The 21st IEEE International Symposium on

Defect and Fault Tolerance in VLSI Systems

Arlington, VA
October 4–6, 2006

Edited by Nohpill Park, Hideo Ito, Adelio Salsano, and Nur Touba

Sponsored by
The IEEE Computer Society Test Technology Technical Council
The IEEE Computer Society Technical Committee on Fault-Tolerant Computing

IEEE COMPUTER SOCIETY

IEEE

The 21st IEEE International Symposium on

Real Time Fault Injection Using Enhanced OCD – A Performance Analysis

André V. Fidalgo[1,2], Gustavo R. Alves[1], José M. Ferreira[2]
anf@isep.ipp.pt *gca@isep.ipp.p* *jmf@fe.up.pt*
[1]*Instituto Superior de Engenharia do Porto*
[2]*Faculdade de Engenharia da Universidade do Porto*

Abstract

Fault injection is frequently used for the verification and validation of dependable systems. When targeting real time microprocessor based systems the process becomes significantly more complex. This paper proposes two complementary solutions to improve real time fault injection campaign execution, both in terms of performance and capabilities. The methodology is based on the use of the on-chip debug mechanisms present in modern electronic devices. The main objective is the injection of faults in microprocessor memory elements with minimum delay and intrusiveness. Different configurations were implemented and compared in terms of performance gain and logic overhead.

1. Introduction

Most of today's safety-critical applications require some type of computer-based device, causing the implantation of microprocessor systems to widen and expand into new areas. As current trends lead to an increase in complexity and decrease in size of electronic systems, their correct operating behavior is becoming harder to guarantee [1]. Circuits are getting more sensitive to noise and other factors, with the appearance of induced errors becoming a real possibility even for devices used in non-hostile environments, making dependability a necessity for a much broader area of applications.

A significant hazard affecting microprocessor systems is referred as Single Event Upset (SEU) and consists of a change of state induced by an ionizing particle such as a cosmic ray or proton in an electronic device. This event may cause a change on the logical value of memory elements, like registers or memory cells. Depending on the system function and architecture, fault tolerance techniques can be used to deal with these events allowing the system to provide acceptable service in the presence of faults. However, even COTS components, providing no fault tolerance on-chip, may require the evaluation of the consequences of SEUs on the running application. All vulnerable critical systems should be verified to insure that they operate within acceptable limits in the presence of such events and validated to check if they accomplish their intended objectives.

A mean to do this is the injection of faults and the analysis of the system response. This can be used both to check fault tolerance implementations and to estimate fault consequences on non-tolerant systems. When testing the dependability of real time systems a new set of problems must be considered and testing in general and fault injection in particular become generally harder and more critical [2]. This paper proposes a solution to improve real time fault injection on microprocessor systems, based on the use of On Chip Debug (OCD) infrastructures. These are becoming increasingly common on electronic devices, as their use for debug is increasing and recent implementations include real time trace and memory access capabilities.

The rest of the paper is organized as follows: the next section gives a short overview of real-time debug and fault injection, presents an ongoing effort to standardize OCD infrastructures and preliminary work on the injection of faults on compliant devices; section 3 presents some proposals to enhance real-time fault injection on microprocessor systems; section 4 presents the experimental results obtained and finally section 5 discusses these results and indicates future research directions.

2. Real Time Fault Injection

2.1 Overview

Existent microprocessor fault injection techniques are commonly classified in three broad groups, namely, (1) Simulation based fault injection; (2) Software based fault injection (SWIFI) and (3) Physical fault injection. Real time usually designates systems that must provide adequate responses within a specified window of time. In this case, dependability is simultaneously harder to implement and more troublesome to evaluate. Not only must the correctness of the results be checked but also the accurate meeting of the deadlines.

