

Hardening FPGA-based systems against SEUs: A new design methodology

L. Sterpone

Politecnico di Torino/Dipartimento di Automatica e Informatica, Torino, Italy
Email: luca.sterpone@polito.it

M. Violante

Politecnico di Torino/Dipartimento di Automatica e Informatica, Torino, Italy
Email: massimo.violante@polito.it

Abstract*— SRAM-based Field Programmable Gate Arrays (FPGAs) are very susceptible to Single Event Upsets (SEUs) that may have dramatic effects on the circuits they implement. In this paper we present a design flow composed by both standard tools, and ad-hoc developed tools, which designers can use fruitfully for developing circuits resilient to SEUs. Experiments are reported on both benchmarks circuits and on a realistic circuit to show the capabilities of the proposed design flow.

Index Terms—FPGA, SEU, fault tolerance

I. INTRODUCTION

SRAM-based Field Programmable Gate Arrays (FPGAs) are programmable devices used for different applications, such as signal processing, prototyping and networking. They have fixed number of wires, switches and look-up tables (LUTs): all these components can be programmed by downloading a configuration memory with a proper bitstream, giving an FPGA the capability to implement nearly any kind of digital circuit on the same chip.

The content of the configuration memory is vital for the correct operations of the circuit the FPGA implements. The circuit is indeed totally controlled by the FPGA's configuration memory, which is composed of static RAM cells. When energetic particles hit the surface of the SRAM-based FPGA, they can alter the bits composing the configuration memory, and therefore the circuit the FPGA implements may change its original behavior.

The problem of making SRAM-based FPGAs resilient to SEUs has been attacked in two ways. An earlier solution consisted in developing radiation-hardened FPGAs by resorting to special manufacturing technologies, as well as suitable SEU-immune

architectures. Although effective, this solution is very expensive and therefore it can be exploited only in those applications where cost is not a primary concern (e.g., military applications). The solution that is currently under investigation by many researchers consists in adopting fault-tolerant architectures to implement hardened circuits while using commercial-off-the-shelf FPGA devices. This solution is very attractive since it is potentially able to combine the needed dependability level, offered by fault-tolerant architectures, with the low cost of commodity devices.

As far as fault-removal techniques are considered, several solutions have been investigated in the past years. The one known as Scrubbing consists in periodically reloading the content of the whole configuration memory [1] with the correct bitstream. To minimize the number of needed reconfigurations, which limit the FPGA's availability, a more complex solution uses the Readback and the Partial Reconfiguration processes. Through the Readback, the content of the FPGA's configuration memory is read and compared with the expected one, which is stored in a dedicated memory located outside the FPGA. If a mismatch is found, the correct bitstream is downloaded in the FPGA's configuration memory. During re-configuration, only the faulty portion of the configuration memory is rewritten [1], thus reducing the re-configuration time.

Several architectures were also proposed, which are all based on introducing hardware redundancy in the circuit the FPGA implements. Among the available architectures, Triple Module Redundancy (TMR) is that attracted most of the attention of researchers. TMR can be implemented easily by using three identical logic blocks performing the same task while a majority voter compares their outputs and decides the correct one.

Although TMR is effective in protecting against SEUs the information the circuits elaborate, it showed some pitfalls when the effects of SEUs in the FPGA's configuration memory are considered. Through detailed analyses of FPGA resources [1], and extensive fault-injection experiments [2], we indeed observed that one SEU affecting the FPGA's configuration memory, and in

* based on "A design flow for protecting FPGA-based systems against single event upsets" by L. Sterpone, M. Violante that appeared in the Proceedings of the IEEE Conference on Defect and Fault Tolerance 2005.

particular those portions of the configuration memory controlling routing resources, may originate multiple errors. This phenomenon depends on many factors: the architecture of the adopted FPGA family, the organization of configuration memory bit, the application that is mapped on the FPGA device, and the memory bit affected by the SEU. In our investigations we considered several test circuits designed according to the TMR architecture, and we observed that about 10% of the faults that may affect the configuration memory produce multiple errors that the TMR is not able to mask [2]. As shown in [3], a clever selection of the TMR architecture helps in reducing the number of escaped faults, but it is still unable to reduce them to zero [4]. To cope effectively with SEUs in the FPGA configuration memory, we presented in [5] an approach that makes use of TMR and of a dependability-oriented place and route algorithm, RoRA, to implement cleverly a circuit on SRAM-based FPGAs in such a way that the effects of SEUs are minimized. The approach is very effective in hardening FPGA-based circuits, but it may require high computational times, due to the complexity of executing the dependability-oriented place and route operations for the whole circuit.

