cannot be covered by a more general implicant, the implicant holds the status of a prime implicant.

The expansion (reduction) of the original PI cover through 0 to 1 (1 to 0) complement is PI expansion (PI reduction).

III. DESIGN CONSIDERATIONS FOR ATMR

Designing ATMR is not a straightforward logic minimization problem. In fact, fault masking performed by ATMR introduces inherent complexity to the design technique. In addition, the development of ATMR transmutes into an optimization problem, as maximum performance benefits are desired. Here, we highlight the required considerations and faced challenges for designing ATMR.

A. APPLICATION-SPECIFIC PHENOMENON

Fault masking mainly aims to provide higher reliability. ATMR must not compromise on fault coverage below the acceptable limits as defined by the error resilience of applications. Thus, the aptness of ATMR is application-dependent, as the critical and non-critical states of the system can be discerned by the application or design engineer for a sensible adoption of ATMR. Consider the case of the LEON-FT microprocessor for the mission-control computer of a spacecraft, where radiation-hardened units are used. On the contrary, the scientific gadgets for a spacecraft are built based on commercial-off-the-shelf (COTS) units having minimum or no error masking. This is because radiation-hardened equivalents of COTS units, if available at all, consume high power with lower performance [15]. However, power is a scarce resource in spacecraft in contrast to COTS units. Furthermore, the risk of failure of individual experiments might be acceptable; however, a critical failure in an on-board mission control system could lead to failure of experiments and the mission itself [15], [16].

B. PRONENESS TO ERROR

Table 2 presents an example of ATMR. Now, if a TMR is built as $f1 = f2 = f3 = G$ and an error arises at f1 for the input 100. In such a situation, f2 and f3 will warrant that the output of the voter remains identical to that of G. However, if f1 of ATMR flips because of an error, a flawed output will be generated, as two of the modules (f1 and f3) will vote for the same output value. Therefore, in contrast to TMR, ATMR is susceptible to errors, and hence, it is of enormous importance to avoid approximations at critical inputs. An error occurrence may manifest an error at the output. It should be noted that the inputs have different levels of importance, as a certain portion of input space is more susceptible to errors than the rest [7]. The formation of ATMR is sensitive to critical input scenarios and the objective of TMR might be defeated. The mechanism of constructing approximate functions for ATMR must emphasize the portions of the input space that are not susceptible to errors and must ensure that the portions of the input space most susceptible to errors remain intact.

References [14], [17], [18], at transistor level, used the notion of sensitive nodes which are a logic circuit's set of p-n junctions sensitive to soft errors. In case of Complementary Metal Oxide Semi-Conductor (CMOS) gate, with a total of n p-n junctions in pull-up and pull-down networks, m sensitive nodes out of n p-n junctions can be hit by a single event transient and demonstrate its effect at output of logical gate [14]. The worked focused on customization the circuits by means of structural reordering, input vector permutation, and transistor topology, specifically such that sensitive vectors will have lesser probability of generating soft errors at the unprotected input vectors of ATMR. The criticality of input space was utilized by partial TMR for FPGA in [19]. References [12] and [9] introduced input vulnerability in ATMR [20], by utilizing automatic test pattern generation (ATPG) to recognize critical inputs and avoid approximation in that space. Reference [21] highlighted that approximations can compromise data accuracy and ATMR methods are restricted by theoretical and mathematical constraints for the assurance of fault masking. In [21], through the representation of data using a reduced number of bits, the usability of ATMR for hardware and software was improved.

C. SELECTION OF THE BEST COMBINATION

For an ATMR, numerous approximate candidates of exact modules must be generated at an abstract level. To form an ATMR, the different combinations of (two/three) approximation candidates must not violate the working principle of ATMR and yield optimal performance gains. As the generation of such a combination itself is a two-/three-fold phenomenon, the ATMR design technique must intelligently choose/formulate either best or a sufficiently good combination of approximations. Reference [11] necessitates the consideration of large variation of approximations and well-defined criteria for identifying optimal ATMR using a combination of approximate modules from list of approximate candidates. References [9], [12], and [20] set a clear criterion for the selection of the best combination by using the blocking property and the literal count. Blocking refers to those input vectors whose output can no longer be complemented and should necessarily hold the same value as the original function which ensures that at each input vector only one approximate module differs from the original circuit. Once candidate approximate functions are obtained, three best approximate functions with minimum number of literals are selected.

D. ATMR VERSUS FATMR

Rationally, FATMR offers a minimum area overhead; however, it is not a rule of thumb, as the decrease depends on the original logic. Consider an input function G=a'cd. For an unprotected vector, the approximate functions of G are f1=abc'd, f2=ab'c'd, and f3=ad(c'+b) For two unprotected vectors, the approximate functions of G are f4=ad and f5= ac'. For constructing an ATMR with four unprotected vectors, some of the possible candidates are f1/f5/f2 and f1/f4/f3. The area overhead for these FATMR versions is 12 literals, which is worse than that of TMR (9 literals). On the contrary, an ATMR of G/f4/f5 will have only 7 literals. As an approximation with one unprotected vector is larger than G itself, FATMR with four unprotected vectors with a sufficiently good combination of approximations is unavailable. A methodology for approximating TMR must intelligently determine whether ATMR, which has one/two approximate modules, or FATMR, which has all approximate modules, is an appropriate choice for attaining the optimal benefits. Reference [17] was based on structural reordering, and FATMR based on Boolean factoring was studied in [18]. Based on the technique in [19], FATMR cannot be developed using the line substitution technique. Reference [19] proved that ATMR technique might show greater reliability than FATMR in FPGA applications using testability measures [22].

E. SCALABILITY

Scalability of ATMR design technique is an important issue. Synthesis-based approximation methods mainly rely on functional considerations offering limited scalability. The exhaustive search space and computational complexity of ATMR translate into methodologies that are suitable for small circuits only. Hence, the design techniques of ATMR must be equally viable for small and large networks.

To improve design scalability, the method presented in [14] was introduced with modified Boolean factoring algorithm of [17] for approximate candidate generation. Reference [11] requires the knowledge of the original Boolean function of the entire circuit explicitly. Furthermore, this method is limited to manipulate functions with up to eight inputs and the use of KL-cuts is suggested for larger circuits. Inexact TMR in [23] used a scalable heuristic independent of the circuit size. Reference [19] departs from the synthesis technique of a technology-dependent network of circuits by using the line substitution technique of approximation. Reference [20] utilized a clustering algorithm of the ABC tool for circuits having more than 15 inputs. Reference [12] was improved by the inclusion of a heuristic of a sorted list of approximation candidates in [9], which was further enhanced in [20] by using an input-vulnerability approach to reduce the search space and in turn make the scheme further scalable. Reference [24] tackled the problem of scalability by focusing on gate approximations instead of approximating a full-fledged combinational logic via searching through Multi-Objective Optimization Genetic Algorithm (MOOGA) [25], [26]. However, the scheme can take a few weeks for the generation of ATMR for circuits consisting of a few hundred gates. Thus, [27] involved heuristics based on testability and observability whose solutions required a minute. As the benchmarks of [27] were small, [28] modified the approach to tackle circuit with 2670 gates in 1 hour and circuit with 1775 gates in 10 hours.

F. PLATFORM INDEPENDENCE

The approximation techniques used in ASIC attempt to reduce circuit size and maximize masking of the output error. In FPGAs, the circuit logic gates are implemented in look-up tables (LUTs). An FPGA LUT is characterized by several inputs and one output. It functions like a block of memory indexed by the inputs of the LUT. The output is the value stored in the location of LUT indexed by the combination of inputs. The LUTs required for implementing the given circuit is dependent on the number of inputs and not on the number of gates. Reducing the circuit size in terms of logic gates may not reduce FPGA implementation area in terms of LUTs [23]. In FPGAs, ATMR can be implemented with lower cost by using FPGA resources that are not in use in the target design. Moreover, an ATMR implementation on FPGA can adapt explicit area overhead and reliability rates where the reconfiguration ability allows the dynamic tuning of the redundancy degree in accordance with the expected error rate and power budget. However, the peculiarities of the FPGA architecture should be considered for optimal approximation, which involves imposing limitations on the faults to be approximated. It ensures that logic transformations that decrease the number of LUTs or the LUT size of approximate logics are allowed [22].

