A Remote Verification Framework to Assess the Robustness of Circuits to Soft Faults

Gustavo R. Alves, Manuel G. Gericota, André V. Fidalgo
Dep. of Electrical Engineering — ISEP\LABORIS
Rua Dr. Antonio Bernardino de Almeida, 431
4200-072 Porto - PORTUGAL
{gca, mgg, anf}@isep.ipp.pt

***Abstract* — The growing number of circuits implemented in Field Programmable Gate Arrays (FPGAs) and the increased susceptibility, due to higher integration levels, of these devices to soft faults caused by radiation at ground level is leading the scientific and technical community to the study of new fault tolerant designs and solutions, and how they can be verified and validated. Using fault injection techniques and enhanced debug tools to inject faults in a circuit and observing its behaviour in the presence of such faults, respectively, is a proven solution for the previous verification and validation problem. This paper presents the underlying concepts for a remote verification framework to assess the robustness of circuits to soft faults. It comprises a verification platform and a set of verification services that can be used in a remote or local fashions.**

I. INTRODUCTION

The use of nanometer scales in FPGAs manufacturing leads to a greater integration and to a per unit power reduction, enabling them to grow both in size and complexity. As a result, FPGAs became an excellent alternative to the more and more expensive Application Specific Integrated Circuits (ASICs) for the implementation of complex circuits, even when high production volumes are forecast. Furthermore, due to their inherent configurability they enjoy an unsurpassed degree of flexibility enabling quicker turnaround time not only during the project phase but also in the field, without the prohibitive costs associated to ASICs in the same situations. These advantages have been reinforced and new possibilities added with the advent of dynamic and partially reconfigurable SRAM-based FPGAs (e. g. the Virtex family from Xilinx), which enable the dynamic customization of hardware functions to a particular system or application "on-the-fly" [1].

The need to reprogram the whole device, halting its operation, was one the major limitations associated with classic SRAM-based FPGAs. Additionally, the contents of all registers (state information) were lost when the component was reprogrammed. In recent FPGA generations, manufacturers addressed these issues by supporting partial device reconfiguration, which can take place concurrently with the system operation. Dynamic reconfiguration goes beyond "in-system reprogramming" since it does not interrupt the operation of the device.

However, the same distinctive features that make FPGAs more appellative also brought them some negative aspects. As a result of size increase the number of configuration memory cells in SRAM-based FPGAs grew exponentially and thus these devices became particularly vulnerable to radiation-induced faults, such as Single Event Upsets (SEU) and Multi-Bit Upsets (MBU) [2-4]. Although these faults do not physically damage the chip, their effects are permanent, since the functionality of the circuits mapped into the device is permanently altered.

In non-reconfigurable technologies, such as ASICs, protection against SEUs is restricted to flip-flops and memory areas, because logic paths among them are typically hard-wired. Nevertheless, Single Event Transients (SETs) — a charge transient induced in a combinatorial gate by the incidence of an heavy ion — may be propagated to flip-flop inputs, where they have a high probability to be registered, causing soft-errors in the user data. Besides, if a SET strikes a clock line, double-clocking may occur, leading to an extemporaneous update that may affect part of or all the flip-flops driven by that line (depending on the charge value and on line attenuation). Because the definition of logic paths in FPGAs relies on memory cells, these paths are also susceptible to SEUs.

Therefore, there is a critical need to incorporate fault tolerance mechanisms in all the circuits implemented in FPGAs. Furthermore, the robustness of those circuits should be verified before going to the field, preventing possible flaws that may lead to critical failures. That verification may be conducted by submitting the circuits to radiation campaigns, carried out with the objective of understanding if the effects of radiation-induced faults are effectively handled by the introduced mechanisms. The problem with radiation campaigns is not only their high cost but also the difficulty of accessing the radiation facilities necessary to carry them out. This prevents assessing the effectiveness, in the closest real working conditions, of many solutions proposed in the literature to make circuits more robust.

