On Improving at no Cost the Quality of Products built with SRAM-based FPGAs

R. Leveugle, M. Ben Jrad
TIMA Laboratory (Grenoble INP, UJF, CNRS), 46 Av. Félix Viallet, 38031 Grenoble Cedex, FRANCE
E-mail: Regis.Leveugle@imag.fr, Mohamed.Ben-Jrad@imag.fr

Abstract

Product or design quality encompasses many aspects. One of them is the robustness with respect to perturbations. This robustness depends on the implementation technology, but can also be improved at design time. This paper is focused on designs implemented in SRAM-based FPGAs that are sensitive to soft errors in the configuration memory. An approach is proposed to increase the dependability with respect to configuration errors, at no cost, by selectively hardening parts of the design. The selection of locally duplicated functions is made so that the protections take advantage of FPGA resources that would not be used by the implemented design. An automated design flow is presented for Xilinx Virtex V devices and fault injection results show that the design dependability may be noticeably enhanced. As an example, more than 40% of the LUTs used to implement a Leon3 Sparc v8 processor can be protected against multiple configuration errors with less than 20% resource overheads at the block level. The final system-level overhead may in many cases be null for a given product, either due to the discrete sizes of available FPGAs or to a different repartition of resource budget between system blocks.

Keywords

SRAM-based FPGA, dependability, soft errors, multiple errors

1. Introduction

The increasing investments necessary to develop a specific cell-based circuit (ASIC) in recent technologies lead to more and more applications relying on programmable devices. Among those devices, SRAM-based FPGAs propose a large amount of configurable logic, high flexibility and low cost. However, due to the large size of the configuration SRAM, these devices are very sensitive to soft errors, i.e. spurious modifications of bits, without any damage in the circuit [1]. Soft errors can be caused by several phenomena, including radiation effects. Initially well known in space applications or aeronautics, they are today an identified threat for most applications even at sea-level, as mentioned in the International Technical Roadmap for Semiconductors. Soft errors in user registers containing data or states can perturb the application behavior but have in most cases a transient effect. When soft errors occur in the configuration memory, the effect is still worse since the error may modify the function of the circuit and remains until a reconfiguration of the device (it is called a remanent error). In many applications, soft errors in the configuration memory must therefore be mitigated to achieve the required reliability or availability level. We will focus in this paper on such configuration errors since they have a much higher probability in such devices than errors in user registers due to the huge number of bits in the configuration.

Aggressive technologies are very often used to design FPGAs with the largest possible logic capacity. In such technologies, multiple-bit soft errors (MBU or MCU) have become at least as usual as single-bit soft errors, or SEUs (Single Event Upsets). MBUs (Multiple-bit Upsets) occur when multiple bits are simultaneously modified in a single logic word. MCUs (Multiple Cell Upsets) occur when multiple bits are modified in several logic words.

Many works have been devoted in the last decade to mitigation techniques for errors in SRAM-based FPGAs. However, the focus is in general on how protecting the system against all possible soft errors, accepting very large overheads to achieve this goal. In this paper, we consider a very different approach based on the maximum exploitation of unused resources to increase the design robustness at no cost. The approach is therefore not intended for high-reliability or high-availability applications, but for cost-sensitive applications seeking the best possible quality. Product or design quality encompasses many aspects. One of them is the robustness with respect to perturbations. So our approach aims at improving the intrinsic robustness of a product built around a SRAM-based FPGA, for applications that are not considered critical but for which a better quality due to a better robustness can lead to a competitive advantage Our approach takes into account SEUs, but also MBUs and MCUs. In this paper, we will only present and evaluate the error detection technique; several system-level strategies can be implemented to recover a correct execution after the alarm signal is asserted, including the reconfiguration of the FPGA.

The principles of the approach were previously proposed and the feasibility was demonstrated on an AT40K target for a small example implemented manually [2]. The main contributions of this paper are (1) to demonstrate that the approach can be exploited on other FPGA architectures, especially Xilinx Virtex V, and (2) to report results on a set of benchmarks including large circuit examples, thanks to an automated design flow.

