A Self-Checking Scheme to Mitigate Single Event Upset Effects in SRAM-Based FPAAs

Tiago R. Balen, *Student Member, IEEE*, Franco Leite, *Student Member, IEEE*, Fernanda Lima Kastensmidt, *Member, IEEE*, and Marcelo Lubaszewski, *Member, IEEE*

***Abstract*—In this work the problem of Single Event Upset (SEU) is considered in a recent analog technology: The Field Programmable Analog Arrays (FPAAs). Some FPAA models are based on SRAM memory cells to implement the user programmability, which makes this kind of device vulnerable to SEU when employed in applications susceptible to the incidence of radiation. In the former part of this work some fault injection experiments are made in order to investigate the effects of SEU in the SRAM blocks of a commercial FPAA. For this purpose, single bit inversions are injected in the FPAA programming bitstream, when an oscillator module is programmed. In a second moment, a self-checking scheme using the studied FPAA is proposed. This scheme, which is built from the FPAA programming resources, is able to restore the original programming data if an error is detected. Fault injection is also performed to investigate the reliability of the proposed scheme when the bitstream section which controls the checker blocks is corrupted due to a SEU.**

***Index Terms*—Field programmable analog arrays, self-checking, self-recovering, single event upset.**

I. Introduction

FIELD Programmable Analog Arrays (FPAAs) are analog integrated circuits based on configurable analog blocks and programmable interconnections. They provide to the analog world the same flexibility as their digital counterparts, Field Programmable Gate Arrays (FPGAs), provide to digital circuits. They allow fast prototyping and offer some interesting features for applications such as adaptive control and instrumentation and evolvable analog hardware [1], [2]. These features can be very useful when the environmental variables can assume a wide range of values and the system must respond properly to these variations. As an example one can consider the avionics applications, where the external temperature and pressure can vary significantly in few minutes of flight. Another possible application is in the space exploration missions, where it can be necessary to re-calibrate the sensor conditioning circuits of spacecrafts to correct errors or improve system performance, for example.

For all these reasons, FPAAs have become an important platform to analog circuit design, thus, it is important to ensure the correctness condition of the analog functions implemented into these devices.

Single Event Upsets (SEU) and Single Event Transients (SET), as well as their effects on digital circuits, have been widely studied from more than a decade [3], [4] and several techniques to implement fault tolerant digital circuits have been devised, for example [5]–[7]. However, not much work has been done so far concerning these problems in analog circuits [8]–[10].

Some FPAA architectures are based on SRAM memory to implement the user programmability. In this case, as it is for SRAM-based FPGAs, a SEU can affect the programming memory and change the device configuration, which can modify the analog circuit behavior.

The preliminary goal of this work is to analyze the effect of SEU in a commercial FPAA from Anadigm [11]. The experiment consists in injecting bit-flips in the FPAA programming bitstream, where an oscillator module is considered as design under test. An error detection circuit allows the evaluation of the effects of these bit inversions in the functional behavior of the circuit. The second aim of this work is to propose a self-checking scheme that allows the FPAA programming data reloading if an error is detected. This scheme is built by using the internal programmable resources of the studied FPAA.

The reliability of the proposed scheme is investigated by means of two sets of fault injection experiments. First, faults are exhaustively injected in the checker bitstream when the blocks under test are considered fault-free. Then, in the second part of these experiments, a functional deviation between the design and redundant blocks is programmed, while faults are injected in the checker bitstream. Some conclusions are drawn considering the results of these experiments.

This paper is organized as follows: in Section II the effects of SEU in generic SRAM-based FPAAs are discussed. Section III describes the fault injection experiments for an oscillator block programmed in the commercial FPAA considered in this work. Section IV presents the proposed self-recovering scheme, while Section V shows the checker fault injection experiments. Section VI concludes this work.

Manuscript received September 07, 2008; revised December 29, 2008. Current version published August 12, 2009. This work was supported in part by CNPq and CAPES Brazilian Agencies.

T. R. Balen is with the Centro Universitário Lasalle—UNILASALLE, Canoas, RS, Brazil, and also with the Universidade Federal do Rio Grande do Sul CEP 90040-060 Porto Alegre, Brazil (e-mail: balen@unilasalle.edu.br; tiago.balen@ufrgs.br).

F. Leite was with the Electrical Engineering Department, Universidade Federal do Rio Grande do Sul, CEP 90040-060 Porto Alegre, Brazil. He is now with Mectron Engenharia, 12227-000 São José dos Campos, SP Brazil

M. Lubaszewski is with the Electrical Engineering Department, Universidade Federal do Rio Grande do Sul, CEP 90040-060 Porto Alegre, Brazil (e-mail: luba@ece.ufrgs.br).

