Chapter 4
A Fault Injection technique oriented to SRAM-FPGAs

H. Guzmán-Miranda, J. Barrientos-Rojas, and M.A. Aguirre

Abstract Fault injection is an accepted method for emulating the effect of ionizing radiation on digital electronic circuits. It can be oriented either to ASIC designs or to SRAM-FPGA designs. When the target device is an SRAM-FPGA the injection has to be assessed both in the functional plane and in the configuration plane. It has been demonstrated that the classical protections oriented to the functional structure are not enough, so the configuration plane has to be analyzed, in the same way. This paper describes the adaption of the FT-UNSHADES2 platform as a fault injection system that tests faults in the configuration plane. The mechanism that assesses the effect of faults in the configuration is read-modify-write, in cycles of inject and repair, based on partial reconfiguration.

In this paper the authors categorize that there are four types of possible faults in the FPGA that should be considered: unrelated, non-damage, outer-propagated and inner-propagated. Faults in the unrelated and non-damage configuration bits are affordable and can be fixed using scrubbing techniques. The damage and propagated faults propagate from the configuration plane to the current data processed and a complete scrubbing followed by a master reset should be asserted to recover the functional behavior of the device.

Other result found is the relationship between the faults in the functional observability and the configuration bits. A result that only can be found if the injection system can distinguish between the faults over the above mentioned planes.

4.1 Introduction

SRAM-FPGAs are digital electronic devices that provide an attractive solution to many aerospace applications [1]. They introduce certain flexibility to the airborne systems and space payloads which allow the actualization and improvement of the electronic subsystems, and also deal with the possible obsolescence of their components [2].

H. Guzmán-Miranda (✉) • J. Barrientos-Rojas • M.A. Aguirre
Departamento de Ingeniería Electrónica, Escuela Superior de Ingeniería,
Universidad de Sevilla, c/ Camino de los Descubrimientos s/n, Sevilla 41092, Spain
e-mail: hipolito@gie.esi.us.es

F. Kastensmidt, P. Rech (eds.), *FPGAs and Parallel Architectures for Aerospace Applications*, DOI 10.1007/978-3-319-14352-1_4

The main drawback of this kind of devices is their extreme sensitivity to ionizing radiation due to the huge quantity of memory cells that compound their structure and the large critical area exposed.

Faults in configuration bits (CB) react in a different way than faults in user registers. While faults in user registers are treated as transient anomalous values that produce corrupted states in the cadence of the circuit, faults in the configuration bits have to be treated as structural modifications which remain permanent until the configuration is overwritten or repaired. Classical protections introduced in the design structure, like Triple Modular Redundancy (TMR) [3, 4] are still insufficient, because configuration faults can affect simultaneously to circuits belonging to several clock domains, or propagate the fault to user logic.

Errors in the configuration are much more probable than the user register ones, due to the abundance of sensitive points. They can be detected by means of a complete *Readback* of the device, in the same way than a normal SRAM-memory [5].

However, the number of CBs related to a particular design is a small fraction of the total CBs. Xilinx has developed a special mitigation method based on the so called Soft Error Mitigation (SEM) core [6]. It combines with an option in the *bitgen* application known as "essential bits". This option generates two files that determine the CBs that are related to the design. Essential bits are obtained by means of a static analysis of the design; this analysis calculates those CBs that are related to the implementation of the design, regardless of their actual value, and are strictly part of the configuration of any element of the FPGA. LUTs, BRAM contents and FF contents are not included in this file.

We take the advantage of this option for the adaption of the FT-UNSHADES2 tool to the injection of faults oriented to the essential bits. We will characterize the tool with a small example.

The rest of the paper is organized as follows: A general introduction about how the fault injection procedures are, when an SRAM FPGA is the user platform. In the third section the FT-UNSHADES2 system is described and also the skills implemented in the system to target the FPGA as object of injection and finally a case of study is introduced to show the system behavior.