Section 2 summarizes the approaches previously presented in the literature to mitigate configuration errors. Section 3 recalls the basics of the proposed approach and presents. the automated design environment developed for Xilinx components. Results are presented and discussed in Section 4.

2. Previous Works

Techniques have been proposed to improve the dependability of designs implemented in reconfigurable FPGAs, and especially SRAM-based FPGAs. These techniques can be grouped in several classes:

Class 1: hardened manufacturing process. Some FPGA families are manufactured on processes less sensitive to soft

errors [3, 4]. Unfortunately, such devices are more expensive and in general smaller than those manufactured on standard processes. In addition, the choice of such components available in the market is limited. This approach is in general considered only for very specific applications.

Class 2: hardened memory cells. Some works have been presented based on replacing the 6T (or 5T) CMOS SRAM cells by either hardened cells less sensitive to soft errors [5] or by new memory technologies [6]. As in the previous case, such components are not widely available on the market and are more expensive. Few applications can afford manufacturing a specific FPGA for tolerating soft errors [5].

Class 3: hardened FPGA structure. An extension of the previous class is based on the insertion, in the structure of the FPGA, of error detection or error correction mechanisms. Academic proposals have been published (e.g., [7]) but FPGA vendors have also implemented such mechanisms in commercial components, e.g., the Cyclic Redundancy Check in Altera Stratix IV family [8] or the Frame ECC in Xilinx Virtex V. Such mechanisms are in general limited to the detection of one or two erroneous bits in the configuration (or in a frame of the configuration) and are often activated periodically, thus a large error detection latency.

Class 4: hardened design architecture. Modifying the architecture of the design implemented in the FPGA is another approach to detect or correct errors [3, 9-12]. Most approaches in this class are based on the use of massively redundant architectures for the design. The most common approach is to use triplication and voting (TMR); this has been proposed and automated for example for Xilinx devices with the XTMR implementation [3]. Many works are based on this level of redundancy, with several granularity levels [11] and sometimes limited to duplication on some I/Os [13]. Some works also tried to achieve detection by duplication, associated with some correction techniques [9]. The main drawbacks for this class of approaches are the huge overheads in terms of resource usage and in terms of power consumption. Since the main advantages of SRAM-based devices over for example Flash-based devices are related to density and cost, their advantages are considerably reduced when massive redundancy is applied. Also, using these approaches leads to the objective of maximum protection against soft errors; a good trade-off between overheads and robustness is usually not considered while it is the main constraint for many applications.

Class 5: hardened implementation. Several works were devoted to improving the implementation of the design onto the FPGA, in particular reducing the number of critical bits in the configuration thanks to a clever placement and routing [14-17]. However, this improvement is in general targeted to designs already implemented with massive redundancy, and aims at avoiding single point failures due to routing paths with resource sharing between the replicated elements [14]. In a few cases, the routing modifications may include redundancy insertion [15]. Placement has also been optimized so that partial reconfiguration can be used to correct the circuit after error detection, but as a complement to TMR [12].

Class 6: hardened system. Another possibility is to cope with soft errors at the system level, instead of protecting the design itself. This is possible in the case of FPGAs with readback capabilities, so only part of the FPGAs available in the market. For those components, it is possible to periodically check the configuration by reading it and comparing with a golden bitstream [3]. Such a scrubbing is for example possible in Virtex V components, but not in Stratix IV devices. Scrubbing is often used in association with massive redundancy, to avoid error accumulations that may lead to failures in spite of the redundancy. In case of errors in the configuration, partial reconfiguration capabilities of some FPGAs can be exploited to correct the errors with a limited impact on the application execution. When readback mechanisms are not available in the selected FPGA, the configuration can also be periodically refreshed; this may however lead to unacceptable performance loss since the application has to be periodically interrupted and the whole configuration has to be rewritten without knowing if it is useful. In both cases, some controller must be implemented in the system to trigger the FPGA scrubbing or reconfiguration so this approach is quite intrusive in the system design.