F. L. Kastensmidt is with the Informatics Department, Universidade Federal do Rio Grande do Sul, CEP 90040-060 Porto Alegre, Brazil (e-mail: fglima@inf.ufrgs.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2009.2013347

0018-9499/$26.00 © 2009 IEEE

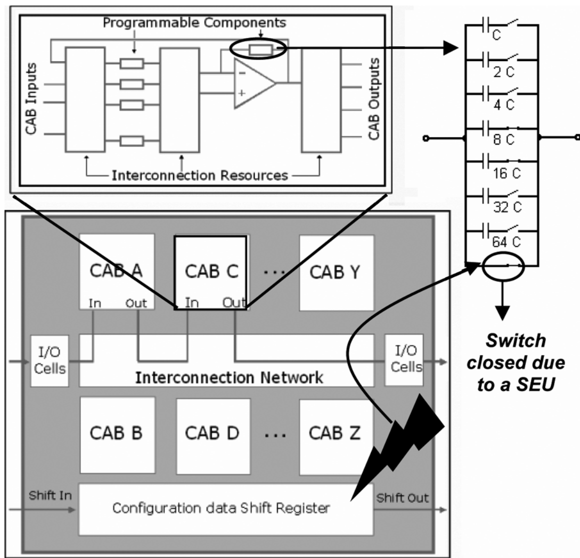

Fig. 1. Variation in the programmed circuit due to a SEU in the programming memory of a SRAM-based FPAA.

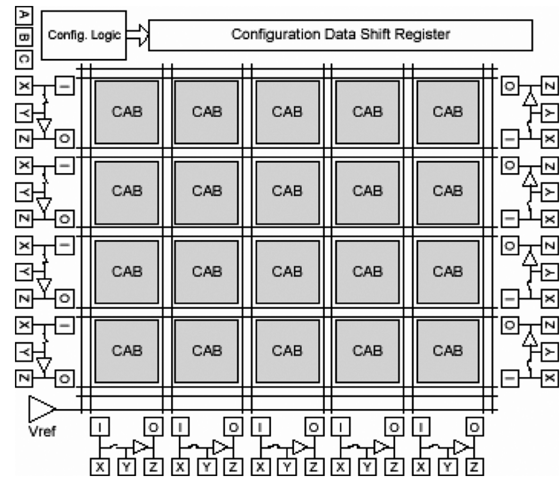

Fig. 2. Block Diagram of AN10E40 FPAA [11].

II. The Effect of SEU in SRAM-Based FPAAs

A typical FPAA structure comprises Configurable Analog Blocks (CABs), I/O blocks, an interconnection network and memory blocks for device programming. In some FPAAs the programming memory is based on SRAM blocks, which is the case of the Anadigm FPAA [11], the scope of this work.

Usually the programmability of FPAAs is allowed through switches that set the routing and the values of components within the array. The state of such switches is defined through the value of a bitstream stored in a shift register that is loaded during the device configuration in the power up cycle. If the programming shift register is based on SRAM-type memory cells the impact of a charged particle in one or more transistors of the memory blocks can result in a bit-flip in the previous stored bitstream. This inversion may change the state of a switch used in the circuit and modify parameters like values of components or change the routing in or between the CABs of the FPAA. In some cases a SEU in the programming memory can result in a very different configuration from that previously programmed, which can be critical to the system operation. Fig. 1 illustrates such event considering a typical SRAM-based FPAA architecture.

Fig. 1 shows the typical FPAA and CAB architectures. Each CAB is composed of an analog programmable component array, local and global interconnection blocks and at least one operational amplifier with global and local programmable feedback loops. The components within the array can be implemented as simple wires, passive or active components or other more complex parts. In general, the programmable parameters of CABs are gain of amplifiers, values of resistors and capacitors, as well as setting global and local feedback loops. In some FPAA models, the resistors are implemented through the switched-capacitor technique and the value of its resistance can be programmed by the value of the capacitor and its switching frequency. The value of the capacitors (either switched or static ones) is programmed through a bank in which a set of programmable switches is used to connect or disconnect the capacitors to configure the desired value.

In the example of Fig. 1, a previous programmed circuit is set with a capacitor whose relative value is 24C $(8\text{C} + 16\text{C})$. If a SEU occurs in the programming memory cells it is possible that a component used in the circuit implementation has its value changed. In this hypothetic case a bit-flip was considered in one of the switches that compose a programmable capacitor bank, provoking a short in the programmed capacitor.

One can see that a SEU can be catastrophic in SRAM-based FPAAs since, in some cases, the correct functioning of a single switch is crucial to the system operation. Besides modifications in the programmed values, a bit-flip in the memory cells can result in the disconnection of components, connections of undesirable components in the circuit (parasitics) and even an interruption in the signal path. Such interruption can invalidate the affected analog blocks or even the entire system in which the FPAA is inserted.

If one or more bits of the original configuration are modified during the operation of the circuit the only way to restore the original configuration is reloading the bitstream into the FPAA programming shift register. In some FPAA models this reload can be done in fractions of milliseconds.

III. Fault Injection Experiments With AN10E40 FPAA

A. Fault Injection Procedure

In order to perform some experiments to study the effect of bit-flips in the programming memory of FPAAs a commercial device is used. The device studied in this work is the AN10E40 from Anadigm Company [11], a switched capacitor FPAA. It has 20 CABs distributed in a 4×5 array. Each CAB can be connected to any other CAB or to one of the 13 I/O cells through an interconnection network. The block diagram of the AN10E40 is shown in Fig. 2, in which one can see the global wiring surrounding the 20 CABs.