Real time fault injection must be performed with the target system running at full speed, with minimum intrusiveness and delays, which makes it difficult to use most traditional fault injection approaches. The use of OCD infrastructures for fault injection can overcome some of the limitations present on other approaches [3]. However, classic OCD debug mechanisms require halting the processor execution to access its internal resources (i.e. memory, registers), which is undesirable on real time applications. To address this limitation, recent trends in debug are leading to the appearance of enhanced OCD infrastructures that provide real time run-control and memory access and that can be reused for real time fault injection.

2.2 The NEXUS Standard

The OCD infrastructures implemented by different families of processors share some common characteristics that form a core feature set, which usually include 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 chip that requires some external hardware to be used, the basic requirement being some kind of communication link between the chip and the host machine. The access to the OCD infrastructure is made through an interface port usually requiring an external debugger in between. In most cases, OCD fault injection techniques rely on halting the processor, either by the use of control signals or breakpoints, and subsequently modifying the targeted registers or memory locations to insert the intended faults.

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 Standard for a Global Embedded Processor Debug Interface” [4]. The motivation behind this effort is allowing development tool vendors to more easily provide a standard set of tools. To achieve this it is necessary a consistent set of capabilities and a similar communications interface that can be shared across device architectures. Presently the NEXUS standard is still in proposal phase and just a few commercial devices implement it, including Freescale’s MPC56X and MPC555X series. For maximum flexibility, the standard allows for four different levels of compliance, with each level adding extra features. Table 1 presents the debugging features that are required for fault injection purposes along with the compliance class where they are mandatory according to the NEXUS standard.

Table 1 – Debugging Features Used for Fault Injection

Debug Features	Class	Usability for FI
Run-Control	**1**	**External Triggering**
Breakpoints	**1**	**Internal Triggering**
Watchpoints	**1**	**Real Time Triggering**
Static Register and Memory Access	**1**	**Static Fault Insertion**
Program Trace	**2**	**Fault Effects Diagnosis**
Dynamic Register and Memory Access	**3**	**Real Time Fault Insertion**
Data Trace	**3**	**Improved Fault Effects Diagnosis**

Real-time fault injection would be impossible with class 1 compliant devices, as all debugging operations must be executed with the CPU halted. Class 2 compliant devices allow the monitoring of the application execution in real time, which can be used for fault effects diagnosis. The watchpoints can be used for fault triggering, but as the actual fault insertion cannot be performed without stopping the CPU this cannot be used for full OCD based fault injection. Class 3 adds real time access to memory, making it possible to execute real time fault injection on compliant devices.

2.3 Preliminary Work

To evaluate the advantages and limitations of real-time fault injection on NEXUS compliant microprocessors, preliminary work was performed using in-production devices. The approach was similar to other already documented in this area [5] and uses both a commercial microprocessor target and a debugger. The target system used was an evaluation board based on the Freescale's MPC565 CPU, which is a 32 bit microcontroller with widespread use on the automotive industry. The debugger used is an iSystems IC3000 (iTracePro version) and its integrated debugging software Winidea 2005. The fault campaigns were manually generated and translated into Winidea scripts. The fault injection environment used is presented in Figure 1.

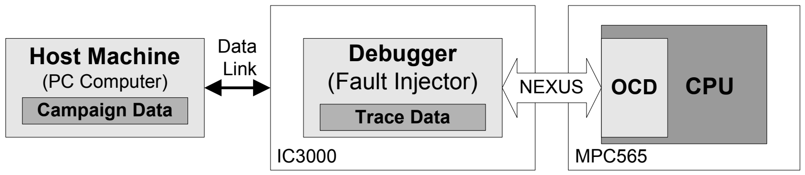

Figure 1 - Fault Injection Environment (MPC565)

In the figure, Campaign Data refers to the scripts describing the fault injection campaigns and Trace Data represents the program trace information output by the OCD and stored on the debugger for posterior analysis and program flow reconstruction. A typical fault injection operation follows the steps indicated in Table 2.