This paper presents a new flow that makes use of the results we achieved in [5] to design realistic circuit that are insensitive to SEU effects. The design flow is based on standard tools for design entry, synthesis and design implementation. Moreover, in-house developed tools are used for implementing the TMR architecture, and for identifying circuit area that may behave incorrectly when affected by SEUs. Finally, RoRA is used to re-route critical circuit areas.

The main advantages of the design flow proposed in this paper is that the usage of the CPU-expensive RoRA is limited only to those circuit areas that are critical when affected by SEUs, i.e., only to that subset of interconnections that may corrupt the correct TMR behavior when affected by SEUs. Since this subset is limited (about 8% of total interconnections on the average), most of the circuit implementation may be demanded to more efficient standard design tools (e.g., those provided by FPGA vendors like Xilinx's ISE). As a result, the proposed design flow produces hardened circuits with very low computational efforts: we can thus attack successfully the design of realistic circuits.

This paper is organized as follows. Section II presents the proposed design flow, while Section III, IV, and V present the concepts and the tools our design flow is based on. Section VI reports an experimental evaluation of the proposed design flow, while Section VII draws some conclusions.

II. THE DESIGN FLOW

The design flow we developed adopts standard tools provided by FPGA or EDA vendors for performing the typical tasks needed for transforming a specification coded in HDL, or provided through schematic entry, into a bitstream suitable for being downloaded into the FPGA's configuration memory. Ad-hoc developed tools

are used in combination to standard tools for guaranteeing that the obtained bitstream is resilient to SEUs: they are used to rework only those portions of the circuit that are particularly sensitive to SEUs, i.e., that modify the circuit behavior when affected by SEUs. The rationale behind this approach is that standard tools are very effective in performing synthesis, placement and routing, and produce high quality designs (in terms of clock frequency, area occupation, or power consumption). It is therefore worthwhile to use them to carry out most of the tasks involved in the design process. Moreover, only a limited subset of the whole circuit has to be addressed to cope with SEU-induced problems. For this limited subset it is worthwhile resorting to ad-hoc developed tools that provide robust circuits, although their computational cost is usually higher than that of standard tool. By exploiting standard tools for carrying out most of the design task, while resorting to computational-expensive ad-hoc tools only for a limited subset of the whole circuit, it is possible to minimize the design time, while meeting high dependability levels.

In the following we describe the design flow we developed, which targets Xilinx devices. Although able to communicate with Xilinx's tools, the ad-hoc tools we developed are loosely-coupled with Xilinx's ones, and therefore the proposed design flow can be extended to other devices coming from other vendors.

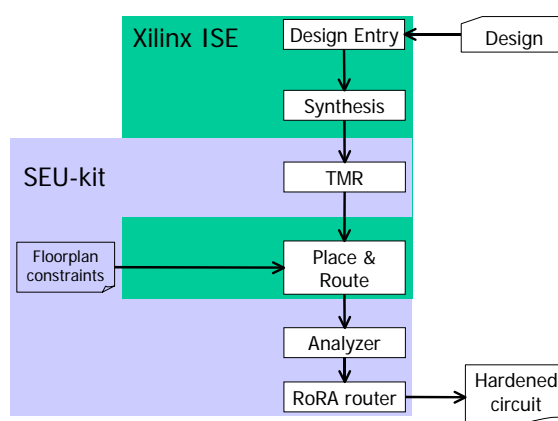


Figure 1. The proposed design flow

The design flow we developed is shown in Fig. 1, and it is composed of two main modules:

1. Xilinx ISE is the collection of Xilinx's design tools that is normally used by designers for obtaining a circuit implemented by any Xilinx FPGA. The collection comprises tools for design entry, synthesis, and place and route. Please note that, although Xilinx's synthesis tool is used, other solutions can be adopted for this purpose (like for example Simplicity's Synplify, or Synopsys's FPGA compiler).
2. SEU-kit is a collection of tools we developed for implementing the TMR architecture, for analyzing the obtained circuit, and running the RoRA router [5]. SEU-kit is composed of the following tools:

- TMR, which takes the circuit as input and produces an equivalent version that is hardened according to the TMR architecture. The circuit (including its input signals) is replicated three times, and a voter is added to decide the output on the basis of the three replicas' outputs.
- Analyzer [6] analyzes the placed and routed circuit, ready for being downloaded on the FPGA, to check whether critical areas exist that may corrupt the correct operations of the TMR architecture when affected by SEUs.
- RoRA router [5] is a reliability-oriented route tool that modifies the critical circuit areas identified by the Analyzer, and that produces a new version of the circuit where all the criticalities have been resolved.

Moreover, SEU-kit includes a set of floorplan constraints that help the ISE's Place & Route tool in producing a more robust circuit against SEU effects.

