Real-time fault injection using enhanced on-chip debug infrastructures

André V. Fidalgo, Manuel G. Gericota, Gustavo R. Alves, José M. Ferreira

ABSTRACT

The rapid increase in the use of microprocessor-based systems in critical areas, where failures imply risks to human lives, to the environment or to expensive equipment, significantly increased the need for dependable systems, able to detect, tolerate and eventually correct faults. The verification and validation of such systems is frequently performed via fault injection, using various forms and techniques. However, as electronic devices get smaller and more complex, controllability and observability issues, and some- times real time constraints, make it harder to apply most conventional fault injection techniques. This paper proposes a fault injection environment and a scalable methodology to assist the execution of real-time fault injection campaigns, providing enhanced performance and capabilities. Our proposed solutions are based on the use of common and customized on-chip debug (OCD) mechanisms, present in many modern electronic devices, with the main objective of enabling the insertion of faults in micro- processor memory elements with minimum delay and intrusiveness. Different configurations were implemented starting from basic Components Off-The-Shelf (COTS) microprocessors, equipped with real-time OCD infrastructures, to improved solutions based on modified interfaces, and dedicated OCD circuitry that enhance fault injection capabilities and performance. All methodologies and configurations were evaluated and compared concerning performance gain and silicon overhead.

Keywords

Dependability, Fault injection, On-chip debug, Real-time systems, Microprocessors

1. Introduction

Most of today's safety–critical applications require some type of computer-based device, broadening the application range of microprocessor systems. As electronic systems increase in complexity and decrease in size, their correct behavior is becoming harder to guarantee [1]. The higher sensitiveness to noise and other factors increases the probability of errors, even for devices used in non-hostile environments. The most frequent hazard affecting microprocessor systems is usually referred as a Single Event Upset (SEU) and consists of a change of state of a flip-flop, induced by an ionizing particle such as a cosmic ray or proton. This event may change the logical value of memory elements, such as registers or memory cells [2].

The verification and validation of dependable systems requires the study of failures and errors in order to evaluate their probability of occurrence and subsequent effects. The possibly destructive nature of a failure and the long error latencies make it difficult to identify their causes in the operational environment, and recommend the organization of experiments under precisely controlled conditions. Depending on the system function and architecture, hardware [3] and software [4] fault tolerance techniques can be used to minimize the effects of SEUs, enabling the system to provide acceptable service in their presence. All vulnerable critical systems should be verified to ensure operation within acceptable limits in the presence of such events, and validated to check if they accomplish their intended objectives. Fault injection can be used both to evaluate fault tolerance implementations and to estimate fault consequences on non-tolerant systems.

When dealing with microprocessors, the main limitations imposed on fault injection are control, internal access, intrusiveness and performance. Ideally a fault injection methodology should allow precise control of fault insertion, both in time and space, complete replicability of experiments, and access to all microprocessor resources. Simultaneously it should require no modifications to the target software or hardware, and should execute in real time. As this is not technically feasible, all fault injection environments are based on acceptable (or possible) trade-offs. Access to the area where faults are to be inserted is a major problem, often requiring either ad hoc [5], intrusive [6], or low-controllability [7] approaches. The first and second solutions require special hardware or modifications to running software, offer restricted coverage, and may be difficult to execute in real-time. The third solution is usually based on contactless fault injection techniques, making fault synchronization and replication hard or impossible to guarantee. OCD infrastructures have been used as an efficient alternative

to handle such problems [8] and the addition of circuitry to evaluate the vulnerability to SEU effects is increasingly accepted at the design stage [9].

This paper proposes a set of fault injection solutions enabled by debug features that are now present in recent microprocessor devices. The proposed fault injection environment was designed to be non-intrusive and to allow real time emulation of SEU effects in the microprocessor memory. Real time operation requirements may indeed justify the use of modified OCD infrastructures in order to provide better fault injection capabilities and/or performance. The rationale behind the proposed solutions is that microprocessor systems dependability would benefit from enhancements aimed at improving fault injection operations, making them viable from both economical and technical viewpoints. The modified OCDs proposed in this paper are based on the use of wider data link with an external debugger, or on the use of a dedicated fault injection module, with low overhead and higher autonomy. More intrusive fault modules were also considered as a way to increase fault coverage on safety–critical devices, enabling the insertion of precisely controlled faults on internal registers or protected memory.

The next section summarizes the state of the art and preliminary research. Section 3 presents our proposed solutions, including the experimental environment and application methodology. Section 4 presents the experimental results obtained during the course of this work. Finally, Section 5 presents the main conclusions, and suggests directions for future research.

2. State of the art

2.1. Real-time fault injection in microprocessors

Real time usually designates systems that must provide adequate response within a specified time window. In this case, dependability is harder to implement and more troublesome to evaluate. The correctness of the results must be checked and accurate meeting of deadlines is mandatory, without modifying or stopping the target system.

Real-time fault injection must be executed with the target system running at full speed, with minimum intrusiveness and delays. Most traditional fault injection approaches cannot be adequately used under these constraints. Simulation based fault injection can be useful on early stages of development, but it is often time-consuming and intrinsically dependent on the quality of the available model [10,11]. Additionally, it is very difficult to implement a model that accurately represents all the delays and other timing aspects, and a different technique must be used once a prototype (or production model) is available. Software fault injection adds fault insertion routines, causing extra delays and limiting the fault targets to those areas accessible by the application code. Although work on this area has shown that it can be used for some real-time systems [12], it presents considerable limitations in terms of intrusiveness and coverage. The need to slow down or stop the running application also makes it inconvenient to apply most contact fault injection techniques, since they degrade system performance. Most technical solutions to this problem rely on contactless fault injection [7] or on special dedicated infrastructures [13], both of which are complex and expensive. Contactless techniques present controllability and replicability problems, concerning precise control of the instant and location of a fault. Dedicated fault injection infrastructures come together with silicon overhead and often require special prototype versions of the target system, hardly or even not adaptable to the final product. Additionally, access to internal blocks where faults are more probable, generally the memory elements and communication buses, is also problematic, particularly without disturbing the running applications.

Recent approaches to real-time fault injection include improved software techniques [14], halting the target with minimal delay for near real-time fault injection [15] or taking advantage of recent FPGA capabilities [16,17]. As many of today's microprocessors incorporate dedicated OCD circuitry, designed to operate independently of the target system resources, their use for fault injection purposes is becoming increasingly popular.

2.2. Fault injection via OCD

The OCD implementations present in different families of microprocessors share common characteristics that form a core feature set, usually including run control, breakpoint support, and memory and register access. Some devices offer more advanced features such as watchpoints, program trace and real time debug capabilities. In general terms, an OCD is a combination of hardware and software embedded onto the microprocessor chip, accessible through an interface port, and usually requiring an external debugger.