Table 2 – Fault Injection Steps on the MPC565

N	Step	Description
1	Set Up	Microprocessor is reset Target application runs from the start The fault injection script runs on the host
2	Fault Triggering	The triggering condition can be: - A time dependant condition on the host - A specific trace occurrence communicated to the host (via debugger)
3	Fault Activation	On the occurrence of the triggering condition the host instructs the debugger to transmit to the OCD: - A read memory command - The target memory address The debugger retrieves the content of the targeted memory position and sends it to the host. The host modifies the value of the memory contents (to insert the fault)
4	Fault Insertion	The faulty data and the memory address are transmitted to the debugger The debugger accesses the OCD to insert the faulty data into the targeted memory position

The obtained results confirmed most of the expected potentialities and simultaneously identified some shortcomings both in fault triggering and performance. It proved possible to insert faults in memory space without affecting the running application and use the trace information gathered as an effective mean to analyze program flow, before and after the actual fault activation. However, as the NEXUS compliant debugger communicates with the host machine through Ethernet or USB connections, and as the fault campaigns must be run on the host machine, this imposes a bottleneck on the time required for an actual memory access. This fact causes the time interval required for reading a memory cell contents and writing back a modified value to be measured in milliseconds. This delay allows the initial data to be overwritten by the application running on the target system, the magnitude of the problem depending of the running application and memory position targeted.

The consequence for the fault injection process is the occurrence of experiments with inconclusive results, derived from the fact that the fault actually inserted doesn't emulate a single bit-flip as intended, but a variable number on the same memory cell. An additional problem is the triggering of a fault, which is inadequate for most applications and makes it unavoidable to have to read the target cell contents prior to fault insertion. Both the described problems are not directly related with the OCD capabilities but rather with the available tools, which lack some features that, not being necessary for debug, would be very useful for fault injection.

3. Experimental Scenarios

3.1 Methodology

From the limitations identified on the preliminary work two scenarios were studied for improving the fault injection capabilities and performance. The first scenario is aimed at minimum intrusiveness and consists on the use of a debugger customized for fault injection. The second scenario is aimed at maximum performance and adds dedicated fault injection capabilities to the OCD infrastructure itself. The objectives intended for both these scenarios were:

- Precise control of the fault location and injection instant.
- Minimum intrusiveness on the target application (hardware or software).
- Minimum number of errors on the fault injection process.
- Maximum performance in terms of fault injection delay and faults per second rate.

The methodology consists of executing several fault injection campaigns on both scenarios. Each campaign is described by a single script and composed of a series of experiments during which the target system runs (a specific application is executed) and a specific fault is inserted at specific trigger conditions. The target system behavior is monitored and information is recorded as comprehensively as necessary (and possible), to later understand and evaluate the effects of the inserted fault(s). The fault model consists of bit-flip faults at specific moments in program execution and the target for each campaign can be either a memory cell or an internal register.

In order to achieve better performance it is desirable to determine beforehand the value that will be present on the target memory cell at the fault insertion moment. This can only be performed when:

- There is complete knowledge of the program flow up to the fault injection instant.
- No external inputs are used (or if used it is possible to observe or control their values).
- The fault injection instant and location can be precisely controlled.

If this pre-determination cannot be done, it is necessary to read the target memory cell immediately before fault injection to determine the faulty value to be inserted to emulate a SEU.

3.2 Customized Debugger

The intended improvements to the fault injection process require modifications to both the target OCD and the debugger so these had to be available as a HDL models. The CPU cores were generated using the cpugenerator building tool [6], which generates customizable RISC cores, allowing direct configuration of all bus sizes, interrupt handling, indirect addressing, data/instruction latency timings and custom instructions definition. Fault tolerance capabilities were included on the target system in the form of software fault tolerance, as this is what would be present on most COTS systems. The fault tolerance was implemented by duplicating all arithmetic operations and performing each on different memory positions. The results are compared afterwards before storing into memory and any difference causes the application execution to be halted with the appropriate error code.

