IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 6 September 2023, accepted 29 September 2023, date of publication 18 October 2023,
date of current version 25 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3325616

METHODS

Formal Verification of Fault-Tolerant Hardware Designs

LUIS ENTRENA[1], (Member, IEEE), ANTONIO J. SÁNCHEZ-CLEMENTE[2], LUIS A. GARCÍA-ASTUDILLO[1], MARTA PORTELA-GARCIA[1], MARIO GARCIA-VALDERAS[1], (Member, IEEE), ALMUDENA LINDOSO[1], (Senior Member, IEEE), AND ROBERTO SARMIENTO[2]

[1]Universidad Carlos III de Madrid, Leganés, 28911 Madrid, Spain
[2]Institute for Applied Microelectronics, Universidad de Las Palmas de Gran Canaria, Campus Universitario de Tarifa, 35017 Las Palmas de Gran Canaria, Spain

Corresponding author: Luis Entrena (entrena@ing.uc3m.es)

This work was supported in part by the European Space Agency (ESA) with ARQUIMEA Ingeniería S.L.U under Contract 4000123942/18/NL/GLC, in part by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M26), and in part by the Context of the V Regional Programme of Research and Technological Innovation (PRICIT).

ABSTRACT Digital circuits for space applications can suffer from operation failures due to radiation effects. Error detection and mitigation techniques are widely accepted solutions to improve dependability of digital circuits under Single Event Upsets (SEUs) and Single Event Transients (SETs). These solutions imply design modifications that must be validated. This paper presents a formal verification method to prove that the applied fault tolerance techniques do actually prevent fault propagation as well as that the fault-tolerant circuit is functionally equivalent to the original version. The method has been implemented in an in-house software tool, VeriHard. It has been successfully applied to verify a wide variety of fault tolerance techniques, such as Triple Modular Redundancy (TMR), Duplication with Comparison (DwC), Safe Finite State Machines and Hamming encoding. Experimental results with benchmarks and industrial cases illustrates the capabilities of the method and its high performance.

INDEX TERMS Equivalence checking, fault tolerance, formal verification, error mitigation.

I. INTRODUCTION

As manufacturing technology progresses, enabling the integration of more complex circuits with higher densities and smaller transistors, ensuring dependability becomes an increasingly important and difficult task. Soft errors are becoming a concern for safety critical applications. Traditionally, this was a problem for circuits working in harsh environments, such as space. In these environments, the use of hardened technologies and fault-tolerant designs is mandatory. Today, soft errors are a concern for an increasing variety of applications even at ground level and the use of fault tolerance is spreading. Security issues are also of increasing importance and are generally closely related to fault tolerance issues.

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

Fault tolerance techniques involve the addition of redundancy in some way to detect and/or correct errors. The most common fault tolerance techniques are increasingly supported in design environments. Design tools currently allow designers to implement some techniques by using specific commands. Other techniques can be manually implemented by designers. However, fault tolerance techniques are subtle and the implementation is not trivial: designers may make mistakes, synthesis tools may inadvertently remove redundancy (because they are designed to remove redundancy and optimize the logic, but not to correct errors), techniques may not be properly applied, etc. As a consequence, the error mitigation performance may be seriously compromised with catastrophic consequences. Therefore, designers need solutions to verify that the fault-tolerant structures work correctly.

Fault injection has traditionally been used to evaluate and validate fault-tolerant designs. However, fault injection

has limited coverage, as it can only validate the design for a subset of errors and test vectors. To ensure fault tolerance is correctly implemented, formal verification techniques are required. In fact, formal verification has become a cornerstone of modern design flows [1]. Despite formal verification is a hard problem, today it can be solved in practice for most real circuits. Nevertheless, the formal verification of fault-tolerant designs has not received much attention yet.

In this work, we propose formal verification techniques that are specifically designed to verify fault-tolerant circuits. The verification of fault-tolerant circuits involves two formal verification problems: verify that the fault-tolerant circuit effectively masks or detects errors; and verify that the fault-tolerant circuit is functionally equivalent to the original circuit before fault-tolerance techniques were applied. To this purpose, we propose a new approach that addresses these two verification problems in sequence.

For the first problem, we note that it is not a conventional formal verification problem, because the goal is not to prove the equivalence of two circuits or a particular property of a circuit under regular operation, but to prove that the circuit is able to detect or correct an error when it occurs. The problem is also complex, as the verification must be repeated for a large set of possible errors. In this work, we solve this problem by formulating it as a formal Equivalence Checking (EC) problem. Then, we show how this problem can be solved by using extended EC techniques that are specifically adapted for this purpose. Furthermore, the complexity of these techniques is similar to that of combinational EC techniques, so they can be used in practice with good performance.

The second problem can be formulated as a conventional EC problem between two circuits. However, after verifying the fault-tolerant structures, redundancies are identified and can be conveniently removed to simplify equivalence checking with the original circuit.

To the best of our knowledge, this is the first time that a general formal verification approach, based on extended EC techniques, is proposed for fault-tolerant circuits. This approach can effectively verify a wide variety of fault tolerance techniques, such as Triple Modular Redundancy (TMR), Duplication with Comparison (DwC), Safe Finite State Machines and Hamming encoding. On the contrary, the few existing approaches are based on checking structural similarity [2], [3], [4], which is limited, or are basic adaptations of existing formal verification techniques, with relatively poor performance.

The remaining of this paper is as follows. Section II introduces the necessary formal verification background and reviews previous work. Section III describes the proposed formal verification approach. Section IV shows how this approach can be used for a variety of representative fault tolerance techniques. Section V describes the experimental results intended to validate the proposed approach. Finally, Section VI shows the conclusions of this work.

II. FORMAL VERIFICATION BACKGROUND AND RELATED WORK

Formal verification of hardware designs is a key topic in Electronic Design Automation (EDA) and has been a matter of research for decades, covering several relevant problems. Among them, Equivalence Checking seeks to formally prove that two circuit designs have exactly the same behavior. As circuit complexity grows, it is necessary to verify that functionality is preserved when a design undergoes the various synthesis and implementation steps. Equivalence Checking is intended to prove equivalence for all possible test vectors, as opposed to simulation-based validation that only tests for a selected set of test vectors.

Practical EC approaches typically assume a known or guessed correspondence between the storage elements of the circuit, so that the checking can be reduced to a Combinational Equivalence Checking (CEC) problem. CEC is a mature technology, with known limitations but also with tools and solutions for practical industrial cases [1].

CEC techniques are strongly associated to the particular formalisms that are used to model the circuit and prove equivalence. Canonical representations, such as Binary Decision Diagrams (BDDs) [5], are very well suited for equivalence checking, as the solution is found by building the diagram of a miter circuit [6], [7]. However, they may be difficult or impossible to build for large netlists and for some types of circuits. For instance, BDD size can grow exponentially for arithmetic circuits such as multipliers [8].

Alternatively, satisfiability (SAT) solvers use Boolean algebraic expressions, typically in Conjunctive Normal Form (CNF), to represent the logic functionality as a set of Boolean constraints. Boolean algebraic expressions are very convenient for mathematical deduction, because they are simple, systematic and efficient. SAT solvers are generally derived from the well-known Davis–Putnam–Logemann–Loveland (DPLL) algorithm, which is based on a search and backtrack approach. The DPLL algorithm iteratively assigns variables and uses Boolean Constraint Propagation (BCP) to obtain new assignments until a satisfying assignment is found or the search is exhausted. SAT solving has been a matter of research for a long time, resulting in several efficient tools such as zChaff [9], Berkmin [10], SATO [11], GRASP [12], MiniSAT [13], and C-SAT [14]. These tools have introduced significant improvements that enable handling large industrial formal verification problems.

Finally, some techniques make use of structural representations, such as And-Inverter Graphs (AIGs). AIGs are used to represent Boolean functions in many EDA problems because they are simple, efficient, uniform and easy to build. They are not canonical, but they can be efficiently matched using hashing techniques [15]. AIG-based structural hashing can save large amounts of computational effort and is widely used in formal verification. For SAT solving, AIGs can be translated into a CNF representation. Moreover, the so-called structural or circuit-based SAT solvers implement Boolean reasoning directly on AIGs [16].

Existing CEC solutions usually combine different methods. Structural hashing is used for fast matching of isomorphic AIG nodes. Random simulation is used to reduce the search effort by identifying potential matching candidates. Then, more powerful but also more computationally expensive SAT solvers and Boolean reasoning methods are used to complete verification. Reference [16] presents a verification framework that combines structural hashing, random simulation, BDDs and structural SAT solvers. In [17], a SAT solver is proposed that combines the strengths of circuit-based and CNF-based SAT solvers. Effective approaches generally try to identify and merge functionally equivalent AIG nodes prior to SAT solving. In [18], the concept of Functionally Reduced AIGs (FRAIGs) is introduced. FRAIGs are built using a reduction-by-construction method based on random simulation and SAT solving. Experimental results show that this method is very efficient for CEC.