OCD infrastructures provide access to internal resources during system operation, being an excellent mechanism for modifying register and/or memory values, i.e. for *inserting faults*, and subsequently retrieving the data necessary to assess the effect of those faults. In most cases, OCD fault injection techniques rely on halting the processor, via control signals or breakpoints [18].

The major problem of on-chip debugging is the lack of a consistent set of capabilities and a standard communication interface across processor architectures. Standard ports (RS232, JTAG) are commonly used for the physical connection [19,20], but their capabilities vary widely. Several standardization efforts for OCD infrastructures and interfaces were initiated on recent years [21–23]. IEEE-ISTO 5001, *The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface* [24], was the first of these efforts and is currently well documented and stable.

To better evaluate the advantages and limitations of real-time fault injection on NEXUS compliant microprocessors, preliminary work was performed using COTS devices. This approach was similar to other research works [8,25], and used a commercial target microprocessor and a debugger.

The obtained results confirmed most of the expected benefits and simultaneously identified some shortcomings, both in fault triggering and performance. It proved that it is possible to insert faults in memory without affecting the running application and to use the trace information as an effective means of analyzing the program flow, before and after fault activation. However, as the fault injection campaigns must be run on the host machine, the operating system (Windows or Unix) and physical connection to the NEXUS compliant debugger (Ethernet or USB) lead to long and non-deterministic memory access times. The consequence is the occurrence of experiments with inconclusive results, since in such cases the fault actually inserted does not emulate a single bit-flip as intended. Depending on the targeted memory area, the actual percentage of inconclusive fault insertions could be as high as 50%, requiring additional debugging and result analysis for validating each experiment.

The triggering source represents an additional source of problems. The use of trace data proved unreliable due to variable communication delays, making it necessary to use an external trigger signal. As a consequence, it was impossible to synchronize the fault insertion and the events of the running application.

To overcome the identified problems, three solutions were developed to enhance real-time fault injection capabilities: (1) a debugger customized for fault injection, (2) higher bandwidth between the debugger and the OCD, and (3) the migration of some capabilities into the OCD infrastructure itself.

3. Proposed solutions

3.1. Target system

The CPU cores used on the target system were created using the *cpugenerator* building tool [26], which produces a customizable VHDL model of generic RISC cores, allowing configuration of all buses, interrupt handling, indirect addressing, data and instruction latency timings, and definition of custom instructions. Three applications were used as workload: (1) a matrix adder (*MAdder*), (2) a vector sorter (*VSorter*) and (3) a generic LUT-based control algorithm (*XControl*). All algorithms are memory intensive, can be adapted to different bus sizes and memory areas, and are relatively simple to debug. Only *XControl* requires external stimuli generation and I/O capabilities on the target. Each application was developed in two versions: normal and fault tolerant. Fault tolerance was implemented by duplicating data in memory and performing each arithmetic operation twice. The comparison of the results obtained from each arithmetic operation provides a limited degree of fault detection, with some overhead in execution time and memory requirements. This approach was selected as it can be easily implemented in most COTS components.

The OCD infrastructure developed for our case study was designed from scratch to be NEXUS compliant. As there is no mandatory implementation, we based it on the infrastructure present on the MPC565 microcontroller, which is a well-documented device. The version implemented on our target system is NEXUS Class 2 compliant with real-time memory access capability (sometimes designated as Class 2+ compliant). The OCD interface uses an AUX port, which provides two message data buses (MDI and MDO) for OCD data input and output, along with independent clock and control signals (MCKO, MCKI, MSEI and MSEO). The OCD infrastructure is divided in three main modules and two bus access modules as seen on Fig. 1. The thinner arrows represent the control and status signals and the thicker arrows represent data and trace information. The FI module represented is not included in the original OCD – it is part of the OCD-FI version explained ahead in Section 3.2.3.

The Bus Snooper and Bus Master modules are responsible for interfacing with the microprocessor buses. Their implementation depends on bus configuration and collision management strategies, and should be customized according to the selected configuration architecture.

The Message Queuing and Management (MQM) module implements the NEXUS message handler and the OCD controller. It translates all debugging operations into messages and vice versa, manages the message queues and provides the necessary control signals to the other modules.

The Read and Write Access (RWA) module is used to access both OCD registers and CPU resources (memory and registers), and access inputs and outputs as directed mapped addresses, as the microprocessor does.

The Run Control and Trace (RCT) module is responsible for CPU run control and OCD management. It receives commands from the MQM and RWA modules and outputs trace data and watchpoint hit signals.

The complete OCD infrastructure provides a common set of debugging features and interface options that can be adapted to different target systems, and upgraded to support additional features or functional blocks.

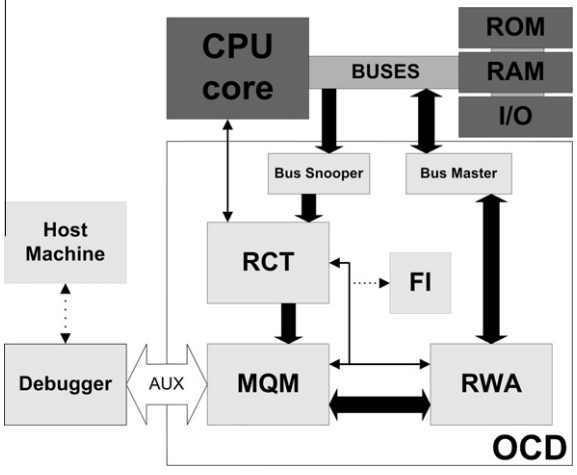

Fig. 1. The OCD infrastructure.

3.2. Fault injection

3.2.1. Environment

Our proposed fault injection solutions were designed to achieve the following objectives:

- Precise control over the fault location and injection instant
- Full observability of fault effects.
- The possibility of replicating experiments.
- Unintrusive to the target application.
- Real time operation (i.e. without stopping the target application).

The experimental environment was designed to implement and evaluate the various fault injection alternatives, maintaining a common architecture and reusing most components with minimum modifications, as presented in Fig. 2.

The Input/Output (I/O) module is required only by applications using external inputs or outputs (e.g. *XControl*) and the FI module is implemented only on the OCD-FI configuration. All environment variants use the same debugger and 32-bit CPU target, differing only in terms of OCD configuration, namely on the MDI bus bandwidth and on the presence/absence of a FI module.

The fault model consists of bit-flip faults, which are inserted at specific moments during program execution, in order to emulate the SEU effects. Faults can be injected in all resources accessible by the OCD, including memory, internal registers and IO registers. Better performance can be achieved by determining beforehand the value that will be present on the target memory cell at the fault insertion instant (herein referred as *predetermination*), but this requires:

- Complete knowledge of the program flow up to the fault injection instant.
- Full observability of external inputs.
- Precise control of the fault injection instant and location.

