Comunicación presentada en **XIV Jornadas de Computación Reconfigurable y Aplicaciones (JCRA14)**. Valladolid, 17-19 Septiembre 2014

Functional Verification for SEU Emulation in FPGA Designs

Igor Villata, Unai Bidarte, Uli Kretzschmar, Gorka Santos, Asier Matallana
Department of Electronics and Telecommunications
University of the Basque Country UPV/EHU
Bilbao, Spain

Abstract

In this paper techniques to detect failures in a FPGA are presented and their application to SEU (Single Event Upset) emulation applications is discussed. SEU emulation in FPGAs consists on programming the device with a configuration file that has an erroneous bit, emulating the effect of a SEU. Once the device has been erroneously programmed a verification method is needed to evaluate the criticality of the modified bits. In this work two verification approaches (hardware verification and software verification) are implemented, experimental results are obtained and conclusions are taken.

1. Introduction

Since smaller and smaller manufacturing technologies are being developed, electronic chips are becoming more and more vulnerable to radiation-induced SEUs. Due to radiation, a high-energy particle can hit the semiconductor storing an erroneous value at a memory cell [3]. The continuous device-size reduction is making more likely for these particles to hit inverse-polarized p-n junctions, which are the most sensitive parts of electronic devices. And thus, overall failure rate is getting higher.

According to [2] a fault is a physical phenomenon that leads to an error. And an error is an incorrect part of the system that can lead to a failure, which means that the equipment does not deliver correctly the service it has been designed for. Electronic devices are widely used in systems called safety critical, where a failure can lead to environmental damage, injury or death. So measures have to be taken when a radiation-induced SEU (fault) leads to a bit flip (error) that may provoke a malfunction (failure), which is not allowed in any way.

SRAM based FPGAs are more vulnerable to SEUs than other semiconductor devices. In [3] it is mentioned that SEU events have a greater impact in SRAM cells than in DRAM or in Flash cells. In [7] the architecture of a SRAM based FPGA is presented and the effects produced by radiation are analyzed. The key element of a FPGA is the configuration memory, where the information about hardware resources is stored. When a used bit in this memory flips the implemented design's functionality changes and it may provoke a failure.

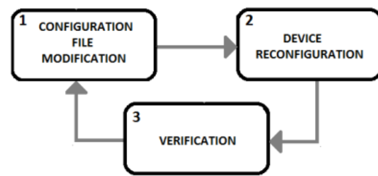
Knowing the failure rate of a system is always an issue of interest for both manufacturers and purchasers. The failure rate of a FPGA is not constant and is dependent on the implemented design.

As it is mentioned in [4] the failure rate is directly proportional to the amount of critical bits of a design. A configuration memory bit is labeled as critical when a modification leads to a malfunction of the system. SEU emulation consists on configuring the device with a faulty configuration file emulating the effects of radiation. In order to know if the introduced error has provoked a failure, a verification system is needed.

2. SEU emulation systems

SEU emulation systems in FPGAs are based on configuring the device with a corrupted configuration file emulating the effects of radiation-induced SEU. First of all one or more bits of the configuration file are modified. Then the FPGA is configured and finally the functionality of the device is tested so as to label the modified bit or bits as critical or non-critical.

At the following figure the functionality of a standard SEU emulation system is represented.



system flow

A SEU emulation system can be used for two purposes; to compare fault tolerant architectures and to estimate the overall failure rate of the design. 100% accuracy is not an essential issue when the main objective is to compare different fault tolerant architectures. Just the comparison between them gives a result that is valid for the developers [6].

When failure rate estimation is done, accuracy is required to be as close to 100% as possible. This means that all the bits that modify the functionality of the design have to be labeled as critical, while bits not modifying it must be considered as non-critical [5].

3. FPGA failure verification methods

3.1. Characteristics of verification methods

The desired characteristics for a failure detection method used for failure rate estimation are the following:

- *Coverage*: It represents the depth of the test. 100% coverage means that all input combinations are tested for all possible internal states. A high coverage is not mandatory when the SEU emulation system is used to compare different architectures. However, if it is used for estimating the circuit failure rate, all bits modifying the functionality of the circuit have to be identified and a high coverage rate is required.
- *Speed*: The time needed to perform a test. It is related to test coverage. A deep test may be slow, but it may be more accurate than a lighter test.
- *Integrity*: Introducing a verification system at the design may introduce modifications at the original design. UUT (Unit Under Test) has to be as close as possible to the final design. Failure rate is dependent on the implementation, so verification methods introducing circuit modifications are less accurate.
- *Universality*: This is the easiness a certain verification system has to be adapted to different designs. A universal test system is able to test different designs without applying significant modifications.