CEC can be applied to sequential circuits, provided that the mapping of storage elements (flip-flops and memories) between the circuits to be checked is known or it can be guessed [19]. In such a case, the sequential equivalence checking (SEC) problem can be transformed into a CEC problem by treating the inputs and outputs of storage elements as primary outputs and primary inputs, respectively. This approach is used by many academic and commercial tools, as it avoids state explosion. The mapping of storage elements can be derived from naming conventions or it can be automated [1].

Nevertheless, a one-to-one mapping may not exist even though the circuits may be equivalent. Synthesis tools may perform optimizations that modify the logic surrounding storage elements or result in a different number of flip-flops. Formal verification techniques have been proposed that can deal with changes in the polarity of the storage elements [20], clock gating [21] or moving logic across flip-flops, as in the case of retiming [22], [23].

In the general case, SEC is still a challenge because of the state explosion problem [24]. Bounded Model Checking (BMC) focuses on finding a counterexample of a length bounded by some integer k. BMC is good for test cases with short counterexamples, but it is incomplete. In [25], several bounded and unbounded model checking approaches are analyzed and compared using a large set of benchmarks.

The verification of fault-tolerant circuits is a particular problem that has not received much attention yet. Mentor [26] proposes a solution based on verifying the equivalence of a miter circuit with two instances of the fault-tolerant circuit when an error is inserted in one of the instances. Fig. 1 shows the construction used to address this problem. With this construction, existing formal verification tools can be applied. This is a general approach, but the performance may be poor, because the equivalence checking problem is addressed from a global perspective and generally requires heavy SEC techniques. In addition, the verification must be repeated for each possible error in the circuit. Experimental results show that a practical circuit may require hours or even days of CPU [27]. This emphasizes the need for more efficient approaches.

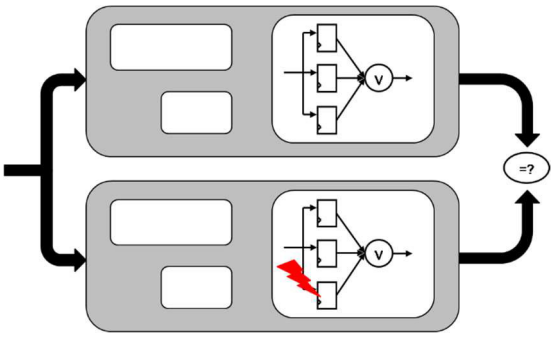

FIGURE 1. Miter construction for verification.

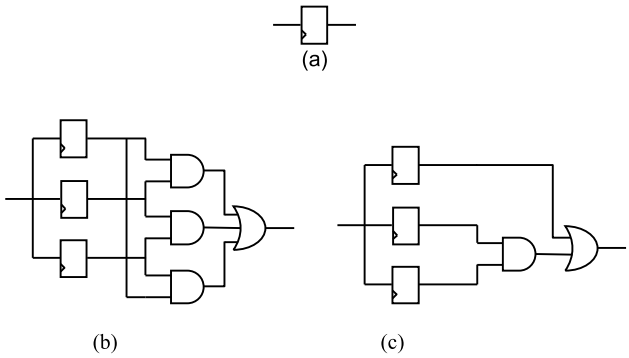

FIGURE 2. Motivating example: (a) a simple flip-flop circuit; (b) a TMR implementation of the flip-flop; (c) a TMR implementation with a simplified voter.

Several solutions have been proposed that rely on checking the structure of the fault-tolerant circuit. In [2], a topological approach is proposed for TMR (Triple Modular Redundancy) designs. The goal is to verify whether the three copies (TMR domains) of logic that feed each voter under analysis are equivalent and the voters are inserted as expected. A structure-based approach is also proposed in [3]. Then, fault injection is applied but without performing gate level simulations. In [4], a commercial formal verification tool is used to check the equivalence of TMR domains with respect to the original design. Then fault injection campaigns are performed to validate fault-tolerance for single faults. Nevertheless, the effectiveness of these approaches has not been clearly demonstrated yet.

III. FORMAL VERIFICATION APPROACH

The problem of verifying that a fault-tolerant circuit effectively masks or detects errors is not directly an equivalence checking problem. A simple motivating example is shown in Fig. 2. The original unprotected circuit is the flip-flop in Fig. 2(a). Two fault-tolerant implementations of this flip-flop using TMR are shown in Fig. 2(b) and Fig. 2(c). Implementation (b) is a correct implementation, while implementation (c) uses a simplified voter that is not able to mask all errors. A formal equivalence checking tool will verify that the three

circuits are functionally equivalent, yet the circuit (c) cannot mask all SEUs (Single-Event Upsets).

To verify that a circuit is fault-tolerant, we need to verify its behavior in the presence of faults. In addition, it is necessary to verify that the application of fault tolerance techniques has not modified the functionality of the circuit. This is a twofold problem that we address in two steps:

1) Prove that errors are detected or corrected in the fault-tolerant circuit as intended by the applied fault-tolerant techniques. If verification is successful, redundant structures are identified.
2) Merge redundant structures and prove that the merged circuit is equivalent to the original unprotected circuit.

Redundancies must be in the circuit for the first step, as they are essential to tolerate faults. Once fault tolerance has been verified, the removal of redundancies simplifies the circuit model and makes further checking easier. Eventually these two steps may be combined and iterated as needed for several types of verification problems.

Fault tolerance techniques generally introduce redundancy in a way that affects the sequential structure of the circuit. For instance, to protect against SEUs, hardware redundancy techniques typically introduce redundant flip-flops. Error detection and correction codes modify state encoding and introduce additional flip-flops or memory bits. From this point of view, the formal verification of fault-tolerant designs is a SEC problem. CEC techniques can be used to alleviate complexity if the mapping of flip-flops is known or it can be inferred. However, the mapping of flip-flops is not immediate, because fault-tolerant circuits generally have many redundant flip-flops. Therefore, the capability to do formal verification in a sequential context is an important requirement.

Our approach to verify circuit behavior in the presence of faults is based on Boolean Reasoning using circuit-based structural SAT. Structural SAT methods perform a systematic search for a consistent assignment on the circuit representation. They have been shown to be very efficient at BCP [17]. Furthermore, they also facilitate the checking of error propagation constraints. As we will see later on, the error propagation constraints are hard to model using CNF clauses.

Instead of checking two copies of the circuit, good (fault-free) and faulty, we use a single circuit in which each node has two different binary values, namely v_g (good value) and v_f (faulty value). Each binary value can be 0, 1 or X (unknown). Thus, a node can have 9 possible values, which are usually expressed as $v = v_g/v_f$. This is known as the nine-valued model [28]. This model is commonly used in ATPG (Automatic Test Pattern Generation) and other related problems.

Circuit-based Boolean reasoning can be extended to sequential circuits using the concept of timeframes. A timeframe is an instance of the combinational logic of a sequential circuit. To study sequential behavior, a sequential circuit is unrolled in a set of timeframes connected by flip-flops. The flip-flop inputs are referred to as pseudo primary outputs (PPOs) of the previous timeframe and the flip-flop outputs are referred to as pseudo primary inputs (PPIs) of the next timeframe. The timeframe in which a fault is injected is called timeframe 0. Other timeframes are numbered according to the distance to timeframe 0.

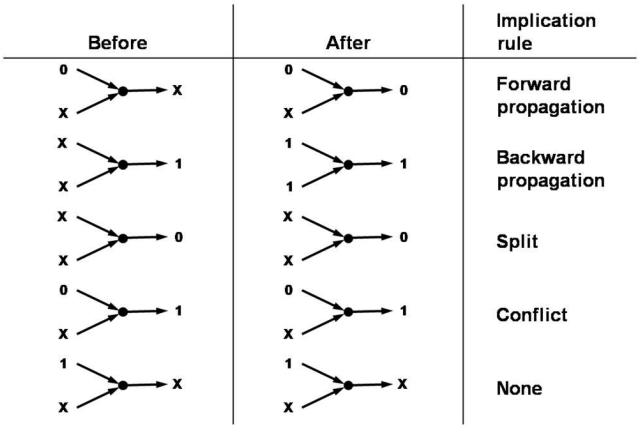

FIGURE 3. An illustrative subset of the implication rules.