A number of fault injection approaches, proposed as alternatives to those (expensive) radiation campaigns, were described in several papers to turn around this situation. In these papers the effects of SEUs are emulated as bit-flips in the bitstream of the configuration memory of the FPGA, either

through changes in the original configuration bitstream or at run-time, through dynamic reconfiguration [5-7]. The greatest advantage of these methods is not only their comparatively smaller costs but also the smaller resources needed to implement them. Furthermore, the higher controllability of the experiments, in contrast to the unpredictability of radiation injection, enables a better diagnosis of the effects of each SEU. However, the implementation of such fault injection approaches still needs a great deal of knowledge concerning the mechanisms of partial reconfiguration and the set up of a rather complex infrastructure to automate the procedure. Since this infrastructure is independent of the functionality of the circuit implemented in the FPGA, a generic infrastructure may be built and made available remotely. Furthermore, this infrastructure may be complemented by other mechanisms such as those required to analyze the behavior of microprocessors (implemented as soft cores, inside the FPGA) under radiation, using on-chip debug (OCD) facilities [8].

The OCD infrastructures implemented by different families of processors share some common characteristics that form a core feature set, which usually includes run-control, breakpoint support and memory and register access. Some devices include more advanced features like watchpoints, program trace and real time debug capabilities. In general, an OCD is a combination of hardware and software on the microprocessor that requires some external hardware to be used, the basic requirement being some kind of communication link between the microprocessor and the host machine. The access to the OCD infrastructure is made through an interface port usually requiring an external debugger in between. When available, trace capabilities provided by the OCD can be an efficient mean to monitor fault propagation and effects [9]. Enhanced diagnosis capabilities, including profiling, are thus important in any fault injection technique. An industry consortium has been working on the establishment of a standard for OCD, which is formally designated as "IEEE-ISTO 5001, The Nexus 5001 Forum Std. for a Global Embedded Processor Debug Interface" [10].

While the OCD infrastructure is mostly used during the development phase, where usually the target circuit is in a safe-radiation environment, sometimes it is also used for field monitoring, where in such case it will also be affected by SEUs. It is therefore important to evaluate the effects of SEUs at the OCD infrastructure itself.

Furthermore, the OCD may also be used for injecting faults in the target circuit. This is another area of undergoing research, namely for complementing other fault injection techniques, which fail to provide the time and spatial controllability provided by OCD infrastructures.

This combined scenario provided the inspiration for a verification framework concept, formed by a verification platform and a set of verification services, to be made remotely available through the Internet.

This paper presents the ideas behind that verification framework and describes the expected benefits for any potential user. The rest of this paper is organized as follows: section 2 presents a conceptual overview of the verification platform; section 3 describes a number of remote verification services associated (or not) with the verification platform; section 4 presents the proposed overall verification framework; and, finally, section 5 concludes and presents the future work directions.

II. THE VERIFICATION PLATFORM

The verification platform comprises several interconnected blocks, whose operation is controlled by a server, as shown in figure 1. This server implements several functionalities essential to guarantee the good operation of the whole verification framework, namely:

1. To manage external request accesses to the platform;
2. To grant access, under certain restrictions and after checking the client's id, to the verification platform and associated services;
3. To upload the users circuit onto the FPGA-based board;
4. To run the services implemented in the verification framework;
5. To cease access and regain complete control of the platform in case of client's misuses or access timeout.

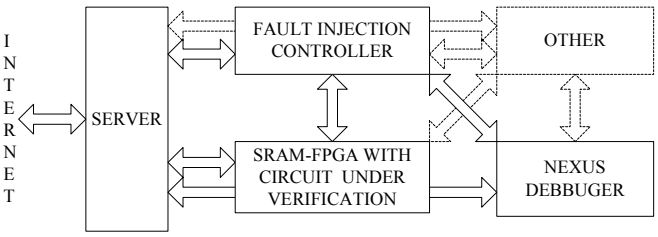

Fig. 1: Verification platform block diagram.