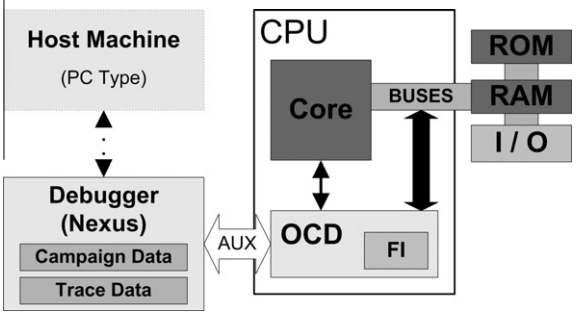

Fig. 2. Fault injection environment.

If predetermination cannot be guaranteed, it is necessary to read the target memory cell data immediately before the fault injection instant, in order to determine which faulty value shall be inserted to emulate an SEU. Each scenario offers various alternatives for this purpose, depending on relevant performance requirements.

Table 1 presents the experimental scenarios that were used during our fault injection experiments. The name of each scenario indicates the specific options selected, e.g. (B) basic OCD configuration, (E) extended OCD configuration, (OF) offline fault injection, (RT) real-time fault injection, (+) no faulty value predetermination required and (FI) fault injection module present.

The FI Method column specifies if faults are injected with the microprocessor halted (offline) or operating at full speed (real-time). The Set-Up delay column indicates the time required for downloading to the OCD all data necessary to each fault experiment. *Set-Up* can be performed while the target application is running, but it must be concluded prior to the occurrence of the triggering condition. The (fault) *insertion* delay column indicates the time interval between the occurrence of the triggering condi-

Table 1
Fault injection scenarios.

Scenario		Bandwidth	Predetermination of the faulty value	Fault injection method	Delays (CLK cycles)	
					Set-Up	Insertion
1	BOF	MDI = 2, MDO = 8	YES	Offline	22	35
2	BOF+		NO		22	44
3	EOF	MDI = 8, MDO = 8	YES		6	9
4	EOF+		NO		6	18
5	BRT	MDI = 2, MDO = 8	YES	Real time	22	35
6	BRT+		NO		22	44
7	ERT	MDI = 8, MDO = 8	YES		6	9
8	ERT+		NO		6	18
9	OCD-FI	MDI = 2, MDO = 8	YES		57	2
10	OCD-FI+		NO		57	4

Table 2
Offline fault injection (BOF and EOF).

#	Step	Description
1	Set-Up	The microprocessor is reset and the target application runs from the start A fault injection script is downloaded into the debugger A breakpoint is set on the target (on the OCD)
2	Fault triggering	The triggering condition is one of the following: – an external halt signal received by the debugger – a breakpoint hit signaled by the OCD (to the debugger)
3	Fault activation (predetermination)	Upon the occurrence of the triggering condition the debugger activates a memory write operation using preset values
3+	Fault activation (no predetermination)	Upon the occurrence of the triggering condition the debugger uses a preset target memory cell address to retrieve its contents (via the OCD) 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
5	Resume	The debugger instructs the target microprocessor to resume execution (via OCD)

Table 3
Real time fault injection (BRT and ERT).

#	Step	Description
1	Set-Up	The microprocessor is reset and the 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 is one of the following: – an external signal received by the debugger – a watchpoint hit signaled by the OCD (to the debugger)
3	Fault activation (predetermination)	Upon the occurrence of the triggering condition the debugger activates a memory write operation using preset values
3+	Fault activation (no predetermination)	Upon the occurrence of the triggering condition the debugger uses a preset target memory cell address to retrieve its contents (via the OCD) 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

tion and the actual insertion of the faulty value (see Tables 2 and 3 for further details).

The proposed solutions were designed to handle real-time fault injection in memory elements, as this is mandatory to reach high values of fault coverage, with maximum compatibility and minimum intrusiveness. However, it is possible to further enhance the OCD-FI infrastructure to support architecture-specific issues and provisions for extending the basic design were considered. Two such extensions were developed for situations where dependability requirements would demand higher coverage, even if degrading performance. Specifically, the OCD-FI (RTREG) extension adds real time access to internal registers, and the OCD-FI (EDAC) extension enables fault injection on memories protected by hardware fault tolerance mechanisms. The two OCD-FI scenarios specifically adapted to evaluate these extensions are presented in Section 3.2.4.

3.2.2. Customized debugger

The customized debugger consists of one controller core and two memory banks for data input and output, as represented in Fig. 3. It provides full support for the execution of scripted commands *and automatically reacts to messages or signals from the OCD*. This is an important feature lacking in most debuggers, as it is not required for common debug operations.

The host machine uploads scripts (fault injection campaigns) to the debugger input memory and later downloads the trace data, taking no part in the fault injection process itself. Direct control is possible through specific signals, which may replace the input or output memories (or both), as source of commands and destination of data.

The output memory can be used to store not only trace data, but also OCD responses and error messages. The input memory size defines the number of fault experiments that can be executed on a single script (campaign), and the output memory size defines the amount of trace data that can be stored. The stored data and the knowledge of the running application code, enable the exact reconstruction of program flow. A communications manager is included in the controller core to translate commands into messages, manage the AUX port and store the messages received from the OCD. The width of the data buses defines the transmission delay required by each message. There are also debugger specific commands that make it possible to insert delays, react to messages from the OCD and autonomously execute bit-flip operations on data words. The execution steps required for each fault injection experiment are listed in Tables 2 and 3, for the offline and real-time scenarios.

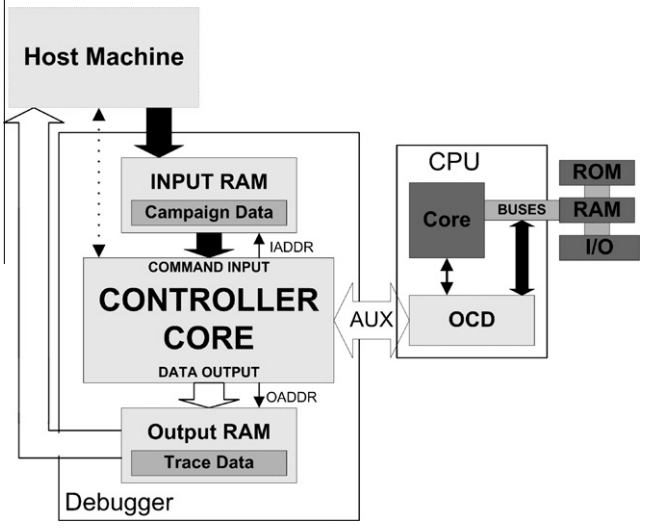

Fig. 3. Customized debugger.

The choice between steps 3 and 3+ depends on the possibility of predetermining the contents of the target memory cell prior to fault insertion, and is made by the fault injection script used.

3.2.3. OCD-FI

Improving the fault injection performance can also be accomplished by enhancing the functionality of the OCD. The OCD infrastructure with fault injection support (OCD-FI) was presented in [27,28], and proposes a workbench that is similar to the customized debugger described in the previous section. As the debugger and target CPU core are identical, the main difference is the presence of an extra fault injection (FI) hardware module embedded into the OCD circuitry.