Class 7: exploitation of unused resources. Recently, a few approaches have been proposed to perform some on-line error detection by using resources that are available in the FPGA but are useless for a given design [2, 18-20]. In [18], the use of hardwired resources is proposed to provide fast and area efficient fine-grained error detection. The goal is mainly to speed up the on-line error detection with respect to traditional fine-grained comparison, and to limit the area overhead to that of coarse-grained approaches. In [2], the idea was mainly to exploit unused resources to achieve a low-cost implementation of fine-grained checking mechanisms based on local duplication of the most critical functions. The feasibility was demonstrated on a case study implemented manually in an AT40K device. A similar idea was presented for Virtex V and Stratix IV Look-Up Tables (LUTs) in [19]. However, the authors focus on a resynthesis algorithm and evaluate their approach only theoretically, based on the synthesis results, without actual implementation in the FPGAs. Furthermore, they only focus on masking some SEUs through AND and OR combinations. The use of unused carry chains is also considered in [20] but here again the work does not include actual implementations; the real feasibility of using these resources is therefore not demonstrated.

The approach considered in this paper fits in Class 7. As in [19], we take advantage of partially used LUTs to implement a partial and local duplication of some functions. But this duplication is not used to tolerate some SEUs. Instead, the goal is to detect as many multiple-bit errors as possible in the configuration memory, without any actual overheads in terms of hardware resources in the system. Also, a complete automated design flow has been implemented and the detection coverage is evaluated by fault injections into actual prototypes. The approach will be more detailed in the next section.

3. Proposed approach

Before presenting in more details our approach and the related design flow, we will summarize the motivations for the type of errors that we considered.

3.1. Targeted errors

Two types of errors can occur in a design implemented in SRAM-based FPGAs. The first type is a perturbation of application data, i.e. errors in registers or memories used by the application. Such errors have in general transient effects. The second type is a modification of the configuration, i.e. a modification of the implemented design. This second type is more critical for several reasons: (1) the error remains until the device is reconfigured, (2) the probability of such errors is high since the number of bits in the configuration memory is much larger than the number of bits containing application data and (3) errors in the configuration cannot be efficiently mitigated by architectural modifications of the design unless resorting to massive redundancy. We will therefore focus in the sequel on errors occurring in the configuration of the functions implemented for the application.

The configuration bits define the logic functions required for the application, and the interconnection of these functions. The first part corresponds to the definition of the LUTs. The second part defines the activation of all the possible interconnection segments in the device. The percentage of configuration bits in this second category is very high. However, we decided to focus more on the first category for two reasons. One is the complexity of implementing redundant routing in commercial design flows without significantly perturbing the final design implementation and reducing the actual device capacity. The second reason is the actual criticality of the errors in the interconnection segments.

Few years ago, we performed several campaigns of laser-based fault injections on Xilinx Virtex II and Virtex II Pro devices. Some results were presented in [21]. These campaigns allowed us to conclude first on the high risk of multiple-bit errors with large spatial multiplicity even for very local perturbations and in spite of the "old" technology of the Virtex II devices. Figure 1 shows as an example the repartition of the recorded error patterns after a scan of a CLB area of the FPGA, using a single laser shot per spatial position and for spot focalizations of 8 and 40 µm. SEUs only account for 4.44% of the error patterns, and a pattern with more than 30 erroneous bits is more frequent than a SEU.

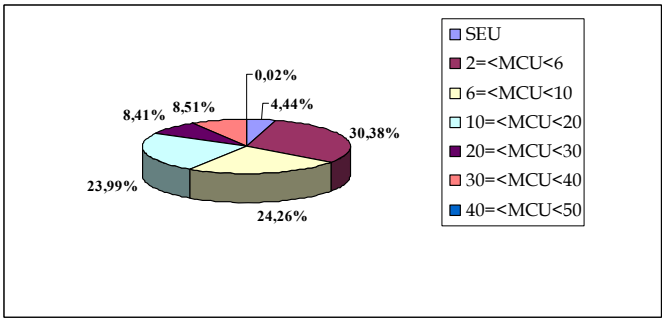

Figure 1: Example of classification of error patterns obtained on a Xilinx Virtex II by laser shots with spot focalizations of 8 and 40 µm.

The second conclusion was that an error in the interconnection configuration does not necessarily impact the application with a high probability.