4.2 Fault Injection in SRAM-FPGA

4.2.1 Fault Injection Oriented to User Registers

SRAM-FPGAs are a very attractive solution for fault injection tasks when the designer wants to analyze how the design structure treats the faults: where the weakest elements are and how the protections work within the circuit structure. There are several proposals in the literature for platforms that develop this concept. Basically they consist of the implementation of a mechanism that produces one or several spontaneous changes in the content of implemented registers (WHERE) at any clock cycle of the execution workload (WHEN), and if there is a predefined method

of injection (HOW). The most important characteristic of this procedure is that the injection is performed over the circuit registers, or registers instantiated due to the high level description of sequential statements. A good survey can be found in [6].

A very well-known system based on this approach is "Autonomous Emulation" system [7], that make use of any kind of SRAM-FPGA, instrument each register the circuit for being tested and make a fast emulation of the system in fault. The platform ASTERICS [8] is another example of how to inject faults reconfiguring.

4.2.2 Fault Injection Over the Configuration Plane

Another category that is completely different (but almost always confusing) is those platforms that are dedicated to study the proper SRAM-FPGA as the target device. This problem is completely different because the SEE can impact not only on the instantiated registers but also in the configuration bits of the elements that are related to the design [9, 10]. The consequences of SEE are totally different than in the former category because the faults remain in the configuration bit over time and will only be removed when the configuration is overwritten. During the time that the fault is active the fault can be propagated to the user logic and then the processed state becomes corrupted.

Overwriting the state of the configuration is done periodically, and the timing is known as scrubbing period, so the time between reconfigurations is the vulnerable time. In a scrubbing cycle, the configuration is overwritten "softly", in such a way that the current state represented by the content of the memory elements of the FPGA remains untouched. This method by itself does not detect if the state of the design is currently corrupted or not, and the scrubbing process takes extra and undesired power due to the internal commutations of the transistors. One goal of the design is to optimize time and power consumption.

Mitigation with scrubbing is not enough, because data remains already corrupted after the soft reconfiguration, so it is necessary to introduce another mitigation technique, in this case, focused on the repairing of data.

Several platforms have been created, mainly for the measurement of the global sensitivity of a design to SEE in a particular FPGA device. The main goal consists in studying the design behavior when the device is configured, and then reconfigure it in a blind manner. The number of errors found versus the number of injections is considered as a measurement of how reliable, running in this device, the circuit is. This is a very inefficient mechanism due to the large amount of configuration bits unrelated to the design. These bits are sensitive from the point of view of the device, but most of them are not, considering the configured action. Many of the injections can be saved if we can distinguish between the related bits and the unrelated.

Few platforms have been developed to test designs running on the SRAM-FPGA (e.g. FLIPPER tool) [11–15], and few correct approaches have been addressed because a platform is needed for the exact device that is going to be flown in the final application. One solution is to study the design as a hard macro of the design,

a part of the identical configuration that will integrate the final device. This is a method to migrate the design within the same technology.

4.2.3 Static vs. Time Zero Analysis

There are approaches that provide information about the reliability of the design just by studying the possible related bits. This is done by software tools like STAR [13] and the Xilinx *bitgen* routines for essential bits determination [16]. The former goes ahead, because it provides rules for a new placement that diminishes the number of critical configuration bits: the RORA tool [18].

Static analysis provides information regardless of whether a particular resource is used or not in the execution of the design. Of course this is the best situation but when the user has to take actions for reducing the number of critical resources the situation is not clear, as there is not idea about the sensitivity of each zone of the circuit to make it more reliable. One possible solution is the use of the SRAM-FPGA executing the design with a representative application workload. The configuration is modified in the clock cycle zero and the effect of the fault is recorded during the workload if there is any propagation path.

If any critical point is not detected, either its effect remains latent within the circuit or the resources are not well stimulated by the workload [5].

Time zero analysis is less restrictive and more realistic than the static one. It identifies the part of the circuit that can propagate faults. It consists of injecting the fault before the execution of the circuit is started. Normally it starts with a reset assertion and if the circuit is modified by the fault in the configuration, the fault is propagated during the workload to any primary output. Platforms watch this sequence of values and detect any anomaly or wrong value. If this is done, the injection is representative of an error rate for a specific implementation of the circuit and workload.

There are few but well known platforms described in the literature. All of them are devoted to the study of fault injection rates injecting using several techniques and internal resources of the Xilinx FPGA.

