Real Time Fault Injection Using a Modified Debugging Infrastructure

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

Dependability is a critical factor in computer systems, requiring high quality validation & verification procedures in the development stage. At the same time, digital devices are getting smaller and access to their internal signals and registers is increasingly complex, requiring innovative debugging methodologies. To address this issue, most recent microprocessors include an on-chip debug (OCD) infrastructure to facilitate common debugging operations. This paper proposes an enhanced OCD infrastructure with the objective of supporting the verification of fault-tolerant mechanisms through fault injection campaigns. This upgraded On-Chip Debug and Fault Injection (OCD-FI) infrastructure provides an efficient fault injection mechanism with improved capabilities and dynamic behavior. Preliminary results show that this solution provides flexibility in terms of fault triggering and allows high speed real-time fault injection in memory elements.

1. Introduction

Dependability is extremely important in safety critical applications. Dependable systems are designed to handle errors that originate from software or hardware faults and must be able to recover from them while maintaining acceptable operating conditions. The potentially destructive nature of a failure and the long error latencies make it difficult to identify the cause of failures in field operation and in the normal time that it takes for a failure to occur. Experimentation with a real device provides a better study scenario and helps to improve its dependability. This experiment-based approach requires knowledge of the system architecture and behavior and especially of the mechanisms implemented to provide tolerance to faults, errors or failures, i.e. the events leading to a service failure on microprocessor based systems [1]. Specific instruments and tools must be used to induce hazards and monitor their effects. In the case of microprocessor systems, access to internal resources is of utmost importance. Many of today's microprocessors support access through dedicated built-in debug circuitry, which is often referred as on-chip debug (OCD). The use of these OCD infrastructures for fault injection purposes is an efficient solution for verifying and validating fault tolerant designs. This paper describes recent research on the extension of a fault injection environment in order to allow efficient real time fault injection. Two techniques were evaluated, one based on a customized debugger and the other going a step further and proposing the upgrade of the OCD infrastructure itself.

The next section provides an overview of fault injection methodologies used on microprocessor systems. Section 3 presents the system used in our case study, consisting of a fault injection oriented debugger, and also proposes a modification to existing OCD infrastructures to enable enhanced fault injection. Section 4 presents the experimental results obtained so far and section 5 discusses these results and discusses future work directions.

2. Fault Injection in Microprocessors

Usage of fault-tolerant components is one way to achieve dependability. In such cases fault injection can be used to:

- Identify design or implementation faults.
- Verify & validate and fault tolerance capabilities.
- Estimate how often failures will occur and evaluate the consequences of such failures.

Fault injection is normally structured in campaigns, each campaign comprising a series of experiments during which the target system is in operation (a specific workload is activated) and a specific fault (or set of faults) is inserted at specific trigger conditions. The target system behavior is monitored and information is recorded as comprehensively as necessary and possible, to understand and evaluate the effects of the inserted faults. 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. Simulation techniques can be used on an early phase of development but are often time-consuming and may lack fault coverage as they are intrinsically dependant on the quality of the model. SWIFI techniques are less expensive but require modifications to the running code (which in fact modifies the target system) and faults can only be inserted in those resources that are accessible by software. Physical fault injection usually allows a better representation of real world faults but it is usually more expensive and less controllable.

The hardest part of microprocessor fault injection is how to access those internal elements where faults are more probable, generally the memory elements and communication buses, without disturbing any running applications. OCD infrastructures provide access to internal resources in parallel with the target hardware and running software, being an excellent mechanism for modifying register and / or memory values (i.e. insert faults) and subsequently retrieve the data necessary for result analysis. The non-intrusive nature of this form of fault injection is in itself an added-value, as it requires no modification to the target system. Most fault injection techniques that use OCD rely on halting the processor, either by the use of external control signals or via breakpoints, and subsequently modifying the target registers or memory locations to emulate a fault. The usual approach involves a host machine running the fault injection campaign and a debugger accessing the target infrastructure. As a technological solution, the major problem with OCD is the lack of a consistent set of capabilities and of a standard communications interface across processor architectures.