When an error occurs in the configuration of an unconnected segment, this segment may first remain unconnected because the configuration of a segment often depends on several bits. Also, even if an extra segment is activated due to the errors, it must be connected to another critical segment to actually interfere with the application behavior.

When the error occurs near a segment used by the application, the segment may be open or may be changed. In these two cases, the application should be perturbed when the related function has to be used. However, in some other cases, segments are added to the initial one. In this case again, the consequence depends on the creation or not of some short circuit.

For the campaign illustrated in Figure 1, less than 4% of the laser shots led to actually suppress or modify an interconnection segment necessary for the application.

In conclusion, the goal of the proposed approach is to detect error patterns in the configuration, taking into account a potentially large spatial multiplicity, and with a special focus on LUT modifications.

3.2. Basic implementation concepts

As previously mentioned, the main idea is to duplicate locally some functions in such a way to take advantage of resources that are originally not necessary for the implemented design.

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 2. 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 replica of the function and to take advantage of the second available output to compare the results computed by the two replica. Such a duplication and comparison scheme allows us to cover a very large percentage of the possible error patterns with large multiplicity, even when both 5-input LUTs are impacted. In addition, this local massive redundancy is for free if it is limited to the partially used LUTs. Another advantage of such a duplication is to avoid any modification in the routing when adding the replica. Of course, the comparison and the generation of the global error detection signal require to use extra LUTs in the device (and this requires routing more signals) but in many cases it is possible to avoid a real cost at the system level; this is discussed in the next section.

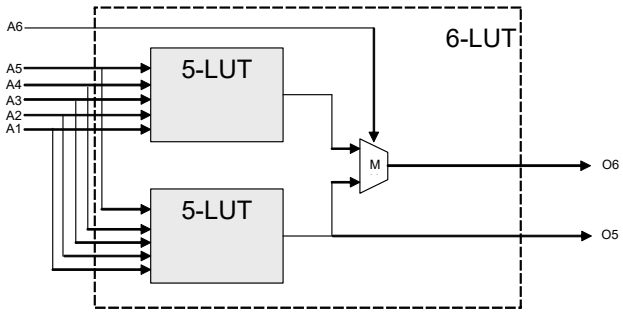

Figure 2: A 6-input LUT in Virtex V components.

The global alarm signal can be used at system level to trigger any suitable procedure, defined with respect to the application constraints. This may be signaling a warning to the user in some low-risk situations, triggering an automatic system reset, or aborting an on-going transaction (e.g., in the case of systems with some security concerns and that may be subject to attacks).

3.2. Global approach towards no cost

FPGA families have of course a limited list of devices, with discrete sizes. Table I illustrates this for a subset of the Xilinx Virtex V family. The consequence is therefore that in many cases part of the FPGA remains fully unused for a given application. Routing constraints and congestions may avoid using some of the resources, but a significant amount of logic is often still available after a given design implementation. Table II illustrates the percentage of logic resources required for a few designs on the smallest available Virtex V device. A design with for example two Leon microprocessors, some filters and some additional blocks would hardly require all available logic resources. The idea to avoid any system-level resource increase is therefore to limit the number of comparators and the alarm generation logic to the available resources.

Table I: Examples of discrete device sizes in the Xilinx Virtex V family

Device	#LUTss	#slices
XC5VLX30	19,200	4,800
XC5VLX50	28,800	7,200
XC5VLX85	51,840	12,960
XC5VLX110	69,120	17,280
XC5VLX155	97,280	24,320

Table II: Percentage of resource usage for some design examples fitted on the smallest available Virtex V device (XC5VLX30).

Design	#LUTs	%LUTs
b14	1161	6
b15	1712	8.9
FIR filter	238	1.2
Leon2	4063	21.2
Leon3	6857	35.7

The selection procedure for achieving zero actual resource overhead with maximum efficiency is therefore the following:

1. Identification of the number of partially used LUTs in the design.

2. Evaluation of the number of available resources for implementing comparators and alarm combination logic. One 6-input LUT is sufficient to compare the results of three protected LUTs with their replica. One 6-input LUT is also sufficient to combine 6 comparator outputs, or 6 intermediate alarm signals in the generation tree. It is therefore easy to compute the required number of extra LUTs to generate the alarm signal. If the unused resources are not sufficient to generate the alarm signal, or cannot be routed satisfactorily, the number of protected LUTs is reduced. The selection should be guided by the identification of the most critical functions (based on the designer knowledge, or on criticality analyses with tools such as SEFEA-ProD [22]).