Analyzer [6] and RoRA [5] are the cores of the proposed design flow, and they are described in Section IV, and V. Before entering in the details of these tools it is however important to give to the reader some background information that are reported in Section III concerning the effects of SEUs affecting FPGAs' memory elements, as well design rules that must be enforced for obtaining SEU-immune circuits.

III. COPING WITH SEU'S EFFECTS IN THE CONFIGURATION MEMORY OF FPGAS

The generic model of an SRAM-based FPGA [4] consists of three kinds of resources: logic blocks, switch boxes and wiring segments. The logic blocks contain the combinational and sequential logic required to implement the user circuit, which is defined by writing the proper configuration memory bits inside the FPGA's configuration memory. Each logic block has a number of input and output signals connected to adjacent switch boxes and logic blocks through wiring segments. The switch boxes are switch matrices, where several programmable interconnect points (PIPs), (i.e., pass transistors) are available, which are controlled by the FPGA's configuration memory. PIPs, called routing segments, provide configurable connections between pairs of wiring segments inside switch boxes.

The resources within an SRAM-based FPGA could be modeled as vertices and edges of a graph [5], which has two types of vertices:

1. Logic vertex: they model the FPGA's logic blocks, each represented as a vertex having I input edges entering the vertex, and O output edges exiting the vertex.
2. Routing vertex: they model the input/output ports of each switch box. For each FPGA's switch box, having I input, and O output, we insert in the graph model I+O routing vertices.

Moreover, the graph has two types of edges:

1. Logic-to-routing edges: they connect one logic vertex to one routing vertex.

2. Routing-to-routing edges: they connect two different routing vertices.

The number of vertices and edges in the graph model depends on the selected FPGA's architecture. Please note that each PIP inside a switch box is modeled as two routing vertices, connected through one routing-to-routing edge. Moreover (as in a real FPGA) a connection between two logic blocks that are exploited to implement a user's circuit may span over several graph's edges. Edges and vertices may be marked, to indicate that the circuit the FPGA implements uses the corresponding FPGA's resource, and un-marked in case they are unused.

The effects induced by SEUs on RAM-based FPGAs have been recently investigated thanks to radiation experiments [7][8][9]. More recently, an analysis that combines the results of radiation testing with those obtained while analyzing the meaning of every bit in the FPGA's configuration memory was published in [1]. From these analyses we observed that SEUs alter both the configurations of logic boxes, and that of switch boxes. In the following, we concentrate our attention only to SEUs that modify the switch boxes. Please note that most of the FPGA's configuration memory is composed of bits controlling switch boxes, and thus the effects of SEUs in this type of resource are particularly relevant.

Within SRAM-based FPGAs, specific configuration memory bits control the routing segments. Depending on the FPGA architecture, it is possible that one and only one configuration memory bit controls two or more routing segment. Thus, a SEU affecting it can modify two or more routing segment provoking multiple errors. To illustrate such a phenomenon, we refer to the graph model we previously introduced where each routing segment is associated to an edge in our graph model. According to our graph model, SEUs affecting the configuration memory bits controlling routing resources can modify two or more edge, thus generating multiple errors.

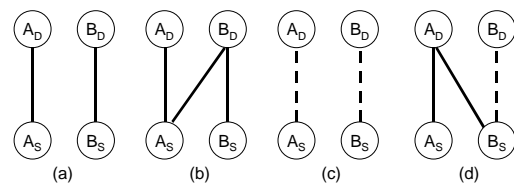


Figure 2. Possible multiple effects induced by one SEU

Given a pair of connections between four routing vertices (AS, AD, BS, and BD), as shown in fig. 2.a, namely AS/AD and BS/BD, we identified the following modifications that could be introduced in the initial graph marking:

1. Short between AS/AD and BS/BD. As shown in fig. 2.b, a new edge is added to the graph that connects either one end of A to one end of B. This effect can happen if:
 - a. AS/AD and BS/BD are composed of graph's edges that correspond to PIPs belonging to the same switch box.

- AS/AD and BS/BD are composed of graph's edges that correspond to routing vertices that can be connected by one wiring segment.
2. Open, which corresponds to the deletion of both edges belonging to AS/AD and to BS/BD, as shown in fig. 2.c
 3. Open/Bridge, which corresponds to either the deletion of the edge AS/AD or BS/BD and to the adding of the edge AS/BD or BS/AD, as shown in fig. 2.d.

In our model, the effect of one SEU in the configuration memory defining the usage of routing resources corresponds to a modification of the graph marking. This means that, depending on the memory bit where the SEU is located, new edges are added the initial marking, or one or more edges are deleted from the marking.

Since one SEU may be responsible for multiple errors, hardening techniques developed accordingly to the single-fault assumption are not adequate to cope with the effects of SEUs in the configuration memory controlling routing resources. As far the TMR architecture is considered, we observed that many situations exist where one SEU provokes multiple routing misbehaviors such that the majority voter is no longer able to mask the effect of the SEU [5]. As an example of this, if AS/AD and BS/BD in Fig. 2.a are two edges belonging to two different modules of the TMR, each SEU resulting in the erroneous configurations of Figures 2.b-2.d will violate the single-fault assumption the TMR depends on.