Boolean reasoning can be performed across timeframes. Each node may have a different assignment in each timeframe, but the assignments to connected PPOs and PPIs of neighbour timeframes must be consistent. This approach has been used in sequential testing [29] and sequential verification [30]. As long as errors are detected or corrected with low sequential depth, it is an efficient approach.

As in other formal verification approaches, we use an AIG [15] representation of the circuit. This representation efficiently supports the required verification techniques, including structural hashing and structural SAT. We have extended the AIG format to support hierarchical netlists and sequential netlists with multiple clock domains.

A. IMPLICATION RULES AND ERROR PROPAGATION CONSTRAINTS

In structural SAT, Boolean Constraint Propagation is transformed into an implication process. Fig. 3 shows a subset of the implication rules (the complete set is omitted for brevity). These rules are a consequence of the AND functional constraints of AIG nodes. They are typically encoded as a lookup table [16] for efficient evaluation.

The application of the implication rules to a node may produce new assignments by forward propagation or backward propagation that trigger further implications. In one case (*Split*), there are two possible solutions and the justification algorithm must attempt both of them. To this purpose, a new decision level is opened in the decision stack. In other cases, the implication leads to a conflict, and the algorithm has to backtrack. The implication may also result in no new assignment (*None*). Eventually, the implication process ends when a conflict is found or all implications have been propagated. Then, a new decision is made and new implications are computed. The verification algorithm iterates until all nodes

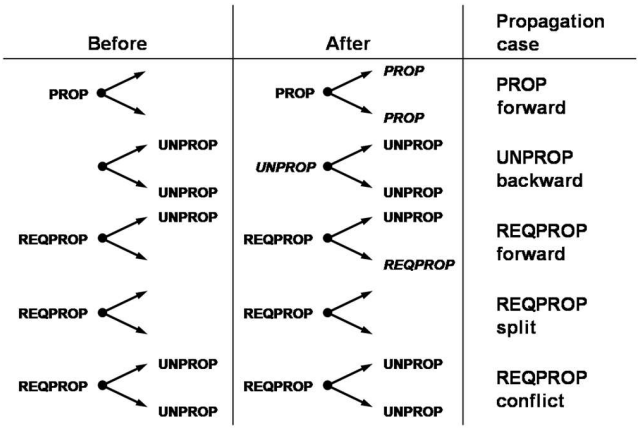

FIGURE 4. Implication rules for P values.

can be consistently assigned, which proves the problem is satisfiable, or until the decision stack is exhausted without having found a solution.

The implication rules must be satisfied in the good circuit and in the faulty circuit. In other words, each node must be implied for both the good and the faulty values. However, error propagation introduces additional constraints. We refer to these constraints as Error Propagation Constraints (EPCs).

EPCs result from the requirements of error propagation. To this purpose, the nodes in a circuit are attributed a P (Propagation) value. We use symbolic names for P values to clearly distinguish them from assigned values. The nodes in the output cone of the error source may propagate an error and receive a PROP value. The nodes outside the output cone of the error source cannot propagate errors and receive a NOPROP value. The PROP value merely results from the structure of the circuit. Whether a PROP node effectively propagates the error or not, it depends on the logical conditions in the upstream and downstream nodes. A PROP node that cannot propagate the error to the primary outputs is called UNPROP. A PROP node that is required to propagate the error is called REQPROP.

Based on these considerations, the following EPC rules must be satisfied:

1) NOPROP nodes have $v_g = v_f$ by definition. For brevity, we denote the assignment with a single value $v = v_g = v_f$.
2) PROP nodes may have $v_g \neq v_f$ or $v_g = v_f$. If a PROP node satisfies $v_g = v_f$, then it becomes an UNPROP node.
3) REQPROP nodes must have $v_g \neq v_f$.
4) Side NOPROP inputs of REQPROP nodes must be assigned a sensitive value ($v = 1$). This is a consequence of EPC rules 1 and 3.

Furthermore, P values are also implied. The implication rules for P values are summarized in Fig. 4. The fanouts of a PROP node are also PROP nodes. Conversely, if all fanouts of a node are UNPROP, the node is UNPROP. For a REQPROP node, implication of P values depends on the number of PROP fanouts. If a REQPROP node has a single PROP fanout, then error propagation requires the PROP fanout to be REQPROP. If there are multiple PROP fanouts, they must be stored in the decision stack in order to try all of them (*split* case). Finally, if all fanouts of a REQPROP node are UNPROP, the fault cannot be propagated and a conflict is found.

B. FAULT SATISFIABILITY

The goal of the fault-tolerance verification problem is to prove that, if a fault is injected, no input vector can produce $v_g \neq v_f$ for any primary output, or to find a counterexample, i.e., an input vector that propagates a 0/1 or a 1/0 value to at least one primary output. It can be formulated as a satisfiability problem: given a target faulty node T, prove that the conditions $v_g(T) \neq v_f(T)$ and $v_g(O_i) \neq v_f(O_i)$ are not satisfiable for any $O_i \in PO$, where PO is the set of primary outputs. In the sequel, we will call it fault satisfiability problem or fault SAT for short. The initial condition $v_g(T) \neq v_f(T)$ can be split in two cases, namely $v(T) = 0/1$ and $v(T) = 1/0$. Each of these two cases is usually called a fault injection.

The proposed approach to fault SAT follows the DPLL SAT algorithm and uses structural SAT with the 9-valued model and timeframe expansion. After injecting a fault, the algorithm exhaustively justifies all assignments and tries all possible propagation paths. Two types of decisions are considered: justification decisions and propagation decisions.

Justification decisions solve backward split cases. They further include four types of cases: justification of good value of a PROP node, justification of faulty value of a PROP node, justification of a NOPROP node (common value for good and faulty), and justification in previous timeframes (common value). Any number of previous and next timeframes can be considered. However, verification can be solved with just one previous timeframe in most cases, because fault-tolerant techniques are usually designed to detect or correct errors within one clock cycle. Propagation decisions solve forward REQPROP propagation split cases. Propagation decisions are specific of fault SAT.

Fig. 5 illustrates the fault verification process for the TMR flip-flop circuit shown in Fig. 2(b). Fig. 5(a) shows a simplified view of the unrolled AIG representation, including two timeframes. Node numbers are shown at the top or right of nodes. Flip-flops are represented by vertical bar nodes. The nodes to the right of the flip-flops are in timeframe 0 while the nodes to the left of the flip-flops are in timeframe -1. Note that in the complete view of the unrolled AIG, all nodes will be replicated in each timeframe. For simplicity, only the nodes involved in the Boolean reasoning are shown in the figure.

Let us select node 1 as the target node. The fault verification algorithm starts by assigning the REQPROP P value to the error source node, i.e. P(1) = REQPROP. Then, the application of EPC rule 3 results in two choices: $v(1) = 0/1$ or $v(1) = 1/0$. The algorithm must try both choices until a solution is found or both choices are proven unsatisfiable.

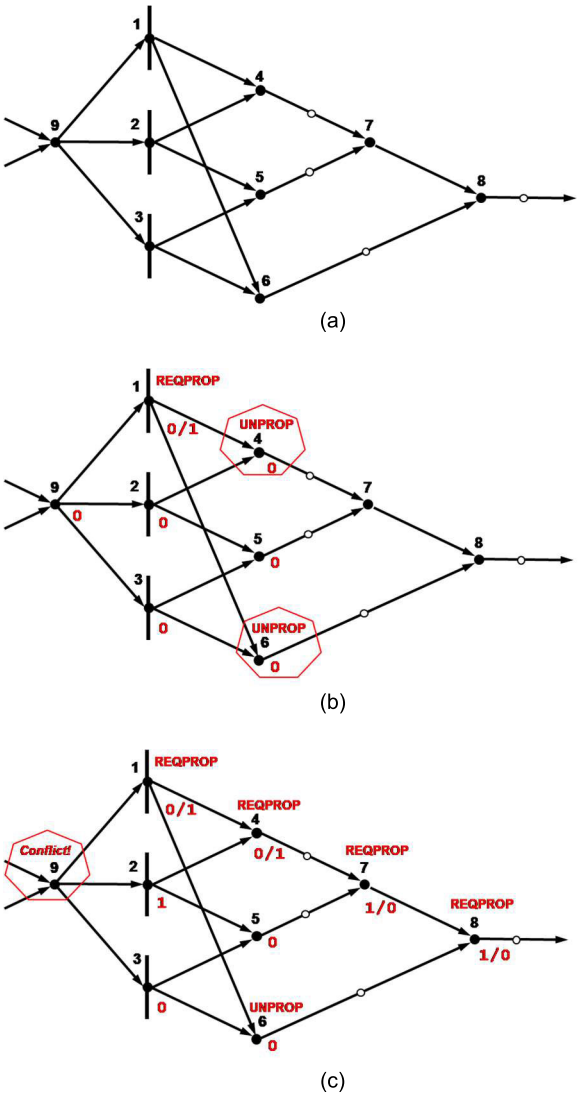

FIGURE 5. Examples of implication in a TMR circuit: (a) simplified AIG view of a basic TMR circuit; (b) implication of fault 0/1 in node 1 (v(1) = 0/1), justification first; (c) implication of fault 0/1 in node 1 (v(1) = 0/1), propagation first.