An industry consortium has been working on the establishment of a standard for OCD, which is formally designated "IEEE-ISTO 5001, The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface" [2]. If widely adopted, it will be possible to use the same debugger to access the core of multiple processors and to use a similar set of debugging features with all of them. This standard is still in a proposal phase, but it represents an interesting possibility for the development of common fault injection methodologies addressing the verification & validation of dependable microprocessor based systems.

Experimental work has been done in our research group and in the DISCA-UPV [3] using real-time fault injection on a MPC565 based system, which is the most widely used NEXUS compliant microprocessor. The results obtained confirmed our expectations and enabled the identification of some shortcomings concerning fault triggering and performance issues. Our experiments confirmed that it is possible to insert faults in the memory space on-the-fly and then use the trace information gathered as an effective means to analyze program flow, before and after fault activation.

However, two problems arose from these experiments, which are important for all real time fault injection devices. As most NEXUS compliant debuggers [4] [5] communicate with the host PC through Ethernet or USB connections, this communications channel imposes a bottleneck on the time required for memory access. The time required to read the contents of a memory cell and to write back a modified value is in the order of 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. In the case of small applications or frequently used memory cells, it becomes impossible to insert the desired type of fault without halting program execution. The second problem consists of the triggering of a fault. The required information is not readily available (even when using watchpoints or reading trace data without halting the processor), as it must reach the host machine before it can be acted upon. This additional delay effectively prevents its use for fault triggering, limiting the available options to time-based approaches.

This last problem can be solved by adding reactive behavior to the debugger, to enable it to perform a write operation upon the detection of a specific signal or message from the target system. The insertion of a specific fault on a memory cell used by a running application is mainly a performance problem. Reducing the writing delay of the fault injection process minimizes the probability of the cell being accessed by the target in the meantime.

3. Case Study

3.1 Target System

The use of a NEXUS compliant device benefits from the useful debugging features defined in the standard and increases the domain of immediate applicability of the proposed solutions. As neither the current commercial NEXUS debuggers nor the compatible CPUs are easily modifiable, an alternative microprocessor core where a NEXUS compliant OCD infrastructure could be implemented was selected. A customized debugger was also necessary, as available devices require specific libraries for each target. The OCD and the debugger were developed in the form of VHDL modules, to ensure portability and to maintain a high level of compatibility with different target architectures. In this way a complete proof-of-concept solution was tested and the requirements for its migration to existent systems (or to those under development) were evaluated.

The cpugenerator [6] building tool was selected to create the microprocessor targets. This tool is publicly available through opencores [7] and allows the automatic creation of 4, 8, 16 or 32 bit RISC microprocessor cores, enabling the configuration of several parameters such as bus type, interrupt support features and memory configuration.

The OCD version implemented on the target system is NEXUS Class-2+ compliant (all Class 2 plus some Class 3 features) and includes some customization capabilities to make it compatible with different CPU configurations, requiring only minor adjustments. The target application for testing is a *matrix_addFT* program, which is a fault tolerant version of a matrix adder. Fault tolerance is achieved by duplicating each arithmetic operation and then comparing the obtained results; any difference will trigger an error detection routine. Although not as powerful as hardware fault tolerance, this solution enables some degree of dependability without requiring modifications to the hardware, at the cost of memory space and performance penalty.

The NEXUS standard defines a minimum set of debugging features, the interface port and the communication protocol. The implemented features include all common OCD features plus real time access to memory. The interface with the outside world is made using the AUX port option, which provides two message data buses for OCD data input and output along with independent clock and control signals. Two additional event pins allow halting the processor and exact timing for watchpoint / breakpoint signaling. The communication protocol was implemented as defined in the standard, with all mandatory messages being implemented and two optional messages added for internal register access and OCD configuration. The OCD infrastructure is divided in three main modules and two bus access modules as represented in Figure 1. The thinner arrows represent control and status signals and the thicker arrows represent the flow of data and trace information. The FI module represented is not included in the original OCD and is used to build the OCD-FI version explained further ahead on this paper.