Again the next step is to provide rules for a new and more reliable implementation. The work should be done iteratively to minimize the criticality of the implementation. Next section will present the option of dynamic injection. The idea is to open the injection to any clock cycle of the workload.

4.3 FT-UNSHADES2 in FPGA Mode

Authors intentionally have omitted the platform FT-UNSHADES2 [17]. This platform traditionally has been described and classified in the set of tools dedicated to test SEE oriented to inject faults in the user registers that belong to the custom logic,

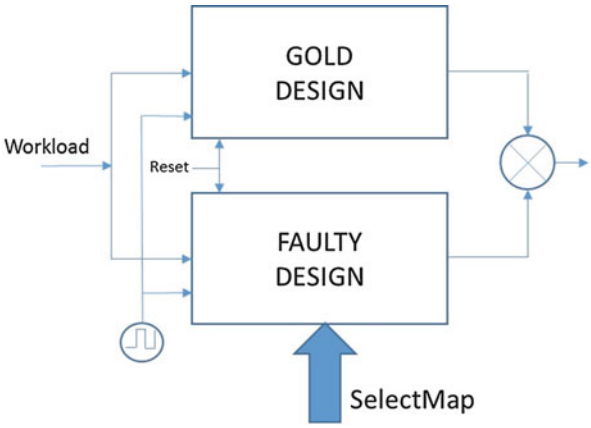

Fig. 4.1 Hardware for injection model

but in this section we are going to describe the adaptation of the tool to the test of designs implemented in FPGAs, so the injection procedures are produced over the *configuration bits*, instead of the *user registers* of the FPGA.

The principle of the method is essentially the same: use partial reconfiguration to read, modify and write a particular frame of the configuration map where the CB is allocated. The identification of the injection point provides a rich information about the reliability of the design, or some critical parts of it (Fig. 4.1).

The adaptation of one method, called *ASIC mode,* to the other, called *FPGA mode,* is at API level. Very low level commands are basically the same. The structure of the system is still based on two identical FPGAs running in parallel, synchronized, both receiving the same sequence of stimuli, and only one of them receiving the injections. The comparison is cycle by cycle at the primary outputs. This procedure is performed repeatedly, always with a known starting state at cycle 1. Every execution of the workload is called *run*. At each run one or several injections are performed selecting the target registers (WHERE and HOW) and clock cycles (WHEN) to inject.

The effect of a fault can be inspected either by on line comparison with the primary outputs coming from the twin FPGAs (*error* faults) or by reading the internal state of all the registers of both FPGAs and comparing their values one to one. This method detects the internal *latent* faults.

In *ASIC mode* the faults are injected only in user registers, faults can be compensated through functional structures, so they can be repaired if the circuit is prepared to. At every injection cycle, the signal reset is asserted in order to initialize the registers content.

In *FPGA mode* the faults are injected in CBs. The abundance or possible target bits (tens of millions) makes the problem very difficult to deal with if there is no previous selection of these CBs. Xilinx has provided a tool very similar to STAR that extracts the CBs that are related to the actual implementation of the design. The rest of CBs are unrelated and should not affect to the design behavior if they receive a bit flip. The tool provides in fact two files, one marking the bits that are related and

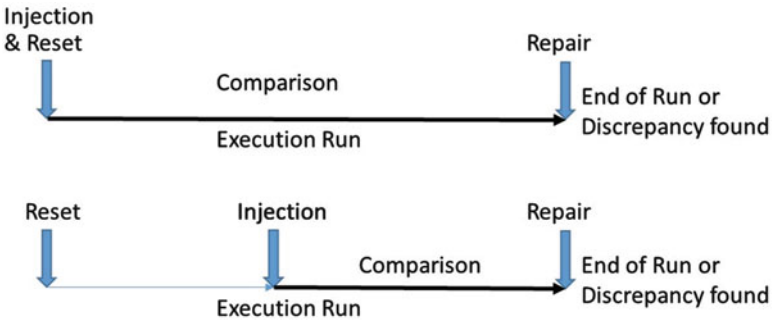

Fig. 4.2 Dynamic injection execution model. (**a**) Time zero injection (**b**) Variable time injection

other with the theoretical value of those configuration bits. These files are part of a mechanism of on-line repairing of the SEE in the configuration plane. Xilinx has developed this procedure for Virtex 5, 6 and 7 families.

