Automated Design Flow for No-Cost Configuration Error Detection in SRAM-based FPGAs

M. Ben Jrad, R. Leveugle
Univ. Grenoble Alpes, TIMA, F-38031 Grenoble
CNRS, TIMA, F-38031 Grenoble
46 Avenue Félix Viallet - 38031 Grenoble Cedex - FRANCE
Mohamed.Ben-Jrad@imag.fr, Regis.Leveugle@imag.fr

***Abstract*— Soft errors in the configuration memory of SRAM-based FPGAs cause significant and remanent application disturbances. However, classical mitigation techniques based on massive redundancy are too costly for most applications. The method presented in this paper is based on selective redundancy in partially used LUTs. It can be applied so that no hardware is added at the system level and it has been automated in standard design flows for Xilinx and Altera families. The detection of soft errors in the configuration is performed within one clock period. Experimental results on benchmark implementations are discussed, showing the good ratio between coverage and block-level overheads, with almost no impact on power and delay. The differences between applying the approach to Virtex V and Stratix IV devices are also discussed.**

Keywords—SRAM-based FPGA; soft errors; multiple errors; dependability; configuration error detection

I. INTRODUCTION

FPGAs are increasingly used in all applications, including some with reliability or availability constraints. Compared with other configuration technologies, SRAM-based FPGAs provide a large amount of configurable logic, high flexibility thanks to reconfigurability and low cost. However, due to the large size of the configuration SRAM and to the aggressive technologies used to manufacture such FPGAs, these devices are very sensitive to soft errors. Soft errors are spurious modifications of bits due to e.g., radiation effects or electromagnetic interferences. They do not result in any damage in the circuit [1] but can cause a significant disturbance of the service provided to the application user. Initially a concern in space or for aeronautics, soft errors are with up-to-date technologies a real concern even at sea level. In ASICs, such errors occur in user data or circuit state, but have mostly transient effects. In SRAM-based FPGAs, such errors occurring in the configuration memory have a different impact since they remain at least until the next reconfiguration, and in some cases require a power-up cycle to recover a correct behavior. Such "remanent" errors can modify the implemented function, either by changing the elementary logic functions or due to interconnection corruptions.

When an application has dependability constraints, soft errors in the configuration memory must therefore be mitigated. Even when the product has no critical function, an improved reliability can lead to a competitive advantage. Intentional fault-based attacks may also be a concern in secured applications; but not all applications with some security concerns can afford high costs for protection. So many applications can benefit from low cost improved robustness in case of soft error occurrence in the configuration SRAM. The work presented here is focused on configuration errors since they have a much higher probability in the targeted devices than errors in user registers due to the huge number of bits in the configuration SRAM. Errors in user registers can be mitigated with the same architectural approaches as in ASICs.

Many works have been published on the mitigation of configuration errors. Almost all these works are done for critical applications and seek a perfect detection or tolerance of some error types (in most cases, Single Event Upsets, or SEUs, corresponding to a single bit-flip). The price to pay is in general very high in terms of resources and power since the approaches are mostly based on massive redundancy (in most cases triplication, or TMR). Manufacturers have also included some mitigations in some of their devices, e.g., some level of ECC encoding or CRC of the configuration. These detections, when available, are limited in terms of error multiplicity and/or error detection latency (in case of periodic checking).

Our approach has different aims. The goals are:

- Applicability to standard commercial FPGAs,
- Low overheads at block level and no hardware overheads at system level,
- Partial detection of configuration errors with potentially high multiplicity, the achieved detection level being defined according to the device and application characteristics in order to satisfy the previous objective,
- Automation in standard design flows.

The main constraint is therefore to avoid any system-level overheads, and then achieve the maximum detection level. In consequence, the proposed method is not targeting high-reliability or high-availability applications, but cost-sensitive applications seeking the best possible dependability. To achieve these goals, we have developed a specific tool based on the maximum exploitation of unused resources and we

978-1-4799-2079-2/13/$31.00 ©2013 IEEE

discuss in this paper the results obtained for two different FPGA families, i.e. Xilinx Virtex V and Altera Stratix IV.

Section II briefly summarizes the approaches previously presented in the literature to mitigate configuration errors and justifies the error model we will focus on. Section III details the proposed methodology and presents the automated design environment developed for the Xilinx and Altera targets. Section IV discusses the main results.