The OCD infrastructure was designed from scratch to be compliant with the NEXUS 5001 standard proposal. The version actually implemented on our target system is based on the one present on the MPC565 and is also NEXUS Class-2 compliant with added real time memory access capability (sometimes referred as Class-2+). It is configurable to adapt itself to the target system, being compatible with different CPU configurations, requiring only minor adjustments. The two configurations implemented use the 32 bit CPU version varying only in terms of OCD implementation, namely on the MDI (Message Data In) bus width. The MDI and MDO (Message Data Out) are part of the AUX port and responsible for transmitting data between debugger to the OCD. This is presented in Table 3 along with the data for the MPC565 for comparison.

Table 3– CPU Clock and OCD Configurations

OCD Configuration	CLK (MHz)	MDI (bits)	MDO (bits)
OCD32 (MDI2)	30	2 bits	8
OCD32 (MDI8)	30	8 bits	8
MPC565 (READI)	40	2 bits	8

Larger MDI buses imply faster writing into memory (and registers) and therefore faster fault injection and configuration. Conversely they will also require more input pins and higher logic overhead. The MDO bus is responsible for the transmission of data from the OCD and affects mainly the trace capabilities, not having a big effect on the fault injection process.

The target applications used were fault tolerant versions of a matrix adder and a vector sorter. These were selected because they are memory intensive algorithms, simple to debug and can be easily adapted to different targets and memory sizes. The customized debugger consists of one controller module and two memory banks for data input and output. As the debugger main purpose is fault injection campaign management, it has reduced support for direct control, the emphasis being on executing scripted commands and reacting automatically to messages or signals form the OCD. This last possibility is an important feature that is lacking in most debuggers, as it is not required for common debug operations. The fault injection environment is presented in Figure 2.

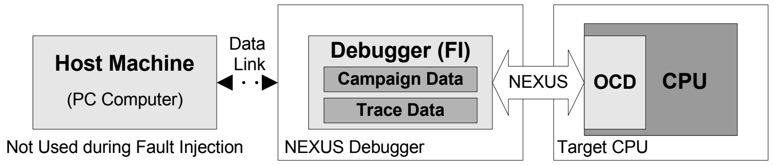

Figure 2 – Fault Injection Environment (Customized Debugger)

The host machine is used to upload scripts (fault campaigns) to the debugger memory and later download the trace data, taking no part in the fault injection process itself. At this phase the debugger supports only the NEXUS messages actually implemented on our OCD, including also debugger specific commands that allow the insertion of delays, wait states and reaction to messages from the OCD. The steps executed for a fault injection operation are presented in Table 4.

Table 4 - Fault Injection Steps Using a Customized Debugger

N	Step	Description
1	Set Up	Microprocessor is reset Target application runs from the start A fault injection script is downloaded to the debugger A watchpoint is set on the target (on the OCD)
2	Fault Triggering	The triggering condition can be: - An external signal received by the debugger - A watchpoint hit signaled by the OCD (to the debugger)
3A	Fault Activation (Predet.)	On the occurrence of the triggering condition the debugger activates a memory write operation using preset values.
3B	Fault Activation (no Predet.)	On the occurrence of the triggering condition the debugger uses a preset target memory cell address to retrieve from the target (via the OCD) the target memory cell contents The debugger applies a data mask to determine the faulty data value to be written into memory
4	Fault Insertion	The debugger transmits to the OCD: - The target memory cell address - The data value to be written

The option between steps 3A and 3B depends on the possibility of predetermining the target memory cell contents prior to the fault insertion. The injection of faults into internal registers requires the replacement of the watchpoint by a breakpoint and an additional step to resume microprocessor operation. The messages output from the OCD are also stored on a separate memory that can latter be used by an external tool for program flow analysis and fault effects diagnosis. The size of the input memory bank defines the number of fault operations that can be executed on a single script (campaign), and the size of the output memory defines the amount of trace data that can be stored. The data provided by the trace plus eventual error messages plus the knowledge of the running application makes it possible to reconstruct the exact program flow.
Preliminary results obtained from the execution of fault campaigns using the customized debugger showed that in some instances the value written into memory still didn't conform to the bit-flip model that was used. Detailed analysis of the relevant operations showed that the cause was that the running application wrote over the target memory position before the fault injection was completed.