For the first choice, we have $v(1) = 0/1$. The implication process may continue by justifying $v_g(1)$ or by propagating the error through nodes 4 and 6. The order in which implication and not affect the result. However, different implications may be triggered depending on the selected implication order. We will show the results of two different implication orders to illustrate the application of the implication rules and constraints.

Fig. 5(b) shows the implications obtained when the process starts by justifying $v_g(1) = 0$. This requires $v^{-1}(9) = 0$ in the previous timeframe. Further implication of this assignment results in the implications shown in timeframe 0. In particular, $v(4) = 0/0$ and $v(6) = 0/0$. According to EPC rule 2, nodes 4 and 6 become UNPROP. A conflict is found because all fanouts of node 1 are UNPROP. The algorithm then backtracks and proceeds to try the remaining choice

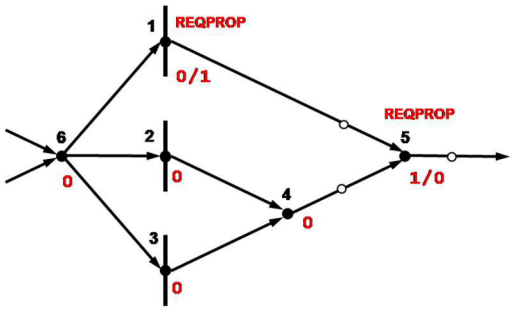

FIGURE 6. Implication of TMR circuit with bad voter.

$v(1) = 1/0$. The implication of this value is not shown, but it also results in a conflict. As the two choices are unsatisfiable, it is proved that no error injected in node 1 can be propagated to any primary output.

Alternatively, consider the case where the forward propagation of P(1) = REQPROP is implied first. Node 1 has two PROP fanout nodes, namely nodes 4 and 6, so it is a split propagation case. First, we try to propagate the $v(1) = 0/1$ error value through node 4, setting P(4) = REQPROP and $v(4) = 0/1$. The implication results are shown in Fig. 5(c). By the application of EPC rule 4, we have $v(2) = 1$. Further application of implication rules $v(7) = 1/0$, $v(5) = 0$, $v(3) = 0$, P(8) = REQPROP, $v(8) = 1/0$, $v(6) = 0/0$ and P(6) = UNPROP. In summary, to propagate the fault 0/1 at flip-flop 1, the other flip-flops, 2 and 3, are required to have different values. When these values are justified in the previous timeframe, we get a conflict, because $v(1) \neq v(2)$. The propagation of the $v(1) = 0/1$ error value through node 6 is very similar and results in a conflict as well.

Fig. 6 shows the AIG of the bad voter design in Fig. 2(c) and the implication of $v(1) = 0/1$. In this case, the implication process is able to find a solution to propagate the fault and the result is satisfiable. In fact, this is the only existing solution, because $v(1) = 1/0$ is not propagatable. Faults in the flip-flop nodes 2 and 3 are also not propagatable.

C. OVERALL FAULT-TOLERANCE VERIFICATION ALGORITHM

Fig. 7 shows a flow chart of the overall fault-tolerance verification algorithm. The main steps are as follows:

1) Build the AIGs for the fault-tolerant design and the original reference design.
2) Functional reduction: check the logic to detect equivalent functions. Merge combinational logic that is found equivalent.
3) Add external constraints, if any.
4) For each fault site, run fault SAT. If the fault is satisfiable, report the satisfying test vector.
5) If the fault is not satisfiable, merge redundant logic.
6) Equivalence checking between the original and the fault-tolerant merged circuit.

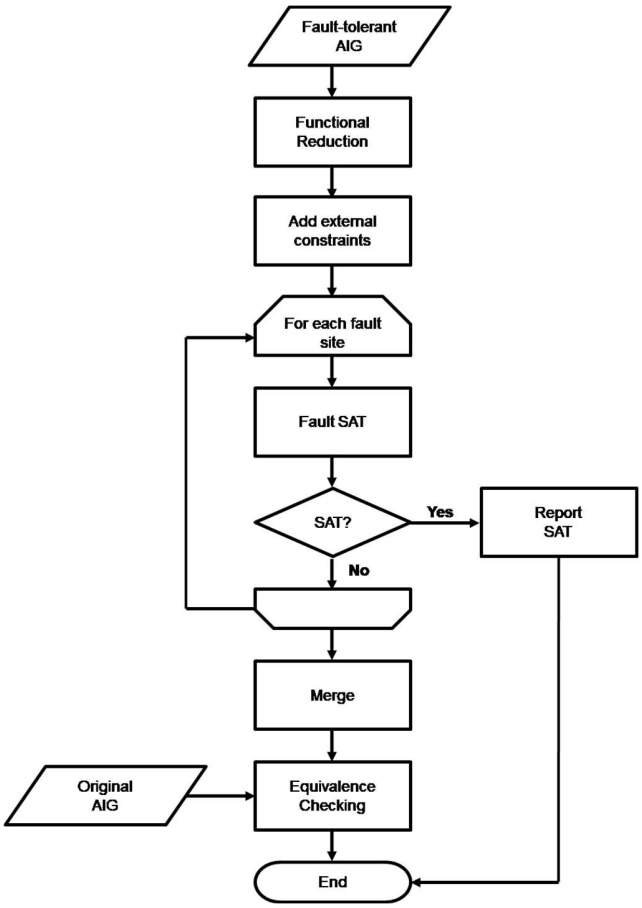

FIGURE 7. Fault-tolerance verification algorithm.

The first step is to build the AIG model of the fault-tolerant design and the reference design. As the fault-tolerant design will be tested for many faults, it is important to simplify the AIG as much as possible. To this purpose, the AIG is functionally reduced [18], using a combination of the following techniques:

- Structural hashing for fast matching.
- Random simulation to identify potentially equivalent nodes.
- Structural SAT-based equivalence checking to confirm or discard equivalence for pairs of potentially equivalent nodes.

The functionally equivalent nodes are reduced to simplify the representation. It is important to note that functional reduction is not applied to the redundant logic that specifically implements fault-tolerance. For instance, in the TMR example of Fig. 5, the functional reduction algorithm will easily detect that the three flip-flops are functionally equivalent. However, the three flip-flops are not merged at this step. It is only after the fault verification has proved no error can be propagated, that the three flip-flops are merged (step 5). This is a key idea that can be generalized as follows. For SEU fault-tolerant circuits, combinational logic is functionally reduced but storage elements are preserved until the fault verification step is completed. Then, redundant storage elements are merged for the final equivalence checking step. For SET (Single Event Transient) fault-tolerant circuits, i.e., circuits intended to tolerate errors in the combinational logic, functional reduction is applied only within combinational logic blocks that belong to the same logic instance or domain, in order to preserve block redundancy. Redundant logic blocks are only merged for the final equivalence checking step.

Some types of verification tasks may require additional constraints. These constraints are expressed as required values on particular nodes. If necessary to model constraints, new nodes, called virtual nodes, may be added to the AIG. Examples of constrained verification tasks will be shown in the next section.

The fault SAT algorithm uses implication-based Boolean reasoning (structural SAT), as explained in the previous section. Other techniques may be used because the problem is formulated as a formal verification problem. One of the main contributions of this work is to formulate the formal equivalence checking of fault-tolerant designs as a formal verification problem that can be solved with CEC techniques. However, the use of implication-based Boolean reasoning in an AIG representation facilitates the modeling of error propagation constraints.

After merging the redundant logic (step 5), the fault-tolerant circuit should have a very similar structure to that of the original circuit. In particular, the matching of storage elements is easier after redundant elements have been removed. If this matching is successful, then the equivalence checking problem can be solved using CEC techniques. In particular, we build a miter circuit and apply a functional reduction approach until all outputs are proved equivalent. Alternatively, this final equivalence checking step can be implemented using commercial tools.

Variations for different fault-tolerant techniques are described in the next section.

IV. APPLICATION TO FAULT TOLERANT TECHNIQUES

In this section, we show how the proposed approach can be applied to verify several common hardware fault-tolerance techniques.