II. Previous Works and Error Models

A. State-of-the-art

Our goal is to improve design dependability on mainstream commercial FPGAs. In consequence, approaches based on hardened manufacturing technology, hardened memory point structures or dedicated error detection included in the basic programmable cells, are irrelevant since no physical or structural intervention is possible in our context.

As previously mentioned, some configuration error detection mechanisms have been implemented by the manufacturers but not in all families. In addition, these mechanisms suffer from some limitations about detection latency and/or coverage of multiple erroneous bits. There is therefore an interest in adding complementary detection mechanisms during the design phase.

System-level detection strategies can be based on periodic scrubbing and reconfiguration of erroneous parts of the configuration bitstream. In some FPGAs, no readback capability is implemented to avoid cloning; in that case only a full, periodic, reconfiguration can be achieved with a noticeable impact on system performance. In all cases, such a strategy implies a large detection or correction latency. What will be proposed is complementary and may be used to trigger reconfiguration when necessary, with a short latency after error occurrence.

Many works have considered design architecture modifications to avoid effects of SEUs or some multiple-bit errors [2-8]. They are mainly based on massive redundancy with in some cases complementary optimizations or mechanisms. The coverage of errors is very good but at the expense of huge overheads in terms of resources and power. Such approaches are therefore not an option for the type of applications we target.

Recently, a few approaches have been proposed based on the use of resources that are available in the FPGA but are useless for a given design [9-12]. In [9-10], the authors focus on a resynthesis algorithm and on masking some SEUs through AND and OR combinations in order to increase the Mean Time To Failure. The approach is evaluated only theoretically. In [11] the idea was mainly to exploit unused resources to achieve a low-cost implementation of fine-grained checking mechanisms based on local duplication of functions identified as the most critical ones. The feasibility was demonstrated on a case study implemented manually in an AT40K device. In [12], the use of hardwired resources is proposed to provide fast and area efficient fine-grained error detection in Xilinx devices.

The work presented in this paper goes further by showing the feasibility of improved dependability at no cost, within an automated design flow based on standard tools, for several device families.

B. Target Error Model

When defining a target error model in FPGA configuration, two characteristics must be considered: the location of the errors and their multiplicity.

Considering the elements used to implement the logic functions, errors may be located in memory cells configuring either the functions themselves (Look-Up tables, or LUTs), or the interconnections between the functions.

We have partly based our target error model on previous experiments made using lasers perturbing a Xilinx Virtex II component [13]. In such a component, less than 20% of configuration bits are in LUTs, so a large percentage of errors are in bits configuring interconnections. However, many errors in interconnection configuration do not lead to application failure. The main reasons are:

- An error in a bit configuring interconnection segments does not necessarily change the segment states, especially when these segments were initially unconnected. The reason for this particular technology is that most segments need two specific configured bits to be connected.
- An extra segment connected due to the errors has no functional effect if it does not create some short-circuit with other connected segments.

In consequence, on a set of more than 5400 error patterns (i.e., simultaneously modified bits) recorded after single laser shots, it was found that less than 4% of the modifications in interconnection bits were actually critical. On the opposite, bits modified in LUTs used by the application are all potentially critical, at least in some phases of the application execution, and such modifications can be identified in a simpler way than errors in interconnections. We will therefore focus in the following on errors in the LUT configurations (for LUTs actually used in a given design).

In the same set of error patterns, it was identified that only 4.44% of the error patterns were SEUs, and SEU patterns were less frequent than patterns with more than 30 erroneous bits.

These results were obtained in the context of intentional attacks and are not representative of all possible types of perturbations. But they demonstrate that errors with high multiplicity can actually occur, and that errors in LUT configurations are the most critical, at least for this type of FPGA. In the following, we will therefore aim at detecting classical errors such as SEUs, but also multiple errors, and we will focus on LUTs.

III. Proposed Methodology and Tools

A. Basic Concepts of Detection Approach

The basic principle is to insert a limited amount of redundancy so that effective errors occurring in the configuration can be detected and an alarm signal can be

generated. This alarm can be used in many ways, directly inside the FPGA to trigger e.g., an endo-reconfiguration, or at the higher hierarchical level to trigger e.g., a global system recovery or reset.