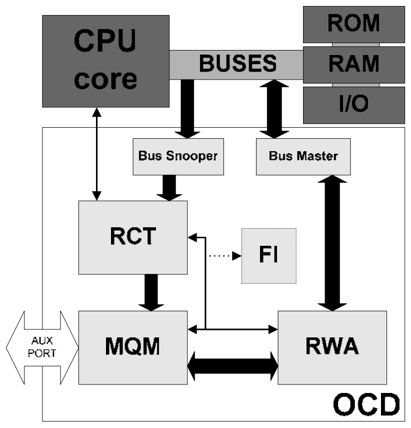

Figure 1 – The OCD Infrastructure

The MQM (Message Queuing and Management) 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 message queues are implemented using FIFO (First-In First-Out) memory blocks and in the case of an overflow, an error message is sent from the MQM module to the debugger, via the NEXUS port.

The RCT (Run Control & Trace) module receives commands from both the MQM and RWA modules and outputs trace data and watchpoint hit signals. It also controls the CPU core clock and the signals required to identify branch and exception occurrences on the running application. It is possible to use up to two instruction and one data breakpoint and both types can be activated at the Nth occurrence of their trigger condition. Additionally a watchpoint may be generated in the same manner as either type of breakpoint. The RCT is linked to the Bus Snooper which is used to monitor data and instruction bus activity, to allow program trace and breakpoint / watchpoint generation. Program trace is performed using branch trace messaging as defined in the NEXUS standard, accounting for executed instructions and signaling branch and exception occurrences.

The RWA (Read & Write Access) module is used to access both the OCD registers and the CPU resources (memory and registers). A register (RAW_REG) is used to store the data and address of the next read / write operation, as this information takes several clock cycles to be transmitted by the MQM module. Conflicts in RAM access are handled by the bus master with the OCD taking priority on access by default. As inputs and outputs are handled by the processor as directly mapped addresses it is possible to access those resources in the same manner as they would be accessed by the microprocessor.

3.2 Fault Injection Environment

In the case of microprocessor systems with built-in debugging mechanisms two areas where the fault injection capabilities can be improved are the debugger and the OCD itself. In an effort to be as comprehensive as possible, experiments were conducted on both areas and the results compared. The selected fault model is used in most common fault scenarios for microprocessor based critical systems [8] and consists of single bit-flip faults in random memory elements, occurring at random moments during the application execution. The fault trigger can be any instruction occurrence of the application currently running, covering the entire execution time. The fault location can be any resource accessible via the OCD, including memory, internal registers and stack. All experiments are structured into fault injection campaigns, and each of them defines a set of fault injection operations where a specific fault location and trigger are selected. In each of such operations the processor is reset and the application runs from start. Each campaign is generated by an external tool and then described as a script that includes the necessary messages to be sent to the OCD infrastructure, both for configuration and data collection. The initialization phase loads the application into memory and sets up the OCD infrastructure as required by the specific operation. The target memory value at the moment of the injection must be determined beforehand, using either the knowledge of the running application code or a prior fault-free execution up to the fault triggering instant, and then using the OCD to read the relevant memory cell contents. In this manner it is possible to determine the value that should be stored, so that a single bit-flip is caused on the target with a single write operation.

The fault injection campaigns can be used for experimental evaluation of the target device fault tolerant characteristics and preliminary results were analyzed to evaluate the fault injection procedure itself. The normal fault injection scenario consists of the NEXUS compliant target microprocessor, a host machine running the fault injection campaigns and a debugger connecting both. This scenario is represented on Figure 2, where boxes #1 and #2 may represent simulation modules, physical devices or parts of the same FPGA.