A. TMR

TMR is a well-known fault-tolerant technique that can be implemented in several ways (Fig. 8) [2]. The simplest solution consists in triplicating and voting every flip-flop in the circuit (Fig. 8(a)). This solution is known as Local TMR (LTMR) and it is effective against SEUs. Distributed TMR (DTMR) is a stronger solution in which the combinational logic (CL) and the voters are also triplicated (Fig. 8(b)). DTMR is intended to protect against SETs. It is also used for configurable logic, to protect against errors in the configuration memory that may affect the configuration of combinational logic. Global TMR (GTMR) is the most general TMR implementation and it additionally includes the

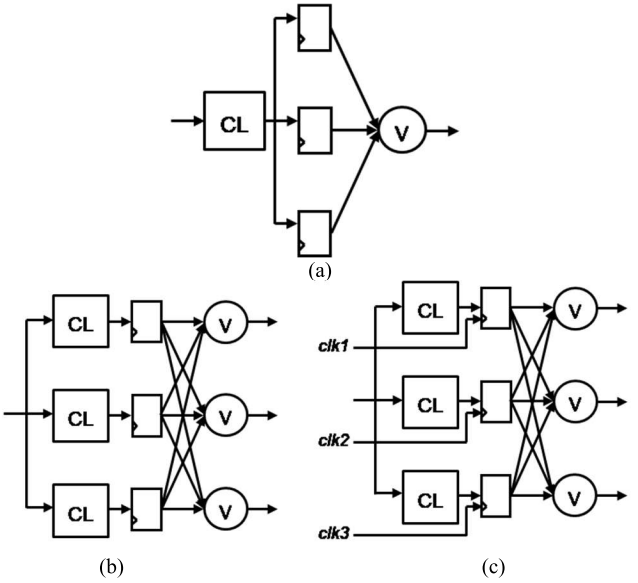

FIGURE 8. TMR approaches: (a) LTMR; (b) DTMR; (c) GTMR.

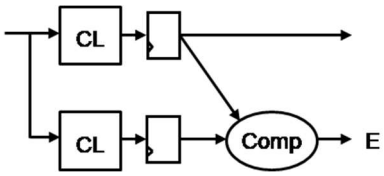

FIGURE 9. Duplication with comparison.

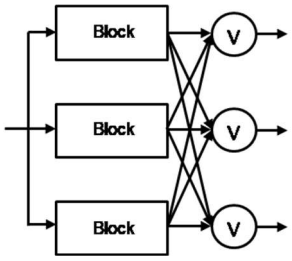

FIGURE 10. Block TMR.

triplication of the global signals, such as the clock and the reset (Fig. 8(c)). The reliability of TMR can be compromised by faults in more than one of the replicated devices, which is an increasingly frequent problem as node-size decreases, allowing for multiple bit upsets to appear from a single particle strike. However, TMR-based solutions are the most used hardening techniques.

Equivalence Checking of LTMR designs can be accomplished with the proposed fault verification algorithm. This algorithm is very efficient for local redundancy because verification can be solved locally. As shown in previous examples, the complexity of the verification is independent of the complexity of the design under test. For instance, the reasoning shown in Fig. 5 will solve fault SAT no matter how deeply the TMR structure is embedded in the circuit. Conversely, when a miter approach is used, such as the one shown in Fig. 1, the verification tool shall traverse the entire circuit to reach the target LTMR structure.

For DTMR, two verification runs of the algorithm's steps 2 to 5 are performed, first for faults in the combinational logic and then for faults in the flip-flops. In the first run, the functional reduction step is limited to each TMR domain. This way, the combinational logic domains are preserved for fault verification of the combinational logic. After combinational logic fault verification, the combinational logic domains are merged if they are found equivalent. This step reduces the circuit to a LTMR equivalent. In the second run, functional reduction reduces combinational logic but preserves the redundant flip-flops. Then, SEU faults are checked in an efficient manner. Finally, the equivalent flip-flops are also merged before the final equivalence checking step.

This approach also works for GTMR, except for the clock signal, which is not verified by injecting a fault. In case multiple clock domains are used, the verification is performed indirectly during the merge step. Namely, we check that flip-flops are not matched (they have different clock inputs) but they are matched if the clocks are matched (they are equivalent when the clock is equivalent). This check is made at step 5.

B. DUPLICATION WITH COMPARISON (DWC)

Duplication with Comparison focuses on error detection instead of error correction. To this purpose, the circuit is duplicated and the outputs are compared (Comp), as shown in Fig. 9. The comparator may be duplicated to protect from errors in the comparator.

The DwC verification problem is specified as follows: prove that the fault SAT problem is not satisfiable when the constraint $v_g(E) = v_f(E) = 0$ is included, being E the active-high error signal. In other words, prove that there is no test vector for which the error propagates, and the error signal does not detect it. For an active-low error signal, the constraint is $v_g(E) = v_f(E) = 1$.

The DwC verification problem can be solved using the proposed algorithm. The error constraint is added in step 3 of the algorithm.

C. BLOCK REDUNDANCY AND HIERARCHICAL VERIFICATION

Fault-tolerant techniques may be applied to large blocks of hardware. Fig. 10 shows the structure of a Block TMR (BTMR) solution, in which entire blocks are triplicated and voters are placed only at the outputs of the blocks. This solution can produce significant hardware savings. It could also be an interesting solution for IPs whose internal architecture is not known or it cannot be modified. However, note that internal block errors may be masked but they are not corrected unless the blocks themselves are fault-tolerant. As errors may propagate inside the block, formal verification of fault-tolerance only makes sense at the upper level.

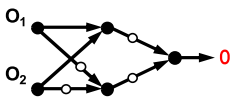

FIGURE 11. AIG model of block output constraint.

The proposed algorithm can be used to verify Block TMR or Block DWC as follows. Blocks are treated as special black box nodes in the AIG. During the functional reduction step, equivalent blocks are identified by a generalization of the structural hashing technique. Two blocks are equivalent if they are instances of the same component and have equivalent inputs. If these conditions are satisfied, then the outputs of the two blocks are equivalent. However, following the proposed methodology, equivalent blocks are not reduced during the functional reduction step. Instead, the equivalence of the outputs of matched blocks is introduced as constraints. Namely, if O_1 and O_2 are equivalent outputs of Block 1 and Block 2, respectively, the constraint $v(O_1) = v(O_2)$ must be satisfied. Fig. 11 shows the AIG used to model this constraint.

For the verification of BTMR, block inputs and outputs are considered as PPOs and PPIs, respectively. Then, the goal of the verification problem is to prove that for each fault at a block output, there is no test vector that can propagate the error, subject to the constraint that outputs of equivalent blocks must have equal values. Finally, the equivalent blocks are merged into a single block before the final equivalence checking step.

This approach can be extended to hierarchical verification. In the case of a complex circuit, with a hierarchy of components, the circuit can be flattened before verification. However, flattening removes information about the structure of the circuit that can be very useful to facilitate verification tasks. In particular, fault-tolerant techniques usually require creating redundant copies of portions of the circuit. A usual implementation approach consists in grouping the target logic into a new component and adding a new hierarchy level with redundant instances of the component. If this new hierarchy level is preserved for the verification, an efficient block-based verification approach can be used.

Generalizing this approach, verification can be accomplished in a recursive fashion. The verification of high level components invokes the verification of lower level components. When lower level components are verified, they are merged, so the verification at higher levels is simplified. If block-based verification is not possible at a particular hierarchy level, because equivalent blocks are not found, the circuit can be partially flattened down to the current verified level. Eventually, the recursive procedure ends at the top hierarchy level with a fully verified and merged circuit, that is ready to be checked with the original circuit.

D. SAFE FSM

Another group of commonly used fault-tolerant techniques aims for the hardening of Finite State Machines (FSM). These techniques are based on detecting illegal states or transitions (safe FSM), which can be combined with modifications of the FSM state encoding in order to ease error correction (Hamming-3 encoding) or detection (one-hot encoding). In this work, we have focused on developing verification methodologies for the safe FSM and Hamming-3 encoding approaches. Other approaches can be supported in a similar manner.

The proposed formal verification algorithm proposed can be tailored to verify the correctness of these FSM hardening techniques. However, some additional information must be known beforehand. In the general case, this information is comprised by the set of state flip-flops, the set of valid state encodings (denoted as S_{valid}), and any signal constraints which may alter the normal FSM behavior, such as reset or enable signals. The set of state flip-flops and encodings can be provided either by the designer or by the synthesis tool (e.g. Synopsys Synplify provides reports of the FSMs in the design). Alternatively, the set S_{valid} could be determined by reachability analysis, although this approach is not applied here other hand, signal constraints can either be provided by the designer or guessed using name matching against typical signal names (e.g. ''reset'', ''rst_n'', etc.). These constraints are taken into account during the FSM verification. In addition, other information may be required depending on the targeted FSM hardening technique, as explained in the sequel.