The proposed redundancy only seeks detecting errors in configurable logic blocks; embedded macros (hard processor IPs, embedded memories or specific DSP primitives) are not considered here. We recall that the goal is to improve dependability as much as possible at no cost; there is no claim to cover errors in all elements.

Design modifications are made by adding in the netlist the maximum possible redundancy leading to avoid actual costs. The first step is to identify parts of the combinatorial primitives unused by the nominal implementation of the application. These unused parts are transformed into spares in order to achieve partial function duplication for free. The next sections will show on two different architectures how this principle can be applied.

The approach can detect single-bit errors (SEUs) in the LUT configuration memory but also many multiple-bit errors either when grouped in one LUT or when the erroneous bits have uncorrelated roles in the device configuration. Also, the probability to have equivalent error patterns in the initial function and in its duplica, avoiding detection, is very small. If considered useful for a given type of architecture, a dual implementation of the initial function and its duplica may be easily done without additional cost; however, the probability of detection would only be improved in the specific case of intentional perturbations performed with a very good knowledge of the attacked area.

B. *Implementation on Xilinx Virtex V Architecture*

LUTs in Virtex V components have 6 inputs, but are in fact composed of two 5-input LUTs with connected inputs, as illustrated in Figure 1. In many cases (90% of the cases for a design example used during a preliminary evaluation), only one 5-input LUT is actually used after place and route. The idea is therefore to use the second 5-input LUT as a free replica of the function. The second output is used to compare the duplicated computation with the one used by the application.

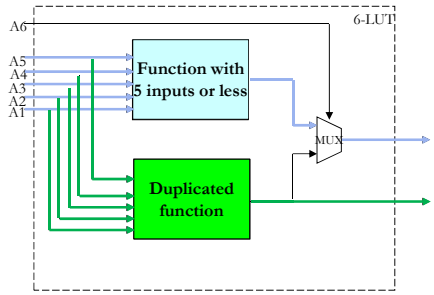

Fig. 1. Partially used 6-input LUT and replica insertion in Virtex V.

C. *Implementation on Altera Stratix IV Architecture*

In the Stratix IV devices, combinatorial functions are implemented in blocks called ALMs (Adaptive Logic Modules). Each ALM has 8 inputs with a "fracturable" LUT that can be divided into two adaptive LUTs (ALUTs) using Altera's patented LUT technology. Each ALM is capable of implementing:

- A 7-input logic function in a selected set,
- Any 6-input logic function,
- Two independent outputs of multiple combinations of smaller LUT sizes for efficient logic packing (two independent functions consisting of smaller LUT sizes, such as two independent 4-input LUTs),
- Complex logic-arithmetic functions without additional resources.

Quartus II software design suite takes advantage of this fracturability and optimizes it for performance, efficiency, power, and area. However, we have observed on our benchmarks that on an average 35% of the available resources remain unused, because no new sub-function with the same inputs has to be implemented. In this case, the fracturability can be used to implement the duplica since the same inputs are required. This is illustrated in Figure 2.

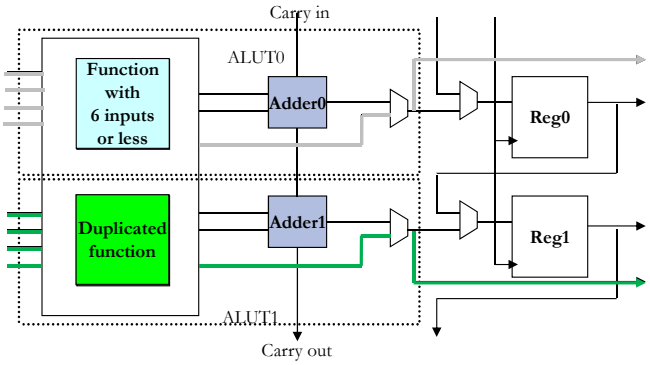

Fig. 2. Partially used ALM and replica insertion in Stratix IV.

D. *Generation of Alarm Signal*

Extra logic elements have to be used in order to compare the outputs of duplicated functions, and then to combine all the comparator outputs into a global alarm signal. The principle is illustrated in Figure 3.