3.2. Online FPGA fault detection methods

Online fault detection methods are those that can detect faults during normal operation. Their main objective is to avoid the occurrence of a failure. These methods are usually redundancy based. This redundancy is achieved by duplicating hardware or processing redundant information (e.g. parity or hamming codes). These methods can also be able to mask the faults providing always a right output. This happens on TMR (Triple Modular Redundancy) type architectures. A summary is done at [8].

These methods are not suitable for critical bits estimation because substantial modifications have to be added to original design and overall failure rate is modified.

3.3. Offline verification methods

These methods are executed when operational function of the circuit is not being performed. Test vectors are sent to circuit's inputs and circuit's functionality is checked at outputs.

Offline verification methods satisfy the characteristics mentioned at point 3.1 and they are widely analyzed at point 4.

4. Offline verification methods

Test vectors are sent to circuit's inputs and outputs are monitored so as to determine if the

circuit fulfills the functionality it has been designed for.

4.1. Software offline verification

A software sends input vectors to the circuit under test and monitors the outputs. In [1] it is mentioned that a purely software approach has a poor testability resolution. As a result a good software test can be very long and slow.

As FPGAs are getting bigger and bigger, more and more complex circuits are being implemented in them. And the more complex a circuit is, the more difficult and expensive is to develop a good software test.

Another drawback for software-based test is the universality. The software that decides if the circuit under test is working properly is fully application-dependent. In consequence, different circuits need different test softwares, and code can't be reused.

In [5] the test process is implemented at a PC, which communicates with the FPGA through RS232 interface.

If a SoC combining FPGA and a hard microprocessor is used (e.g. Xilinx Zynq) software test can be done inside the chip. Due to the following reasons there is a small accuracy reduction. The implemented design has to be modified in order to route the signals towards the microprocessor. In addition, critical bits belonging to IOBs (Input-Output Blocks) are not detected.

4.2. Hardware offline verification (BIST)

Offline verification can also be done by hardware, which is named as BIST (Built-in Self Test). In [1] a summary is done about different kinds of BISTs. In figure 2 the architecture of a basic BIST scheme is represented.

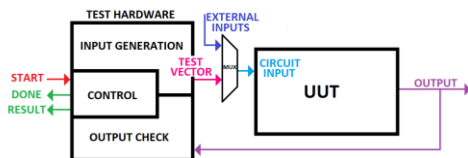


Figure 2: BIST architecture

Parts of a BIST are the following:

- *Unit Under Test (UUT)*: Is the circuit whose functionality is being tested.
- *Input generation*: This part is responsible of providing test vectors to the UUT. Vectors can be stored at memories or generated by hardware elements such as counters, adders or FSMs (Finite State Machines)
- *Output check:* This part analyzes the outputs of the UUT and decides if it is working properly. It is usually based on comparing the received output with a golden output.
- *Control*: Is the heart of the system. The Built-in Self Test is activated by an external agent using the START signal. Then the control unit takes the control of the circuits and activates the input multiplexer to send to the UUT the test vectors. When the test is finished it returns the control to the external agent activating the DONE signal and providing a result.

The main advantage of this approach is the universality. The output can be compared with the golden output at each clock cycle. So the output checker works regardless of the functionality of the circuit under test. In a software-based approach the software that decides if a certain circuit is working properly is fully application dependent.

Simulation tools of FPGA manufacturers such as Xilinx allow doing a netlist-level simulation. In this way it is possible to know the activity rate of each net under a known input vector. As a result the coverage level of a certain input vector can be evaluated. To achieve a 100% coverage activity must be present at each net of the design. The test must have a duration that allows the activity of all the nets to propagate to outputs. For a software-based approach it is difficult to estimate the coverage level of a test vector, and if it can't be assured that the coverage-level is close to 100% the failure-rate results can't be trusted.

BIST is much faster than a software approach, which means that high levels of coverage are achieved at a much shorter time.

The additional hardware needed to perform the verification test can be placed outside or inside the FPGA. If it is placed inside the FPGA during the validation process and then removed, validated design is different from the operating design.

However, this circuitry can be very small. In that case the gap between the two implementations would be small and the result would be accurate.

But validating an implementation different from the operating implementation is not recommended in any case. A solution is to keep the verification hardware at the operating implementation. It provides to the circuit a self-test functionality and the results obtained from validation process are valid.