As a result of the analysis we performed on FPGA architectures, and on the organization of configuration memories, we defined the following rules that must be enforced by the place and route algorithm in order to develop a fault-tolerant graph marking that is resilient to multiple errors:

1. All the circuit modules and connections must be replicated three times.
2. The outputs of the three replicas must be voted according to the TMR architecture.
3. Circuit's modules and connections must be placed and routed in such a way that, given the corresponding graph marking, each new edge that is added (or deleted) to (from) the graph cannot provoke:
 - Short between connections belonging to different circuit replicas.
 - Open or Open/Bridge between connections belonging to different circuit replicas.

In order to obtain circuits immune to Open and/or Open/Bridge effects it is sufficient to place the different circuit replicas in dedicated FPGA's area. We achieve this goal by constraining the adopted place and route tool to partition the FPGA's resources in four sets: one devoted to implement the majority voter the TMR mandates, and the other three devoted to implement the circuit replicas, where each replica is implemented by using the resources of one and only one set.

As far as the Short effect is considered, a dedicated router is needed since the standard routing tools (like the

Xilinx's ISE one) are generally not completely controllable by the users.

IV. THE ANALYZER TOOL

Overview of the tool

The main purpose of the proposed tool, that is introduced in [6] is to analyze the effects SEUs in both the user's memory elements and the FPGA's configuration memory early in the design phase, in particular, as soon as the placed and routed model of the designed circuit is available.

Fig. 1 depicts the architecture of the approach we developed, and it shows the following elements:

- *Static Analyzer*: it is the tool that checks whether the placed and routed circuit is sensitive to SEUs affecting either the memory elements the designers embedded in the circuit, as well as the configuration memory of the SRAM-based FPGA implementing the circuit.
- *Circuit Description*: it is a file containing the structural description of the circuit, which consists of logic functions (either combinational or sequential) and connections between them. Both the logic functions and the connections between them are described in terms of FPGA's resources.
- *Layout Description*: it is a file containing the description of where each resource in the Circuit Description is placed and routed on the FPGA area.
- *Dependability Rules*: it is a database of constraints that must be fulfilled by the placed and routed circuit in order to be resilient to the effects provoked by SEUs. More details about the Dependability Rules are reported in the following sections.
- *Report of Violations*: it is a file that lists all the violations of the Dependability Rules that the Static Analyzer identified. Each entry of the file describes the memory element, and the FPGA's resource responsible for the violation.

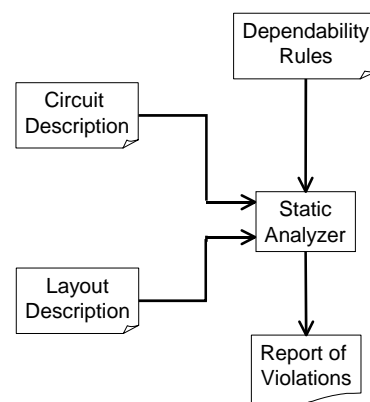


Fig. 1. The proposed approach. The Figure reports the developed Static Analyzer tool, the input information and the output results.

Given the Circuit and Layout Descriptions, the Static Analyzer verifies whether all the constraints described in the Dependability Rules are fulfilled. In case any

violation is found an entry is stored in the Report of Violations file.

Modeling the circuit and its layout

The information stored in the Circuit and Layout Description files models all the details available when the circuit under analysis is placed and routed on the selected FPGA device. To store and handle efficiently this information we defined a graph-based representation that is able to capture all the aspects of the architecture of typical FPGAs, and that can be tailored to devices coming from different manufactures (i.e., Xilinx, Altera, etc.).

While developing the graph-based representation we referred to the generic FPGA's architecture shown in Fig. 2.

The model is common to the architecture of several families of SRAM-based FPGAs [10], and it consists of three kinds of resources: logic blocks, switch boxes and wiring segments.

Wiring segments are group of wiring devoted to transfer information among logic blocks. Wiring segments are organized in the *horizontal plane* through an FPGA from east to west, and *vertical plane* through an FPGA from north to south. Wiring segments are used in conjunction with switch boxes to deliver information between any locations inside FPGAs.

The logic blocks contain the combinational and sequential logic required to implement the user circuit, while the switch boxes and the wirings segments provides the interconnection between them. Each logic resource within a CLB is controlled by m configuration bits of the SRAM-based FPGA's organized regularly in their configuration memory.