When the connection with the remote client is established, the server generates the interface through which he/she uploads his/her own circuit onto the FPGA and then controls the services available in the verification framework. These services correspond to: A) the ability to inject faults into the FPGA configuration memory; B) using an external debugger to diagnose the effects of injected faults, including possible effects on the OCD itself; and C) using the OCD and the debugger to inject faults on the target registers and memories.

The fault injection controller is responsible for implementing the first mentioned service. At present, it is thought to be a software module, running on a host machine and controlling the FPGA configuration memory through an IEEE1149.1-compatible Test Access Port (TAP). The link between the host machine and the TAP can be implemented either through a simple PC parallel port (less bandwidth) or through a dedicated hardware module (e.g. an USB-JTAG adapter – higher bandwidth). Although physically connected to the FPGA-based board, the fault injection controller could also be used for remotely injecting faults in a board located near to the user. In this case, the board TAP had to be connected to an Ethernet-JTAG adapter and its IP address

indicated to the fault injection controller. The command/data signals could then be re-directed to that adapter while the fault injection controller would be running at the verification platform.

An external debugger connected to the microprocessor (implemented as a soft core inside the FPGA) debug port is capable of monitoring the circuit operation while subject to the fault injection campaigns. This service B is intended for diagnosing the effects of injected faults not only on the functional circuit but also on the OCD itself.

Finally, the same debugger may be used for injecting faults on the functional circuit, an additional service C that can not be run in parallel with service A, because it is important to guarantee the perfect operation of the OCD.

The possibilities and requirements associated to each service are in described in more detail in the following section.

III. THE VERIFICATION SERVICES

A. Fault Injection on the FPGA configuration memory

The fault injection procedure consists on the insertion of bit-flip faults on the configuration memory using the reconfiguration capabilities of the target FPGA. Depending on the purpose of the affected bit the effects on the target system can be classified as follows:

- A modification to a logical function, namely by altering Look-Up Tables (LUTs) contents. This is considered a change to a static element;
- A modification to the routing configuration, adding, removing or altering one or more interconnects. Again, this is considered a change to a static element;
- A modification to memory elements contents, used as such by the target system. This is considered a change to a dynamic element.

An important aspect, at this point, is that circuit registers may be composed of either memory elements or Configurable Logic Block (CLB) flip-flops. Notice that, while it is possible to read the content of a flip-flop, it is not possible to write it through the configuration memory, and therefore it is not possible to inject faults in CLB flip-flops (constraint 1). Additionally, the injection of faults can be performed in real time only if targeting frames configuring just static elements (constraint 2). Providing that the target system is suspended during fault injection, this methodology can be further applied to components that can change during execution (i.e. memories). The suspension is required to allow reading back the contents of relevant elements. The content of the target element should be modified, according to the desired fault model, while the contents of the other elements pertaining to the circuit should remain unchanged.

For better understanding constraint 2, consider for instance the Virtex FPGAs from Xilinx, where the configuration memory is divided into one bit wide vertical frames that span from the top to the bottom of the array. Each column of CLBs comprises multiple frames, which combine internal CLB configuration and state information, with column routing and interconnection information. A representation of the configuration memory frame partitioning is shown in figure 2. The LUTs located in the CLBs can also be configured as memory modules (RAMs) for user applications. However, the extension of this fault injection concept to the injection of faults in LUT/RAMs is not viable due to the time interval between frame read and frame write and also due to configuration memory architecture. The content of the LUT/RAMs could be read and written through the configuration memory, but if there is a write attempt during the interval between the read-modify-write cycle needed to perform the fault injection, there is no possible mechanism, other than to stop the system, capable of ensuring the consistency of the fault injection procedure, as can be inferred from [11]. Furthermore, since frames span an entire column of CLB slices, the same LUT bit in all of them is updated with a single write command. One must thus ensure that all the remaining data in the slice is constant. Even if not being subject to fault injection, LUT/RAMs should not lie in any column that could be affected by the fault injection procedure.