3.3 Modified OCD

Addressing the issue of improving the performance of the fault injection process can be accomplished by transferring some functionality to the inside of the OCD. The resultant modified infrastructure with added fault injection support is designated as OCD-FI and the basic version is described in [7]. The fault injection environment is similar to the one used for the customized debugger, with the debugger and the target CPU core being identical. The main difference is the inclusion of an extra hardware module (FI) into the OCD circuitry as presented in Figure 3.

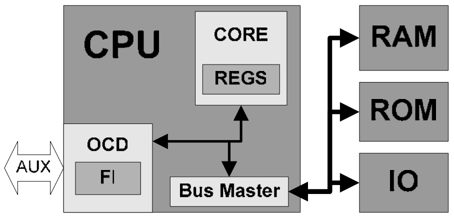

Figure 3 – OCD and FI Module

The FI module reuses some capabilities of the OCD to automatically activate a fault insertion on the occurrence of a watchpoint hit. Once enabled the FI module takes control of the OCD until the actual fault insertion. During the entire process the trace data generation is not affected continuing to operate as if a "real" fault occurred. This module was designed to be adaptable OCD infrastructures in general and NEXUS compliant devices in particular. It requires some OCD features, namely: (1) Watchpoint support; (2) Real-time memory access and (3) Memory read / write preloading. If these features are not available, the implementation of the FI module may entail substantial modifications to the OCD infrastructure.

The Basic FI module implementation incurs on minimum logic overhead but requires the target memory value to be determined beforehand and both the address and faulty data to be preloaded into the OCD-FI infrastructure. To eliminate this limitation an enhanced version (designated as Plus) can be used, adding the capability of reading the target memory contents and modifying its value within the OCD-FI. Both implementations are presented on Figure 4.

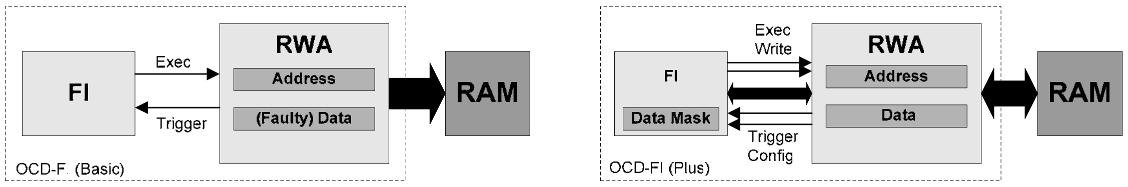

Figure 4 – OCD-FI Configurations

On its Plus version the FI module stores an internal data mask and requires more control and data signals between itself and the other OCD components. It also requires the target memory address and a fault data mask to be pre-loaded and on the triggering of the fault the target memory cell value is read and modified (based on the data stored in the mask). Subsequently the faulty value is written back to memory. The sequence of steps to inject a fault is described in Table 5.

Table 5 - Fault Injection Steps Using the OCD-FI

N	Step	Description
1	Set Up	Microprocessor is reset Target application runs from the start A fault injection script is downloaded to the debugger A watchpoint is set on the target (on the OCD) The OCD-FI fault injection mode is enabled and preloaded with the required data
2	Fault Triggering	The triggering condition is: A watchpoint hit signaled internally by the OCD-FI
3A	Fault Activation (Predet.)	On the occurrence of the triggering condition the OCD-FI activates a memory write operation.
3B	Fault Activation (no Predet.)	On the occurrence of the triggering condition the OCD-FI retrieves the target memory contents. The OCD-FI determines the faulty memory value by applying a fault mask (internally)
4	Fault Insertion	The OCD-FI inserts the faulty data into the targeted memory position

The selection of steps 3A or 3B depends on the configuration (or version) of the OCD-FI being used. The insertion of faults into internal registers requires replacing the watchpoint for a breakpoint and the FI module automatically instructs the resuming of normal operation after fault insertion.