Each logic block, that effectively describe the logic function embedded in the Control Logic Blocks (CLBs) within the SRAM-based FPGAs, has a number of input and output signals connected to adjacent switch boxes through wiring segments.

The programmable interconnection network consists of wiring segments that can be connected or disconnected by several *programmable interconnect point* (PIPs). The FPGA's configuration memory controls the PIPs that are organized to form switch matrices and are located inside switch boxes forming the FPGA routing structure.

The basic PIP structure consists of a pass transistor controlled by a configuration memory bit. There are several types of PIPs: *cross-point PIPs* that connect wire segments located in disjoint planes (one in the horizontal plane and one in the vertical plane), *break-point PIPs* that connect wire segments in the same plane, *decoded* and *non-decoded multiplexer (MUX) PIPs*, and *compound PIPs* which consist in a combination of n cross-point PIPs and m break-point PIPs each controlled separately by groups of configuration memory bits.

The routing structure of the entire FPGA could be described by a graph, called *routing graph*, where vertices model logic blocks and switch boxes while the

edges correspond to wiring segments or PIPs. In the model, shown in Fig. 3, we have two types of vertices:

- *Logic vertices*: they model the FPGA's logic blocks.
- *Routing vertices*: they model the input/output ports of the switch box.

We also have two types of edges:

- *Routing edges*: they model the PIPs, each represented as an edge that connects two routing vertices.
- *Wiring edges*: they model the wiring segments. They allow logic vertices to connect to adjacent routing vertices.

The vertices corresponding to the FPGA's resource that are used to implement a circuit are colored. In case fault-tolerant architectures like TMR are used, different colors are used to mark the vertices of each circuit replica, as well as the majority voters.

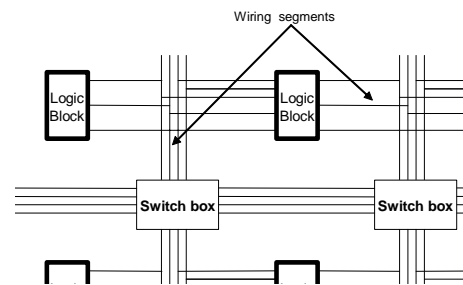


Fig. 2. Generic FPGA architecture model.

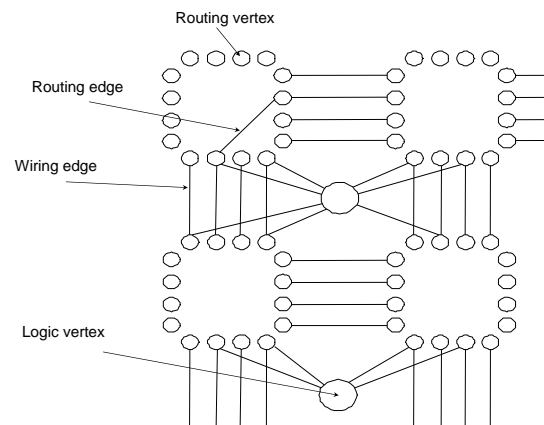


Fig. 3. The routing graph is composed of *logic vertices* modeling logic blocks, *routing vertices* modeling input/output ports of switch boxes, *routing edges* modeling PIP, and *wiring edges* modeling wiring segments.

According to our approach, the designer produces both the Circuit Description and the Layout Description, and then the back-end of the Static Analyzer tools parses the two files and builds the routing graph for the circuit under analysis. The designer obtains the Circuit Description through standard synthesis tools (like Synopsys' FPGA compiler II, Symplicity's Simplify, or Xilinx ISE) starting either from a schematic, or a behavioral model coded in a hardware description language. Similarly, the Layout Description is obtained through the place and route tools for the adopted FPGA device (like Xilinx's par). Moreover, the information coming from the place and route tools is complemented with information

concerning the type of fault-tolerant architecture used to harden the circuit. In the current implementation of our approach we support the TMR [11] design architecture (and its more advanced implementation XTMR [12], only). When a circuit hardened according to one of these techniques is under analysis, the designer specifies in the Layout Description the type of usage of any FPGA's resource allocated to the circuit. For each used resource, the designer specifies whether it is used for implementing a majority voter, or one of the three circuit replicas.

The routing graph is able to convey the information about SEUs' effects, too. The effects of a SEU in the configuration memory are modeled by changing the coloring of the graph vertex corresponding to the resource affected by the SEU. The same approach is used to model the effect of one SEU altering a memory element used by the circuit. Moreover, effects on the routing are modeled through the addition or subtraction of edges (routing or wiring edges) to/from the routing graph.

The association between the elements of the routing graphs and the corresponding bits of the configuration memory mandates a detailed knowledge of FPGA's architecture and of its configuration memory. For this purpose, we studied the architecture of the Xilinx's configuration memory, we decoded it, and we identified the relationship between configuration-memory's bits, FPGA's resources and routing-graph's elements.