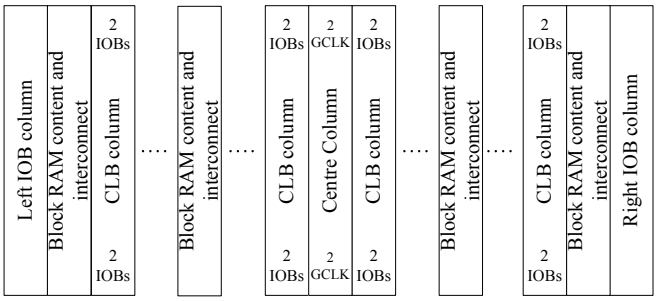

Fig. 2: Configuration memory organization.

The knowledge of the target configuration and the capability of precisely controlling the fault insertion and its position on the target system allows three main approaches, depending on the objective, namely:

- The target system is considered to be the entire FPGA device and the objective is the evaluation of the effects of SEUs on the configuration memory itself;
- The target is a model of an ASIC device, but the fault types are not constrained, the objective being the evaluation of all malfunctions that can be emulated, by modifications to the configuration memory;
- The objectives are limited to the specific effects of SEUs on the target system, with faults into configuration memory adhering to a specific fault model (e.g. bit-flips in memory elements of the target system).

To closely simulate radiation effects (radiation-induced faults are not deterministic), the fault injection controller randomly selects a reconfiguration frame and changes the state of one of its bits. The frame is then written back in the

configuration memory. A log with information about the position of the bit changed during the fault injection is kept by the server. This fault injection log is needed for the client to diagnose the causes of a circuit failure, if it occurs during the fault injection procedure.

B. *Diagnosis of faults effects, including on OCD infrastructures*

While the use of OCD mechanisms for diagnosing fault effects is considered a normal debug scenario, the rules and procedures for the diagnosis of faults affecting the OCD infrastructure are still being developed and documented applications are rare. Although some research on the possibility of faults in test infrastructures has identified a number of problems and proposed some solutions [12], the added functionalities and complexity of modern OCD implementations requires further study.

Early OCD implementations were focused on controllability, assuming that status data could be obtained through other means (i.e. logic analyzers) or stopping the target system. Recent trends in debug and fault injection require real time operation, relying on trace capabilities embedded on the OCD infrastructures.

OCD functionalities that may assist the diagnosis process can be summarized as:

- Register and Memory reading;
- Real Time Trace;
- Breakpoints and Watchpoints.

As an example of recent trends on OCD technologies we may consider the NEXUS proposal for standardization of OCD interface and functionalities. The proposed infrastructure should allow all traditional OCD capabilities, including trace and watchpoint support and also real time access to memory elements, being potentially very useful. In fact, the overall observability provided by the OCD is a measure of its effectiveness for diagnosis purposes.

Testing for OCD infrastructure faults can be executed in two different ways, namely online and offline. This last option is used when the target system is not operating and consists on testing the correct operation of the infrastructure, allowing the detection of permanent errors on ASIC devices and/or configuration memory errors, if using FPGA devices. The online diagnosis of OCD infrastructure errors assumes that the target system is operating and is even more constrained. It can be expected that in operational use the OCD is usually idle, allowing the use of "dead time" for fault detection and/or diagnosis.

When operating offline, it is possible to use specific test patterns or sequences to verify the correct behavior of at least part of the OCD infrastructure. Elements like the communications port, overall control logic and data paths can be reasonably tested. Some limitations exist, and some functions may be impossible to verify. As an example, most OCD registers are accessible only for writing. Testing all possible values, in all operational conditions, would also not be feasible within a reasonable timeframe. The testing of all interconnections may require excessive time, particularly in systems with large memory areas. Reactive logic area may be difficult to test as it would require the microprocessor (or its components) to apply stimulus, and this would require the programming and execution of a considerable number of test algorithms. In short, if relying on the data that is output by the OCD itself, it will be difficult to detect and diagnosis all possible faults.