This network comprises horizontal and vertical buses organized in 5 rows and 6 columns, each one composed of two wires. Besides the global buses, the connections between the CABs can

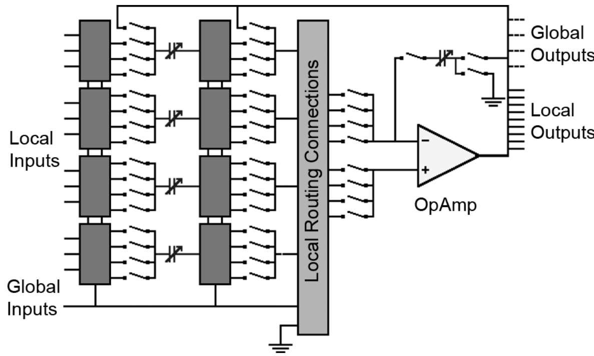

Fig. 3. Schematic view of the AN10E40 CAB [11].

IPmodule	Bytes
Simple gain stage	003f c040 0022 ff24 1000 0ff3 fc00 0018 2270 01c0 0805 2090 8000
Rectifier	003f c040 0082 ff20 1000 0ff0 0000 0080 2a70 01c0 0000 2090 9500

Fig. 4. Examples of Default Bitstream of IP-Modules.

be made by means of local interconnections. A schematic view of the AN10E40 CAB is depicted in Fig. 3. Each CAB has 5 capacitor banks that can implement a programmable capacitor or a programmable resistor (switched-capacitor).

The values of the programmable components of the CABs cannot be changed directly by the user, since the programming and constructive details are unknown. In order to configure the desired circuit into the FPAA one can use the Anadigm Designer Software, which provides a set of pre-built IP modules. One can link these modules and set parameters like block gain, central frequency of filters, integration constants and thresholds of comparators, for example.

However, in one of the data directories of the programming software it is possible to find the files that contain the default bitstream for each module available in the programming library. Fig. 4 shows the default bitstream for two of these IP-modules (a simple gain stage and a rectifier).

In the example of the Fig. 4, both considered analog IP-modules are built using only one CAB of the FPAA. Each CAB comprises 208 programmable switches [11], therefore, each one of these blocks is programmed through a stream of 208 bits. There are other analog modules built with 2 or 3 CABs, therefore, one single analog function can be programmed with up to 624 bits. As there are 20 CABs in this device the total amount of memory dedicated to the CABs programmability is 4160 bits.

The other programmable resources of the device (global routing, IO, programmable voltage reference and clock) are set by means of 2704 switches, therefore the whole bitstream of the AN10E40 is composed of 6864 bits [11].

The fault injection experiments are carried out by modifying the default bitstream of the IP-modules. For this purpose a copy of the files that contain the bitstream was made in such a way that two module libraries are now available, one with the fault-free bitstream and the other with single bit-flips in the configuration bitstream. This approach does not allow injecting faults in the bits that program the global interconnections or the IO cells of the device because the modified files only comprise the bitstream of the CAB switches.

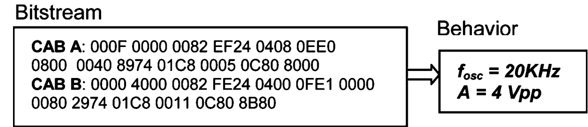

Fig. 5. Default bitstream of oscillator and expected behavior.

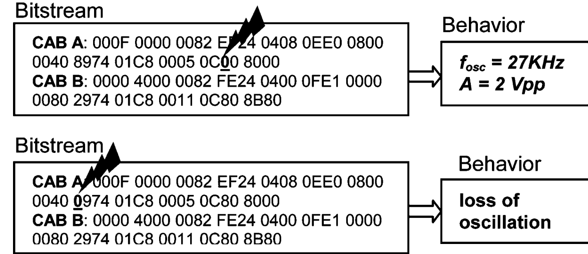

Fig. 6. Examples of modified bitstream of oscillator and resultant behaviors.

The analog module used in this experiment is a sinewave oscillator. The default bitstream of both CABs used in this module implementation and the expected parameters of the output signal are shown in Fig. 5 (where $f_{\rm osc}$ is the oscillation frequency and A is the amplitude of the signal).

Each digit of the above bitstream is a hexadecimal representation, therefore comprising 4 bits. The faults where injected in the oscillator by changing (inverting) the values of one bit at a time of each CAB. A total of 416 faults where injected (208 in each CAB). Examples of the implication of single flips in the considered bitstream are shown in Fig. 6.

B. Error Detection and Experimental Results

In order to assist the evaluation of the fault injection experiments a specific error detection circuit was built using the internal programmable resources of the FPAA. Such error detection circuit is based on a very selective Band-Pass (BP) filter that attenuates the signal if the frequency of the oscillator (tuned with the central frequency of the filter) is different from that previously programmed. The output of the BP filter is rectified and filtered in order to generate a DC level, which is compared to a reference window.

Variations in the amplitude or frequency of the oscillation signal will generate a DC signal whose amplitude are out of the considered window, thus being detected by the comparators. Fig. 7 shows the block diagram of the error detection scheme. The oscillator frequency and amplitude are programmed as 20 kHz and 4 Vpp respectively. The filter central frequency and gain are programmed as 20 kHz and 0 dB, while the programmed quality factor is 100. The reference window is set to ± 10 mV around the DC level generated in the output of the rectifier/filter block. With this scheme variations of ± 10 mV in the amplitude and ± 100 Hz in the oscillator frequency are detected by the evaluation circuit.