FT-UNSHADES2 has taken these files as reference for the FPGA mode for a technique based on inject and repair cycles. The points of injection are determined by the essential bits file and these bits are the ones attacked. The method is based on the idea that when a CB is attacked, this change of value will not affect another configuration bit, otherwise the technique is not strictly valid, because the effect of a fault would remain present in the FPGA after a reset. The attack model is described then, as follows:

1. Selection of the configuration bit and clock cycle that will be attacked (WHERE and WHEN).
2. Initial reset, and execute the application until the injection instant.
3. Using partial reconfiguration, the frame that corresponds to the CB is retrieved from the FPGA
4. (alt) this step can be substituted by the theoretical value coming from the .ebc file.
5. Write the opposite value in the desired CB
6. Resume the execution and compare primary output values.
7. While execution, compare with Gold theoretical values.
8. If a discrepancy is found or end of run is reached, repair the CB, following the step 3.

This mode is repeated in many execution runs following the procedure established in the method of injection selected. If time zero is selected, then the injection is produced just at the beginning of the experiment. If time is a variable, then the system is driven to any clock cycle following the programmed selection pattern (Fig. 4.2).

The user can proceed to send a complete configuration at any certain number of injections in order to refresh it and erase any unexpected lateral effect.

Also the system allows avoiding the step 7 and studying possible accumulated effects.

The most important difference between this system and other developments is the consideration of the time as variable. It is very important to dedicate effort to the elaboration of the test vectors, because they must be representative of the real application, in order to make the results of the test more realistic and valuable.

The second advantage of the current platform is that the designer can compare between how the faults behave in the same framework from the point of view of ASIC mode and FPGA mode, and compare both. This is especially interesting, because in normal flight, the faults are detected using a specific detection circuit and monitored at any primary output.

The current system is based on Virtex 5 technology, and all the transactions are performed through the SelectMap port in parallel mode.

4.4 A Case of Study

This chapter will explain a case of study that characterizes the system. All the results come from the FT-UNSHADES2 platform. We have developed a set of examples to characterize the process. The examples *b01*, *b13*, *b20* and *keccak* sponge function, the former are complex circuits taken from the ITC99 benchmark suite and the latter is part of a cryptocodec found in internet. All of them are examples that have available the high level description code with a stimuli set. In the case of keccak example we have used two different sets of vectors to show the dependence of the observability on the application.

Previous to the experimental activity a study about the essential bits has been performed. For a blind attack, a complete sweep of all the used frames and all the configuration bits has been performed injecting in a blind way, say, if they are in the subset of essential bits or not. Then the essential bits were attacked. All the critical bits were detected in both subsets matching almost perfectly, with the unique difference of several bits in some frames of the blind sweep, corresponding to the LUTs and FF contents, that are not part of the essential bits. This experiment was performed over *b01* and *b13* circuits.

The results of these previous experiments confirm that the essential bits are a good subset for an effective fault injection campaign, as promised by Xilinx. However there are user memories that are not included in the essential bits subset. These bits should be added to those bits that are critical.

The first analysis has been performed to compare static analysis versus dynamic analysis. This experience pursues to compare the basic injection process. The number of injection points is given by the essential bits static analysis generated from the *bitgen* tool. In our examples set, the target device is XCV5FX70T, containing 18,936,096 bits.

For all the benchmarks, the first cycle is the assertion of the reset signal. This vector erases the possible functional value stored in previous execution runs and starts the current one from a known state (Table 4.1).