4. Experimental Results

All components were implemented as VHDL modules and synthesized using Xilinx ISE 7.1i. All simulations were run on post synthesis models using Modelsim 6. Synthesis was executed identically for all components using balanced settings. The experimental scenarios include two different MDI configurations and two FI module implementations and are summarized in Table 6. The number following the MDI acronym is the MDI bus width, the FI acronym indicates the presence of a FI module on the OCD and the plus sign indicates the ability to inject faults without predetermination of the faulty value to insert. On a scenario including only the customized debugger this capability depends solely on the used script. On the scenarios where the OCD-FI is used it depends on the implemented version of the FI module. It should be noted that although the FI module Plus version can use the predetermination of the faulty value this wasn't used for the tested scenarios.

Table 6 – OCD and FI Module Configurations

Scenario	Designation	OCD Version	FI Module	Scenario	Designation	OCD Version	FI Module
OCD only	**MDI2**	MDI2	None	**OCD-FI**	**MDI2_FI**	MDI2	Basic
	MDI2+	MDI2	None		**MDI2_FI+**	MDI2	Plus
	MDI8	MDI8	None		**MDI8_FI**	MDI8	Basic
	MDI8+	MDI8	None		**MDI8_FI+**	MDI8	Plus

Due to the debugger memory constraints, each fault campaign consists of 10 experiments, each injecting a single bit-flip fault, emulating a single SEU event. Several campaigns were executed on each scenario and an analysis of the obtained results allows the following conclusions:

- Faults can be injected with absolute precision in terms of target location.
- The instant of injection is dependant on the delay between the meeting of the trigger condition and the actual fault insertion. As this delay is constant for each configuration, and can be precisely determined, it is possible to insert faults in precise instants in time.
- In some cases the CPU overwrites the target memory cell before the fault injection can be completed. This leads to an erroneous fault injection and these experiments must be discarded (as an inconclusive result) from the dependability evaluation process.

- The previous effect is dependant on the delay between fault triggering and fault insertion, and as such it varies between scenarios and configurations.
- The results for the same fault injection campaigns are identical between scenarios, apart from the inconclusive result cases.
- It is possible to use the trace information generated by the OCD to reconstitute program flow.
- The OCD can be used for posterior analysis of the algorithms final (or intermediate) results allowing these tasks to be automated into the fault campaign scripts.

Table 7 presents the durations of the set up and fault injection processes, for each configuration.

Table 7 – Set Up and Fault Injection Delays (CLK cycles)

Configuration	Set Up	Fault Injection	Configuration	Set Up	Fault Injection
MDI2	22	35	**MDI2_FI**	57	2
MDI2+	22	44	**MDI2_FI+**	57	4
MDI8	6	9	**MDI8_FI**	15	2
MDI8+	6	18	**MDI8_FI+**	15	4

Set up represents the time required to prepare the fault injection process so that the trigger condition is set and can be acted upon. This can be performed before the target application starts but if extra performance is required it can be executed with the target system running, being only necessary to insure that set up can be concluded prior to the fault trigger condition. Fault injection represents the time between triggering and the actual insertion of the faulty value into memory. Table 8 presents the occurrence percentage of inconclusive results by configuration and target application.

Table 8 – Inconclusive Results Percentages

Configuration	Inconclusive Results		Configuration	Inconclusive Results	
	MatrixAddFT	VectorSortFT		MatrixAddFT	VectorSortFT
MDI2	4 %	5 %	**MDI2_FI**	0 %	0 %
MDI2+	5 %	7 %	**MDI2_FI+**	1 %	1 %
MDI8	2 %	3 %	**MDI8_FI**	0 %	0 %
MDI8+	3 %	4 %	**MDI8_FI+**	1 %	1 %

Although current OCD implementations don't allow injecting faults on registers in real time with the microprocessor running, it is possible to minimize the time interval during which the microprocessor is halted. Table 9 presents the time interval during which the microprocessor must be halted for the injection of a fault on an internal register.