For a total of 416 injected bit-flips into the oscillator module only 140 faults (33.65%) affected the functional behavior of the circuit in a way that the amplitude or frequency of the oscillator deviates from the nominal programmed value. For the first CAB of the oscillator 57 of the 208 injected bit-flips (27.5%) modified the circuit behavior. For the second CAB 83 injected bit-flips

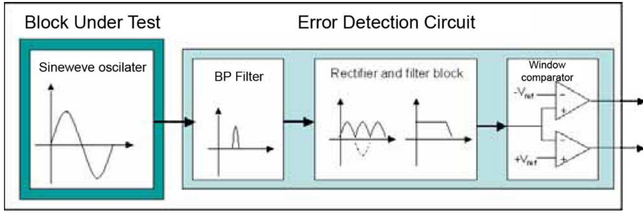

Fig. 7. Oscillator and error detection circuit.

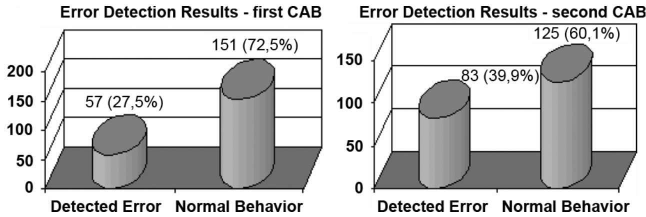

Fig. 8. Error detection results for the injected bit-flips.

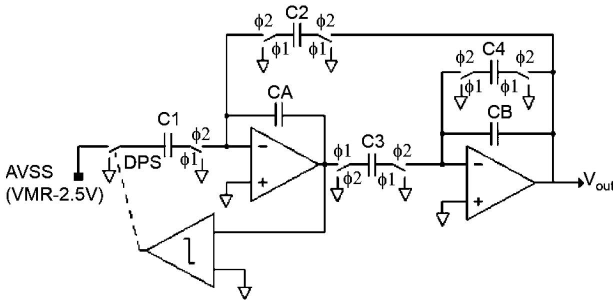

Fig. 9. Oscillator schematic (built with two CABs) [12].

(39.9%) resulted in a detectable behavior change, as it can be seen in Fig. 8.

This low occurrence of errors caused by the injected faults is due to the fact that the CAB resources are partially used to implement the oscillator. Therefore, many programmable switches of the CABs are not in the signal path and do not affect the behavior of the circuit. Fig. 9 shows the schematic of the oscillator according to the Anadigm IP-Modules Manual [12] and Fig. 10 shows the possible implementations of this scheme by using the CAB depicted in Fig. 3. The comparator in Fig. 9 is a control block and is not considered in this fault injection experiment.

One can see that this implementation does not use all the programmable components or the entire local routing of the CABs. For this reason, a SEU in a bit that controls some component or branch not used in the circuit can maintain the functionality of the programmed blocks unaltered. Furthermore, the difference observed in the error detection rates of the two oscillator CABs are due to the higher number of programmable resources used in the second block, as one can see in Fig. 10.

IV. A Self-Checking Scheme With AN10E40

The error detection circuit used in the previous fault injection experiments is applicable to the specific circuit considered as block under test, in that case an oscillator module. Furthermore, the test setup is an off-line scheme since the purpose of that circuit is only to ease the evaluation of bit-flip effects

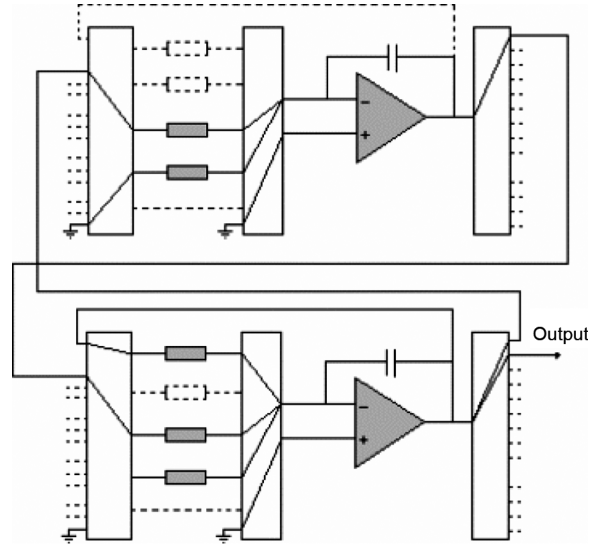

Fig. 10. Example of not used resources in the oscillator implementation.

in the programming bitstream of the considered module. However, it is possible to take advantage of the FPAA programmability to build on-line checkers and consequently to implement self-checking and self-recovering mechanisms.

If a bit-flip occurs in a memory cell, a possible way to restore the original programming data is to reload the default bitstream into the FPAA configuration shift register. The programming shift register is loaded during the power up or reset cycles of the device.

The simplest way to perform the AN10E40 programming is through a serial interface. In this case the configuration data is stored in an external serial ROM. Since this configuration scheme is widely used to set up the FPGAs boot configuration, the use of a standard serial EEPROM can be a low cost programming alternative.

The self-recovering scheme proposed here consists in an on-line checker that forces the device reset, and consequently the programming data reloading, if a functional deviation is detected in the circuit. The whole scheme is based on redundancy. The programmed circuit is duplicated, then, the checker subtracts the outputs of the circuit and its replica, generating an error signal. This error signal is then compared to a pre-defined reference window. If the error signal deviates from the tolerance limits the window comparator activates the FPAA reset and the original configuration is reloaded. A very simple external logic is also required.