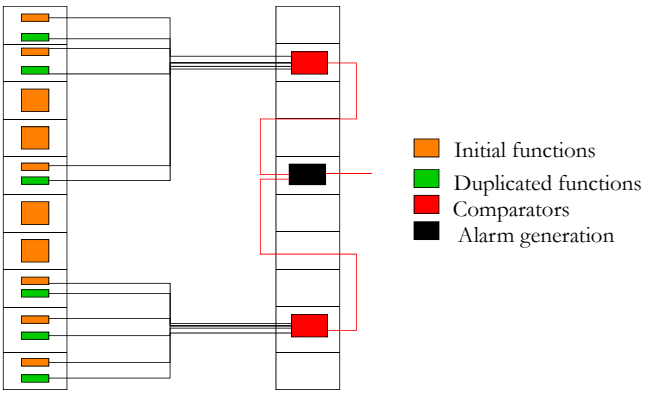

Fig. 3. Principle of global alarm signal generation.

E. Conditions for No-Cost at System Level

At block level, overheads are due to the comparators and to the logic tree used to generate the alarm signal. The minimal number of extra LUTs or ALMs can be easily computed. One is sufficient to implement a comparator for 3 duplicated functions, or to combine 6 comparator outputs.

Due to the discrete size of the devices in a given FPGA family, a given number of LUTs or ALMs are not useful in many cases. Some of them cannot be used due to e.g., routing congestion. But often many remain available. The idea is to limit the number of duplicated LUTs to the number of available elements to generate the alarm signal, so that the more dependable version of the design can be implemented on the available FPGA resources, avoiding any hardware cost at system level. The achieved error coverage is a consequence of available resources. For the results given in this paper, we limit the number of duplicated LUTs to the partially used LUTs. If more LUTs remain available, it would be possible to also duplicate fully-used LUTs. On the opposite, in case the overheads for the alarm generation are two large, it is possible to duplicate less LUTs to implement less comparators. In that case, the duplicated functions must be selected among the most critical ones, as defined by the designer or by a specific tool such as SEFEA-ProD [14].

Many other trade-offs are possible, that cannot be discussed in this paper. In particular, only a global alarm signal is generated in the examples presented in this paper. In case more unused outputs are available, or in case of an internal use of the alarms for an endo-reconfiguration, it is possible to generate local alarms. The two advantages are less resources required for the alarm generation, and a better diagnostic of the erroneous area to reconfigure.

We will focus in the following on the addition of redundant logic at block-level. The lowest the overheads are at block-level, the best coverage can be achieved at no-cost at system level. Also, it is easiest to adapt the resource budget of other less critical blocks in the same system to compensate.

F. Generic Design Flow

The generic design flow is illustrated in Figure 4 and is common to all targets, although the tools actually used at each step depend on the selected FPGA family. A specific tool has been developed to automate the insertion in the netlist of the redundant logic and the generation of error signals. In addition, this tool generates placement and routing constraints to optimize the implementation on the selected target. The synthesis and place & route tools are those provided by the FPGA manufacturers, without specific synthesis or routing.

G. Design Flow Constraints on Xilinx Virtex V

In order to avoid actual overheads when performing duplication, even when limited to partially-used LUTs, it is necessary to ensure, during the placement of the functions, that each duplica is actually associated with its initial function, in the same 6-input LUT. This is the role of the place & route constraints generated by our dedicated tool.

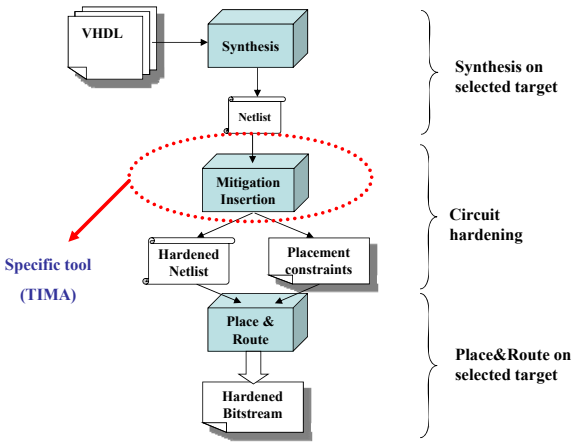

Fig. 4. Generic design flow.