Table 9 – Halt Delay for Fault Injection on Registers (CLK cycles)

Configuration	Halt Delay	Configuration	Halt Delay
MDI2	41	**MDI2_FI**	3
MDI2+	60	**MDI2_FI+**	5
MDI8	12	**MDI8_FI**	3
MDI8+	20	**MDI8_FI+**	5

Set up is still required and takes the same time as for a real time fault injection in memory. Table 10 present the equivalent logic gate count of the different components. The first percentage column represents the relative weights of the CPU core, OCD infrastructure and FI module relative to the complete target system and the second represents the FI module weight percentage relative to the complete OCD.

Table 10 – FI Module (Basic) Overhead

Config	MDI2_FI	%		MDI8_FI	%		MDI2_FI+	%		MDI8_FI+	%	
CPU	53,926	75,3	■	53,926	70,8	■	53,926	75,1	■	53,926	70,6	■
OCD	17,601	24,6	99,6	22,201	29,1	99,7	17,601	24,5	98,3	22,201	29	98,6
FI	75	0,1	0,4	75	0,1	0,3	309	1,7	1,7	309	0,4	1,4

Both the FI module implementations are relatively small in terms of logic gates. Their overhead is very small even when only compared with the OCD infrastructure.

5. Conclusions and Future Work

OCD infrastructures present a non intrusive mean to access the microprocessor resources providing mechanisms for the triggering and insertion of faults and subsequent analysis of their effects. When dealing with real time systems, performance becomes fundamental and the used tools require enhanced capabilities. From the available results it is possible to assess that the use of OCD infrastructures with the required capabilities allows efficient real time fault injection on memory space. The solutions proposed on this paper allow precise control over the fault target both in time and space. The reuse of already implemented debug mechanisms on-chip makes the proposed solution efficient both in terms of performance and required resources. In general, execution is fast, minimizing the probability of inconclusive experiments and allowing good fault per second rates for cases where the mass injection of faults is required. Intrusiveness is minimal as neither the target microprocessor nor the running program are modified, the choice of modifying the OCD being given as an option. As an extra advantage, this solution allows the entire fault injection scenario, including manager, fault injector and target system, to be implemented on a single FPGA device. Although other solutions may provide better performance they usually require special resources or large logic overheads. The choice between the different configurations depends on the target hardware and software. Larger bandwidth for debug messaging can considerably improve fault injection performance, and the inclusion of on-chip fault injection capabilities can further improve reaction time. As in debug, the answer to a specific problem will be some type of compromise between required capabilities and acceptable overhead.

Some limitations are still present. Real time fault injection is only possible on those resources accessible by the OCD, mainly memory, but this is also where SEUs are more probable. The inexistence of an accepted standard may impose a considerable tuning effort to adapt the debugger and FI module to each particular case. The expected evolution towards OCD standardization would allow generalizing the solution. Future research will focus the broadening of the proposed solution to different architectures and addressing issues like real time access to internal registers and support for hardware fault tolerance.

6. References

[1] M. Rebaudengo, M.S. Reorda, M. Violante, B. Nicolescu, Velazco; "Coping with SEUs/SETs in microprocessors by means of low-cost solutions: A comparison study"; IEEE Transactions on Nuclear Science, Vol49, No3; June 2002

[2] B. Rahbaran, A. Steininger, T. Handl; "Built-In Fault Injection in Hardware – The FIDYCO Example"; 2nd IEEE International Workshop on Electronic Design, Test and Applications (DELTA); Perth, Australia; January 2004

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

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

[5] P. Yuste, D. de Andrés, L. Lemus, J.J. Serrano, P.J. Gil, "INERTE: Integrated NExus-Based Real-Time Fault Injection Tool for Embedded Systems"; The International Conference on Dependable Systems and Networks; San Francisco, USA; June 2003.

[6] G. Ferrante; "CPUGEN 2.00" (www.opencores.org), 2003

[7] A. Fidalgo, G. Alves, J. Ferreira; "A Modified Debugging Infrastructure to Assist Real Time Fault Injection Campaigns"; 9th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS); Prague, Czech Republic; Mar 2006