Fig. 11 shows the block diagram of the proposed scheme, considering a band-pass (BP) filter as target design block. Although in this work a BP filter is considered, this scheme applies to any functional block that can be programmed into the device. The frequency response of the BP filter considered as design block is shown in Fig. 12.

The error detection circuit (checker) and the redundant block are built from the internal FPAA resources. Therefore, as the device comprises a limited number of CABs, this restricts the number of programmable blocks available to build the functional design module. Fig. 13 shows the Anadigm Designer

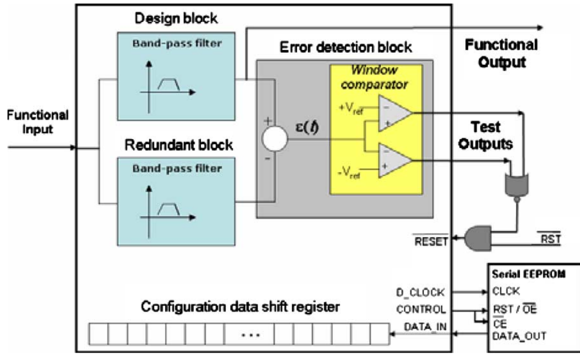

Fig. 11. Block diagram of the self-recovering scheme with the AN10E40 FPAA.

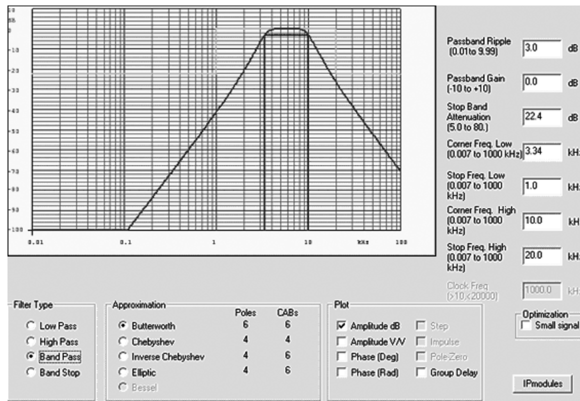

Fig. 12. Specifications of the BP filter programmed as functional design block.

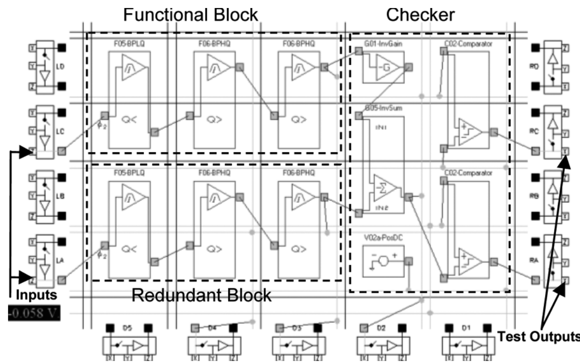

Fig. 13. Block diagram of the self-recovering scheme programmed into the AN10E40 FPAA.

software screen with the block representation of the proposed scheme.

Fig. 14 shows the waveforms at the output of both filters and the error signal acquired during a functional fault injection experiment. The setup is the same depicted in Figs. 11 and 13. A functional deviation of $+10\%$ in the higher corner frequency parameter (from 10 to 11 kHz) is injected in one of the two BP filters. A 10 kHz square signal is applied at the functional input. The considered tolerance window in this experiment is $\pm 58\ \mathrm{mV}$. Fig. 15 shows the output of both comparators of the checker when the error signal $(\varepsilon(t))$ exceeds the tolerance window limits.

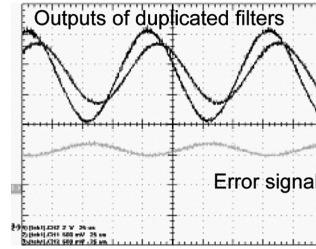

Fig. 14. Waveforms at the output of the filters and error signal acquired during a fault injection experiment.

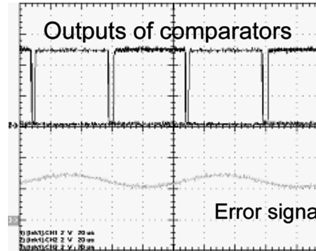

Fig. 15. Output of comparators when the error signal exceeds the tolerance limits.

From Fig. 13 one can see that the checker circuit requires 8 CABs when built with the Anadigm Designer software basic IP library. Thus, 12 CABs are available to implement the functional circuit and the redundant part. This way, the number of CABs available to implement the desired functional circuit is 6. This is a low number since only 30% of the whole device can be used in the functional design when using this self-recovering scheme. However, the checker is not a complex circuit, thus, if a customized design is used (IP provided by vendor) and the reference voltages of the window comparator are set externally, the error detection circuit can be built with 3 or 4 CABs (20% of available CABs). The remaining 80% are divided between the functional and redundant block, which means that 8 CABs can be available to the user defined design.

Depending on the design complexity, 8 CABs can be a reasonable number of blocks, since one can implement 4 second order filters, for example, or even a set of different linear and nonlinear functional blocks. Additionally, in critical applications, the implementation cost can be a secondary concern, as it is the case of radiation-exposed circuits. This way, if it is necessary to use more than one FPAA device, the increasing on system costs can be compensated by the fault tolerant characteristic associated to redundant schemes.