3. Duplication and alarm generation using only unused LUTs available in the FPGA. In this paper, we limit the number of duplicated LUTs to the partially used LUTs. An extension may consider adding the duplication of more LUTs, in case enough resources are wasted in the original design. This case is not explicitly considered in the results of this paper.

In this paper, we also consider that only a global alarm signal is generated. Other solutions are possible if the alarms are managed inside the FPGA (for example, to trigger an endo-reconfiguration) or if several I/Os are left available by the design. Increasing the number of alarm signals would reduce the number of LUTs necessary for their generation, potentially permitting the protection of a larger number of LUTs, and may give more precise information on the error localization, for example to trigger a partial reconfiguration of the device. Many trade-offs are possible but will not be explicitly explored in this paper.

Such a procedure leads to error detection without any actual overheads at system level in terms of hardware resources since the protection is limited to available resources. The only potential overheads are in terms of dynamic power consumption since more logic blocks are computing in the FPGA during the application run and in terms of maximum frequency due to the extra routing of the alarm signal. This will be discussed in section 4.

3.3. Design flow

The generic design flow is illustrated in Figure 3 and has been used on several FPGA families. A specific tool has been developed to automate the insertion 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.

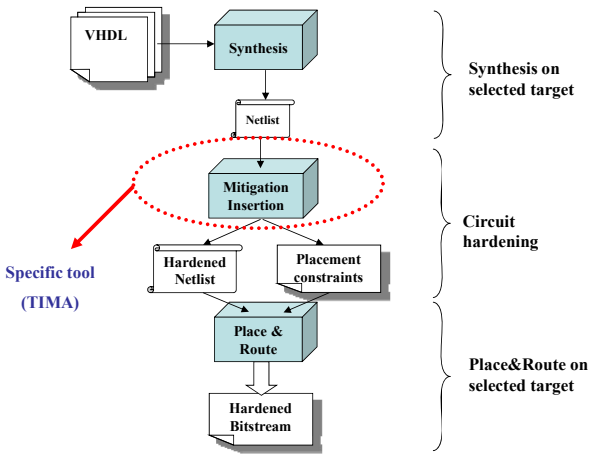

Figure 3: Generic design flow.

On Xilinx Virtex V, a detailed analysis on a set of benchmarks showed that between 36% and 100% of the

combinatorial functions can be duplicated using only free resources. In order to avoid any actual overheads, it is however necessary to ensure, during the placement of the functions, that each replica is actually associated with its initial function, in the same 6-input LUT. This is the role of the constraints automatically generated by our tool.

4. Implementation results

4.1. Implemented designs

Experiments have been carried out for a set of blocks including the ITC'99 benchmark set and some other design examples: typical processing blocks (FIR filter and a sequential multiplier) and open-source Sparc v8 microprocessors (Leon2 and Leon3).

4.2. Resource usage

Implementation results on Xilinx Virtex V are summarized in Table III. Only partially used LUTs have been duplicated. Overheads indicated are with respect to the initial block complexity and are due to the error signal generation. The coverage is related to the percentage of LUTs partially unused in the original block implementation. For the two largest examples (microprocessors Leon2 and Leon3) it was possible to protect more than 40% of the LUTs with less than 20% of additional resources. If enough resources are not available after implementing the whole original design in the FPGA, the coverage may be reduced. Also, the size budget of other blocks may be revised.