Although our work is based on Xilinx's devices, the approach we developed is general, and it can be adopted for modeling devices coming from other manufacturers.

The Static Analyzer

When analyzing a circuit, the Static Analyzer performs three distinct phases.

The first phase reads the Circuit Description and the Layout Descriptions, and computes the routing graph corresponding to the circuit under analysis. The obtained routing graph is colored according to the information contained in the Layout Description file. Since the circuit is designed according to the TMR principle, three different colors are used for vertices belonging to each TMR replica. Moreover, an additional color is used for the vertices implementing majority voters.

The second phase checks the effects that may be generated by SEUs that affect the resources corresponding to the graph's vertices. This phase analyzes all the graph's vertices, and it consists of three steps:

1. The analyzed vertex is considered as SEU sensitive.
2. A SEU propagation tree is generated. The tree contains all the paths stemming from the SEU sensitive vertex to all the graph's vertices corresponding to the circuit's outputs.
3. If the leaves of the propagation tree include more than one graph coloring (other than that of the SEU sensitive vertex and that of the vertices implementing the majority voter), the resources

corresponding to the analyzed vertex are added to the Report of Violations. Indeed, when this condition is met we have that the SEU effects propagated to two or more circuit replicas, and therefore the TMR principle is no longer enforced.

The third phase checks the effects provoked by SEUs affecting those configuration memory bits that control the routing resources, and it is executed similarly to the previous one. By exploiting the known relationship between configuration-memory's bits controlling the routing resources and the routing-graph's edges, we compute a new routing graph for each of the possible SEU affecting the routing resources. Then, we compute the propagation tree for each graph vertex in the faulty routing graph, allowing the propagation of the SEU effects in all the possible ways to the circuit's outputs. If the leaves of the propagation tree include more than one graph coloring (other than that of the SEU sensitive vertex and that of the vertices implementing the majority voter), the resources corresponding to the analyzed vertex are added to the Report of Violations.

When analyzing the user memory, the Static Analyzer performs three distinct phases.

The first phase is the same performed while analyzing the configuration memory.

The second phase verifies that the user memory is implemented according to the TMR principle. This phase is performed in two distinct steps:

1. Each cell of the user memory is inserted in one of three lists (one for each circuit replica).
2. The three lists are compared, and if one of the user memory's cells is not contained in all the three lists, it is marked as SEU sensitive.

The third phase checks the effect provoked by SEUs affecting those bits belonging to the user memory. This phase is performed in two different steps:

1. Every user memory's cell marked as SEU sensitive by the previous phase is taken into account.
2. A SEU propagation tree is generated. Starting from the vertex connected to the considered user memory's cell, it spans over different routing graph's resources. If the leaves of the propagation tree include more than one graph coloring (other than that of the SEU sensitive user memory's cell), the user memory's cell is added to the Report of Violations.

At the end of the execution of the Static Analyzer the Report of Violations file contains the list of FPGA's resources (e.g., PIPs, Logic Blocks, etc) and the locations within the configuration memory of every SEUs that provoke erroneous behaviors of the circuit the SRAM-based FPGA implements.

V. THE RORA ROUTER

The RoRA router, which was introduced in [5] to solve the problem of the Short effect, is based on the approach developed for the Pathfinder negotiated congestion

algorithm [11][12]. Basically, the RoRA router works on the graph model we developed, and it routes each connection between two logic vertices through the shortest path it can find. The path is composed of routing vertices, routing-to-routing edges, and logic-to-routing edges. During path selection, RoRA labels dynamically the graph's routing vertices, in such a way that it avoids the instantiation of two connections that may be subject to Short effects.

RoRA works on the same circuit description Analyzer adopts, and during its operations it labels each routing vertex as free, used or forbidden with the following meanings:

1. Free: the routing vertex is not used by any connection.
2. Used: the routing vertex is already used by a connection.
3. Forbidden: a routing vertex RV is forbidden if and only if:
 - It belongs to set S_i ($R \in S_i$) where $i=1, 2, 3, 4$.
 - At least one routing-to-routing edge, or one logic- to-routing edge exists between RV and another vertex RV' belonging to S_j ($R \in S_j$), where $i \neq j$.

If RV is added to the circuit marking and a SEUs results in a Short between RV and RV' we have two different modules of the TMR having two connections shortened together, and thus the two modules will work erroneously.

Routing algorithms are commonly based on the construction of a routing tree, where the root is associated to the logic vertex correspondent to the source of the connection, while the leaves are associated to the logic vertices correspondent to the destination of the connection. The maze routing [13] is usually used for this purpose. The routing tree is expanded progressively: starting from a tree composed of the source vertex, only, new vertices are added, until all the destinations of the connection have been added to the tree.