V. Fault Injection Into the Checker

The basic assumption to justify looking only at the checker in the remainder of this work is that the probability of multiple faults to affect the design block and the redundant block of the duplication scheme at exactly the same programming bits is extremely low. Thus, a given fault or (set of faults) will maintain the outputs of both functional blocks unaltered or cause the deviation of one or both blocks to be greater than the tolerance limits. This ensures that even under a collection of successive SEUs affecting the programming bits of the functional blocks,

it will be highly probable that an error indication will be delivered to the checker. Additionally, analyzing further the faulty behavior of the checker becomes essential since it is a universal block that can be used in any analog self-checking scheme independently of the analog function implemented.

Next, some definitions employed in digital self-checking schemes [13] are used to analyze the self-checking properties of the checker proposed in the previous section. These definitions consider a functional design block G (the checker, in our particular analysis), with input code space A, output code space B and a given fault set F, and are as follows:

- Definition D1: "G is *fault secure* with respect to F if, for all faults in F and all code inputs, the output is either correct or is a noncode word."
- Definition D2: "G is *self-testing* with respect to F if, for each fault in F, there is at least one code input that produces a noncode output."
- Definition D3: "G is totally *self-checking* (TSC) with respect to F if it is fault secure and self-testing with respect to F."

For analog circuits we may consider that the code space corresponds to the limits expected to a given signal, regarding features such as amplitude, fundamental frequency and harmonic content (features that impact the signal waveform). Concerning the redundant scheme proposed in this work and the voltage at the outputs of the design block (V_{oD}) and the redundant block (V_{oR}), the checker input code space must satisfy the following equation:

$$|V_{o_D} - V_{oR}| < |V_{ref}| \tag{1}$$

where V_{ref} is the absolute value of the window comparator references.

In the previous section the mechanisms that make it possible the proposed checker to detect behavioral deviations between the functional design block and its replica were demonstrated. Although no exhaustive fault injection campaign has been applied to the proposed checker to this point, the results obtained in the first part of this work for the oscillator module and the condition imposed by (1), allow us to conclude that it is highly probable that the checker be *fault secure* (considering a single fault hypothesis). However, these same results lead us also to the conclusion that it is highly probable that the checker is not *self-testing* (and consequently, not TSC) because some faults of the considered fault model can affect a unused resource of the DUT, and, for these faults, the output signals will belong to the code space regardless the faulty DUT.

Nevertheless, according to [14], in a self-checking scheme there is no need for the checker to be TSC, it suffices that it be strongly code disjoint, according to Definitions D4 and D5 below:

- Definition D4: "A network is *code disjoint* if it always maps code inputs onto code outputs and noncode inputs onto noncode outputs."
- Definition D5: "A circuit G is *strongly code disjoint* for a fault set F if before the occurrence of any fault, G is *code disjoint* **and** for every fault f in F, either a) G is *self-testing* **or** b) G modified by a fault f maps always noncode inputs onto noncode outputs, and if a new fault in F occurs, for the obtained multiple fault case a) or b) is true."

Fig. 16. Results of fault injection in the checker when the DUT is fault-free.

To check whether the block we are dealing with is *strongly code disjoint*, two sets of experiments are performed. First, bit-flip faults are injected in the checker bitstream when the DUT is considered fault-free. This experiment investigates the *self-testing* propriety of the checker. In the second set of the experiments, the procedure is repeated when a functional deviation is injected in one of the functional blocks. For both sets of experiments the input stimulus is a 2 V (peak) and 10 kHz square signal. The main objective of the latter is to investigate the checker reliability and the fault aliasing phenomenon that can occur when a fault in the functional block is not detected due to a fault that occurs at the same time in the checker. The results of these two sets of experiments are reported next.

A. Fault Injection in the Checker While DUT Is Fault-Free

In this set of experiments, bit-flips are injected in the 5 blocks that compose the checker, comprising 8 CABs. As each CAB is programmed through 208 configuration bits, and the fault injection is performed exhaustively, a total of 1664 faults are injected. Unlike some FPGAs that have special tools that allow the bitstream manipulation according to the FPGA resources, there is no such tool in the FPAA. In our case, the fault injection procedure was made in an exhaustive way, to guarantee that all programmable resources were affected by a bit-flip.

The DUT is kept with its original configuration (Fig. 12) with the functional and redundant blocks working properly. In this case the error signal must lie within the tolerance limits (checker input code space) and the output of the comparators of the checker must be at the zero value. However, if a fault affects the checker an error indication may occur, but, in this case, the checker detects an error originated in its own blocks.

From the 1664 injected faults, 283 (17%) generated an error indication at the checker outputs while 1381 (83%) did not change the checker outputs. These results are shown in Fig. 16.

These results show that only 17% of the injected faults in the checker programming bitstream where detected while most faults did not affect the checker output behavior. At a first look this can be considered a bad result, since the checker is not *self-testing* in respect to the fault model considered. However, as mentioned in Section II.B, and exemplified in Fig. 10, this low error detection rate is due to the low usage of the internal resources of the CABs that compose the checker. Additionally, this low error detection rate does not mean that the checker is unreliable since these experiments where performed considering the DUT and its replica as fault-free blocks. The reliability of this scheme is tested in the following described experiments in which the ability of the checker to detect deviations in the

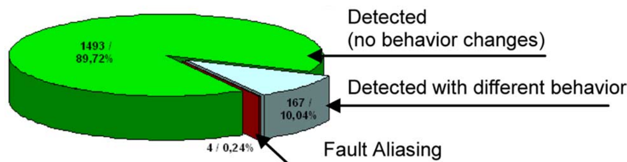

Fig. 17. Results of fault injection in the checker when a functional deviation is injected in the DUT.