The techniques that can be used to identify faults affecting an OCD infrastructure and the areas where these can be detected are summarized as follows:

- Reading of OCD registers;
- Writing into OCD registers;
- Writing and reading microprocessor registers or memory;
- Inserting and testing watchpoints or breakpoints.

In the specific case of NEXUS compliant infrastructures we can propose a multi-level procedure for OCD fault detection. Three modes with different intrusiveness levels are considered:

- Non intrusiveness and limited availability of the OCD infrastructure – in this mode it is assumed that debug operations are being executed in parallel with the fault diagnosis and therefore operations requiring OCD resources can only be executed whenever these are not being used for other purposes (i.e. debug). Only non-intrusive operations can be executed;
- Non intrusiveness and full availability of the OCD infrastructure – in this mode all OCD resources are available for fault diagnosis, but still only non intrusive operations can be executed;
- Unrestricted OCD access and use – in this mode fault diagnosis takes precedence over all other operations including the target system execution. All OCD resources can be used, including run control, breakpoints and memory/register access.

Table 1 presents the OCD operations that are available in NEXUS compliant infrastructures, their possible uses for fault diagnosis and the expected intrusiveness. Basic operation refers to the communication channel, message decoding and control structure and are tested in all operations, as an error on one of these elements should prevent successful completion of the required operation. Overhead refers to the required messages via the OCD communications interface and indicates the feasibility of using the indicated operation in parallel with other debug tasks.

Depending on the intrusiveness level considered the execution of fault detection operations can be done in parallel with the target execution and scheduled in order to synchronize with its operation or debug tasks being performed. Different alternatives are possible and further research is necessary in order to reach a complete diagnosis framework.

Table 1 – NEXUS operations for fault diagnosis

Operation	Tests	Intrusiveness
Device Identification	OCD register access (for reading)	Low overhead, no effect on target
Watchpoint Insertion	OCD register access (for writing) Triggering Logic	Low overhead, no effect on target
System Halt/Restart	Run Control	Low overhead, requires target suspension
Breakpoint Insertion	Triggering Logic Run Control	Low overhead, requires target suspension
CPU register read/write	CPU register access Run control	Low overhead, requires target suspension
Memory read/write	CPU memory access	Low overhead, no effect on target
Program Trace	CPU register access OCD register access (for writing)	High overhead, no effect on target
Data Trace	CPU memory access OCD register access (for writing)	High overhead, no effect on target

It should be noted that, although the referred operations are based on the NEXUS architecture, they can be used with most OCD implementations with eventual adjustments or restrictions. Similarly, the use of more advanced OCD versions or architectures may provide additional resources. It is predictable that with present OCD capabilities and access restrictions, most fault diagnosis tasks would be limited to fail/pass testing and would present limited coverage.

C. Fault injection via OCD

As OCD infrastructures provide access to internal resources, in parallel with the target hardware and running software, they are also an excellent mechanism for modifying register and / or memory values, for fault injection purposes. In most cases, OCD fault injection techniques rely on halting the processor, either by the use of control signals or breakpoints, and subsequently accessing the targeted registers or memory locations to insert the intended faults [13].

Furthermore, to address the debug requirements of real time systems recent microprocessor devices are being equipped with enhanced OCD infrastructures that provide on-the-fly run control and memory access. These capabilities are very useful for real time debugging and can also be reused for fault injection.

Recent work using both a regular and a modified NEXUS compliant infrastructure [14, 15] showed that it is possible to use these to assist real time fault injection into memory and also into the microprocessor internal registers, in this case with a minimal performance loss and requiring a temporary target suspension. Different variants were studied, with the basic approach being based only on a regular NEXUS infrastructure (OCD) and a compliant debugger.