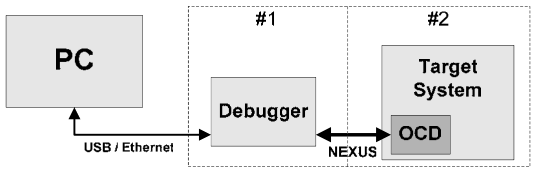

Figure 2 – Fault Injection Scenario

Each fault injection operation consists of loading the input memory bank with a series of instructions describing the steps required for its execution. These include the necessary OCD set-up and application loading steps. After these are completed the debugger waits for the triggering condition to be met which will be signaled by a watchpoint hit signal. Although the debugger allows an instant reaction to this signal, the actual fault insertion still requires the transmission and decoding (by the OCD) of at least one complete message. During the entire operation the output memory records the trace messages that are sent by the OCD, to enable the reconstruction of program flow and fault effect analysis. After the application runs the OCD may be used to check if all final results are correct. All steps can be done with the target processor running normally, but the fault activation may only take place after this set up is performed. The program trace is not affected and operates normally before, during, and after the fault injection process, reacting exactly as if a "real" fault was inserted.

3.3 FI Module

The On-Chip Debug and Fault Injection (OCD-FI) concept proposed on this paper consists of an additional hardware module that automatically inserts faults on the occurrence of a triggering condition, without further commands from the debugger. This Fault Injection (FI) module is represented in Figure 3 and is implemented within the OCD circuitry reusing some of the debugging functions that are already implemented.

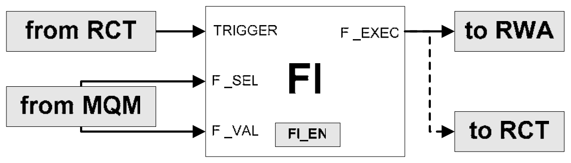

Figure 3 – Fault Injection Module

Two requisites have to be met by the OCD infrastructure to enable the usage of this module, namely:

- Write operations must be executed on the activation of a specific control signal that can be controlled by the FI module. This may be performed by pre-loading one or more registers prior to the write operation itself.
- The OCD must signal watchpoint occurrences and this signal must be available to the FI module. Alternatively, a breakpoint signal may be used, but for real time operation the actual halting of the execution must be inhibited.

Depending on the actual OCD architecture, some additional logic may be necessary for signal multiplexing and status control. Once enabled the FI module monitors the watchpoint or breakpoint signal, so that it can activate a fault injection action. The input signals FI_SEL and FI_VAL are used to access the FI_EN register, which enables and configures the FI module. The TRIGGER input prompts the execution of the FI operation. The output signal (FI_EXEC) is used to activate a memory write operation in order to insert a single bit-flip fault at a given address. This approach requires that both the data value to be written and the respective memory address that were previously determined be preloaded in the OCD register (RAW_REG) that is used for data writing. The required data must be downloaded to the OCD infrastructure prior to the watchpoint occurrence, and the RAW_REG register must not be rewritten until the actual fault activation. Once the fault is inserted, the FI module disables itself and all the OCD resources can then be used normally. Two dedicated (optional) NEXUS messages are used, one to enable and configure the FI module and the other one to set up the address and data values for the actual fault injection. Fault triggering can be done using either a breakpoint or a watchpoint. The watchpoint option allows the injection of faults without stopping the target system but can only be used for memory, as access to internal register requires the system to be halted. For the insertion of faults in internal registers or the stack it is necessary to add a breakpoint with the same address as the watchpoint, to ensure that the processor is halted when fault injection takes place. In this case, the signal used for fault activation can also be used to restart program execution, as represented in Figure 3 in the form of a dotted line. In this case the OCD-FI infrastructure allows the insertion of faults in all resources mapped in the OCD, with a minimum time delay. The FI module can also be programmed prior to the application start or in runtime, the only limiting factor being the fault activation instant.

4. Experimental Results

Three CPU configurations were used differing only in terms of bus width, all including full interrupt support and internal stack. All configurations include separate ROM and RAM banks on the target system, the first for storing the program code and the later for application data. The fault campaigns were structured as follows:

- For the sake of simplicity, fault campaigns are divided between those where activating the fault injection can be done with the processor running and those where it must be done during set up.
- The instruction address that triggers each fault injection is randomly generated from the accessed ROM space and each target memory position is also randomly selected from the used RAM space.
- The OCD is configured once at the beginning of the campaign, which is then loaded into memory and the experiments executed sequentially.
- The results are retrieved after all the experiments are complete, their analysis being performed externally to check if the final results are correct and if the fault was detected.