As some ATMR techniques are platform-restricted, careful design tactics must be adopted for diverse platforms such as ASIC and FPGA. Platform independence is essential. For instance, [19] is appropriate for programmable systems owing to need for under-/over-approximations.

IV. APPROXIMATE-COMPUTING-BASED TMR

Reference [13] introduced the use of approximate logic circuits for TMR by using testability estimations. Algorithmically, the circuit is transformed to unate form, then the line testability is estimated using stuck-at fault simulation, and over-/under-approximate circuits are generated using testability estimations. Stuck-at Fault for a Boolean function is an abstract fault model. A logic stuck-at 1 means it produces 1, a logical error (logical 1), when the line is applied a logic 0. A logic stuck-at 0 means it produces 0, a logical error (logical 0), when the line is applied a logic 1. Fault simulation is typically used to evaluate the fault coverage obtained by set of test patterns capable of detecting many faults in a circuit. Automatic test pattern generation (ATPG) find a set of test patterns that detects all faults considered for that circuit. The methodology does not require a specific synthesis tool and need logic transformations that expose more low testability lines.

A. BOOLEAN-FACTORING-ALGORITHM-BASED ATMR

1) ATMR USING APPROXIMATE CIRCUIT, TRANSISTOR TOPOLOGY, AND INPUT PERMUTATION APPROACHES

TMR can benefit from approximated circuits to generate redundant modules as in [13]. Reference [14] coined the term ATMR. The validity of the working principle of ATMR is set by restricting the generation of over-/under-approximation functions with the following restriction: $F \subseteq G \subseteq H$. Subsequently, the ATMR approach utilizes the knowledge of unprotected vectors to reduce sensitive nodes in the transistor level of design by performing two customizations as given in Fig. 3: 1) input reordering: an alteration in the transistor inputs without altering the topology or logic function; 2) transistor reordering: an alteration in the circuit topology circuit without changing the logic. The final stage of ATMR combines the best circuits thus determined.

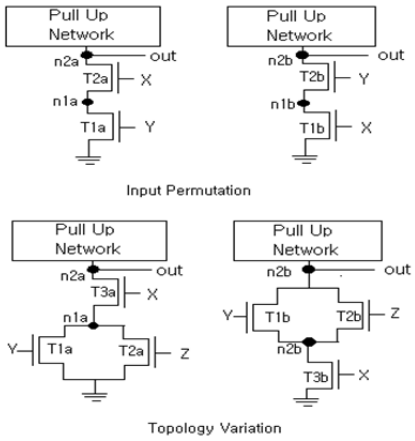


FIGURE 3. Structural reordering [14].

Advantage: Transistor-level approach for node sensitivity

Limitation: Necessitates analysis of a large variety of circuits

2) ATMR USING BOOLEAN FACTORING AND STRUCTURAL REORDERING

In [17], the technique of area overhead reduction of [14] is further analyzed with approximate modules by using a different methodology for computing approximate functions and the choice of best combination of approximate circuits. A slightly modified form of the Boolean factoring algorithm has been used for generating approximate candidates as shown in Table 3. The Boolean factoring algorithm is based on functional composition paradigm. First, the Boolean function is decomposed into factors, which are then combined using an order concept that classifies functions as under-/over-approximations. Cofactors and cube factors are combined for obtaining the allowed function set.

Variables of correct polarity are stored in a bucket using bonded-pair representation where new bonded-pairs are formed using existing simpler bonded-pairs to find the target logic with fewer literals. Upon the extraction of candidates, the structural reordering of [14] is used.

Advantage: The approximation algorithm separates over- and under-approximations.

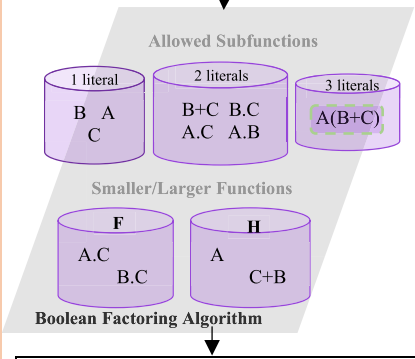
Limitation: Necessitates analysis of large variety of circuits

TABLE 3. ATMR Techniques-Example cases.

BOOLEAN-FACTORING-ALGORITHM-BASED ATMR

ATMR using Boolean factoring and Structural Reordering [17]

G=A*B+A*C

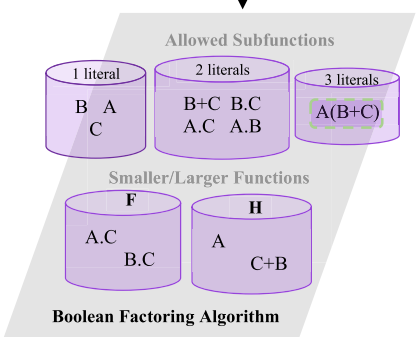


Approximation Candidate Analysis			
Approx	Function	Literal Count	Unprotected
F	AC	2	1
	BC	2	1
H	C+B	2	3
	A	1	1

ATMR Composition
G/F=AC/H=A (2 unprotected)

FATMR for reducing area overhead in TMR [18]

G=A*B+A*C



		Candidate Analysis			
ABC	G	H1=A	F1=AB	F2=AC	H2=C+B
000	0	0	0	0	0
001	0	0	0	0	1
010	0	0	0	0	1
011	0	0	0	0	1
100	0	1	0	0	0
101	1	1	0	1	1
110	1	1	1	0	1
11`	1	1	1	1	1

FATMR Composition
H1/F1/F2 (3 unprotected)

Exploring ATMR/FATMR with low area overhead [11]

G=(a'+b')((c'b)+((a+e)((a.(c+e))+d)))

FC-ATMR Tool

Approximate functions

$F_1 = \bar{a}b\bar{c}$
$F_2 = a\bar{b}\,(d+e)$
$F_3 = a\bar{b}\,d + \bar{a}e$
$F_4 = (\bar{a}+\bar{b})(b\bar{c}+de)$
$F_5 = (\bar{a}+\bar{b})\,((a+e)(a(c+e))+d)$
$H_1 = \bar{a}+\bar{b}$
$H_2 = \bar{b}+(\bar{a}(\bar{c}+e))$
$H_3 = a\bar{b}+(\bar{a}(\bar{c}+e))$
$H_4 = a\bar{b}+(\bar{a}(\bar{c}+ed))$
$H_5 = de+(a\bar{b}+(\bar{a}\,b\,\bar{c}))$
$H_6 = d\bar{a}e+a\bar{b}+\bar{c}\bar{a}b$

Study large variety of possible circuits

ATMR Compositions
G/F5/H4 (9 unprotected)
G/F5/H2 (13 unprotected)
F1/F5/*H6* (16 unprotected)

PI EXPANSION/REDUCTION-BASED ATMR

Systematic ATMR using PI expansion/reduction through Heuristics [9]

G=(a'+b')((c'b)+((a+e)((a.(c+e))+d)))

PI Expansion/Reduction

Approximate functions

$f_1 = b\bar{c}(\bar{a}+\bar{d}\,\bar{e})+\bar{b}(ac+de)$
$f_2 = \bar{a}(b\bar{c}+de)+a\bar{b}$
$f_3 = \bar{a}b\,(\bar{c}+e)+d(a\bar{b}+\bar{a}e)$
$f_4 = a\bar{b}(c+e)+d(\bar{a}+\bar{b})$
$f_5 = \bar{a}(b\bar{c}+e)+\bar{b}(a\,c+e)$
$f_6 = a\bar{b}$
$f_7 = \bar{a}b+\bar{b}\,(ac+de)$
$f_8 = \bar{a}b(\bar{c}+d)+de$
$f_9 = de$
$f_{10} = \bar{a}b+\bar{b}\,(d+a\,(c+e))$
$f_{11} = \bar{a}\,b\,\bar{c}+\bar{b}\,(ac+d)$
$f_{12} = a\bar{b}+b(\bar{a}+cde)$

Blocking property and literal-sort candidates

ATMR Compositions
$f_4/f_2/f_5$ (9 unprotected)
$f_6/f_7/f_8$ (13 unprotected)
$f_9/f_{10}/f_2$ (16 unprotected)
$f_9/f_{11}/f_{12}$ (18 unprotected)

Input vulnerability-aware ATMR [20]

G=(a'+b')((c'b)+((a+e)((a.(c+e))+d)

Vulnerable Input Vectors
Stuck-at-0:
….
Stuck-at 1:
…..