Faults can be injected in the whole FPGA, so verification side errors may appear if a fault is injected at a bit belonging to verification hardware. If the verification subsystem is damaged it may never activate the done signal. If this happens, the agent controlling the verification hardware must take the flow control after having waited a prudential time. However, since this hardware is very small it is considered a minor effect.

The other possibility is to place the BIST hardware outside the FPGA. To keep the characteristic of universality the verification hardware needs to have the same clock as the design at the FPGA. So this circuitry needs to be placed at the same board as the FPGA. But in many cases modifications at PCB level are not available.

Design process is more complex for external validation than for internal validation for both hardware and software approaches. Configuring additional equipment and circuits means that different development tools and techniques have to be used adding complexity to the design. Additional pins and routing are needed so as to add this external equipment, so minimal modifications are added to the original design and results are not necessarily better.

In the following table characteristics of different combinations of offline verification approaches are presented.

Verification	*Coverage*	*Speed*	*Integrity*	*Universality*
Internal HW	High	High	Affected*	High
External HW	High	High	Minimally affected	High
Internal SW	Low	Low	Minimally affected	Low
External SW	Low	Low	Minimally affected	Low

Table 1: Offline verification approaches

*If test hardware is removed after performing the test the integrity is affected. If it is not removed there is no trouble.

5. Experimental tests

Two verification configurations have been implemented, internal hardware verification and internal software verification. Both are implemented at a Xilinx Zynq SoC, which combines a hard dual ARM microcontroller referred as PS (Processing System), with Programmable Logic (PL).

In the PS runs a software that inserts faults at configuration file and configures the PL using PCAP (Processor Configuration Access Port). Communication between PS and PL is done through GPIO (General Purpose Input/Output) peripheral of the PS.

5.1. Internal software verification

Input vectors and output check are done at a software running at the PS. This software also controls the selection signal of input multiplexer and disables the circuit's outputs when a test is being done. In the figure 3 the architecture of proposed internal software verification is shown.

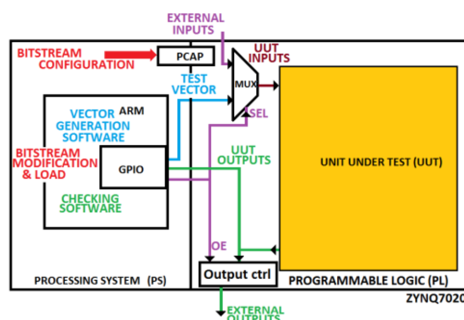


Figure 3: Internal software verification

It may be possible to develop using a Zynq device a universal software verification system. For example, saving the outputs for all clock cycles at a memory and sending them to the PS through a DMA. Those outputs are compared at the PS with a previously saved golden output vector.

But that is not much different from a hardware verification system. Note than the memory and the DMA are implemented at the PL, and only the comparison is done at the PS. This implementation is done so as to compare the coverage and speed level of a typical software implementation with a hardware implementation. A typical software verification system means that the test software is running at an agent (for example a PC) that can only access a few pins of the unit under test and does not have the same timing as the FPGA.

5.2. Internal hardware verification

Input generation and test check parts are implemented in hardware at the PL. The software stored at the PS activates a start signal in order to transfer the control to the verification hardware and waits until the test is finished. Then it reads the result and starts a new test.

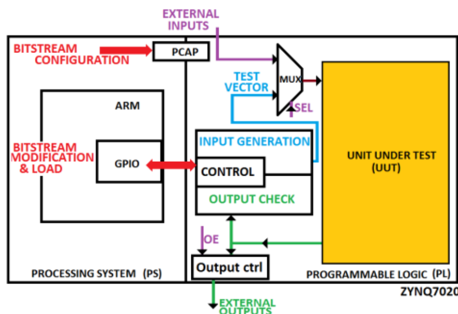


Figure 4: Internal hardware verification

The input generation hardware consists on a FIFO where an input pattern is stored. The coverage level of this pattern has to be previously validated. The time needed to perform a test is the length of the FIFO multiplied by the clock frequency.

For output check another FIFO is used. This FIFO has the same size as the input generation FIFO in order to check the outputs for the same time the inputs are generated. The content of this FIFO is fulfilled with the golden outputs of the circuit in case of no errors. While a test is being done these golden values are compared to circuit outputs and if a mismatch is observed during test time an error flag is activated.

The interface between PS and PL has at least three signals. A start signal set by the PS that launches the verification system; a done signal set by the PL that indicates that the test has finished; and a result signal, which is active if a failure has occurred.

The test time is limited to the size of the FIFOs. When a certain UUT needs a long vector to achieve a high coverage level the FIFOs can be too long. In that case other input generation and output check configurations would be more suitable.