Table 4.1 Characterization of each benchmark

Benchmark	Registers	Workload	Essential bits
b01	10	245	3,216
b13	66	7,640	14,572
b20	434	10,933	475,230
keccak1	1,683	856	622,168
keccak10	1,683	8,798	622,168

Table 4.2 ASIC mode results

Benchmark	Inject.	Errors	Percentage	Time (s)
b01	10,000	7,666	76.6	5
b13	10,000	8,072	80.7	42
b20	100,000	16,105	16.1	428
keccak1	50,000	45,421	90.8	27
keccak10	50,000	46,420	92.0	239

Table 4.3 FPGA mode in time zero

Benchmark	Inject.	Errors	Percentage	Time (s)
b01	10,000	6,765	67.6	25
b13	10,000	5,960	59.6	426
b20	200,000	1,525	0.75	4,840
keccak1	622,168	40,578	6.52	3,637
keccak10	622,168	49,190	7.91	30,896

The pair workload/circuit is firstly tested as "ASIC mode" in order to test the fault propagation capabilities of each benchmark. The keccak example is used twice with different input vector databases. One is a single frame of data, and the second comprises ten frames (Table 4.2).

This experiment shows how the circuit structures propagate the faults. B01, B13 and keccak are examples that provide a high level of observability of faults, because they can be easily propagated to the primary outputs. It is very important to test, for each design-stimuli pair, their respective fault propagation capacity. Attacking the user registers, it is possible to measure this effect.

The keccak example shows that there is a dependence with how the stimuli set helps this propagation.

The next table shows the examples injecting only over those frames and configuration bits that belong to the CLBs. The injection technique implemented is the previously described inject and repair one. Table 4.3 shows the results for *Time Zero* experiment:

For comparison, the same experiment has been made but randomly selecting the injection *cycle*. A small decrease of the percentage of detected faults is expected, due to faults that could not have enough clock cycles to propagate to the outputs. Table 4.4 shows this effect with a smaller percentage of errors in all the examples. This situation is much more realistic than the previous one.

Table 4.4 FPGA mode in random time

Benchmark	Inject.	Errors	Percentage	Time (s)
b01	10,000	6,275	62.7	26
b13	10,000	5,366	53.6	440
b20	200,000	1,239	0.60	4,937
keccak1	1,000,000	49,131	4.91	5,936
keccak10	2,000,000	153,612	7.68	99,671

The keccak10 experiment is performed about three times per configuration bit. This shows that the experiment becomes similar to the time zero one. As the percentage shows, the time zero will be an upper bound of the real experiment, more pessimistic than the random time experiment.

The first conclusion is that not all the essential bits present errors. That means that the essential bit set is compounded by two subsets: the first one, is the critical ones, where faults introduce errors in the processing data and are detected at the outputs affecting to the processed data. The second is related to those bits whose error produces perturbations only in the propagation time of the connections, so they only change the parasitic capacitances of the wires. They are difficult to detect, but easy to prevent. In fact the critical ones are the candidates of being measured and if possible, mitigated. They give the real vulnerability degree of the design running in the current FPGA. The first group needs to be repaired using any logical mitigation technology plus the necessary scrubbing process to erase the errors.

These results also show that the FPGA mode is strongly related to the ASIC mode. The global observability of a design shows the propagation capacity of a particular design to the detection mechanism, that in these examples are simply the primary outputs. The experiments over B01, B13 and keccak circuits have high controllability and observability so it is expected that faults have an easy propagation to primary outputs. However B20 has a bad architecture for propagating faults. These numbers do not show that there is a high difference between the time zero experiment and dynamic experiment, but they show that the capacity of a design to propagate the perturbation is a very important measurement of its behavior.

4.5 Conclusions

This paper presents, for the first time, a flexible platform that is ready to perform fault injection over designs that are synthesized specifically for FPGA. The paper discusses the differences between the ASIC and FPGA modes, where there is a connection between them. Also this paper shows the procedure for the robustness assessment of a design, and how to implement the design in one device and translate it to another that belongs to the same family. It is also shown the influence of the workload in the processing data, showing that the workload has to be representative of the final functionality. This paper shows how different models of SEU tests can offer results depending on the timing scheme of the study.

Further work will study larger and more complex designs where new conclusions can be extracted.

Acknowledgments The authors would like to thank Junta de Andalucía, Spain for funding the EDELWEISS: Design of a Highly Efficient Intra-Satellite Wireless Communications System project (reference P11-TIC-7095), the European Space Agency for funding the FT-UNSHADES2 project (reference PI-0072/2010), and Luis Sanz for his help and insight with the mass processing of the fault injection results.