Pre-Blocking

PI Expansion/Reduction for Input-Vulnerability aware Approximate functions

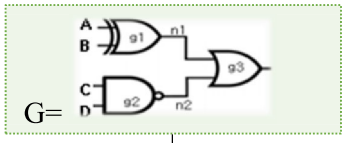
Blocking property and literal-sort candidates

ATMR Compositions
….
….

TABLE 3. *(Continued.)* **ATMR Techniques-Example cases.**

GATE-APPROXIMATION-LIBRARY-BASED ATMR

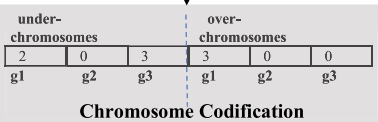
ATMR using ApxLib and MOO [24]



APXLIB

Possible Gate Approximations

Gates	Under-Apx	Over-Apx
g1	Const-0 [cod : 1] NOi21 (v1) [cod: 2] NOi21 (c2) [cod: 3]	Const-1 [cod : 1] OR2 [cod:2] NAND2 [cod: 3]
g2	Const-0 [cod : 1] Inverter (i1)[cod: 2] Inverter (i2)[cod: 3]	Const-1 [cod : 1]
g3	Const-0[cod : 1] Line/Buffer (i1) [cod: 2] Line/Buffer (i2) [cod: 3]	Const-1 [cod : 1]

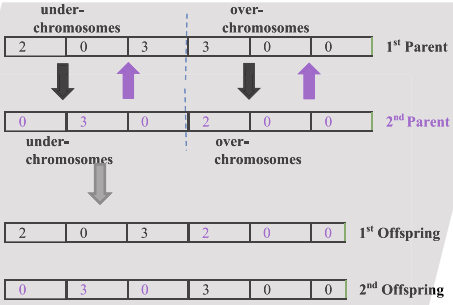


Chromosome Codification

ATMR Composition

F based on Under-chromosomes:
g1 = (A'+B)', g3= Buffer
F= g3 (g1)

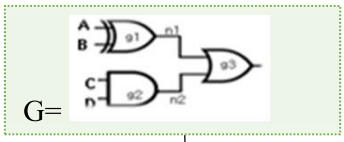
H based on Over-Chromosomes:
g1=(AB)', g2=(CD)', g3=OR
H= g3(g1, g2)



Crossover Mechanism

New ATMR Composition

ATMR using ApxLib and testability [27]



Candidate Gates Evaluation Using Testability and Observability

ABCD	Original output	Stuck at 0		Stuck at 1	
		g1	g2	g1	g2
0000	1	1	**0**	1	1
0001	1	1	**0**	1	1
0010	1	1	**0**	1	1
0011	0	0	0	**1**	**1**
0100	1	1	1	1	1
0101	1	1	1	1	1
0110	1	1	1	1	1
0111	1	**0**	1	1	1
1000	1	1	1	1	1
1001	1	1	1	1	1
1010	1	1	1	1	1
1011	1	1	1	1	1
1100	1	1	**0**	1	1
1101	1	1	**0**	1	1
1110	1	1	**0**	1	1
1111	0	0	0	**1**	**1**

Testability and Observability measures

Gate	Influence measure	Type of Approx
g1	6.25%	Under-approx
g1	12.5%	Over-approx
g2	12.5%	Over-approx
g3	12.5	Over-approx
g2	37.5%	Under-approx
g3	87.5%	Under-approx

Different Bit measurements

g1(suitable for over/under approximation), g2 and g2 (suitable for over-approximation)

Candidate approximations (APIXLIB) Evaluation

For instance:
g1 → underapproximation → Constant 0
……
……

Approximation Selection for ATMR Composition

3) FATMR FOR REDUCING AREA OVERHEAD IN TMR

The TMR approximation technique of FATMR was devised in [18] as depicted by example of Table 3. The approximations were obtained using a Boolean factoring algorithm with an order concept. The number of literals was used to evaluate approximate candidate list. Subsequently, a structural reordering tool was used to investigate masking of ATMR design and highlight sensitive circuit nodes and unprotected vectors. Thus, the best and worst structured circuits were determined.

Advantage: Possibility of drastic area reduction

Limitation: Requires judicious choice for ATMR/FATMR

4) EXPLORING ATMR/FATMR WITH LOW AREA OVERHEAD

In [11], which was an enhancement of [18], the innovation originated from the algorithm for searching for the best-case set-ups of approximate logics. In accordance with the literal count for G, various valid combinations are possible. Hence, a tool to generate the ATMR/FATMR circuit automatically and introduce faults at the logical and transistor levels was developed to obtain ATMR designs formed by various F and H functions and ATMR formed by two F functions and one H function or two H functions and one F function in accordance with the literal count for G. The tool acquires the set of ATMR/FATMR, automatically maps these structures to a gate netlist, and later generates a transistor-level design for evaluating the fault masking ability of each ATMR and FATMR structure developed and determining the amount of unprotected p–n junctions of ATMR. Notably, the method of spreading the approximate modules significantly affects overhead reduction as illustrated by example in Table 3.

Advantage: Manipulates circuits with up to eight inputs

Limitation: No systematic approach for finding the best combination

B. PI EXPANSION/REDUCTION-BASED ATMR

1) SYSTEMATIC ATMR USING PI EXPANSION/REDUCTION

Reference [12] searches for the best approximate TMR structure such that the error rate is within an acceptable threshold and the area overhead is minimum. The technique of PI expansion and reduction is used as shown in Fig. 4. Initially, the minterm/maxterm to expand/reduce original PI (MEROP) list is generated. This list is then shortened in accordance with the allowed complements, the approximate function is generated, and the previously complemented MEROPs are blocked to maintain the validity of the principle of ATMR. It should be noted that the block state reflects the input vectors for which the output can no longer be complemented, and which must have the same value as the original function. Iteratively, three approximate functions are generated where a locally optimal selection is made at every phase to obtain the global optimum at the final phase. It was observed that the circuit inputs and literal count reduction have a direct proportionality relation with the equivalent error rate threshold, as the same error rate threshold allows an exponentially greater number

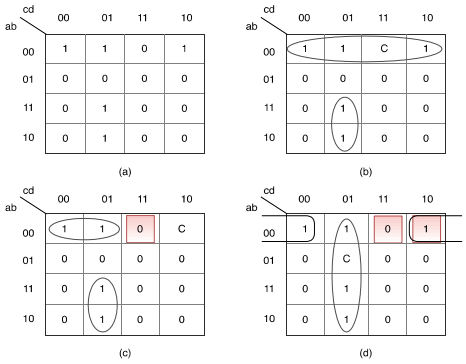


FIGURE 4. Approximations using PI expansion/reduction: a) Original; b) Approximation 1; c) Approximation 2; d) Approximation 3 [9].

of complements for a logic with a higher number of inputs. The method is exhaustive but yields the best approximations due to an extensive search space.

Advantage: Systematic approach

Limitation: Exhaustive search space

2) SYSTEMATIC ATMR USING PI EXPANSION/REDUCTION THROUGH HEURISTICS

Reference [9] furthers the work of [12] to target circuits having a higher number of input vectors and investigates fault injection. Heuristics were proposed to obtain three approximate modules of ATMR successively using the logic optimization of PI reduction and expansion as shown in example of Table 3. The aim is to obtain approximate logic successively for modular redundancy by enhancing the methodology given in [12].

A MEROP list with the number of literals sorted is generated, which contains the minimized approximate functions and the complemented MEROP location. The good-enough approximations are at the top of the list and the remaining functions are discarded. Subsequently, until the allowed number of complements is achieved, a complement in the list members is repetitively introduced and processed for generating the approximate function. As the locations of complemented MEROPs are different from each other and the literal count is the lowest, the combinations of approximate modules are chosen.