Table III: Implementation and fault injection results (limited to LUTs) in Xilinx Virtex V devices

	Application	Protected LUTs	Additional resources	Number of injected error patterns	Detection
ITC bench.	b01	90%	40%	100	87.10%
	b02	100%	50%	16	100%
	b03	57.89%	26.32%	1444	59.40%
	b04	72.81%	29.82%	12996	70.20%
	b05	47.83%	19.57%	33856	42.90%
	b06	100%	50%	64	100%
	b07	60.61%	25.25%	9801	60.14%
	b08	47.83%	21.74%	529	52.21%
	b09	83.67%	32.65%	-	-
	b10	50%	23.68%	1444	52.40%
	b11	55%	24 %	10000	53.71%
	b12	36.86%	15.29%	65025	37.95%
	b13	71.43%	30.61%	2401	79.29%
	b14	51.85%	20.84%	234792	50.74%
	b15	41.35%	16.8%	-	-
Others	Multiplier	49.46%	21.51%	8649	50.69%
	FIR filter	71.85%	29.41%	-	-
	Leon3	43.17%	17.30%	-	-
	Leon2	45.24%	18.16%	-	-

4.3. Protection efficiency

Emulation-based fault injections have been performed for some benchmarks with the environment presented in [23] to validate the level of protection. Some results are shown in Table III. Only errors (single and multiple) in the LUT configurations have been injected during the first campaigns and the results correspond to what was expected, i.e. are of the order of magnitude of the percentage of protected LUTs. Since fault injection is a dynamic evaluation, the injection time may lead or not to detection for a given error. Also, the injection was not exhaustive; statistical fault injection was performed leading to a margin of error (the injections were performed for a 10% margin of error with 95% confidence).

We also performed fault injections in order to evaluate the efficiency of the approach with respect to error patterns not limited to LUT configurations. The campaigns were performed with a statistical selection of the error location and time, for a 10% margin of error with 95% confidence. The selected benchmark is the Leon3 processor running an AES encryption program. The first campaign was based on the injection in the configuration of single bit-flips (SEUs) and multiple bit-flips limited to a single configuration word (MBUs). Results are shown in Table IV, for errors injected in the area of the component actually used by the benchmark,

and with several MBU multiplicities (2, 4, 6 and 8). The percentage of silent faults is always very large, although it decreases with the error multiplicity. Failures are identified when the computation does not terminate properly. Data errors mean that a result is obtained but is incorrect. The global percentage of detection increases with the multiplicity. Some errors lead to false positives, i.e. an alarm signal asserted while the final computation result is correct. This is due to errors injected directly in the alarm signal generation or in the replica, whose outputs are not taken into account to compute the final result. It can be observed that, in spite of a replication limited to LUTs, the approach is able to detect a noticeable percentage of MBUs impacting all configuration bits, especially in the case of large multiplicity.

A similar campaign was based on the injection of multiple bit-flips in several configuration words (MCUs). The physical proximity of the words was however considered in order to represent a local perturbation of the device. The first modified bit for each experiment was randomly selected then, the other modified bits were randomly chosen in an area of 3 frames * 5 words surrounding the first modified bit. This basically corresponds to an injection in a small rectangular area of the component. Results are shown in Table V, for several MCU multiplicities (2, 4, 6 and 8). The percentage of silent faults is again very large, although it decreases with the error multiplicity. The global percentage of detection increases with the multiplicity but is smaller than for MBUs. Since MCUs are spread over several configuration words, the detection level is close to what was observed for SEUs and 2-bit MBUs.

Table IV: Fault injection results in Xilinx Virtex V devices, for SEUs and MBUs in the configuration

	SEU	MBU: 2	MBU: 4	MBU: 6	MBU: 8
# injected faults	9192	9191	9189	9187	8985
Silent	8109	7812	7419	7289	7101
False positive	93	91	129	214	240
Undetected data error	0	0	3	12	32
Detected data error	0	0	0	0	0
Undetected failure	909	1157	1409	1370	1124
Detected failure	81	131	229	302	488
% Silent	88.22%	85.00%	80.74%	79.34%	79.03%
% False positive	1.01%	0.99%	1.40%	2.33%	2.67%
% Undetected data error	0.00%	0.00%	0.03%	0.13%	0.36%
% Detected data error	0.00%	0.00%	0.00%	0.00%	0.00%
% Undetected failure	9.89%	12.59%	15.33%	14.91%	12.51%
% Detected failure	0.88%	1.43%	2.49%	3.29%	5.43%
% Detected non-silent faults (including false positive)	8.08%	10.50%	14.86%	18.92%	34.96%

Table V: Fault injection results in Xilinx Virtex V devices, for MCUs in the configuration (limited to a local area of 3*5 configuration words)