Apart from the setup of fault insertion data (triggering and location) and external analysis of the results, the autonomous OCD-FI solution enables full control of fault activation and insertion, without the need for external signals. Fig. 4 presents a simplified view of the full OCD and FI module, including the control signals, data paths and registers used during the fault injection process.

The thick black arrows represent data exchange with external components (bus management modules are not represented for the sake of simplicity). The thick white arrows represent the main OCD-FI internal data paths used; SD represents the setup data, TD the trace data and FID the fault injection data. The Trigger control signal is used to confirm the occurrence of a watchpoint, RW is used for reading and updating the RWA registers and Exec for requesting the insertion of the faulty value into memory.

As our fault model is limited to bit-flip fault insertions, it only requires the execution of an XOR operation, between the data read by the RWA module immediately before the fault triggering instant, and the data mask preloaded on the FI module, which defines the bit(s) to flip. Due to performance requirements, the data link between the FI and RWA modules must be implemented via a dedicated bidirectional bus (FID).

The FI module reuses the OCD event detection (RCT) and memory writing (RWA) capabilities to automatically activate fault insertion upon the occurrence of a watchpoint. Once enabled, the FI module takes control of the entire OCD-FI infrastructure until the fault is inserted. Trace data generation is not affected during the entire process, continuing to operate as if a real SEU had occurred.

The FI module was designed to be adaptable to OCD infrastructures in general and NEXUS compliant devices in particular. It requires the OCD to implement (1) Watchpoint support; (2) Real-time memory access and (3) Memory read/write preloading capability.

If the required operations are available, the FI module implementation requires no substantial modifications to the OCD infrastructure, and is able to read the target memory and modify its contents. Using the OCD-FI for autonomous fault injection requires preloading of the target address (memory or register), and either (1) the data to be inserted or (2) a data mask defining the bit(s) to flip. If not predetermined, the faulty value can be generated upon the occurrence of the triggering condition. The faulty value is subsequently written back to the target cell. The sequence of steps to inject a fault is described in Table 4.

Steps 3 or 3+ are once again chosen by the fault injection script that configures the OCD-FI according to the intended scenario, enabling or disabling the predetermination capability. Inserting faults into internal registers requires the watchpoint to be replaced by a breakpoint, and the FI module to request that normal operation be resumed after fault insertion (this signal is ignored when inserting faults in real time).

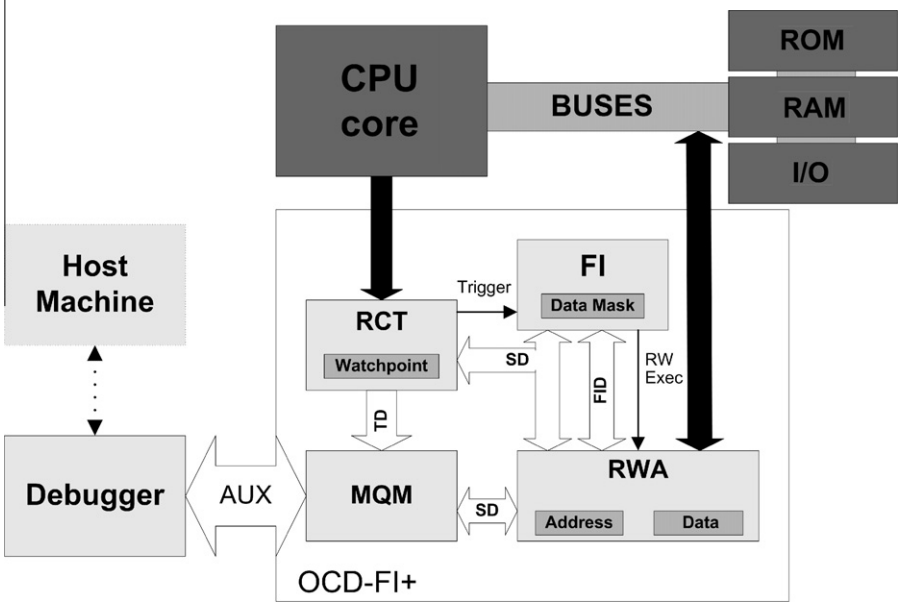

Fig. 4. OCD and FI module.

Table 4
Fault injection steps using the OCD-FI.

#	Step	Description
1	Set-Up	The microprocessor is reset and the 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
3	Fault activation (predetermination)	Upon the occurrence of the triggering condition the OCD-FI activates a memory write operation using preset values
3+	Fault activation (no predetermination)	Upon the occurrence of the triggering condition the OCD-FI uses a preset target memory cell address to retrieve its contents The OCD-FI applies a data mask to determine the faulty data value to be written into memory
4	Fault insertion	The OCD-FI directly inserts the faulty data in the previously addressed memory position

3.2.4. Extensions

The fault injection environment and methodology described so far were designed to handle real-time fault injection on NEXUS compliant devices, targeting either unmodified COTS devices or those requiring only minor modifications, incurring on minimum silicon overhead and no performance penalties. If additional fault injection capabilities are required for dependability evaluation, the OCD-FI infrastructure can be extended to add extra features, or to interface with additional components. In general, such extensions are adapted to each specific problem, and may degrade performance and eventually require additional modifications to the target CPU or OCD. Two scenarios where such extensions may be required are (1) targets equipped with hardware fault tolerance mechanisms, and (2) situations where real-time fault injection in internal registers is critical.

3.2.4.1. Error detection and correction (EDAC). In many critical systems, hardware fault tolerance is implemented by adding EDAC mechanisms between microprocessor and memory. Such solutions add extra bits to protected memories using special error correcting codes (e.g. Hamming codes). EDAC mechanisms generate the extra bits on write operations and check them on read operations. Depending on the number of extra bits, it is possible to detect and correct a variable number of errors [29].

To accurately evaluate EDAC-based fault tolerance features, it must be possible to emulate SEU effects by inserting single bit-flip errors into memory without affecting any other data or EDAC bits. As OCD infrastructures usually access memory through the EDAC mechanism, fault injection as envisaged is not possible, since single bit-flip errors are automatically corrected. The extension to the OCD-FI requires the ability to generate both the data to be written into memory and the codes used for error detection and correction. Fig. 5 presents the common OCD and CPU memory access buses and the alternate configuration required by the OCD-FI (EDAC).

This extension requires the OCD-FI to be able to use both configurations, operating as a common OCD when being used for debug purposes, or when faults target non-protected areas. The OCD-FI (EDAC) extension is enabled and configured when the fault injection experiment is Set-Up, and should be used whenever the fault targets EDAC protected memory areas.

3.2.4.2. Real time register access (RTREG). The problem of real-time fault injection on internal registers is more complex and requires modification of the microprocessor register file to allow simultaneous read and write operations. The RTREG extension requires additional collision control logic and predetermination of the faulty value to be inserted, as illustrated in Fig. 6.