These techniques could be used to partially compensate the limitations identified for the injection of faults into dynamic elements present in FPGAs. A combination of both approaches could enhance fault coverage, overall performance and real time operation. Although we have no results to sustain this hypothesis, it is possible to understand that these two approaches are orthogonal, i.e. while service A relies on the physical infrastructure that is part of every FPGA (the ability to reconfigure it through the TAP), the present service C relies on the logic present in the target circuit. Notice that although a debug infrastructure is usually not considered to be part of the functional logic (i.e. it does not implement any circuit function, but rather it is used to debug the circuit function), it is always a part of the circuit, i.e. it is there and it can potentially be used for purposes other than debug. In any case, using OCD mechanisms for injecting faults requires more user intervention, as information will be dependent on the target circuit, e.g. the user must know the address of the register/memory cell where the fault should be inserted to define the debugger commands to use.

IV. THE VERIFICATION FRAMEWORK

The described verification platform and services require a flexible methodology to handle the variable capabilities and limitations of both the fault injection (diagnosis) environment and the target system. As services may be used individually or combined (services A and B can be used simultaneously), there should be a comprehensive interface for the user, allowing him/her to understand: a) how the verification platform can be accessed and the circuit uploaded; b) the sequence of steps associated with each service, and how they can be executed in parallel; c) how to obtain and understand the results provided by each service; and finally, d) how to use a certain service in a remote fashion, without downloading a circuit onto the verification platform, i.e. how to use the service with a circuit implemented in an FPGA-board located near the user (and not near the server where the service is being executed on).

As initially stated, this paper does not describe an implemented framework, but rather presents the ideas around it. The complete sequence of steps associated with each service is not available yet, but still it is possible to present a tentative list, for instance, respecting to service A.

1. Upload the FPGA with the circuit whose robustness the client wants to verify;
2. Initialize the fault injection controller;
3. Apply stimulus to the circuit inputs and read responses from circuit outputs through I/O buses. Internal values can also be obtained by downloading (readback) the FPGA configuration memory;
4. Inject one fault;
5. Evaluate the circuit behaviour in the presence of an injected fault (detected/undetected, catastrophic, etc.)
6. Repeat steps 3 to 5 during an entire fault injection campaign;
7. Download the fault injection log.

The previous list highlights the need for the remote user to interact with the verification platform in order to program it and apply the stimulus / read the responses during the

execution of a fault injection campaign. This could be something simple, for instance in the case where the circuit is a microprocessor running an algorithm that gets all its input information from a memory and outputs the results to a memory, or more difficult, e.g. the situation where the circuit interacts with an external system, physically located near the user. In the latter situation, it would however be possible to devise the solution depicted in figure 3, where the fault injection controller would be running on the remote server, and, through an Ethernet/JTAG converter, interacting with the user circuit implemented in a local FPGA-based board [16].

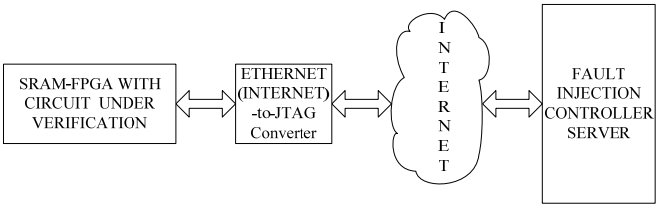

Fig. 3: Running the fault injection controller, in a remote fashion, with a circuit implemented in a local FPGA-based board.

Being possible to run the verification services with the user circuit implemented in a local board (i.e. local from the user's point of view) or in a remote board, many possibilities are open for the definition of a verification methodology. We therefore use the expression verification framework to denote the "basic structure underlying a system or concept", in our case, the verification platform and services. Notice that the possibility to use a certain resource through the Internet is not new, as in fact many physical resources are now shared through the Internet [17].

V. Conclusion and Future Work

This paper presented a framework for making remotely accessible a verification platform and a set of verification services. Only the underlying concepts were discussed, some of them in a very superficial manner. Nevertheless, the services are based on previous work already described in literature [14, 15, 18], the main novelties presented here being: a) the conjunction of different services related to fault injection; b) the ability to execute them in a circuit uploaded by a remote user in a FPGA-based board, connected to a local server; or in alternative c), in a remote fashion, in which case the user will need to have installed on his/her location an Ethernet/JTAG converter.