H. Design Flow Constraints on Altera Stratix IV

The automation of the process was much more difficult on Stratix than on Virtex. The main reasons are:

- the Quartus suite does not allow generating a netlist that can be modified to insert the duplica and the error generation logic. Several approaches have therefore been explored. The first one consisted in using Quartus functions (through a TCL script) to insert the required logic; this approach was not successful, due to errors generated during the process, after insertion of the duplica and before the insertion of the comparison logic. It was therefore necessary to use another synthesis tool (Precision from Mentor Graphics in our case) to generate a netlist. This netlist is modified by our home-made tool and then imported as an EDIF netlist in Quartus. The placement and routing constraints have however to be generated as a TCL command script for Quartus and the implementation must take into account that Quartus launches a new synthesis after importing the EDIF netlist.

- the optimization strategy in Quartus leads to a tough control of the placement and routing. After adding the duplica, an automatic placement does not lead to the expected results because the duplicated functions are not well associated. If constraints are added only on the duplicated functions, then the placement of the other cells is completely modified and the overheads are noticeably increased. It is therefore necessary to put constraints on the placement of almost all the cells.

Analyzing the synthesis results on Quartus and Precision, it should be noted that one tool or the other may have the advantage for a given design, and no clear advantage is noticed on an average. However, the results with Precision on an average allow us to implement more "free" replica, and therefore to increase the fault coverage.

IV. Case Studies and Result Discussions

A. Implemented Blocks

Experiments have been carried out mainly on the ITC'99 benchmark set. Some other design examples have also been considered, including typical processing blocks (FIR filter and

a sequential multiplier) and open-source Sparc v8 microprocessors (Leon2 and Leon3).

B. *Implementation Results in Xilinx Virtex V*

Some results on Virtex V are summarized in Figure 5. Only partially-used LUTs have been duplicated, leading to a duplication rate between 37% and 100%. Block-level overheads correspond to the error signal generation and are between 15% and 50%. For the two largest examples (microprocessors Leon2 and Leon3) it was possible to detect errors in more than 40% of the LUTs with less than 20% of additional resources. Such block-level overheads are compatible with avoiding any hardware overheads at the system level in many cases. Fault injections of single and multiple errors in the LUT configurations have validated the detection mechanisms, with a coverage that is, as expected, directly related to the percentage of duplicated functions. MBUs, i.e. multiple-bit errors in a configuration word, were also injected in the full configuration bitstream of Leon3, perturbing both LUTs and interconnections. The detection coverage was found to be 10.5% for 2-bit MBUs, but increases with multiplicity and 34.96% of 8-bit MBUs were detected. The proposed approach is therefore able to detect a significant percentage of configuration errors even when not limited to LUTs.

C. *Implementation Results in Altera Stratix IV*

Results on Stratix IV are summarized in Figure 6 for the ITC benchmarks. Only partially-used ALMs have been duplicated, leading to a duplication rate between 5% and 88%. Block-level overheads are due to the error signal generation and are between zero and 43%, with an average of 19.83%. This result was obtained when enforcing strong constraints on the placement and routing thanks to our specific tool; the average of overheads without these constraints is 25.78% for the same set of benchmarks.

D. *General Discussion*

The feasibility of the approach is demonstrated for the two technologies, but the results show a difference in efficiency, in spite of the efforts put on generating placement constraints. On the Stratix IV platform, errors in the LUTs of the example b03 can be detected at more than 80% without any overheads (with or without placement constraints); this demonstrates the feasibility, in some cases, to significantly increase the intrinsic robustness of an application for free. But this example is very small. Table I summarizes for the ITC benchmarks the ratio between the number of extra LUTs needed to generate the alarm signal and the number of duplicated LUTs (representative of the achieved coverage). For the Stratix IV platform, a large dispersion can be noticed, with a minimum of 0 for two examples and a maximum of 92% for b04, that means that almost as many additional LUTs are necessary than the number of checked functions although the duplication by itself is free because it is limited to partially used ALMs. On the opposite, on the Virtex V platform, the ratio is quite stable between 39% and 50%. This difference mainly comes from the difficulty to efficiently control the placement and routing phase on Stratix IV. On an average, for the ITC benchmarks, the ratio is 55% on this technology while it is only 44% for Virtex V. Also, the average coverage of LUTs is only 40% on Stratix IV (with a minimum of 5.26% for b07) while it is 64% on Virtex V. It is possible to interpret this as an efficient use of ALM resources, limiting the free duplications. But it further limits the interest of the proposed approach on this platform.