The collision manager must ensure that the fault is injected only when the target register is not already being accessed for writing, and that the outputs are immediately updated if being accessed for reading. Once the triggering signal is received, the OCD-FI (RTREG) waits for an opportunity to insert the faulty value into the target

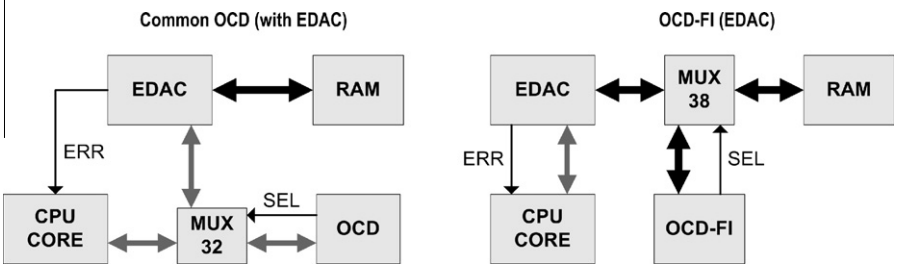

Fig. 5. Typical and OCD-FI (EDAC) interfaces.

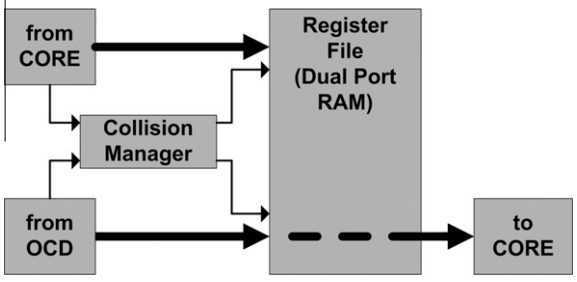

Fig. 6. Modified register file.

register, signaled by the collision manager. This procedure may cause problems with some combinations of triggering instants and target registers, which may prevent the faulty value insertion before the microprocessor accesses the relevant register. Since delaying a microprocessor action is undesirable under real time operation, the application code must be previously analyzed to exclude fault experiments that would cause such collisions. This mechanism has additional limitations, as it adversely affects the microprocessor dynamic performance (i.e. maximum operating frequency), and it is not possible to access intensively used registers (i.e. program counter). It can however be useful in situations where real-time fault injection in internal registers is more important than performance, and where the coverage limitations are acceptable.

4. Experimental results

4.1. Basic, extended and OCD-FI scenarios

4.1.1. Fault injection campaign execution

All modules were implemented in VHDL and synthesized using Xilinx's ISE version 7 [30]. All simulations were run on post place-and-route models using Modelsim 6 [31]. Synthesis was executed identically for all components using balanced area versus performance settings.

Due to debugger memory limitations, each fault injection campaign consisted of 10 experiments, injecting one bit-flip fault that emulates a single SEU. One hundred campaigns were executed on each scenario using our three target applications (*MAdder, VSorter, XControl*) in their normal and fault tolerant versions. Each campaign required 2 KB of input memory and 256 KB of output memory on the debugger.

Tables 5 and 6 present the results of the fault injection campaigns, classified by scenario and target application. All the scenarios that use offline fault injection (BOF, BOF+, EOF, EOF+) returned exactly the same results, which are presented on the first line of Table 5 (OFF row). Fault effects were classified into the following categories:

- U_{ERR}: undetected error – an erroneous final result not detected by the (eventual) fault tolerance routine (all errors will be U_{ERR} if there is no fault tolerance routine)
- D_{ERR}: detected error – the fault tolerance routine detected an error during execution. The application ended with an error detection signal.
- N_{ERR}: no error – the application ended correctly. This result includes both the errors that are still present in memory when the experiment ended and those overwritten by the running application.

Fault classification was performed after campaign execution, analyzing the contents of the debugger output memory. Trace information and the results of each application were compared with expected values to identify the occurrence of errors and their detection.

The execution of all experiments listed above and the results obtained led to the following conclusions, relative to the controllability and observability of our proposed solutions:

- The instant when the fault is inserted depends upon the delay between the occurrence of the trigger condition and the actual fault insertion operation. As this delay is constant and known for each configuration, it is possible to achieve precise control of fault insertion.

Table 5
Fault injection results (in.%).

Scenario	*MAdder*					*VSorter*					*XControl*				
	Non-FT		SW-FT			Non-FT		SW-FT			Non-FT		SW-FT		
	U_{ERR}	N_{ERR}	D_{ERR}	U_{ERR}	N_{ERR}	U_{ERR}	N_{ERR}	D_{ERR}	U_{ERR}	N_{ERR}	U_{ERR}	N_{ERR}	D_{ERR}	U_{ERR}	N_{ERR}
OFF	*19*	*81*	*28*	*13.9*	*58.1*	*98*	*2*	*97*	*2*	*1*	Not possible[a]				
BRT	19.4	80.6	28.3	13.8	57.9	98.1	1.9	96.8	2	1.2					
ERT	19.2	80.8	28.1	13.9	58	98	2	96.9	2	1.1					
OCD-FI	19	81	28	13.9	58.1	98	2	97	2	1					
BRT+	19.5	80.5	28.4	13.8	57.8	98.2	1.8	96.7	1.9	1.4	29.3	70.7	29.1	1.5	69.4
ERT+	19.3	80.7	28.2	13.8	58	98.1	1.9	96.8	1.9	1.3	29.6	70.4	28.9	1.2	69.9
OCD-FI+	19.1	80.9	28.1	13.9	58	98	2	96.9	1.9	1.2	29.8	70.2	28.8	1.1	70.1

[a] As the XControl application requires the use of external I/Os, predetermination is not practical, making fault injection campaigns impossible for the indicated scenarios.

Table 6
Occurrence of INC results (in.%).

Scenario	*MAdder*		*VSorter*		*XControl*	
	No FT	SW-FT	No FT	SW-FT	No FT	SW-FT
OFF	*0*					
BRT	3.1	4	0.9	2.2	Not possible	
ERT	1.4	2.3	0.6	1.1		
OCD-FI	0.2	0.2	0.1	0.2		
BRT+	3	4.8	1.2	2.8	2.1	3.2
ERT+	2	3.7	0.8	2.1	1.5	2.4
OCD-FI+	0.4	1.7	0.2	1.2	0.3	1.3

- All experiments can be repeated on similar scenarios (i.e. using the same target application), on exactly the same conditions, and replicated as often as necessary.
- It is possible to use the trace information generated by the OCD to reconstruct program flow. Fault effect classification can be executed via the OCD using trace data and memory reads.

Overall, our proposed real time methodology allows a high degree of controllability over the fault injection process and adequate observability for fault classification, even when operating in real time.