5.3. Units under test

Two circuits have been tested applying this two configurations. The first is a bypass between the inputs and the outputs. This test is done in order to quantify the amount of errors provoked by the testing setup. The second tested circuit is an adder chain.

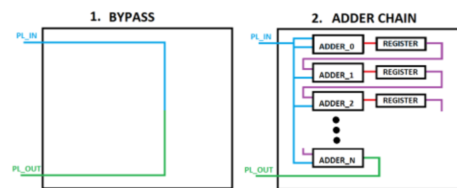


Figure 5: Units Under Test

for both circuits when a hardware approach is applied.

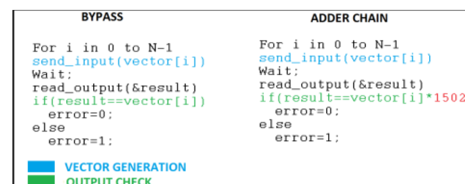


Figure 6: Software verification pseudocode

For this example it is quite simple to develop the code. The result is obtained just multiplying the input with the number of adders of the chain. But if the functionality is more complex it is not so simple. For example, the adder-chain can be used as a FIR filter. And it looks quite difficult to develop a software that checks if a FIR filter is working properly.

5.4. Obtained results

First the bypass design has been tested for both hardware and software configurations. Then the adder chain has been tested with 1500 adders.

The adder chain has been tested with different vector sizes. The software approach has been tested with 10, 30 and 100 vector size and the hardware approach with sizes of 2048 and 16384.

			Error injections	*Detected errors*	*time*
Bypass	software	10	17782464	1219	8h
Bypass	software	30	1782464	1442	10h 15min
Bypass	hardware	128	17782464	2443	45min
Bypass	hardware	16384	17782464	5631	2h 30min
Adder chain	software	10	17782464	1698093	8h
Adder chain	software	30	17782464	1784175	10h 15min
Adder chain	software	65	17782464	1799464	12h
Adder chain	software	100	17782464	1815405	30h
Adder chain	hardware	2048	17782464	2494407	1h 15min
Adder chain	hardware	16384	17782464	2574836	2h 30min

Table 2: Obtained results

6. Conclusions

Both hardware and software verifications are valid when they are used in order to compare two different architectures. But for estimating critical bits accurately hardware verification is more suitable because the coverage level can be validated through simulation. Hardware verification is much faster than software verification. Software verification is fully application-dependent, which means that different testbenches are needed for testing different applications. Hardware verification can be used for testing different applications just modifying the vector generation part.

Introducing a verification process modifies the implemented design and hardware verification introduces more errors than software verification, but the modification of the failure rate is low when verification hardware is small. The verification hardware gives the possibility to the system to perform an offline auto-test while it is in normal operation.

Acknowledgements

This work was carried out in the R&D Unit UFI11/16 of the UPV/EHU, and supported by the Ministerio de Ciencia e Innovacion of Spain within the projects TEC2011-28250-C02-01/2, and by the Basque Governments Department of Education, Universities and Research within the research fund of the Basque university system IT394-10.

References

[1] Agrawal, V.D.; Kime, C.R.; Saluja, K.K., A tutorial on built-in self-test. I. Principles, Design & Test of Computers, IEEE , vol.10, no.1, pp.73,82, Mar 1993

[2] Avizienis, A; Laprie, J.C; Dependable Computing: From Concepts to Design Diversity, Proceedings of the IEEE, Volume 74, Issue 5, pp 629-638, 1986

[3] Baumann, R.C. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies, IEEE Transactions on Device and Materials Reliability, Volume 5, no. 3, pp 305-316, Sept 2005

[4] Chapman, K. Virtex-5 SEU Critical Bit Information Extending the capability of the Virtex-5 SEU controller, Feb 2010

[5] Kretzschmar, U. ; Astarloa, A.; Jimenez, J.; Garay, M.; Del Ser, J. Compact and Fast Fault Injection System for Robustness Measurements on SRAM-Based FPGAs, Industrial Electronics, IEEE Transactions on , vol.61, no.5, pp.2493,2503, May 2014

[6] Sterpone, L. ; Violante, M. A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs, IEEE Transactions on Nuclear Science, Volume 54 , Issue: 4, pp. 965 – 970, August 2007

[7] Sterpone, L. Electronics System Design Techniques for Safety Critical Applications, Lecture Notes in Electrical Engineering, Volume 26, 2008

[8] Stott, E.; Sedcole, P.; Cheung, P. Y K, Fault tolerant methods for reliability in FPGAs, Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on , vol., no., pp.415,420, 8-10 Sept. 2008,