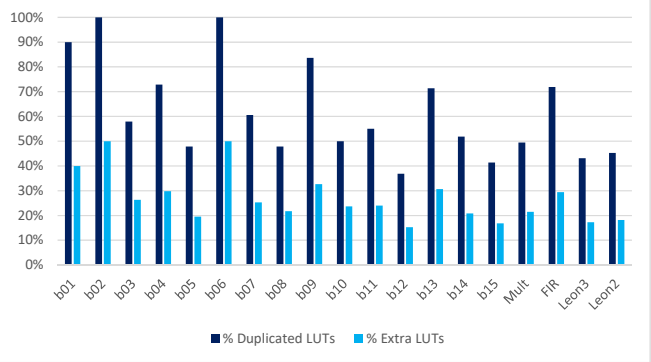

Fig. 5. Implementation results in Xilinx Virtex V devices.

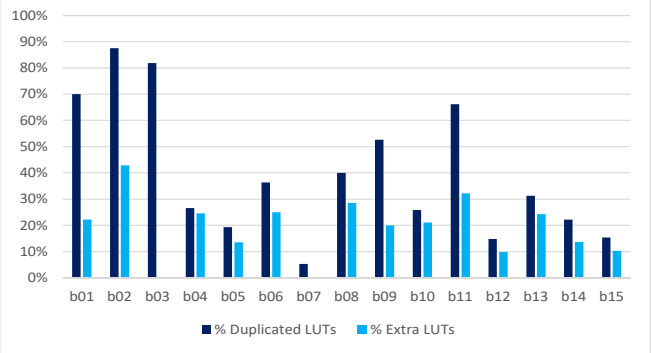

Fig. 6. Implementation results in Altera Stratix IV devices.

We also evaluated on the Xilinx V platform the overheads in terms of delay and power. This was done on the most complex example, i.e. the Leon3 processor, running an AES encryption application.

The evaluation done with the Xilinx tools gave a maximal frequency of 56.99 MHz for the original design and 58.70 MHz for the design with the alarm generation. Although we do not claim that our placement constraints will always help improving the circuit speed, this example demonstrates that the detection can also be free in terms of performances. The alarm is generated in less than one clock period. In case the configuration perturbation occurs at the beginning of a clock cycle, the alarm can immediately identify a wrong result at the end of the cycle. The perturbation may also occur during a cycle where the error is logically masked, i.e. the outputs of the LUT and its duplica are the same in spite of the configuration errors due to the value of the input signals at that time. In that case, the alarm will be activated at the first following cycle during which the error has a functional effect. Finally, if the error is not masked but occurs late in the clock cycle, the alarm may not be asserted early enough to be identified during this clock cycle; in that case the detection latency is slightly increased and the error will be detected at the

next cycle during which the error has a functional effect. In all cases, the latency is much smaller than what can be achieved with a periodic checking of the configuration.

The power consumption evaluated with the Xilinx tools at 50 MHz is 685.6 mW for the original Leon3 running an AES encryption and 686 mW for the design with error detection. The increase is therefore negligible.

A last point to be mentioned is the availability decrease that may be induced by the proposed approach. When an error occurs in one duplica, the application is not directly impacted and without error detection it would continue without interruption. With the proposed approach, an error in the previously unused logic will trigger a "false alarm", and therefore the application will be impacted. But on another hand, an undetected error in the initial logic may lead to many wrong results and significant consequences on the application behaviour. A trade-off has therefore to be made between the rate of undetected errors and the rate of false alarms.

TABLE I. OVERHEAD RATIO FOR VIRTEX V AND STRATIX IV

	Ratio added LUTs vs. duplicated LUTs	
	Virtex V	***Stratix IV***
b01	0.44	0.32
b02	0.50	0.49
b03	0.45	0.00
b04	0.41	0.92
b05	0.41	0.70
b06	0.50	0.69
b07	0.42	0.00
b08	0.45	0.71
b09	0.39	0.38
b10	0.47	0.82
b11	0.44	0.49
b12	0.41	0.66
b13	0.43	0.78
b14	0.40	0.61
b15	0.41	0.67

V. CONCLUSION

The results presented in this paper demonstrate the feasibility of detecting some configuration errors with a very small latency and without any hardware overheads at the system level. The overheads at the block level are kept quite low by exploiting unused resources, but the implemented mechanism still allows detecting the critical errors in LUT configurations and a significant fraction of multiple-bit errors also impacting routing. The extra power due to the added logic was found to be negligible and no performance overhead was recorded on the most complex example evaluated in this work. The approach has been automated and optimized for two different FPGA families from two different manufacturers. The insertion in a standard design flow makes the use seamless for a designer.