The RoRA router algorithm uses the maze routing approach with a fundamental difference in the creation of each routing tree. The key step of the RoRA router is performed during the routing tree expansion, where those vertices that are labeled as forbidden are not used. Moreover, the set of forbidden vertices is updated after the creation of the new routing tree.

VI. EXPERIMENTAL ANALYSIS

The purpose of this section is to evaluate the effectiveness of the proposed design flow in designing circuits that are hardened against SEU effects. In particular, we focused on the effects of SEUs in the FPGA's configuration memory since it is the most critical aspect. The number of bit devoted to the configuration memory is indeed much higher than that devoted to the user memory (for implementing registers, or memory blocks). As a result, an SEU is more likely to happen within the configuration memory than in the user memory.

Two experiments were performed. The first one consisted in designing three simple circuits according to the approach presented in [5], and according to the design flow presented in this paper. By comparing the attained results we can quantify the improvements that our new design flow allows with respect to the original solution we presented in [5]. The second experiment consisted in designing a realistic circuit with the intent of analyzing the viability of our design flow in attacking the design of a hardened real-life design.

A. Evaluating the proposed design flow

To evaluate the capability of the proposed design flow we considered three purely combinational case studies: an adder with two 8-bit wide operands, an adder working on two 16-bit wide operands, and a multiplier with two 8-bit wide operands. We also performed some experiments on two FIR Filters with 8 and 32 taps in order to evaluate the proposed approach capability on sequential circuits. We designed it according to two approaches: the one presented in this paper, and that presented in [5].

TABLE I.
COMPARING EXECUTION TIMES

Circuit	[5] [sec]	Proposed approach [sec]
Add8	105.0	56.6
Add16	556.2	101.0
Mul8	1,312.2	265.4
FIR 8-taps	3,584.9	361.6
FIR 32-taps	3,677.1	365.4

The device we used in our evaluation experiments is a Xilinx Spartan® XC2S30PQ144, whose configuration memory is composed of 336,768 bits organized in 1,165 frames of 288 bits each. The configuration memory controls 132 I/O blocks and an array of 12x18 slices.

Fault injection experiments [3] showed that in both cases we obtained hardened circuits with respect to SEUs affecting the configuration memory of the adopted FPGA. Conversely, as results reported in Tab. I show, significant differences in execution times were observed. These results show that demanding the synthesis, and place and route operations, of most of the circuit to standard tools, while relying to reliability-oriented tools for addressing critical areas only, is far more efficient than routing the whole circuit with the approach presented in [5]: the execution times are indeed reduced by a factor ranging from about 2 to a factor of about 6.

B. Design of a realistic circuit

To evaluate the effectiveness of the proposed approach on a realistic FPGA-based design we considered an IP-core that implements the Control Area Network protocol. The IP-core was developed and validated in [14] and it is compliant with the CAN protocol specifications: it is thus a good example of a realistic FPGA-based design.

As a first step of this experiment, we designed the hardened version of the adopted IP-core according to the design flow proposed in this paper. To show the versatility of our approach, we considered several different FPGA families all coming from Xilinx: the

Spartan II, the Virtex I, the Spartan 3, and the Virtex II. Similarly to what is normally done during the development of a realistic design, while selecting the FPGA device to use, we opted for the smallest device able hold our design. Table 2 reports the selected FPGAs, the percentage of FPGA’s resources that are used by the TMR-version of the CAN controller, as well as the circuit’s maximum frequency. Please note that for all the devices we considered, the IP-core uses at least the 98% of the total available resources. Although the selected FPGAs are almost full, RoRA was still able to find a different routing for the critical signals. Moreover, we analyzed the circuit’s frequency (by exploiting the Xilinx’s timing analyzer tool) before and after the execution of RoRA, which reworked the optimal circuit implemented produced by Xilinx’s ISE to make it robust against SEUs. For all the considered architectures, we always observed negligible reduction (less than 1%) of the maximum frequency. These results suggest that the proposed design flow can be used effectively to attack the design of realistic circuits, even in those case were few resources are available for reworking the circuit produced by the standard design tools.

In order to quantify the effectiveness of the proposed design flow in protecting FPGA-based systems over a simpler approach where only the TMR architecture is adopted, we designed the selected IP-core according to the TMR, only. In this case, only the standard Xilinx ISE tools are used in combination with the TMR tool we developed, while the other components of the SEU-kit are not exploited (i.e., floorplan constraints, Analyzer and RoRA are not used). This scenario depicts the case of designers willing to adopt the TMR architecture and resorting only to standard tools for circuit design, plus ad-hoc tool for replicating the circuit and adder the needed majority voter.