The clustering technique of intermediate circuit representation with the internal nodes of Boolean circuits having 10–15 inputs was utilized for circuits with greater than 15 inputs.

Advantage: Can handle up to 14 inputs

Limitation: Vulnerable to inputs

3) INPUT VULNERABILITY-AWARE ATMR

The concept of input vulnerability for ATMR was proposed in [20] by recognizing the critical input space using ATPG and restricting the vulnerable input space as unavailable for the technique of ATMR based on PI reduction expansion. The entire input space does not have the same level of importance. Rather, a portion of the input space can be more vulnerable to

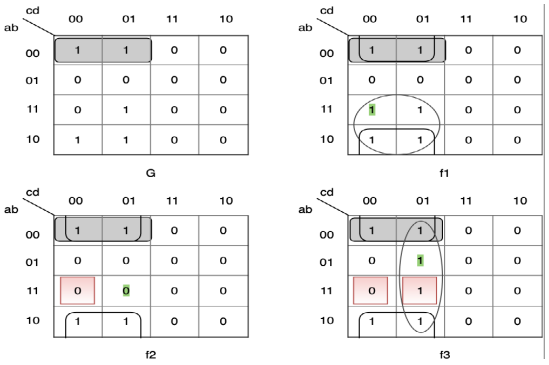


FIGURE 5. Input vulnerability aware ATMR [20].

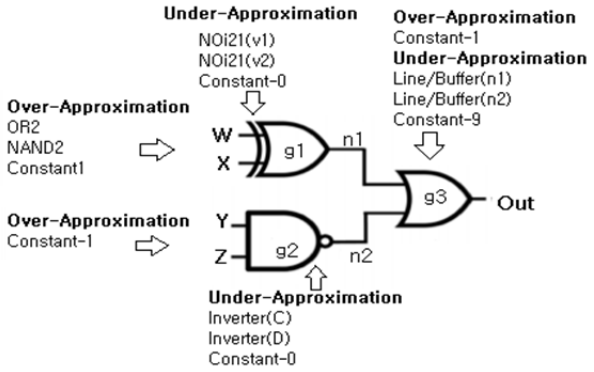


FIGURE 6. Approximate library technique [24].

errors than the rest, as the error can propagate to the output. It was highlighted that ATMR is more sensitive to errors as compared with TMR, and hence, approximations at critical inputs need directed attention. The method was based on two stages, namely, the pre-blocking stage and the approximate module extraction and selection stage (AMES) as presented in Figure. 5. Pre-blocking guarantees that ATMR generation will be input-vulnerability aware. Through the ATPG tool, the input vectors whose output is vulnerable to errors are identified and are made unavailable for the approximation process. The AMES is based upon [9]. In addition to the heuristics of [9], the pre-blocking aids in a further reduction of the search space for approximation candidates. For example, the Karnaugh Map in Fig. 5(a) presents the original circuit where a'b'c'd' and a'b'c'd are vulnerable inputs and will not be approximated. For generation of first approximate circuit f1, abc'd' has been complemented. For generation of second approximate circuit f2, abc'd' has been blocked and abcd' has complemented. Finally, for the third approximate circuit, abc'd' and abc'd have been blocked and a'bc'd has been complemented. It was reported that the ATMR formed consisted of sufficiently good approximate functions with higher fault coverage while overcoming the previous limitation of exhaustion of search space. The method is illustrated in Table 3.

Advantage: Input vulnerability-aware

Limitation: Needs prior knowledge of circuit

C. GATE-APPROXIMATION-LIBRARY-BASED ATMR

1) ATMR USING APXLIB AND MOO

In [24], instead of using functions for the extraction of approximations, the approximate library technique generates approximations through the replacement of gates using a predefined library. For optimization, genetic Algorithms (GA) with Multi-Objective Optimization, (MOO) is known as MOOGA, is used. The genetic algorithm performs blind search within all possible solution space and then with the MOO sorting, the best configurations are selected. The MOO sorting algorithms are a good approach to optimize a problem which has interdependent metrics. This combination of techniques allows the evaluation of a larger number of ATMR circuits. Fig. 6 presents a gate netlist for approximations. Through testability and observability, the XOR2 gate is recognized to have lesser significance in the circuit, which indicates that it is a good gate for approximation.

Every gene value reflects the approximation by which a given gate is altered. These alterations are called mutations. Through the usage of the mutation scheme with ApxLib, invalid individuals are rarely generated. In the genetic algorithm, by using a crossover, two individuals (genotypes) are chosen from the population where some portion of these individuals is exchanged for the formation of a new individual. Thus, a new ATMR was created by exchanging F and H among the individuals as shown in example of Table 3. NSGA-II which is MOOGA [27] performs evaluation of individuals based upon the diversity within the dimension of every objective for the optimization of two metrics entirely. The metrics under consideration were the circuit size and fault coverage, which needed to be minimized and maximized, respectively. The analysis of the circuit configures the chromosome and the approximations for every gate, resulting in first-generation individuals. Iteratively, the next population is generated, and evaluation is performed to verify the correctness of the approximation, its size, and fault masking ability. Furthermore, dynamic genes are evaluated for their usefulness in the circuit for reducing the computational effort. Subsequently, MOO-sorting is performed for the selection of the best circuits for the subsequent stage. Conclusively, crossover and mutation are performed with the best circuits of the population. The steps are repeated for hundred cycles of generation [24].

Advantage: Does not require prior explicit knowledge of circuit

Limitation: Takes hours for computation

2) ATMR USING APXLIB AND TESTABILITY

Given the computational exhaustiveness of [24], the results of [27] is an improvement over it. The work proposed the use of the approximate library technique with the heuristics of testability and observability to build approximations in a faster and deterministic way.

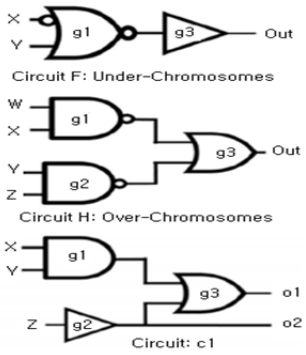


FIGURE 7. Testability and Observability Analysis [27].

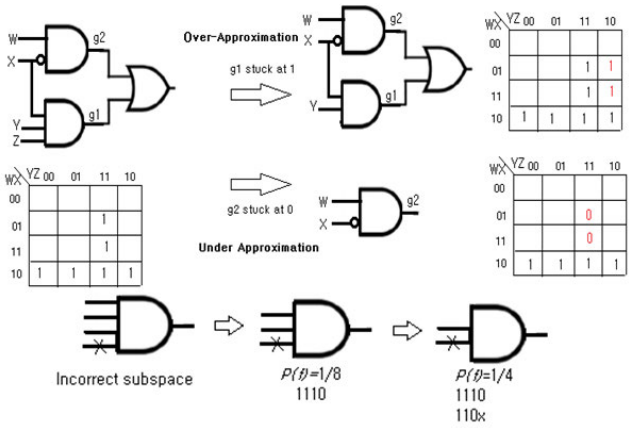


FIGURE 8. Probabilistic and evolutionary method [29].

Once a gate is reflected as the best approximation candidate, the heuristic chooses the most appropriate gate transformation. Through the testability and observability schemes, the impact of each gate on the outputs of the circuit is evaluated as given in Fig. 7. and illustrated though example of Table 3. A stuck-at fault in the gate output has less testability, if only a few input vectors can be used for fault testing. The gate modification with the lowest testability leads to a good approximation circuit. After the gate selection, the approximation to be used should be chosen. The gate transformation is performed through an evaluation of the percentage of output bits that alter in comparison with the output bits of the original circuit via exhaustive analysis.

Advantage: Takes minutes

Limitation: Intended for smaller circuits

3) ATMR CIRCUITS USING APXLIB AND HEURISTIC FOR LARGER CIRCUITS

Reference [28] furthers the work of [27] for larger circuits through a deterministic approach. This work involves the evaluation of gate parity to attain functional ATMR.