References

1. Heiner J, Sellers B, Wirthlin MJ, Kalb J (2009) FPGA partial reconfiguration via configuration scrubbing. In: Proceedings of the field programmable logic conference 2009, PL'09, pp 99–104
2. Guzman-Miranda H, Sterpone L, Violante M, Aguirre MA, Gutierrez-Rizo M (2011) Coping with the obsolescence of safety- or mission-critical embedded systems using FPGAs. IEEE Trans Ind Electron 58(3):814–821
3. Rollins N, Wirthlin M, Caffrey M, Graham P (2003) Evaluating TMR techniques in the presence of single event upsets. In: Proceedings for the 6th annual international conference on military and aerospace programmable logic devices (MAPLD) Washington, DC, NASA Office of Logic Design, AIAA, Sept 2003, p P63
4. Wirthlin M, Johnson E, Rollins N, Caffrey M, Graham P (2003) The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets. In: Proceedings of the 2003 IEEE symposium on field-programmable custom computing machines, 9–11 Apr 2003, pp 133–142
5. Quinn HM, Black DA, Robinson WH, Buchner SP (2013) Fault simulation and emulation tools to augment radiation-hardness assurance testing. IEEE Trans Nucl Sci 54(1):252–261
6. Morgan K, Caffrey M, Graham P, Johnson E, Pratt B, Wirthlin M (2013) SEU-induced persistent error propagation in FPGAs. IEEE Trans Nucl Sci 60(3):2119–2142
7. Lopez-Ongil C, Garcia-Valderas M, Portela-Garcia M, Entrena L (2007) Autonomous fault emulation: a new FPGA-based acceleration system for hardness evaluation. IEEE Trans Nucl Sci 54(1):252–261
8. Velazco R, Mansour W, Pancher F, Costa Marques G (2011) ASTERICS—a platform for the simulation of radiation effects on processors by fault injection. Open access paper: https://www.rd-access.eu/edatools/system/files/_edaTools/ubooth_submission/2011/209.pdf
9. Bernardi P, Sonza Reorda M, Sterpone L, Violante M (2004) On the evaluation of SEU sensitiveness in SRAM-based FPGAs. In: Proceedings of the international on-line testing symposium (IOLTS), 2004, pp 115–120
10. Lima F, Carmichael C, Fabula J, Padovani R, Reis R (2001) A Fault injection analysis of Virtex FPGA TMR design methodology. In: IEEE European conference on radiation and its effect on component and systems, 2001, pp 275–282
11. Nazar GL, Carro L (2012) Fast single-FPGA fault injection platform. Defect and fault tolerance in VLSI and nanotechnology systems (DFT), 2012 IEEE international symposium on, 3–5 Oct 2012, pp 152–157
12. Bolchini C, Castro F, Miele A (2009) A Fault analysis and classifier framework for reliability-aware SRAM-based FPGA systems. In: Proceedings of the international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT), 2009, pp 173–181
13. Sterpone L, Violante M, Rezgui S (2006) An analysis based on fault injection of hardening techniques for SRAM-based FPGAs. IEEE Trans Nucl Sci 53(4):2054–2059
14. Sterpone L, Violante M (2007) A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs. IEEE Trans Nucl Sci 54(4):965–970

15. Alderighi M, et al (2007) Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform. In: Proceedings of the 2007 international symposium defect and fault tolerance in VLSI systems, Rome, Italy, Sept 2007, pp 105–113
16. Xilinx App note Xapp 538. April 2012
17. Mogollon JM, Guzman-Miranda H, Napoles J, Barrientos J, Aguirre MA (2011) FTUNSHADES2: A novel platform for early evaluation of robustness against SEE. Radiation and Its effects on components and systems (RADECS), 2011 12th European conference on, 19–23 Sept 2011, pp 169–174
18. Sterpone L, Aguirre M, Tombs J, Guzmán-Miranda H (2008) On the design of tunable fault tolerant circuits on SRAM-based FPGAs for safety critical applications. Proceeding of the design automation and test in Europe conference DATE 2008. Munich, Germany. 2008, pp 336–341