Experiments also showed that in spite of the feasibility, some platforms (or their associated development tools) lead to more difficult implementations of the approach. In particular, many constraints had to be taken into account to succeed in applying the approach to Stratix FPGAs. In addition, the results are less interesting than on Virtex V platforms. Looking at other Stratix families may perhaps help in reaching better conclusions for this type of architecture, but at that time the approach seems to have a better potential on Xilinx platforms.

Further work includes experiments with a laser to validate the practical coverage achieved by the detection logic.

ACKNOWLEDGMENT

This work is supported by French research funds in the frame of the CATRENE OPTIMISE project.

REFERENCES

[1] R. Baumann, "Soft errors in advanced computer systems", IEEE Design & Test of Computers, vol. 22, no. 3, May-June 2005, pp. 258-266

[2] http://www.xilinx.com/esp/mil_aero/index.htm

[3] F. Gusmao de Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, R. Reis, "Designing fault tolerant techniques for SRAM-based FPGAs", IEEE Design & Test of Computers, vol. 21, no. 6, November-Décember 2004, pp. 552-562

[4] F. Gusmao de Lima Kastensmidt, L. Sterpone, M. Sonza Reorda, L. Carro, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs", Design, Automation and Test in Europe Conference (DATE), March 7-11, 2005, pp. 1290-1295

[5] C. Bolchini, A. Miele, M. D. Santambrogio, "TMR and partial dynamic reconfiguration to mitigate SEU faults in FPGAs", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, 2007, pp. 87-95

[6] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G. R. Sechi, R. Weigand, "Evaluation of single upset mitigation schemes for SRAM bassed FPGAs using the FLIPPER fault injection platform", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, 2007, pp. 105-113

[7] L. Sterpone, M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs", IEEE transactions on Computers, vol. 55, no. 6, June 2006, pp. 732-744

[8] C. Bolchini, A. Miele, C. Sandionigi, N. Battezzati, L. Sterpone, M. Violante, "An integrated flow for the design of hardened circuits on SRAM-based FPGAs", 15th IEEE European Test Symposium, Prague, Czech Republic, May 24-28, 2010, pp. 214-219

[9] J.-Y. Lee, Y Hu, R. Majumdar, L. He, M. Li, "Fault-tolerant resynthesis with dual-output LUTs," Asia South Pacific Design Automation Conference (ASP-DAC), 2010, pp. 325-330

[10] J.-Y. Lee, Z. Feng, L. He, "In-place decomposition for robustness in FPGA," International Conference on Computer-Aided Design (ICCAD), 2010, pp. 143-148

[11] J. B. Ferron, L. Anghel, R. Leveugle, "Towards low-cost soft error mitigation in SRAM-based FPGAs: a case study on AT40K", 3rd IEEE Latin American Symposium on Circuits and Systems (LASCAS), Playa del Carmen, Mexico, February 2012

[12] G. L. Nazar, L. Carro, "Fast error detection through efficient use of hardwired resources in FPGAs", 17th IEEE European Test Symposium, Annecy, France, May 28-June 1, 2012, pp. 26-31

[13] G. Canivet, J. Clédière, J.B. Ferron, F. Valette, M. Renaudin, R. Leveugle, "Detailed analyses of single laser shot effects in the configuration of a Virtex-II FPGA", 14th IEEE International On-Line Testing symposium, Rhodes, Greece, July 6-9, 2008, pp. 289-294

[14] J.B. Ferron, L. Anghel, R. Leveugle, A. Bocquillon, F. Miller, G. Mantelet, "A methodology and tool for predictive analysis of configuration bit criticality in SRAM-based FPGAs: experimental results", 3rd International Conference on Signals, Circuits & Systems (SCS), Djerba, Tunisia, November 6-8, 2009