Initially, a table of testability and observability measurements is generated. From this table, the best candidate gate available on the first line of the table is evaluated for possible gate transformations. These measurements are recomputed when: 1) a certain number of gate approximations are performed or 2) the initial gate measures had been utilized for candidate evaluation. This procedure continues until 1) the table is empty or 2) a certain number of approximations are performed. Once specific criteria are satisfied, a new table is generated based on testability and observability analyses. The process continues until no more approximations exist. To tackle larger circuits, the ATMR is developed for every new generated table.

Advantage: Applicable to larger circuits

Limitation: Greedy approach

D. PROBABILISTIC AND EVOLUTIONARY APPROACHES FOR ATMR

Reference [29] proposed a dynamic-probability-estimation-based approach, which greedily operates for the removal of logic with the minimum probability of error generation by a coupling of the approximation technique and the error estimation technique through stuck-at faults as shown in Fig. 8. For computing probabilities, a set of mandatory assignments (SMA) of the approximated line is maintained such that the probability of incorrect subspaces for the set of faults approximated to acquire F and H is obtained as the probability of union of the SMAs. In the case of a new approximation, the new SMA is included in the set. At initialization, F and H are exactly G, having a total error probability (EP) of zero and an estimated area (EA) as for TMR. The initial SMA for each fault and the fault probabilities are set. Subsequently, in every iteration of the approximation loop, for approximation, the fault having the minimum probability is chosen, which can yield an under-approximation or over-approximation and is added to AF (set of faults approximated to acquire F), or AH (set of faults approximated to acquire H), respectively. Subsequently, the total probability estimation is updated and new fault probabilities for all the probable faults are computed. Finally, approximation is performed. The SMAs of all the faults in the approximation are updated and possible redundancies created by the approximation are eliminated. The procedure is repeated to reach the EP and EA targets.

Reference [29] further proposed an evolutionary approach for obtaining ATMR solutions that are hard to find with other methods. The method is based on Cartesian genetic programming (CGP) where the circuit is presented as a fixed-sized Cartesian grid of nr $\times$ nc nodes interconnected using a feed-forward network. Genotypes are integer arrays which represent the circuit and phenotypes are the circuits established according to the genotypes. The phenotypes are evaluated by a fitness function. In a genotype, every two-input node is coded with three integers (an address for the first input; an address for the second input; a node function). A CGP is constructed either randomly or by mapping a known solution to the CGP chromosome. In every generation, the best individual is passed on unaltered to the next generation along with its λ offspring individuals generated by a point mutation operator. If multiple individuals with the best fitness

exist, one is randomly selected. The mutation rate m is usually set to modify up to 5% randomly selected genes. The role of mutation is significant in CGP. A multi-stage single-objective approach with constrained resources to obtain the desired approximations is used.

Advantage: Probabilistic - results closer to evolutionary Evolutionary - able to reach global minima

Limitation: Probabilistic - Greedy approach Evolutionary - Higher computational effort

E. ATMR BASED ON DATA SIZE AND PRECISION REDUCTION

1) ATMR WITH DATA PRECISION REDUCTION

In [21], the authors introduced the concept of precision reduction for implementing ATMR. This method involved the use of fewer bits for floating point representations. The proposed method was applied to two benchmarks and a multitude of ATMR designs with different degrees of approximation.

Advantage: For programmable hardware and software

Limitation: Output accuracy highly dependent on the nature of data and application

2) ATMR BASED ON SUCCESSIVE APPROXIMATION AND LOOP PERFORATION IN MICROPROCESSORS

The ATMR technique proposed in [30] is not restricted by mathematical constraints and utilizes the notion of approximation intensity and successive approximation. Through successive approximation algorithms, with each loop, output quality is iteratively improved. For instance, the Newton–Raphson benchmark successively approximates to compute the roots of the function. This algorithm is used as a benchmark application in [30].

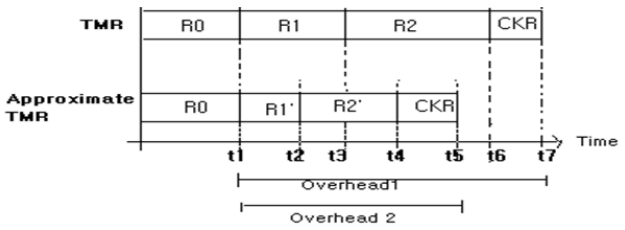


FIGURE 9. ATMR using successive approximation [30].

The ATMR technique of [30] is shown in Fig. 9 where R0 is the original task. Here, R1 and R2 are the exact copies of R0 and R1′ and R2′ are redundant tasks of R0 using fewer iterations. Notably, the TMR overhead includes the additional execution time taken for the execution of the R1 and R2 tasks and the checker (CKR). The tasks using fewer iterations (lesser accuracy) are executed faster. Hence, by decreasing the task execution time, the TMR overhead is decreased. Furthermore, variable data-size reduction can be performed at the software level. The type of variable defines its reading, writing, and arithmetic/Boolean-related operations and their bit size. These factors are correlated to the system performance, which includes energy consumption and fault tolerance. Floating-point data types are naturally approximated,

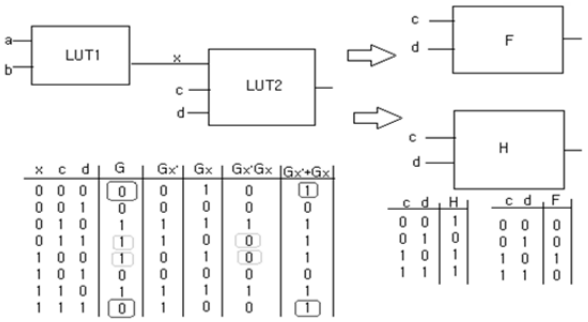


FIGURE 10. Approximation for LUT [19].

as they work by the division of the value into an exponent and significand (mantissa). Hence, the work also evaluated the effect of various data types and their sizes on the Newton–Raphson benchmark protected by their ATMR technique.

Advantage: Based on concept of approximation intensity

Limitation: Essential to find an optimal point, as more iterations can make the system less fault-tolerant

F. ATMR FOR FPGA

1) INEXACT TMR FOR FPGAS

The proposed heuristic/algorithm in [23] receives as inputs the logic circuit G in the BDD format and the variation Δ. First, the functions provided by the CUDD framework [23] are used to compute the maximum number of BDD nodes (gNodes), the number of inputs, and the area in terms of LUTs used by the circuit G after it is synthesized and mapped. Second, both inexact circuits and the sequence of nodes (1,, gNodes) with an interval variation given by $d\Delta/100 \times$ gNodes are synthesized and mapped independently. Third, all the obtained designs for F and H in the previous section are combined. All the combinations that return unique area overhead numbers are stored. For the combinations that have the same area, only the one that maximizes the number of minterms is stored. At the end of this step, the best inexact designs for the range of area overhead factor (0, 2] are computed. These designs are called the key designs. Finally, the fourth step selects from among the key designs the one that maximizes the soft error rate (SER) reduction. Subsequently, the corresponding F and H designs are returned. Notably, the proposed heuristic/algorithm can be easily changed to return the inexact circuit that maximizes the masking factor, according to a given target area overhead factor or, on the other hand, the inexact circuit that minimizes the area overhead, according to a given target masking factor.

Advantage: Scalable heuristic independent of circuit size

Limitation: FPGA-specific methodology

2) PARTIAL TMR IN FPGAS USING APPROXIMATE CIRCUITS

In [19], the line substitution approach for FPGA is adapted based on the netlist structure focusing on the reduction, merging, or elimination of LUTs, which in turn can decrease the interconnection requirements and related configuration

TABLE 4. Design issues targeted by existing ATMR.