4.1.2. Analysis of fault injection results

The analysis of the fault injection results leads to some interesting conclusions relative to software fault tolerance efficiency, and to the effects of real-time fault injection. The following conclusions are worth of mention:

- When faults are injected while the target is halted (offline), the fault classification results are identical for all scenarios and when using real-time fault injection (with or without the FI module), the fault classifications results are only marginally different from one scenario to another.
- The effects of the injected faults are strongly dependent on the target application, and undetected errors are much higher for the *VSorter* application, due to its more intensive use of memory.
- The use of software fault tolerance substantially reduces the occurrence of undetected errors, namely for the *VSorter* (98% reduction) and *XControl* (96% reduction) applications. This reductions is less important in the case of *MAdder* (28% reduction), due to the lower refresh rate of the results.
- The percentage of correct results actually decreases when software fault tolerance is used, due to the larger memory area required and subsequent higher vulnerability to memory faults. However the percentage of undetected errors decreases significantly.

In conclusion, our software fault tolerance provides adequate error detection capabilities, but also reduces the probability of correct service, if used alone. Its effectiveness would benefit from the adoption of fault removal capabilities, possibly by forcing the application to restart upon error detection.

4.1.3. Real-time fault injection limitations

To evaluate the discrepancies between real-time fault injection scenarios, additional experiments were carried out, and those returning different results were replicated using extra debug operations, for this specific purpose. Each experiment was repeated with added data trace or, if necessary, with breakpoints immediately after fault insertion. Although this approach would be time-consuming for fault classification, it enables a finer analysis of the fault injection methodology. Erroneous fault insertions were classified as inconclusive (INC), and represent the cases where the fault injection process was corrupted due to a microprocessor write access to the target cell during fault injection. INC results occur in all three categories that were previously referred (U_{ERR}, D_{ERR} and N_{ERR}). Table 6 presents the percentage of inconclusive results found in each scenario.

The results shown above help us to understand the limitations of real-time fault injection:

- The OFF configurations always produce the most reliable results, as fault injection is performed when the target system is halted.
- In some cases the CPU overwrites the target memory cell before the fault injection operation is complete. This leads to an erroneous fault injection and these experiments should be discarded (as an inconclusive result) for dependability evaluation purposes.
- INC results become more probable as the delay between fault triggering and fault insertion increases, and as such vary within the scenarios and configurations that were considered. The use of an OCD-FI configuration and predetermination of the faulty value significantly reduces the occurrence of this type of results, particularly if used together.

The results obtained confirmed that our proposed solutions are an efficient alternative for injecting faults in memory, both in real time and offline scenarios. The best configuration depends on the target characteristics and dependability requirements. Offline fault injection is preferable for simpler scenarios (i.e. *MAdder*), and real time capabilities may be required for scenarios where external I/O must be included in the fault injection process (i.e. *XControl*).

The minor fault classification inaccuracies caused by real-time fault injection should be taken into account when analyzing dependability results. The importance of such inaccuracies will vary according to the target application and fault classification requirements. Overall, the OCD-FI configuration offers considerably better performance, and predetermination of faulty values should also be used, whenever possible.

4.2. Extensions (EDAC, RTREG) to the OCD-FI scenario

4.2.1. OCD-FI (EDAC)

The target system used for this scenario included the hardware EDAC mechanism between CPU and memory. The implementation of the OCD-FI (EDAC) extension required modifications to the OCD and to its interface. The EDAC mechanism itself requires additional logic and memory resources, and the CPU dynamic performance[1] is slightly degraded. Table 7 presents the results obtained with the OCD-FI (EDAC) extension, using only the non-fault-tolerant versions of the target applications.

The execution of fault campaigns using the EDAC extension provided the following conclusions:

- The OCD-FI (EDAC) extension can be used to automatically inject faults into memory blocks protected by hardware fault tolerance mechanisms.
- The use of an EDAC fault tolerance mechanism effectively eliminates the effects of single bit-flip errors on the target system, since they are all detected and corrected.

Hardware fault tolerance mechanisms like EDAC are increasingly used and must be adequately tested. The ability to directly

[1] *Dynamic performance* refers to the maximum operating frequency, as indicated by the VHDL synthesis tool.

Table 7
FI results for a target equipped with EDAC (in.%).

Predet.	*MAdder*				*VSorter*				*XControl*			
	D_{ERR}	U_{ERR}	N_{ERR}	INC	D_{ERR}	U_{ERR}	N_{ERR}	INC	D_{ERR}	U_{ERR}	N_{ERR}	INC
NO	39.6	0	58.8	1.6	98.3	0	0.8	0.9	29.9	0	69.1	1
YES	39.7	0	59.5	0.8	99	0	0.7	0.3	30	0	69.5	0.5

Table 8
FI results using the OCD-FI (RTREG) extension (in.%).

Scenario	*MAdder*					*VSorter*				
	No FT		SW-FT			No FT		SW-FT		
	U_{ERR}	N_{ERR}	D_{ERR}	U_{ERR}	N_{ERR}	U_{ERR}	N_{ERR}	D_{ERR}	U_{ERR}	N_{ERR}
OCD-FI (RTREG)	*89*	*11*	*62*	*22*	*16*	*60*	*40*	*46*	*14*	*40*

insert faults into memory without disabling its protection is required for adequately classifying fault effects. Our proposed solution enables to access the memory via the EDAC mechanism or to bypass it, which is useful not only for fault injection, but also for debug and classification.

4.2.2. OCD-FI (RTREG)

Current OCD implementations do not allow the injection of register faults in real time while the microprocessor is running, but it is possible to minimize the time interval during which the microprocessor needs to be halted. For real-time fault injection on internal registers, the OCD-FI (RTREG) infrastructure can be implemented on the target device. One hundred customized fault campaigns were selected from a larger set that was designed and executed using this infrastructure. All faults were defined to target the accumulator register for two reasons: its contents are easier to predict using program code knowledge, and it makes a good example, as it is the most intensively used register. All fault insertions were manually generated to ensure adequate synchronization. This procedure requires a prior study of the target application, in order to determine the instruction addresses that can be used as fault triggers. A given address will qualify if no change to the target register occurs during the fault insertion. Table 8 presents the results obtained using the OCD-FI (RTREG) extension.

The following conclusions are worth of mention:

- When targeting CPU internal registers in real time, triggering must be adjusted to ensure that faults can be inserted before the running application attempts to write on the target register.
- The instruction addresses that can be used as fault triggers depend on the target microprocessor, the running application, and the target register. The selection requires precise knowledge of the application code and instruction delays. For the accumulator register, using our workload applications, an average of 45% of the code space used qualifies for triggering.

The use of the RTREG extension shows that the injection of faults in internal registers is an important and complex problem. Registers are very sensitive to errors, and in critical systems it may be necessary to add extra hardware to protect them, and/or to more effectively test their sensitivity to faults. In some critical systems, adding on-chip support for register fault injection may be useful and justify the added intrusiveness and performance degradation.