DUT, while bit-flip faults affect the checker blocks, is investigated (*strongly code disjoint* property).

B. Fault Injection in the Checker While DUT is Faulty

In this set of experiments bit-flip faults are injected in the checker, just like the procedure described before. However, differently from that experimental setup, in this essay the DUT is considered faulty. A functional deviation is injected in one of the duplicated blocks. The higher corner frequency of the BP filter is deviated in $+10\%$ from its nominal value (from 10 kHz to 11 kHz). This variation is sufficient to result in an error indication at the outputs of the checker if it is working properly. The goal of these specific experiments is to investigate weather fault aliasing can occur when a bit-flip affects the checker blocks, or, in other words, if the checker can be considered *strongly code disjoint*. In this case, fault aliasing is considered as the inability of the checker to indicate that there is discrepancy among the functional and redundant blocks when actually there is an injected functional deviation in one of them.

Results show that from the 1664 injected faults only 4 (0,24%) caused the fault aliasing phenomenon, while the remaining faults (99,76%) did not affect the ability of the checker to indicate the functional deviation injected into the DUT, as shown in Fig. 17. This can be considered a good result concerning the checker reliability, since it represents a low probability of fault aliasing, although it cannot yet be formally considered *strongly code disjoint*.

It is possible to conclude that this low aliasing probability is due to the natural redundancy of the proposed checker, since there are two comparators to indicate a possible discrepancy in the duplicate blocks. The experiments show that, in most cases, when one of the comparators fails to indicate an error, the other is able to detect it. These specific cases, and other faults detected when the outputs of comparators are different from the fault-free case (noncode words at the checker comparators outputs), represent near 10% of the total injected faults and are shown in Fig. 17 as "detected with different behavior."

Although the checker presents a natural redundancy, from Fig. 11 one can see that it has a weak point, which is not duplicated: the output of the subtractor block, more specifically the adder block (as mentioned before, the subtractor is composed of an adder with an inverter in one of its inputs). If a fault in this block (or in neighboring blocks) makes the error signal to be zero when actually it should be not null, fault aliasing will occur. In fact, for all faults that generate aliasing, the error signal at the output of the subtractor is within the considered tolerance window.

In order to identify how this four faults cause the aliasing, another set of experiments is carried out. The first step is to identify what blocks of the checker the affected bits belong to. Although the weak point mentioned before is the output of the subtractor block (adder), only two of the four aliasing cases are originated from a fault in the adder bitstream. The other two cases are due to faults in the inverter (one of the subtractor inputs) and in the comparator whose reference is positive (top comparator in Fig. 13). Results showed that these faults forced the error signal to be zero. However, the inverter fault did not affect the inverter behavior and the same occurred to the comparator. A possible reason for that is the interaction between the neighboring blocks, by means of local interconnections. As the inverter and the comparator are neighbors to the adder, faults in these blocks can activate the local interconnections that link these blocks to the adder and possibly deviate the error signal to the internal reference, for example.

The hypothesis of interaction with neighboring blocks was confirmed by changing the location of the blocks in the programmable array and injecting the same faults that generated aliasing. In this case the bit inversions that had previously generated fault aliasing, no longer affected the checker correct behavior. In fact, when changing the location of the blocks it is possible that only the location of the sensitive bits in the programming bitstream is changed. Therefore, if interconnection constructive and programming details are available, it may be possible to distribute the checker blocks in the array in a way that this neighbor interaction is reduced and, finally, make the checker *strongly code disjoint* by construction!.

VI. Conclusions

Some recent analog programmable components, the FPAAs, may have its programmability based on SRAM memory blocks. This fact can make the SEU problem in FPAAs as critical as it is for the FPGAs.

The preliminary goal of this paper was to study the effects of bit-flips potentially caused by SEU in the programming memory of SRAM-based FPAAs. The early results of this part of the work were presented in a previous paper [15]. In this study a series of bit-flip faults were injected in the considered device by modifying the value of the default bitstream of a programmable analog module (oscillator). The experiments showed that a single bit inversion can result in a very different configuration of that previously programmed, and, in some cases, the whole analog application can be affected. The experiments also showed that a SEU affecting a memory cell that controls a not used resource can result in a correct functional behavior and the fault can remain undetectable. Only 33.65% of injected faults resulted in a detectable modification of behavior given to the high number of not used CAB programmable resources in the module implementation. However, when a SEU occurs in some memory cell that controls a used resource the consequences can be catastrophic to the system.

In the second part of the paper, the major and novel contributions of this work were presented. It was shown that when using the device considered in this work, a simple scheme can be implemented to allow on-line error detection and self-recovering of programming data if a functional behavior modification occurs. In this scheme the analog functional hardware is duplicated and a built-in error detection circuit compares the output

of both blocks. If the signals at the output of both blocks are significantly different the error detection circuit activates the reset sequence of the FPAA and the programming data is reloaded, correcting the error.

The reliability of the self-recovering scheme proposed was studied by injecting bit-flip in the section of the programming bitstream that controls the checker block. First, fault injection was performed when the functional and redundant block are working properly (error signal is within the tolerance window). In this case 17% of the injected faults make the checker outputs deviate from the fault-free operating values, while the remaining faults did not change the checker behavior, which means that the checker is not *self-testing* regarding the fault model considered. However it is not considered a major problem, if the design block is working properly.