The verification of fault-tolerant FSMs follows the same steps of the verification algorithm shown in Fig. 7. In step 4, specific checks are run to verify the correctness of the target FSM fault-tolerant technique, using the proposed SAT approach. If the FSM is successfully verified, redundancies are removed by merging whenever possible and the merged circuit is checked for equivalence against the original version of the FSM. However, it must be noted that redundancy removal in this kind of techniques is not always straightforward. In addition, there may be subtle differences between the synthesis of hardened and unhardened versions of an FSM due to sequential don't-cares, which can affect the state encoding or the transition logic. Therefore, the CEC approach proposed in this paper is expected to work for fault-tolerant FSMs only if certain conditions are met. Without loss of generality, at least the encoding of every valid state in the original and hardened circuits must coincide.

The first FSM hardening technique considered is the Safe FSM. With this technique, an invalid state detector is incorporated to the FSM logic. When it detects any state belonging to the set of invalid states, $S_{invalid}$, $S_{invalid}$, it triggers a recovery mechanism which puts the FSM into a known safe state, several ways of implementing this approach. On the one hand, it can be implemented in a pure combinational way, with detection and recovery occurring in a single clock cycle. On the other hand, detection and recovery phases can be separated by using a reset flip-flop (or flip-flop chain), delaying the return to S_{safe} for one or more clock cycles. The latter is the solution typically adopted for synthesis tools such as Synopsys Synplify when they have to automatically apply

this kind of hardening. The proposed formal verification tool is capable of handling both approaches.

To prove the correctness of the safe FSM implementation, our algorithm must know which is the S_{safe} state, S_{safe} state, which must belong to S_{valid}. This information must be provided by is verified by proving that $\forall s_i \in S_{invalid}, s_i^{-1} \Rightarrow s_{safe}^0$, where s^j denotes the state of the FSM in timeframe j. If the Safe FSM is implemented with reset flip-flops, then the check is split in two parts: 1) prove that $s_i^{-1} \Rightarrow v^0\left(r_{dff}\right) = 1 \forall s_i \in S_{invalid}$, and 2) prove that $v^{-1}\left(r_{dff}\right) = 1 \Rightarrow s_{safe}^0$, where r_{dff} denotes the active-high reset flip-flop node.

Merging of the safe FSM detection and recovery logic is performed depending on the approach followed for its implementation. If a reset flip-flop was used, the merging is as simple as removing such flip-flop and all its associated logic. On the contrary, if a pure combinational approach is used, the hardening logic would be mixed with the original state transition logic, which makes much more difficult to correctly identify and remove the target nodes. Currently, the proposed formal verification approach only performs merging of safe FSMs when a reset flip-flop is implemented. In order to prove equivalence between the merged and the original FSM, it is necessary that the state transition logic of invalid states is implemented in the same way in both circuits, which depends on the optimizations made by the synthesis tool.

E. HAMMING-3 FSM ENCODING

Hamming-3 FSM encoding is a well-known hardening approach for FSMs. With this approach, the FSM is extended with a certain amount of parity flip-flops. These are assigned in such a way that, for every valid state, its extended encoding has a Hamming distance of at least 3 with respect to the extended encoding of any other valid state. Therefore, this approach can correct any single error affecting the FSM state. This is achieved through an error correction logic.

In addition to the state flip-flops and the original encoding of every valid state, the verification algorithm must know which are the parity flip-flops of the FSM. This information can be provided by the user (in case of manual design) or guessed by name matching (in case of automated implementation by synthesis tools). In principle, the error correction logic of the Hamming encoding should ensure that any single error in the FSM flip-flops will be masked. Therefore, the proposed fault verification algorithm is used over the FSM flip-flops in a first attempt to prove the correctness of the Hamming-3 encoding, like in a LTMR design. However, synthesis optimizations may simplify the error correction logic for some invalid states, which possibly makes the fault verification to fail. In that case, a reduced fault verification is then performed, testing only the bit flips over the extended valid state encodings. In order to do so, first the parity bits corresponding to every valid state must be either provided by the designer or inferred, where the latter approach is done by SAT.

If the Hamming-3 FSM is proven to be correctly hardened, merging is performed by removing the parity flip-flops and simplifying the subsequent error correction logic. Again, to prove CEC between the original and merged netlists, it is required that the encoding of every valid state matches and the state transition logic of every invalid state is synthesized in the same way in both circuits.

V. EXPERIMENTAL RESULTS

The proposed method has been implemented in an in-house tool, named **VeriHard**. The tool is a console application, coded in C++ and prepared to run under Windows and Linux through a command-line interface.

The developed tool uses structural VHDL netlists as inputs. It contains three software modules corresponding with the three main steps:

1) The *parser* module translates the VHDL netlists into AIG format. This first step generates the intermediate files that will be analyzed and processed in the next steps.
2) The *verihard* module formally verifies the correctness of fault tolerance techniques. This step generates another intermediate file, also in AIG format, where the existing redundancies have been removed (a merged netlist). In case a defective mitigation technique is detected, the input vector that shows the defective behavior is reported.
3) The *eqhard* module formally verifies that the processed hardened netlist and the original netlist are functionally equivalent.

Libraries of technology files are required in order to allow the *parser* to translate the structural VHDL netlists into AIG format files. For these experiments, technology libraries for 3rd and 4th generation of Microsemi FPGAs (ProASIC3L/ProASIC3E, IGLOO2, RTG4) have been developed.

Several experiments have been performed in order to test the whole algorithm, including test circuits for every mitigation technique described in section IV as well as cases of defective implementations.

The experiments were applied over a set of test circuits with different characteristics. On the one hand, simple benchmarks have been used to verify the functionality of each software module. These benchmarks include ITC'99 benchmarks (denoted as *Bxx*) [31] and in-house simple circuits (denoted as *Hxx*). On the other hand, industrial cases have been used to prove the applicability of the proposed approach to real examples (denoted *Ixx*).

Every simple circuit has been hardened using several correctly implemented techniques: LTMR, DTMR, BTMR, GTMR, DWC and LTMR selectively applied to a defined set of flip-flops (selective LTMR). Additionally, those circuits that contain FSMs have been hardened by applying different safe encoding mechanisms: using as safe state the reset state, using as safe state the 'others/default' state, and using 3-Hamming encoding. Furthermore, defective versions

TABLE 1. Characteristics of the test circuits.

Circuit	#PI	#PO	#FFs	#Nodes	#Comp.
B05	3	36	111	1109	-
B07	3	8	126	888	-
B10	13	6	75	476	-
H01	3	1	3	14	-
H02	4	4	12	40	-
H03	3	1	9	38	-
H04	3	4	24	175	-
H05	4	8	54	359	1
H06	4	8	69	346	-
H07	3	6	18	107	1
I01	142	7	0	248	5
I02	121	126	0	309	7
I03	61	63	4	203	13
I04	114	90	21	922	10

of each mitigation techniques have also been generated to test the capability of the method to find errors in hardening techniques. Overall, there are from 8 to 14 different hardened versions of every simple circuit, half of which are correct implementations and the other half are defective implementations. With respect to the industrial cases, versions with single mitigation techniques have been generated.

Table 1 summarizes the main characteristics of the test circuits: primary inputs (PI), primary outputs (PO), flip-flops (#FF) and nodes (#Nodes) in the AIG files, and the number of different components (#Comp.) for those circuits that present a hierarchical architecture. For the simple circuits, data correspond to the TMR version, while for the industrial cases, data correspond to the top-level entity of the original version. For example, in the industrial case I01, the design consists of a total amount of 117 component instances what gives an idea of its complexity. Those circuits have not been flattened to simplify the formal verification process by applying it to each component.

The experiments were run on an AMD® Fx™-4300 quad-core processor at 3.8 GHz under Debian GNU/Linux 10. The synthesis of the source code to generate the structural VHDL netlist was performed by using Synplify Premier.

A. EXPERIMENT 1: VALIDATING VERIHARD MODULE

Every version of simple benchmarks and industrial cases with single mitigation or detection techniques was analyzed with the *verihard* software module. It formally verifies if the applied fault tolerance technique does actually prevent or detect the fault propagation through the circuit. Tests were also used to check the approach does return neither false positive results (the circuit is classified as fault tolerant when it is not) nor false negative results (the circuit is classified as not hardened but it is actually fault tolerant).

Results show that the VeriHard tool is able to formally verify the mitigation/detection capabilities of the applied techniques in every case. Furthermore, it always successes in identifying all the defective cases analyzed. In these cases, the tool reports an input vector as counterexample what helps designers to find the mistake in the design. No false positive or false negatives results have been generated.