4.3. Performance and overhead

4.3.1. Overhead

A Virtex-2 FPGA was used for experimental analysis due to the high implantation of this FPGA family for microprocessor-based systems, and the silicon overhead and the maximum operating frequency achieved are summarized in Table 9. The use of more recent and/or higher performance FPGAs causes an general increase in performance and small variations of the synthesis results, but has no effect on the fault injection process and the relative merits of each fault injection solution are fundamentally the same on all FPGAs. The reference scenario (shadowed line in the table) is the case where only the CPU core and basic OCD infrastructure are implemented, since this is the typical COTS situation.

The figures presented in Table 9 refer to a target CPU that is a based on a RISC architecture using a limited instruction set. The use of more complex microprocessors would lower the OCD overhead, since the area required is mostly dependent on the debug features implemented, and on target bus widths (that should remain constant).

In comparative terms, the extra overhead required for enhanced input bandwidth on the OCD (ERT) is fairly large (over 6%). Since the OCD-FI configuration presents much better results (less than 0.5%), it is preferable for real-time fault injection purposes. As would be expected, the inclusion of an EDAC mechanism slightly increases the microprocessor area, and also reduces its maximum

Table 9
Silicon overhead and dynamic.

CPU core	OCD	OCD-FI	EDAC	RTREG	Logic area Eq. gates	Overhead (%)	Max f (MHz)
X					53,926	75.4	37
X			X		55,018	76.9	32
X	BOF/BRT				71,527	100.0	36
X	BOF/BRT		X		72,619	101.5	32
X	EOF/ERT				76,127	106.4	36
X		X			71,842	100.4	36
X		With EDAC ext	X		73,184	102.3	32
X		With RTREG ext		X	76,392	106.8	27
X		With both ext	X	X	77,484	108.3	25

operating frequency. The degradation of these parameters, imposed by the EDAC and the RTREG versions of the OCD-FI infrastructure, are however within acceptable limits, considering that they are intended for safety–critical applications.

4.3.2. Comparison with other fault injection environments

For the fault model and the real-time requirements that were considered, the most frequently used fault injection techniques are either software or radiation based, although for our specific target system (available as a VHDL model), simulation based techniques would also be possible. A comparison between these approaches and our proposed solutions may be made as follows:

- Our solutions can be used either in simulation, in a programmable device (FPGA) or in an integrated circuit (ASIC), fitting the technology scenarios that cover the whole product development cycle.
- Most hardware based real-time fault injection methodologies would be more complex and expensive to implement, and sometimes require a customized hardware version. Some of our proposed solutions require modifications to the target hardware, but their low overhead facilitates market acceptance.
- Relative to radiation based fault injection or other contactless techniques, our proposed solutions have significant advantages in terms of experiment controllability and replicability. Precise control of fault location and injection instant is possible, facilitating experiment replication and deterministic results.
- Software based techniques are more intrusive, present similar fault injection delays, and offer more limited coverage.
- The need to handle erroneous fault classification results is common to all fault injection techniques, and more so when operating in real time. As in other approaches, problems can be minimized using statistical techniques or extra classification operations, whenever possible.

Quantitative comparison of fault injection methodologies is always complex, due the specific nature of each methodology and the considerable differences between target architectures and fault injection techniques. As an indicative example, Table 10 presents a list of measurable parameters for four fault injection techniques that can be used on our target system, considering similarly sized fault campaigns, but with unavoidable variations in terms of fault model, triggering and results.

The fault injection scenarios considered were derived from those presented on Section 4, adapted to each fault injection technique, with the target system being the same for all experiments. Execution time represents the total duration of all experiments, including setup and data collection, but not data analysis which is performed separately in all techniques. Fault coverage represents the percentage of memory elements where fault injection is possible and controllability represents the minimal element where faults can be inserted with precision. Costs were estimated for the execution of all experiments, considering the OCD-FI as a reference, and including manpower, software and hardware costs. It is important to note that the presented values may vary considerably for other fault injection environments or targets. For instance, fault coverage for SWIFI is lower than normal in our example due to memory protection issues and simulation execution time and coverage can also vary a lot depending on the software and/or target model used.

Table 10
Fault injection techniques comparison.

Technique	Execution time	Fault coverage (%)	Controllability	Cost estimate
OCD-FI	17 min	92	Individual bit	100
SWIFI	22 min	55	Individual bit	85
Simulation	1 h52 min	100	Flip-flop	215
Radiation	3 h30 min	100	Memory block	3000

4.3.3. Real time features

The proposed solutions were designed for real time operation, and are particularly advantageous when the fault injection experiments are designed in such a way, either to increase representativity or due to technical constraints.

- As the proposed solutions require no modification to the target applications or hardware, all workloads execute exactly as they would if not performing fault injections. The OCD infrastructures are not used during normal execution, and their use for fault injection adds no overheads or delays.
- Most traditional fault injection techniques are often unable to cope with real-time requirements, which are supported by our proposed solutions. Simulation based techniques or those requiring halts to the target system are particularly hard to use on real-time systems.
- Software based fault injection has been used in some scenarios, mainly when it is possible to use the timing characteristics of the operating system for near real-time fault injection. However, as task scheduling becomes tighter it becomes much harder to insert faults without imposing a small delay, with consequent loss of performance and representativity.
- Contactless techniques are non-intrusive by nature, and generally won't affect target performance. However, the experiment setup times are much higher due to the complexity of the fault injection equipment. The main issue when using these techniques is usually controllability, being impossible to target specific memory cells or adhere to precise fault injection timings.
- The use of OCD for real-time fault injection obviously requires these capabilities to be available, but these are becoming increasingly popular on modern devices, mainly microprocessor-based systems.
- When compared with similar NEXUS-based real-time fault injection techniques [8], our proposed solutions offer enhanced performance, with the subsequent minimization of inconclusive experiments.

In short, we presented the reasons why we believe that our proposed solution has the potential to be the best choice for fault injection on critical real-time systems, particularly if included early on the system design process.

5. Conclusions

OCD infrastructures offer a non-intrusive means of accessing internal microprocessor resources, and provide a useful mechanism for triggering and injecting faults, and for subsequently analyzing their effects. Performance becomes a fundamental issue when dealing with real-time systems, demanding enhanced capabilities from the debugging tools. Our proposed solutions and experimental work brought into evidence that it is possible to use OCD infrastructures for efficient real-time fault injection in memory space and internal registers. Our work has shown that it is possible to achieve precise control over the fault target, both in time and space. Reusing already available OCD infrastructures is an added-value in terms of performance, development costs and required resources. Execution is generally fast, minimizing the probability of inconclusive experiments, and enabling high fault/second rates, when mass injection of faults is required. Intrusiveness is minimal, as neither the target microprocessor nor the

running application are modified, and modification of the OCD is offered as an option. As an extra advantage, this solution allows the entire fault injection scenario, including environment, fault injector and target system, to be implemented on a single FPGA. Although other solutions may provide better performance [17], they usually require special resources or imply much larger silicon overhead. The best configuration depends on dependability requirements and on the target architecture – 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. The OCD-FI infrastructure can be easily extended to cope with target-specific requirements. As in many other situations, the best solution calls for a compromise between required capabilities and acceptable overhead.