ATMR Technique [Ref No]	Level	Design Issues			
		Scalable	*Error Proneness*	*Selection of combination*	*Platform independency*
Unate functions governed by testability [13]	Hardware	Yes	Yes	No	Yes
Transistor topology and input permutations [14]	Hardware	No	Yes	No	No
Selection of best combination of approximate circuits using complex gates and structural reordering [17]	Hardware	No	Yes	No	Yes
Only approximate modules -Boolean factorization [18]	Hardware	No	Yes	No	Yes
Approximation for all modules with reduced area overhead [11]	Hardware	No	Yes	No	Yes
Approximate modules for SRAM-based FPGAs [23]	Hardware	Yes	No	No	No
Line substitution for partial TMR of FPGA [19]	Hardware	Yes	Yes	Yes	No
Successive PI expansion/reduction for best combination [12]	Hardware	No	No	Yes	Yes
Probabilistic error estimation of error and evolutionary approach [29]	Hardware	Yes	Yes	Yes	Yes
Approximating K cores of multicore processor [31]	Hard/Software	Yes	Yes	n/a	Yes
Approximate modules from APXLIB using MOOGA [24]	Hardware	No	Yes	Yes	Yes
Approximate modules from APXLIB through gate testability and observability measures [27]	Hardware	No	Yes	Yes	Yes
Testability measures for approximation in FPGA [22]	Hardware	Yes	Yes	No	No
PI expansion-reduction: literal sort [9]	Hardware	Yes	No	Yes	Yes
Input-vulnerability Aware successive PI expansion and reduction [20]	Hardware	Yes	Yes	Yes	Yes
Data size/precision reduction for microprocessor[30]	Hard/Software	Yes	No	n/a	Yes
Apxlib with testability and observability as heuristics [28]	Hardware	Yes	Yes	Yes	Yes
Precision reduction [21]	Hard/Software	Yes	No	n/a	Yes
Fault-tolerant voter for ATMR [16]	Hardware	n/a	Yes	n/a	n/a

bits. Substitution of the input or output of the flip-flop by a constant lead to the removal of flip-flops and a simplification of voting logic. For the removal of an LUT, the output of the LUT can be set to a logic constant. Fig. 10 presents an example where, for the removal of LUT1, LUT2 is approximated on x. Transformation of approximation and cofactors can be computed with the truth table of LUT2, as shown in Fig. 10.

First, the logic representation of the circuit is acquired, and the stuck-at fault simulation is thus conducted. The fault simulation output is utilized as the criterion for the estimation of the approximation criticality and the selection of the less critical portions. Subsequently, the circuit is decomposed into the combinational block, consisting of LUTs, and the sequential blocks (FFs). An underapproximation and over-approximation circuit is generated using the combinational block based on the previous criticality estimation. However, hardening with TMR is performed for the sequential components. Finally, the components are merged, and the circuit is resynthesized to optimize the approximation logic.

Advantage: Targets critical input space

Limitation: FATMR is not possible.

3) ATMR FOR FPGAS BASED ON TESTABILITY ANALYSIS

Reference [22] emphasizes that overheads caused by the ATMR technique can be adjusted through unused resources of the FPGA and FPGA reconfiguration capabilities can be utilized to tune dynamically the tradeoff between resource utilization and the error masking rate requirements, which may differ in accordance with the actual error rate to satisfy the target reliability constraints.

Advantage: scalable, fine granularity and flexible balance between area overhead and reliability

Limitation: FATMR is not possible.

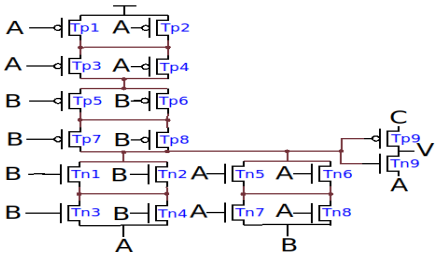


FIGURE 11. Fault-tolerant voter for ATMR [16].

Apart from these discussed methodologies, as the voter is an integral part of TMR, other studies on AC for TMR have considered a fault-tolerant voter. Reference [16] highlights that, in case of ATMR, a faulty voter is a direct threat to system reliability. Consider the ATMR of Table 2 and assume that the modules forming ATMR are fault-free. In contrast to TMR, the fault-free conditions for ATMR are 000, 001, 010, 100, 011, 101, 110, and 111, as ATMR allows one of the approximations to diverge from the original circuit at each input vector. Hence, cases will exist where two inputs of the voter will match, and one input will differ. For the circuit input vector 010, in the non-faulty voter state, f2 and f3 vote to forward a value of 1 to the output, although f1 generates a value of 0. At the occurrence of faults at the voter input fed from f2, 1 is flipped to 0, and f1 and f2 will vote an erroneous output of 0. Hence, ATMR must be tolerant to (a) faults at the voter circuit nodes (internal outputs) and (b) faults inverting the voter inputs. Reference [16] uses a fault-tolerant voter for ATMR by using pass transistor logic (PTL) to keep transistors and possible faulty nodes to a minimum. As a minimum number of transistors were used, transistor-level redundancy

TABLE 5. Comparison Of ATMR methodologies.

Key idea	Forte, Keywords (Reference #,Year)	Bench mark	Plat form	Evaluation Metric	Fault Injection tool	Synthesis/ Simulation Tool	Claim
The concept of approximate logic circuits for TMR pursuing an optimal balance between fault masking and overheads is introduced by invoking the notion of unate functions governed by testability estimations	Unate functions, testability estimation, Single-Event Transient,Soft error, Error detection and correction, testability ([13], 2012)	LGSynth93	ASIC	Area overhead	AMUSE system	Synopsis SAED 90 nm / HOPE	Tradeoffs between error coverage and area overhead
Explores the approximate modules for TMR through the usage of complex gates in conjunction with technique of employing different transistor topologies and input permutations	Complex gates, transistor topologies, permutation, TMR, Ampproxiated circuit, Transient Faults ([14],2013)	4/6-input logic function	ASIC	Protected p–n junction ratio, area overhead	NGSPICE	NGSPICE	Area overhead reduction 150% Fault coverage 99%
Targets the selection of the best combination of approximate circuits for TMR with high fault coverage while using complex gates and explores structural reordering	Selection of best combination, TMR, Approximated circuit, Transient Faults ([17], 2014)	6- and 8-input Boolean function	ASIC	Protected p–n junction ratio, area overhead	C++ based tool integrated with ModelSim	ModelSim	Area overhead reduction 120%–200% Fault coverage 95%
Introduces the notion of only approximate logic for the three modules of TMR based on the Boolean factorization method	Only approximate modules, Boolean factoring, TMR; Approximated circuit; Transient Faults **([18],2015)**	6-input logic function	ASIC	Protected vectors/p-n junction, area overhead, best/worst structure	C++ based tool integrated with ModelSim	ModelSim	Area overhead reduction 120%–200% Fault coverage 95%
Uses approximate logic for all the three modules to construct the TMR to reduce the area overhead close to a minimal value	Boolean factorization, Selection of best composition TMR; Approximated circuit; Transient Faults **([11], 2015)**	5-input function, 4-bit ripple adder	ASIC	Unprotected vectors/total vectors, unprotected p–n junctions, masking coverage,transistors , area overhead	Python integrated with ModelSim	FC-ATMR tool, Python-based ABC Integration	**ATMR:** Area overhead165% protected p–n junction 98.8% **FATMR** protected p–n junction 94.6% area overhead 88%
Introduces fault tolerance for SRAM-based FPGAs through approximate modules in conjunction with TMR	Binary decision diagram, Field programmable gate arrays, Tunneling magnetoresistance, Table lookup, Logic gates, Circuit faults, Logic circuits, Boolean functions ([23], 2016])	LGSynth93 (alu4, clip, duke2, apex4, cps, pdc, seq)	FPGA	Search space, masking factor, area overhead factor, SER reduction	-	ABC synthesis tool, CUDD framework, C++	Computation time reduction 84.4%
Uses the line substitution technique of approximation for the provision of partial TMR in FPGA by replacing some lines with logic constants	Line substitution, Approximation methods, Tunneling magnet-oresistance, Circuit faults, Field programmable gate arrays, Logic circuits, Logic functions, Table lookup ([19],2016)	B13 (ITC'99)	FPGA	Radiation test, Mean time to failure,	HOPE	Artix 7 Xilinx	Resource reduction 15%
Introduced the method of PI expansion and reduction driven by an acceptable error rate threshold to obtain the best combination of the three approximate modules for ATMR successively	Systematic methodology, PI expansion and reduction, Approximate TMR, Fault tolerance, Approximate circuit ([12], 2016)	5–6-input combinational circuit	ASIC	Acceptable error number of literals	-	Python EDA	Up to 44% literal reduction with 14% unprotected vectors
Approximate logic circuit generation technique for TMR using 1) a greedy approach of probabilistic estimation of the error and 2) an evolutionary approach	Probabilistic and evolutionary approach, Approximate logic circuit, error mitigation, evolutionary computing, single-event transient (SET), Single-event upset (SEU) ([29], 2016)	LGSynth93 (b12, rd73, t481, apex3, apex4, m3, misex3, table3, table 5)	ASIC	Error probability, area overhead, masking rate, Hamming distance	HOPE	Synopsis NanGate 15nm	Improved scalability