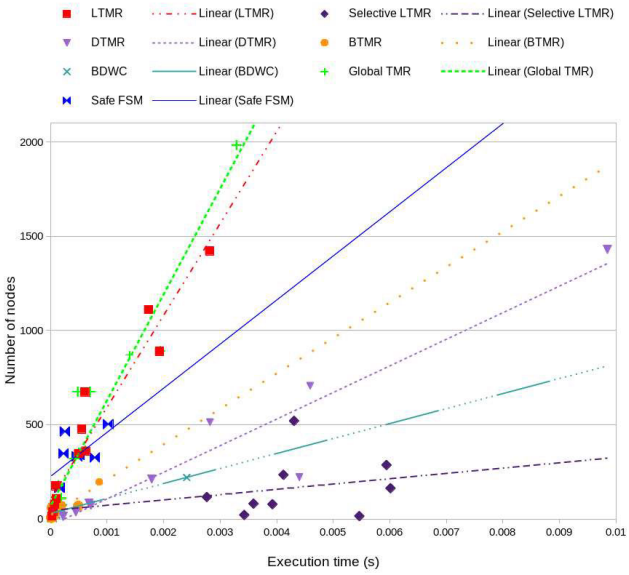

FIGURE 12. Execution time with respect to the number of nodes of some of the studied hardening techniques.

For the 3-Hamming encoding techniques, the performed implementation infers the parity codes for every valid state by using SAT. In one case, this was not enough for guessing a unique combination of parity bits for every valid state. This would be solved if the designer provides those codes as part of the valid state or by using reachability analysis. Anyway, it is a matter related to the implementation but not to the approach itself.

Fig. 12 shows the run time data with respect to the number of nodes in the AIG of each verified case, excluding file input/output time. All the test cases were fully verified in less than 7.5 seconds, being the median of all the execution times 0.5 ms for circuits with a correct implementation of the fault tolerance technique. This result is two orders of magnitude better than the result provided in [27] for similar circuits with TMR (more than 22 minutes).

The regression lines printed in the graph can be used to classify the hardening verification processes according to their verification speed.

From the data in the figure, we can clearly see that local TMR (LTMR) is the fastest verification algorithm, whereas the slowest is the selectively applied LTMR. This contrast can be easily explained. The process of verifying a circuit in which only some FFs have been hardened involves reading and processing constraints (the triplicated flip-flops) stored in an external file, which is a time consuming task.

Fig. 13 shows the run time data with respect to the number of nodes in the AIG of circuits that were classified as incorrect by VeriHard. As in the case of Fig. 12, the regression

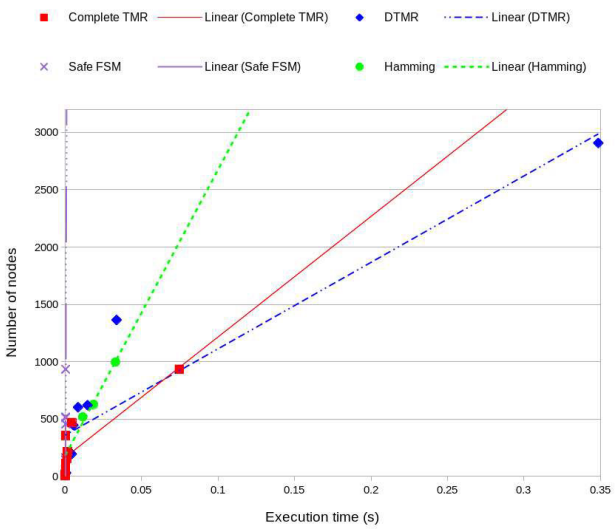

FIGURE 13. Execution time with respect to the number of nodes needed in some hardening techniques to detect incorrect hardening implementations.

lines of this graph reflect the speed of a certain verification algorithm, in this case, they measure the efficiency in detecting incorrect implementations. The maximum execution time was 5.01 seconds, and the median value is 1.5 ms for defective implementations.

B. EXPERIMENT 2: VALIDATING EQHARD MODULE

After verifying the hardening capabilities of the fault tolerance techniques, the verification of functional equivalence between the original and the hardened version was performed.

The equivalence checkings are successful for those TMR, DTMR, BTMR and DWC circuits without FSMs. For circuits with FSMs, the result of the equivalence checking depends on the synthesis over the unreachable states. The synthesizer could optimize in a different way the combinational unreachable states logic for the different versions (original, hardened). Therefore, two circuits will be considered as not equivalent if there is a difference in the logic of the invalid states. This warns designers that the original and hardened circuits could evolve to different invalid states in case of errors.

The equivalence checking process fails when there are logic changes across flip-flops (retiming). These cases are not supported by VeriHard tool yet since the implemented solution is based on Combinational Equivalence Checking.

Fig. 14 shows the run time data with respect to the number of nodes in the hardened AIG for the tested circuits. A logarithmic relationship between the number of nodes and the execution time is expected, and this is supported by the graph. This graph provides information about how the execution time will scale for larger circuits.

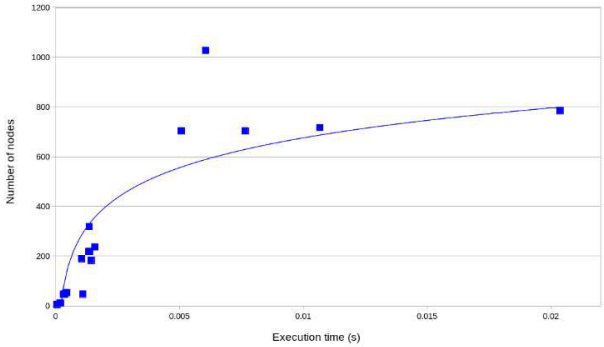

FIGURE 14. Execution time with respect to the number of nodes needed to verify the functional equivalence of correctly hardened designs.

It is noticeable that the execution time for verifying fault tolerance techniques is of the same order of magnitude than the run time for EC, even despite the process being repeated for every fault to evaluate. This is due to the verification strategy used, that is applied locally, as it was explained in section IV-A.

Nevertheless, as explained before, the equivalence checking step is expected to be performed in a hierarchical way, checking the individual hardened components against their original versions and without flattening the whole circuits. With this approach, the size of components is expected to be small, which would reduce the computation efforts. Therefore, this solution would scale well.

VI. CONCLUSION

This work formulates the formal equivalence checking of fault-tolerant designs as a formal verification problem that can be solved with CEC techniques. Furthermore, it applies Boolean reasoning, structural satisfiability, and specific implication rules and error propagation constraints to formally verify fault tolerance in a circuit.

The proposed approach covers two different goals:

- Proof of the mitigation capabilities.
- Proof of the functional equivalence between the hardened version and the original one.

The proposed approach has been implemented in a software tool (*VeriHard*). It is a command-line tool developed in C++, without any external dependencies. It has been successfully proven for a wide variety of fault tolerance techniques (TMR, DTMR, DWC, Safe FSM, 3-Hamming encoding, ...). The execution times are lower than state-of- the-art tools because verification techniques are applied locally and can take advantage of hierarchy. Therefore, this solution is expected to scale well with circuit complexity.

With respect to the equivalence checking of both versions, the current implementation is focused on the use of combinational EC techniques. Future work will be oriented to take advantage of Sequential Equivalence Checking (SEC) for FSMs with different encodings and retiming effects.

REFERENCES

[1] A. Kuehlmann, F. Somenzi, C.-J. Hsu, and D. Bustan, "Equivalence checking," in *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*, L. Lavagno, I. L. Markov, G. Martin, and L. K. Scheffer, Eds. Boca Raton, FL, USA: CRC Press, 2016.

[2] M. Berg and K. A. LaBel, "Verification of triple modular redundancy (TMR) insertion for reliable and trusted systems," in *Proc. MRQW Microelectron. Rel. Qualification Working Meeting*, 2016, pp. 1–26.

[3] G. Beltrame, "Triple modular redundancy verification via heuristic netlist analysis," *PeerJ Comput. Sci.*, vol. 1, p. e21, Aug. 2015.

[4] L. A. C. Benites and F. L. Kastensmidt, "Automated design flow for applying triple modular redundancy (TMR) in complex digital circuits," in *Proc. IEEE 19th Latin-Amer. Test Symp. (LATS)*, São Paulo, Brazil, Mar. 2018, pp. 1–4.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[6] A. J. Hu, "Formal hardware verification with BDDs: An introduction," in *Proc. IEEE Pacific Rim Conf. Commun. Comput. Signal Process (PACRIM) 10 Years Netw. Pacific Rim*, vol. 2, Victoria, BC, Canada, Aug. 1997, pp. 677–682.