The most important set of experiments carried out in this work was the checker fault injection when a functional deviation was also injected in one of the duplicated blocks (making the error signal to exceed the tolerance limits). In these experiments the fault aliasing probability of the scheme (considering the bit inversion fault model) was studied. Results showed a very small aliasing occurrence (0.24% of the injected faults), which means that the checker is intrinsically robust considering the faults that affect its own programming memory blocks. Furthermore, it was discussed how these aliasing cases can be overcame in such a way that the checker becomes *strongly code disjoint* by construction.

We also performed an estimate of the Soft Error Rate (SER) for the FPAA at sea level. In this estimate it was considered a FIT (Failure in Time) rate of 1000/MBit for the FPAA technology $(0.6\ \mu\mathrm{m})$ [16]. Considering that all CABs are occupied and that 35% of the programmable resources are used in the circuit implementation (according to our experimental data), we estimate a SER of approximately 2.4×10^{-9} errors per hour at sea level.

Regarding the time needed to data reloaded when an error is detected, it depends on the programming timing of the device. When using the serial ROM mode for the AN10E40 FPAA configuration, this time can be as fast as 705 μs [11]. This interruption time can be tolerable in several analog applications, like instrumentation and audio, for instance. It is important to point out that the analog operation bandwidth of the AN10E40 FPAA is limited to 500 kHz [11]. Furthermore, there is another way to set up the FPAA programming: by using a dedicated microprocessor. In this programming mode the data is loaded through a byte-wide interface and the complete programming cycle can be accomplished in 125 μs [11]. This approach raises the system implementation costs, but, besides the faster programming time, it allows different configurations to be loaded into the device, which, depending on the application, can be a very useful feature.

Finally, with all these results, it is possible to conclude that, by using the available programmable resources of the FPAA, a reliable and cost-effective scheme can be built in order to mitigate SEU effects in the considered device.

REFERENCES

[1] L. Znamirowski, O. A. Paulusinski, and S. B. K. Vrudhula, “Programmable analog/digital arrays in control and simulation,” in *Analog Integrated Circuits and Signal Processing*. Norwell, MA: Kluwer Academic Publishers, 2004, vol. 39, pp. 55–73.

[2] J. Hereford and C. Pruitt, “Robust sensor systems using evolvable hardware,” in *NASA/DoD Conf. Evolvable Hardware (EH’04)*, 2004, p. 161.

[3] G. C. Messenger, “A summary review of displacement damage from high energy radiation in silicon semiconductors and semiconductors devices,” *IEEE Trans. Nucl. Sci.*, vol. 39, no. 3, Jun. 1992.

[4] F. L. Yang and R. A. Saleh, “Simulation and analysis of transient faults in digital circuits,” *IEEE Journal of Solid-State Circuits*, vol. 27, Mar. 1992.

[5] A. Anghel, D. Alexandrescu, and M. Nicolaidis, “Evaluation of a soft error tolerance technique based on time and or hardware redundancy,” *Proc. of IEEE Integrated Circuits and Systems Design (SBCCI)*, pp. 237–242, Sep. 2000.

[6] C. Carmichael, “Triple module redundancy design techniques for virtex series FPGA,” *Xilinx Application Notes 197*, vol. 1.0, Mar. 2001.

[7] F. Lima, L. Carro, and R. Reis, “Designing fault tolerant systems into SRAM-based FPGAs,” in *Proc. of Design Automation Conf. (DAC’03)*, 2003, pp. 250–255.

[8] P. Adell, R. D. Schrimpf, H. J. Barnaby, R. Marec, C. Chatry, P. Calvel, C. Barillot, and O. Mion, “Analysis of single-event transients in analog circuits,” *IEEE Trans. Nucl. Sci.*, vol. 47, Dec. 2000.

[9] T. L. Turflinger, “Single-event effects in analog and mixed-signal integrated circuits,” *IEEE Trans. Nucl. Sci.*, vol. 43, pp. 594–602, Apr. 1996.

[10] R. Leveugle and A. Ammari, “Early SEU fault injection in digital, analog and mixed signal circuits: A global flow,” in *Proceedings of the Design, Automation and Test in Europe Conf. and Exhibition (DATE’04)*, 2004, pp. 1530–1591.

[11] Anadigm AN10E40 User Manual Anadigm Company, 2002 [Online]. Available: www.anadigm.com

[12] Anadigm Designer IP Module Manual Anadigm Company, 2002 [Online]. Available: www.anadigm.com

[13] D. A. Anderson, Design of Self-Checking Digital Networks Using Coding Techniques CSLiUniv, Illinois, 1971, Rep. 527.

[14] M. Nicolaidis and B. Courtois, “Stongly code disjoint checkers,” *IEEE Trans. Computers*, vol. 37, pp. 751–756, 1988.

[15] T. R. Balen, F. L. Kastensmidt, M. Lubaszewski, and M. Renovell, “Single event upset in SRAM-based field programmable analog arrays: Effects and mitigation,” in *Proc. of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2007, pp. 192–197.

[16] Soft Errors in Electronic Memories—A White Paper Tezzaron Semiconductor, 2004 [Online]. Available: www.tezzaron.com