TABLE 5. *(Continued.)* **Comparison Of ATMR methodologies.**

Introduces N-modular redundancy (NMR) construction software for multicore processor by use of approximate computing concept for some of the cores	Exact and approximate versions of task, different cores, Multicore processors, reliability, approximate computing, real-time embedded systems, N-modular redundancy ([31], 2017)	AxBench and Mibench	Softwar e-based	Normalized energy, normalized execution time	-	ESESC Cycle-accurate multicore processor	Energy reduction 35% Area reduction 40%
Forms ATMR by substituting its modules from a library of gate approximations (ApxLib) based on multi-objective optimization genetic algorithm (MOOGA)	Gate approximation, multi-objective optimization, ATMR, Approximate Circuits, Fault Tolerance, Genetic Algorithm, MultiObjective Optimization ([24], 2018)	LGSynth93 (majority, clpl, cm82a, newtag rd73)	ASIC	Error masking rate, area overhead	Custom-made Python simulator	ABC synthesis tool	Area overhead 176% with 9.6% error rate
Formulates ATMR by substituting its modules from a library of gate approximations (ApxLib) defined by gate testability and observability measures	Gate approximations, gate testability and observability, ATMR Approximate circuits Fault tolerance, Genetic algorithm ([27], 2018)	LGSynth93 (majority, clpl, cm82a, rd73)	ASIC	Execution time, area overhead, error rate	Custom-made Python simulator	ABC synthesis tool	Area overhead up to 100% with 41% error rate
Technique for ATMR in FPGA using testability measures for the generation of approximate logic circuits	Testability, Approximate logic circuit, selective error mitigation, FPGA, Single-Event Upset, Triple Modular Redundancy ([22], 2018)	B13 (ITC'99)	FPGA	Fault Index, Critical bits, LUTs, FFs	HOPE, CAN facilities	Artix 7 Xilinx	Critical bits reduction 58% Combinational area reduction 12%
Systematic ATMR using PI expansion/reduction with a heuristic of literal-sorted approximate candidates	Systematic, PI expansion and reduction, literal sorted, Approximate TMR, Fault tolerance, Approximate computing, Reliable computing ([9], 2018)	Rd73, Z9sym, Sym10, and Co14	ASIC/F PGA	Allowed complements, Transistors/literals, unprotected vectors/ total vectors % of unprotected input vectors, fault coverage	HOPE	MATLAB PyEDA Espresso logic synthesis tool NanGate 45 nm library	Transistor count reduction 26.1% Fault detection 42.1%
Emphasizes the error-proneness of input space in ATMR by making it unavailable for PI reduction/expansion for systematic approximate module generation	Input-vulnerability-aware, PI expansion/reduction ([20], 2019)	il (i : 25, o:12) CC (i : 21, o: 20) CU (i : 14, o: 11) X2 (i : 10, o : 7)	ASIC/F PGA	Complements, fault coverage, undetected/detected faults, transistor count, candidate search space reduction	HOPE	ABC logic synthesis tool	75%–48% fault coverage Search space reduction 41.5% to 95.5%
Integration of data size and precision reduction for ATMR of microprocessors	Successive approximation, number of iterations ([30], 2019)	Newton–Raphson ATMR configurations	Softwar e/hardw are	Execution time, overhead, error distribution	Laser	ARM Cortex A9 processor Zynq-7000 ,	56.9% reduction in execution time
ATMR based on the approximate gate library (ApxLib) technique with testability and observability as heuristics	Testability and observ ability ([28], 2019)	LGSynth93 (5xp1, 9sym, alu4, intb, max1024, prom1)		Area overhead error rate, execution time	Custom-made Python-based simulator	ABC synthesis tool	Execution time reduction
Uses the precision reduction technique for ATMR designs	Precision reduction, FPGA, fault tolerance, approximate computing, high-level synthesis, performance, reconfigurable systems ([21], 2019)	Xilinx Zynq-7000	FPGA/ ASIC	Accuracy, area, reliability	Bit flipping of configurat ion bits	Vivado Design Suite, Zynq7000	DSP usage reduction 80% Number of sensitive configuration bits 75% Accuracy 99.6%
Introduces fault-tolerant voter for ATMR by using pass transistors for compactness and quadded transistor redundancy for improved fault masking	Fault-tolerant voter, pass transistor, quadded transistor redundancy, Fault-tolerant voter, pass transistor, quadded transistor redundancy ([16], 2019)	Voter	ASIC	Vulnerable input vectors QoC, FMR, delay power product, number of transistors	HOPE	ModelSim with Python integration, Cadence Virtuoso	FMR improvement 45.1% QoC improvement 62.5% Reliability improvement 26.6% Transistor count improvement 50% PDP improvement 56%

for the voter inputs was introduced as shown in Fig. 11. As the output stage of the circuit must offer high reliability, reliability analysis of a PTL multiplexer, a sensitive gate, mandates for Monte Carlo simulation. Hence, an ATMR voter design with an improved (high reliability) output stage is desired

Limitation: Low noise margin, need for additional drivers, low-reliability output stage.

In another work [31], a software-based method, LEXACT, was proposed to develop NMR in a multicore processor by using AC. The technique targeted higher system reliability with lower overheads while using only k cores in comparison with the N cores deployed by a traditional NMR, where k<N. In this work, combinations of exact and approximate versions of a task were executed on different cores.

V. COMPARISON

In this section, existing ATMR methodologies are compared. Table 4 highlights the distinctive features of ATMR design techniques with respect to design considerations and challenges discussed in the previous section. Table 5 presents summary and comparison of ATMR mechanisms in terms of benchmarks, platforms, evaluation metrics, synthesis/simulation tools, and their claim. LGSynth93 is the most popular benchmark among these works. The ASIC and FPGA platforms have been equally explored by the existing ATMR studies. The commonly used evaluation metrics are the area overhead and error rate. Several ATMR techniques have relied on HOPE as the fault injection tool whereas a few have developed their own tool. The ABC synthesis tool is generally used for mapping the circuits.

VI. CONCLUSION

In this study, several existing ATMR design techniques were surveyed in detail. The significance of AC and its benefits for fault masking through ATMR were emphasized. The ATMR design techniques were classified, their trends were presented, and the relevant concepts were introduced. Furthermore, the issues and challenges that must be considered by ATMR design methods were discussed extensively. Various existing ATMR techniques were briefly described along with their advantages and limitations. Subsequently, the design techniques were analyzed in terms of benchmarks, platforms, evaluation metrics, and tools used. To the best of our knowledge, this is the first article that discusses the existing ATMR design techniques in detail. Our review highlights that the ATMR techniques must consider all the related design issues due to the sensitive nature of ATMR. Notably, there is no open access tool available for the generation of approximate circuits of ATMR. Specifically, as the problem generation of approximate circuits differs from that of conventional approximate circuits, owing to the relationship of the three modules and the working principle of ATMR, a dedicated tool is required.