	MCU: 2	**MCU: 4**	**MCU: 6**	**MCU: 8**
# injected faults	9193	9193	9193	9193
Silent	7914	6609	5595	5582
False positive	99	149	235	260
Undetected data error	0	0	11	0
Detected data error	0	0	0	1
Undetected failure	1103	2194	3021	2964
Detected failure	77	241	331	386
% Silent	86.09%	71.89%	60.86%	60.72%
% False positive	1.08%	1.62%	2.56%	2.83%
% Undetected data error	0.00%	0.00%	0.12%	0.00%
% Detected data error	0.00%	0.00%	0.00%	0.00%
% Undetected failure	12.00%	23.87%	32.86%	32.24%
% Detected failure	0.84%	2.62%	3.60%	4.20%
% Detected non-silent faults (including false positive)	6.41%	10.29%	10.13%	11.97%

4.4. Impact on power and delay

Analyses were performed on the largest benchmark (Leon3) to quantify the impact of the proposed approach on power and delay. At 50 MHz, the evaluated power consumption is 685.6 mW for the original design and 686 mW for the protected design, so a very negligible increase. The maximum frequency is 56.99 MHz for the original design and 58.70 MHz for the protected design; this increase is within the typical variation margins due to the heuristics used by the CAD tools, but demonstrate at least that there is no significant critical path increase. The computation of the global alarm signal can be achieved within one clock cycle, leading to almost immediate error detection.

5. Conclusion

The results presented in this paper demonstrate the feasibility to improve the quality of a product implemented on SRAM-based FPGAs by taking advantage of unused resources available after placement and routing of the design. The approach can be automated, leading to a seamless use in a standard design flow. Fault injections have confirmed the expected coverage of errors in LUTs and have also shown that more complex error patterns can also be detected. The use of extra logic can be limited so that no actual costs are incurred from a hardware point of view at system level. Many trade-offs are possible and have been briefly presented in this paper; exploring them in more details is part of our future work.

6. Acknowledgements

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

7. 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] 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, 2012

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

[4] http://www.atmel.com

[5] S. Bonacini, F. Faccio, K. Kloukinas, A. Marchioro, "An SEU-robust Configurable Logic Block for the implementation of a radiation-tolerant FPGA", IEEE transactions on Nuclear Science, vol. 53, no. 6, Part I, December 2006, pp. 3408-3416

[6] L. Torres, Y. Guillemenet, S. Z. Ahmed, "A dynamic reconfigurable MRAM based FPGA", ERSA'2010, pp.31-40

[7] Q. Zhao et al., "A novel soft error detection and correction circuit for embedded reconfigurable systems", IEEE Embedded Systems Letters, vol. 3, no. 3, September 2011, pp. 89-92

[8] http://www.altera.com/literature/an/an357.pdf

[9] F. Lima, L. Carro, R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs", 40th Design Automation Conference (DAC), 2003, pp. 650-655

[10] F. Gusmao de Lima Kastensmidt et al., "Designing fault tolerant techniques for SRAM-based FPGAs", IEEE Design & Test of Computers, vol. 21, no. 6, November-Décember 2004, pp. 552-562

[11] F. Gusmao de Lima Kastensmidt et al., "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

[12] 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

[13] M. Alderighi et al., "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

[14] 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

[15] C. Kinzel Filho et al., "Improving reliability of SRAM-based FPGAs by inserting redundant routing", 8th European Conference on Radiation and its Effects on Components and Systems (Radecs 2005), 2005

[16] C. Bolchini et al., "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

[17] M. A. Abdul-Aziz, M. B. Tahoori, "Soft error reliability aware placement and routing for FPGAs", International Test Conference (ITC), 2010, pp. 1-6

[18] 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

[19] 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

[20] 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

[21] G. Canivet et al., "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

[22] J.B. Ferron et al., "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

[23] R. Leveugle, M. Ben Jrad, "A new methodology for accurate predictive robustness analysis of designs implemented in SRAM-based FPGAs", IEEE International Conference on Electronics, Circuits and Systems (ICECS), Athens, Greece, December 12-15, 2010, pp. 1179-1182