TABLE II.
CHARACTERISTICS OF THE FPGA DEVICES USED

FPGA device	Resource occupation [%]	Frequency [MHz]
Spartan II XC2S200	98	225
Virtex I XCV200	98	174
Spartan 3 XC3S200	99	230
Virtex 2 XC2V250	100	180

Table 3 reports the number of faults that escape the TMR architecture obtained by using standard tools only, in comparison with the figures attained for the IP-core designed according to the approach presented in this paper (for simplicity, only the figures concerning the Spartan II and Virtex I architectures are reported. Similar figures were observed for the other architectures). As the reader can observe, the approach presented in this paper produced fault- tolerant circuits, while the TMR alone was not able to achieve this goal. Table 3 also reports the CPU time for completing the design of the circuit according the approach presented in this paper. This figures indicate that a designer can obtain a validated and hardened design in less then 1 our and a half.

TABLE III.
COMPARING EXECUTION TIMES OF REAL DESIGN

FPGA device	Number of escaped faults		CPU time [min]
	Proposed Approach	TMR Approach	
Spartan II XC2S200	0	263	83
Virtex I XCV200	0	265	86

VII. CONCLUSIONS

This paper presented a new design flow for supporting the design of real-life FPGA-based circuits that are hardened against SEUs. The adoption of the TMR architecture, as well as of a dependability-oriented strategy for placing circuit’s components, and for routing circuit’s interconnections guarantee that the obtained circuits are resilient to SEUs affecting the circuit’s memory elements, as well as the FPGA’s configuration memory.

The efficiency of the proposed design flow was assessed by experiments performed on both benchmark circuits and on a realistic circuit implementing the CAN protocol.

As future work, we are planning to evaluate the impact of the proposed approach on the performance in terms of power.

REFERENCES

- [1] Xilinx Application Notes XAPP216, “Correcting Single-Event Upset Through Virtex Partial Reconfiguration”, 2000
- [2] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Cerchia, A. Paccagnella, M. Rebaudengo, M. Sonza Reorda, M. Violante, P. Zambolin, “Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA”, IEEE Design Automation and Test in Europe, 2004, pp. 188-193
- [3] P. Bernardi, M. Sonza Reorda, L. Sterpone, M. Violante, “On the evaluation of SEUs sensitiveness in SRAM-based FPGAs”, IEEE International On-line Testing Symposium, 2004, pp.115-120
- [4] F. Lima Kastensmidt, L. Sterpone, L. Carro, M. Sonza Reorda, “On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs”, IEEE DATE, 2005, pp. 1290-1295
- [5] M. Sonza Reorda, L. Sterpone, M. Violante, “Multiple errors produced by single upsets in FPGA configuration memory: a possible solution”, IEEE European Test Symposium, 2005, pp. 136-141
- [6] M. Sonza Reorda, L. Sterpone, M. Violante, “Efficient Estimation of SEU effects in SRAM-based FPGAs”, to be presented at the IEEE International On-line Testing Symposium, 2005
- [7] J. Rose, A. El Gamal, A. Sangiovanni-Vincentelli, “Architecture of Field-Programmable Gate Arrays”, IEEE proceedings, vol. 81 no.7 July 1993, pp. 1013-1029
- [8] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, J. Fabula, “Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Reconfigurable Computing”, presented at the IEEE Nuclear and Space Radiation Effects Conference, July 2000
- [9] M. Bellato, M. Ceschia, M. Menichelli, A. Papi, J. Wyss and A. Paccagnella, “Ion Beam Testing of SRAM-based

- FPGA's", IEEE Radiation Effects Data Workshop, July 2002
- [10] J. Rose, A. el Gamal, A. Sangiovanni Vincentelli, "Architecture of Field Programmable Gate Arrays", Proceedings of the IEEE, Vol. 81, No. 7, July 1993
- [11] D. K. Pradhan, "Fault-Tolerant Computer System Design", Prentice Hall, ISBN 0-13-057887-8
- [12] "TMRTool User Guide", Xilinx User Guide UG156, 2004
- [13] M. Ceschia, M. Violante, M. Sonza Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, A. Candelori, "Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs", IEEE Transaction on Nuclear Science, Dec. 2003, Vol. 50, No. 6, pp. 2088-2094
- [14] V. Betz and J. Rose, "Directional Bias and Non-Uniformity in FPGA Global Routing Architectures", ICCAD, 1996, pp. 652-659
- [15] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA", IEEE Trans. On VLSI, Dec. 1995, pp. 473-482
- [16] C.Y. Lee, "An algorithm for Path Connections and Its Application" IRE Trans. Electronic Computers, vol. 10, pp. 346-365, Sept. 1961
- [17] J. Perez, M. Sonza Reorda, M. Violante, "Accurate Dependability Analysis of CAN-based Networked Systems", 16th IEEE Symposium on Integrated Circuits and Systems Design, 2003, pp. 337-342