The future work is described along the text and is now being tackled in a modular approach.

References

[1] C. Maxfield, "Logic that mutates while-u-wait", EDN Magazine, N.º 23, November 1996.

[2] L. Sterpone and M. Violante, "Analysis of the Robustness of the TMR Architecture in SRAM-Based FPGAs", *IEEE Transactions on Nuclear Science*, Vol. 52, No. 5, pp. 1545-1549, Oct. 2005.

[3] M. Ceschia, M. Violante, M. S. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, A. Candelori, "Identification and Classification of Single-Event Upsets in the Configuration Memory of SRAM-Based FPGAs", *IEEE Trans. on Nuclear Science*, Vol. 50, No. 6, pp. 2088-2094, December 2003.

[4] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M. S. Reorda, M. Violante, P. Zambolin, "Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA", *Design, Automation and Test in Europe Conf.*, pp. 584-589, 2004.

[5] F. Lima Kastensmidt, L. Sterpone, L. Carro, M. S. Reorda, "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-Based FPGAs", *Proc.of the Design, Automation and Test in Europe Conf.*, pp. 1290-1295, 2005.

[6] L. Sterpone and M. Violante, "A New Reliability-Oriented Place and Route Algorithm for SRAM-Based FPGAs", *IEEE Trans. on Computers*, Vol. 55, No. 6, pp. 732-744, June 2006.

[7] M. Rebaudengo, M. S. Reorda, M. Violante, "Simulation-based analysis of SEU effects on SRAM-based FPGAs", *Proc. of the 12th Intl. Conf. on Field-Prog. Logic and Applications*, pp. 607-615, 2002.

[8] A. Berger and M. Barr, "Introduction to On-Chip Debug", Embedded Systems Programming, pp. 47-48, March 2003

[9] A. Fidalgo, G. Alves and J. Ferreira, "A Modified Debugging Infrastructure to Assist Real Time Fault Injection Campaigns", *9th IEEE Workshop Design and Diagnostics of Electronic Circuits and Systems*, pp. 174-179, Prague, Czech Republic, April 2006

[10] "The Nexus 5001 Forum Standard for a Global Embedded Processor Interface version 2.0"; IEEE-ISTO 5001, 2003.

[11] W. Huang and E. J. McCluskey, "A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations", *Proc. 9th ACM International Symposium on FPGAs*, pp. 183-192, Feb. 2001.

[12] F. Jong and F. Hayden, "Testing the Integrity of the Boundary Scan Test Infrastructure", *IEEE International Test Conference*, Nashville, USA, October 1991

[13] J. Vinter, O. Hannius, T. Norlander, P. Folkesson, and J. Karlsson; "Experimental dependability evaluation of a fail-bounded jet engine control system for unmanned aerial vehicles", *International Conference on Dependable Systems and Networks*, Yokohama, Japan, June 2005

[14] A. Fidalgo, G. Alves and J. Ferreira, "OCD-FI: On-Chip Debug and Fault Injection", *Int. Conference on Dependable Systems and Networks*, Philadelphia, USA, June 2006

[15] A. Fidalgo, G. Alves and J. Ferreira, "Real Time Fault Injection Using Enhanced OCD – A Performance Analysis", *21st IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 254-264, Arlington, USA, October 2006

[16] Actel, http://www.actel.com/products/solutions/remprog/, accessed September 2007.

[17] Luís Gomes and Javier García-Zubía (eds.), "Advances on remote laboratories and e-learning experiences", University of Deusto press, 312 pp., June 2007, ISBN 978-84-9830-077-2

[18] Manuel G. Gericota, Gustavo R. Alves, Miguel L. Silva, and José M. Ferreira, Chapter 10. "Run-time defragmentation for dynamically reconfigurable hardware", in *New Algorithms, Architectures and Applications for Reconfigurable Computing*, edited by Patrick Lysaght and Wolfgang Rosenstiel, Springer, 313 p., Abril 2005, pp. 117-129, ISBN 1-4020-3127-0