REFERENCES

[1] R. R. Osorio and G. Rodriguez, "Truncated SIMD multiplier architecture for approximate computing in low-power programmable processors," *IEEE Access*, vol. 7, pp. 56353–56366, 2019.
[2] G. Zervakis, H. Amrouch, and J. Henkel, "Design automation of approximate circuits with runtime reconfigurable accuracy," *IEEE Access*, vol. 8, pp. 53522–53538, 2020.
[3] S. Kasap, E. W. Wächter, X. Zhai, S. Ehsan, and K. Mcdonald-Maier, "Survey of soft error mitigation techniques applied to LEON3 soft processors on SRAM-based FPGAs," *IEEE Access*, vol. 8, pp. 28646–28658, 2020.
[4] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.
[5] K. S. Mohamed, "Approximate computing: Towards ultra-low-power systems design," in *Neuromorphic Computing and Beyond*. Cham, Switzerland: Springer, 2020, pp. 147–165.
[6] Q. Xu, T. Mytkowicz, and N. S. Kim, "Guest editors' introduction: Approximate computing," *IEEE Des. Test. Comput.*, vol. 33, no. 1, pp. 6–7, Feb. 2016.
[7] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, May 2016.
[8] G. Rodrigues, F. L. Kastensmidt, and A. Bosio, "Survey on approximate computing and its intrinsic fault tolerance," *Electronics*, vol. 9, no. 4, p. 557, Mar. 2020.
[9] A. S. Hassan, T. Arifeen, H. Moradian, and J.-A. Lee, "Generation methodology for good-enough approximate modules of ATMR," *J. Electron. Test.*, vol. 34, no. 6, pp. 651–665, Dec. 2018.
[10] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2011, pp. 365–376.
[11] 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," *Microelectron. Rel.*, vol. 55, nos. 9–10, pp. 2072–2076, 2015.
[12] T. Arifeen, A. S. Hassan, H. Moradian, and J. A. Lee, "Probing approximate TMR in error resilient applications for better design tradeoffs," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2016, pp. 637–640.
[13] A. Sanchez-Clemente, L. Entrena, M. García-Valderas, and C. López-Ongil, "Logic masking for SET mitigation using approximate logic circuits," in *Proc. IEEE 18th Int. On-Line Test. Symp. (IOLTS)*, Jun. 2012, pp. 176–181.
[14] I. A. C. Gomes and F. G. L. Kastensmidt, "Reducing TMR overhead by combining approximate circuit, transistor topology and input permutation approaches," in *Proc. 26th Symp. Integr. Circuits Syst. Design (SBCCI)*, Sep. 2013, pp. 1–6.
[15] I. Polian and J. P. Hayes, "Selective hardening: Toward cost-effective error tolerance," *IEEE Des. Test. Comput.*, vol. 28, no. 3, pp. 54–63, May 2011.
[16] T. Arifeen, A. Hassan, and J.-A. Lee, "A fault tolerant voter for approximate triple modular redundancy," *Electronics*, vol. 8, no. 3, p. 332, Mar. 2019.
[17] I. A. C. Gomes, M. Martins, F. L. Kastensmidt, A. Reis, R. Ribas, and S. P. Novales, "Methodology for achieving best trade-off of area and fault masking coverage in ATMR," in *Proc. 15th Latin Amer. Test Workshop (LATW)*, Mar. 2014, pp. 1–6.
[18] 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 *Proc. 16th Latin-Amer. Test Symp. (LATS)*, Mar. 2015, pp. 1–6.
[19] A. J. Sánchez-Clemente, L. Entrena, and M. García-Valderas, "Partial TMR in FPGAs using approximate logic circuits," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 4, pp. 2233–2240, Aug. 2016.
[20] T. Arifeen, A. S. Hassan, H. Moradian, and J. A. Lee, "Input vulnerability-aware approximate triple modular redundancy: Higher fault coverage, improved search space, and reduced area overhead," *Electron. Lett.*, vol. 54, no. 15, pp. 934–936, Jul. 2018.
[21] G. S. Rodrigues, J. Fonseca, F. Benevenuti, F. Kastensmidt, and A. Bosio, "Exploiting approximate computing for low-cost fault tolerant architectures," in *Proc. 32nd Symp. Integr. Circuits Syst. Design (SBCCI)*, 2019, pp. 1–6.
[22] A. Sánchez, L. Entrena, and F. Kastensmidt, "Approximate TMR for selective error mitigation in FPGAs based on testability analysis," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Aug. 2018, pp. 112–119.
[23] S. Venkataraman, R. Santos, and A. Kumar, "A flexible inexact TMR technique for SRAM-based FPGAs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 810–813.

[24] I. Albandes, A. Serrano-Cases, A. J. Sanchez-Clemente, M. Martins, A. Martínez-Álvarez, S. Cuenca-Asensi, and F. L. Kastensmidt, "Improving approximate-TMR using multi-objective optimization genetic algorithm," in *Proc. IEEE 19th Latin-Amer. Test Symp. (LATS)*, Mar. 2018, pp. 1–6.
[25] A. Serrano-Cases, J. Isaza-González, S. Cuenca-Asensi, and A. Martínez-Álvarez, "On the influence of compiler optimizations in the fault tolerance of embedded systems," in *Proc. IEEE 22nd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2016, pp. 207–208.
[26] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Saf.*, vol. 91, no. 9, pp. 992–1007, Sep. 2006.
[27] I. Albandes, A. Serrano-Cases, M. Martins, A. Martínez-Álvarez, S. Cuenca-Asensi, and F. L. Kastensmidt, "Design of approximate-TMR using approximate library and heuristic approaches," *Microelectron. Rel.*, vols. 88–90, pp. 898–902, Sep. 2018.
[28] I. Albandes, M. Martins, S. Cuenca-Asensi, and F. L. Kastensmidt, "Building ATMR circuits using approximate library and heuristic approaches," *Microelectron. Rel.*, vol. 97, pp. 24–30, Jun. 2019.
[29] A. J. Sanchez-Clemente, L. Entrena, R. Hrbacek, and L. Sekanina, "Error mitigation using approximate logic circuits: A comparison of probabilistic and evolutionary approaches," *IEEE Trans. Rel.*, vol. 65, no. 4, pp. 1871–1883, Dec. 2016.
[30] G. S. Rodrigues, J. S. Fonseca, F. L. Kastensmidt, V. Pouget, A. Bosio, and S. Hamdioui, "Approximate TMR based on successive approximation and loop perforation in microprocessors," *Microelectron. Rel.*, vols. 100–101, Sep. 2019, Art. no. 113385.
[31] F. Baharvand and S. G. Miremadi, "LEXACT: Low energy N-modular redundancy using approximate computing for real-time multicore processors," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 431–441, Apr. 2020.

TOOBA ARIFEEN received the B.E. degree in electrical (telecommunication) engineering from Bahria University, Pakistan, in 2010, the M.Sc. degree in IC design engineering from The Hong Kong University of Science and Technology (HKUST), in 2012, and the Ph.D. degree in computer engineering from Chosun University, South Korea. From 2012 to 2015, she has served as a Lecturer with the Department of Engineering, Iqra University, Pakistan. She is currently a Postdoctoral Researcher with the Computer Systems Laboratory, Chosun University. Her research interests include fault tolerance, approximate computing, deep neural networks, and low-power digital systems.

ABDUS SAMI HASSAN received the bachelor's degree in electronics from COMSATS University, Islamabad, Pakistan, in 2010, the M.Sc. degree in computer engineering from UET, in 2014, and the Ph.D. degree in computer engineering from Chosun University, South Korea. From 2014 to 2015, he was a Lecturer with the Department of Computer Science, COMSATS. He also holds a three-year experience as a System Engineer. He is currently a Postdoctoral Researcher with the Computer Systems Laboratory, Chosun University. His research interests include fault tolerance, approximate computing, deep neural networks, and low-power digital systems.

JEONG-A LEE (Senior Member, IEEE) received the B.S. degree (Hons.) in computer engineering from Seoul National University, in 1982, the M.S. degree in computer science from Indiana University Bloomington, in 1985, and the Ph.D. degree in computer science from the University of California at Los Angeles, in 1990. From 1990 to 1995, she was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Houston. Since 1995, she has been with Chosun University, South Korea. From 2008 to 2009, she has served as the Program Director of the ECE Division, National Research Foundation of Korea. She has authored or coauthored over 100 reviewed journal and conference papers. Her research interests include high-performance computer architectures, approximate computing, self-aware computing, and reliable computing. She is a member of the National Academy of Engineering, South Korea.