[7] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic verification using binary decision diagrams in a logic synthesis environment," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 1988, pp. 6–9.

[8] A. Q. Dao, M. P. Lin, and A. Mishchenko, "SAT-based fault equivalence checking in functional safety verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3198–3205, Dec. 2018.

[9] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2001, pp. 530–535.

[10] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solver," in *Proc. Eur. Design Test Conf.*, Mar. 2002, pp. 142–149.

[11] H. Zhang, "SATO: An efficient propositional prover," in *Proc. Int. Conf. Autom. Deduction*, in Lecture Notes in Artificial Intelligence, vol. 1249, 1997, pp. 272–275.

[12] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 506–521, May 1999.

[13] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. SAT*, in Lecture Notes in Computer Science, vol. 2919. Berlin, Germany: Springer, 2003, pp. 502–518.

[14] F. Lu, L.-C. Wang, and K.-T. Cheng, "A circuit SAT solver with signal correlation guided learning," in *Proc. Eur. Design Test Conf.*, 2003, pp. 892–897.

[15] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1997, pp. 263–268.

[16] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, Dec. 2002.

[17] M. K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik, "Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver," in *Proc. Design Autom. Conf.*, 2002, pp. 747–750.

[18] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," Dept. EECS, UC Berkeley, Berkeley, CA, USA, ERL Tech. Rep., Mar. 2005. [Online]. Available: Available: http://www.eecs.berkeley.edu/~alanmi/publications/2005/tech05_fraigs.pdf

[19] H. Savoj, D. Berthelot, A. Mishchenko, and R. Brayton, "Combinational techniques for sequential equivalence checking," in *Proc. Formal Methods Comput. Aided Design*, 2010, pp. 145–149.

[20] P. Bjesse and J. Kukula, "Automatic generalized phase abstraction for formal verification," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2005, pp. 1076–1082.

[21] H. Savoj, A. Mishchenko, and R. Brayton, "Sequential equivalence checking for clock-gated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 2, pp. 305–317, Feb. 2014.

[22] S.-Y. Huang, K.-T. Cheng, K.-C. Chen, C.-Y. Huang, and F. Brewer, "AQUILA: An equivalence checking system for large sequential designs," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 443–464, May 2000.

[23] G. Cabodi, S. Quer, and F. Somenzi, "Optimizing sequential verification by retiming transformations," in *Proc. 37th Conf. Design Autom.*, 2000, pp. 601–606.

[24] J.-H.-R. Jiang and R. K. Brayton, "On the verification of sequential equivalence," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 6, pp. 686–697, Jun. 2003.

[25] N. Amla, X. Q. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan, "An analysis of SAT-based model checking techniques in an industrial environment," in *Proc. CHARME*, in Lecture Notes in Computer Science, vol. 3725, 2005, pp. 254–268.

[26] M. Handover, "Exhaustively verify SEU mitigation techniques using formal verification," in *Proc. Space FPGA Users Workshop (SEFUW)*, Apr. 2018. [Online]. Available: https://indico.esa.int/event/232/contributions/2141/attachments/1784/2082/Mentor_2018-April_09-Exhaustively_Verify_SEU_Mitigation_Techniques_Using....pdf

[27] A. Traskov, T. Ehrenberg, S. Loitz, A. Ayari, A. Efody, and J. Hupcey, III, "Fault proof: Using formal techniques for safety verification and fault analysis," in *Proc. Design Verification Conf. (DVCon Europe)*, Oct. 2016, pp. 27–32, Paper 10.2.

[28] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. Comput.*, vol. C-25, no. 6, pp. 630–636, Jun. 1976.

[29] W.-T. Cheng, "The BACK algorithm for sequential test generation," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 1988, pp. 66–69.

[30] F. Lu and K. Cheng, "SEChecker: A sequential equivalence checking framework based on *K*th invariants," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 6, pp. 733–746, Jun. 2009.

[31] *ITC'99 Benchmarks*. Accessed: Apr. 3, 2019. [Online]. Available: http://www.cad.polito.it/tools/itc99.html

LUIS ENTRENA (Member, IEEE) received the Industrial Engineering degree from Universidad de Valladolid, Spain, in 1988, and the Ph.D. degree in electronic engineering from Universidad Politécnica de Madrid, Spain, in 1995.

From 1990 to 1993, he was with AT&T Microelectronics, Bell Laboratories, USA. From 1993 to 1996, he was a Technical Project Leader with TGI, Spain. In 1996, he joined Universidad Carlos III de Madrid, Spain, where he is currently a Full Professor, and previously, he was the Head of the Electronic Technology Department and the Director of the Postgraduate Program in Electrical Engineering. He has coauthored over 160 papers in journals and conferences and two patents. His current research interests include online testing, formal verification, fault tolerance, soft error sensitivity evaluation and mitigation, hardware security, and hardware acceleration.

ANTONIO J. SÁNCHEZ-CLEMENTE received the B.S. degree in industrial engineering and the Ph.D. degree in electric, electronic and automation engineering from Universidad Carlos III de Madrid, Spain, in 2011 and 2017, respectively. Since 2017, he has been a Research Member of the Institute of Applied Microelectronics, Universidad de Las Palmas de Gran Canaria, Spain. His work is mainly focused on the development of onboard data processing hardware for space missions. He has been involved in the design of several data compression IP cores belonging to the European Space Agency portfolio and he has coauthored over 30 publications in journals and conferences. In addition, his research interests include fault tolerance, the verification of electronic circuits and systems, and electronic design automation.

LUIS A. GARCÍA-ASTUDILLO received the B.S. and M.S. degrees in industrial engineering and the Ph.D. degree in electric, electronic and automation engineering from Universidad Carlos III de Madrid, in 2017, 2019, and 2023, respectively. He has coauthored over five articles in journals and conferences. His research interests include digital electronics for space applications, radiation hardening by design, error mitigation techniques, and functional verification.

MARTA PORTELA-GARCIA received the degree in electronic engineering from the Complutense University of Madrid, in 2002, and the Ph.D. degree in electrical, electronic and automatic engineering from Universidad Carlos III de Madrid, in 2007, with European mention. In 2003, she joined the Department of Electronic Technology, Universidad Carlos III de Madrid, where she developed research and teaching activities as part of the Microelectronics Design and Applications Research Group, until August 2020. In September 2020, she joined the Arquimea Research Centre (ARC), the private research center of the Arquimea Group. At ARC, she has started a research line focusing on post-quantum cryptography and secure implementations for embedded systems. Her research interests include hardware security, hardware acceleration, and the fault tolerance of digital circuits in space applications.

MARIO GARCIA-VALDERAS (Member, IEEE) received the M.S. degree in industrial engineering and the Ph.D. degree in digital electronics from Universidad Politécnica de Madrid, Spain, in 1997 and 2004, respectively. He has been with Universidad Carlos III de Madrid, since 2001, where he is currently an Associate Professor with the Electronic Technology Department. His research interests include the design of fault tolerant systems for space, hardware acceleration using FPGAs, and digital design techniques and tools.

ALMUDENA LINDOSO (Senior Member, IEEE) received the M.S. degree in telecommunication engineering from Universidad Politécnica de Madrid, in 2001, and the Ph.D. degree from the University Carlos III of Madrid, in 2009. Since 2003, she has been a Professor and a Researcher with the Electronic Technology Department, Universidad Carlos III de Madrid, where she is currently an Associate Professor. Her research interests include hardware acceleration, image processing, and fault tolerance. She has coauthored over 75 publications in journals and conferences. Her current research interest includes the adaptation in terms of reliability of high-performance commercial circuits for aerospace applications.

ROBERTO SARMIENTO is currently a Full Professor in the area of electronic engineering with the Electronics and Telecommunication Engineering School, University of Las Palmas de Gran Canaria (ULPGC), Spain. He contributed to set this school up, he was the Dean of the Faculty, from 1994 to 1998, and the Vice-Chancellor for Academic Affairs and a Staff with ULPGC, from 1998 to 2003. He is the Co-Founder of the Research Institute for Applied Microelectronics (IUMA), where he is also the Director of the Integrated Systems Design Division. He has published more than 100 journal articles and more than 180 conference papers. He has participated in more than 70 projects and research programmes funded by public and private organizations. He has led several projects for the European Space Agency (some of them related to development of IPs for ESA's portfolio of CCSDS 123 and CCSDS 121 standards) and he has collaborations with main companies in the sector, such as Thales Alenia Space, SENER, and GMV. His current research interests include the development of electronics system for on-board satellites and space missions.