Some limitations are still present in our proposed solutions – coverage is limited to the resources accessible by the OCD, but these locations represent a high percentage of the area affected by SEUs. The lack of an accepted standard may impose a considerable tuning effort to adapt the debugger and the FI module to each particular case, but the trend towards OCD standardization will facilitate this effort. Presently, NEXUS [24] is used in commercial devices and already provides useful features for fault injection purposes. However, different technologies may be adopted in the future [21–23]. Assuming that watchpoints and data preloading are available, our proposed solutions are flexible enough to be adapted to different OCD infrastructures, and are adequate to support real-time fault injection in current and future OCD-equipped microprocessors. Ongoing research is focused on broadening our application scope to different architectures and on improving fault coverage issues.

References

[1] M. Rebaudengo, M.S. Reorda, M. Violante, B. Nicolescu, R. Velazco, Coping with SEUs/SETs in microprocessors by means of low-cost solutions: a comparison study, IEEE Trans. Nucl. Sci. 49 (3) (2002).

[2] R. Velazco, F. Franco, Single event effects on digital integrated circuits: origins and mitigation techniques, in: IEEE International Symposium on Industrial Electronics, Vigo, Spain, June 2007.

[3] J. Gaisler, A portable and fault-tolerant microprocessor based on the SPARC V8 architecture, in: International Conference on Dependable Systems and Networks, June 2002.

[4] B. Nicolescu, Y. Savaria, R. Velazco, Software detection mechanisms providing full coverage against single bit-flip faults, IEEE Trans. Nucl. Sci. 51 (6) (2004) (December).

[5] M.S. Reorda, L. Sterpone, M. Violante, M. Garcia, C. Ongil, L. Entrena, Fault Injection-based reliability evaluation of SoPCs, in: 11th IEEE European Test Symposium, Southampton, UK, May 2006.

[6] C. Ongil, M. Valderas, M. Garcia, L. Entrena, Autonomous fault emulation: a new FPGA-based acceleration system for hardness evaluation, IEEE Trans. Nucl. Sci. 54 (1) (2007) (February).

[7] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, G.H. Leber, Comparison of physical and software-implemented fault injection techniques, IEEE Trans. Comput. 52 (9) (2003) (September).

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

[9] B. Rahbaran, A. Steininger e T. Handl, "Built-in fault injection in hardware – the FIDYCO example", Second IEEE International Workshop on Electronic Design, Test and Applications (Delta'04), pp. 327-332, Perth, Australia, Janeiro 2004.

[10] J. Gracia, J.C. Baraza, D. Gil and P.J. Gil, "Comparison and application of different VHDL-based fault injection techniques", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, October 2001.

[11] S. Misera, H.T. Vierhausa, A. Siebera, Simulated fault injections and their acceleration in SystemC, Microprocess. Microsyst. 32 (5–6) (2008). August.

[12] H. Madeira, R. R. Some, F. Moreira, D. Costa, D. Rennels, Experimental evaluation of a COTS system for space applications, in: International Conference on Dependable Systems and Networks, June 2002.

[13] R.J. Martínez, P.J. Gil, G. Martín, C. Pérez, J.J. Serrano, Experimental validation of high-speed fault-tolerant systems using physical fault injection, in: 7th IFIP Working Conference on Dependable Computing for Critical Applications, January 1999.

[14] R. Moraes, J. Duraes, R. Barbosa, E. Martins, H. Madeira, Experimental risk assessment and comparison using software fault injection, in: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007, Edinburgh, Scotland, 25–28 June 2007, pp. 512–521.

[15] M.P. García, C.L. Ongil, M.G. Valderas, L. Entrena, A rapid fault injection approach for measuring SEU sensitivity in complex processors, in: 13th IEEE International On-Line Testing Symposium (IOLTS 2007), Heraklion, Crete, Greece, 8–11 July 2007, pp. 101–106.

[16] L. Sterpone, M. Violante, A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs, IEEE Trans. Nucl. Sci. 54 (4) (2007) 965–970 (Part 2, August).

[17] M.G. Valderas, P. Peronnard, C.L. Ongil, R. Ecoffet, F. Bezerra, R. Velazco, Two complementary approaches for studying the effects of SEUs on digital processors, IEEE Transactions on Nuclear Science 54 (4) (2007) 924–928. August, Part 2.

[18] 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, in: International Conference on Dependable Systems and Networks, June 2005.

[19] C. MacNamee, B. Heffernan, Emerging on-chip debugging techniques for real-time embedded systems, Computing and Control Engineering Journal (2000) (December).

[20] M. Zenha-Rela, J.C. Cunha, L.E. Santos, M. Gameiro, P. Gonçalves, G. Alves, A. Fidalgo, P. Fortuna, R. Maia, L. Henriques, D. Costa, Exploiting the IEEE 1149.1 standard for software reliability evaluation in space applications, in: European Safety and Reliability Conference, October 2006.

[21] Enabling innovative IP re-use and design automation at www.spiritconsortium.org (last visited July 2008).

[22] R. Oshanae, G. Swaboda. Compact JTAG (IEEE P1149.7) overview of cJTAG, a reduced pin debug access protocol, in: 5th IEEE International Board Test Workshop (BTW'06), Denver, USA, September 2006.

[23] J. Rearick, B. Eklow, K. Posse, A. Crouche B. Bennetts, IJTAG (Internal JTAG): a step toward a DFT standard, in: International Test Conference (ITC'05), Austin, USA, November, 2005.

[24] "IEEE-ISTO 5001™, 2003, The Nexus 5001™ forum standard for a global embedded processor debug interface, in: IEEE Industry Standards and Technology Organization (IEEE-ISTO), 2003.

[25] J.C. Ruiz, J. Pardo, J.C. Campelo, P. Gil, On-chip debugging-based fault emulation for robustness evaluation of embedded software components, in: 11th Pacific Rim International Symposium on Dependable Computing, December 2005.

[26] G. Ferrante, 2003. CPUGEN 2.00 <www.opencores.org> (last visited July 2008).

[27] A. Fidalgo, G. Alves, J. Ferreira, A modified debugging infrastructure to assist real time fault injection campaigns, in: 9th IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems, April 2006.

[28] A. Fidalgo, G. Alves, J. Ferreira, OCD-FI: on-chip debug and fault injection, in: International Conference on Dependable Systems and Networks, June 2006.

[29] U. Kumar, B. Umashankar, Improved Hamming code for error detection and correction, in: 2nd International symposium on wireless pervasive computing (ISWPC'07), San Juan, USA, February 2007.

[30] FPGA and CPLD solutions from Xilinx, Inc. <www.xilinx.com> (last visited July 2008).

[31] ModelSim – a comprehensive simulation and debug environment for complex ASIC and FPGA designs. <www.model.com> (last visited July 2008).