Each set of fault campaigns was executed on each of the configurations and repeated using both the original OCD and the OCD-FI infrastructure. After simulating several fault campaigns the following conclusions, relative to the fault injection processes, were reached:

- The OCD-FI infrastructure does not degrade the maximum microprocessor clock frequency, and it is possible to use the same frequency for all clocks.
- Each infrastructure requires a minimum number of clock cycles for system set up prior to each fault injection operation and for the writing operation itself, as represented in Table 1. Set up time assumes that all configuration registers are already set up (prior to the fault injection) and writing time is measured from the watchpoint hit to the writing instant of the intended value into memory.

Table 1 – Fault Injection Delay (in CLK cycles)

	OCD		OCD-FI	
CPU	**Set up**	**Writing**	**Set up**	**Writing**
8 bit	13	14	28	2
16 bit	14	18	32	2
32 bit	14	21	36	2

- If targeting internal microprocessor registers, execution must be halted for only 2 additional clock cycles if using the OCD-FI infrastructure, which increases slightly the time interval required to run each fault campaign.
- If using only the OCD for register access, the time interval during which the processor must be halted is 2 clock cycles higher that the time required for memory writing.
- When using only the OCD some experiments return meaningless results because the CPU writes on the memory cell being targeted before the fault is inserted. This did not happen with the OCD-FI.

The number of equivalent gates for each module and each CPU configuration is presented in Table 2. The number of gates required for the Bus Snooper and the Bus Master modules are included in the RCT and RWA counts, respectively.

Table 2 – Area Overhead

Modules	8 bit CPU		16 bit CPU		32 bit CPU	
	# Gates	%	# Gates	%	# Gates	%
CPU core	9166	N/A	20212	N/A	53717	N/A
RCT	2391	34	3730	31	5113	27
RWA	369	5	516	4	643	3
MQM	4225	60	7715	65	13045	69
FI	75	1,1	75	0,6	75	0,4
OCD-FI	**7060**	**100**	**11961**	**100**	**18876**	**100**
Debugger	766	N/A	817	N/A	1079	N/A

Synthesis results confirm that the logic overhead of the FI module is minimal. It is also possible to see that a simple NEXUS compliant debugger loaded only with fault injection campaigns management and results storage requires comparatively little space on a programmable device. The area of the OCD itself is comparatively large, as the implemented CPU cores are rather simple in terms of registers and instruction support. This effect is less notorious as the CPUs increase in complexity, because the OCD area mostly depends on the size and complexity of the communication buses.

5. Conclusions and Future Work

From the available results it is possible to conclude that the proposed OCD-FI infrastructure is an efficient mechanism for verifying and validating the fault tolerance characteristics of microprocessor based systems. The implementation of a fault injection oriented debugger allows the inclusion of some features that are lacking in the majority of commercial devices, as they are not required for most debug operations. The FI module main advantage is its extremely fast reaction time. When compared with other alternatives, it provides an efficient methodology for fault injection, both in terms of reusability, resource coverage, performance and cost. If the necessary HDL modules are available, the OCD-FI can be used for fault injection in the simulation phase, prototyping phase or in the final device. Faults can be inserted on most CPU resources with a minimum delay, allowing non-intrusive and fast fault injection campaigns. The achieved performance is better when targeting memory space and when the faults are not injected early in the application execution. In this case, fault campaigns can be executed almost as fast as it takes to run the target application and without stopping it. Even in less than ideal circumstances, either by targeting microprocessor registers or injecting early faults in memory, it is a very efficient mean to execute fault injection campaigns. The compliance with the NEXUS proposed standard provides a common basis for development and enhancement of the proposed methodology. In this sense, the OCD-FI concept can easily be extended to any NEXUS compliant microprocessor and even other architectures as the more complex functions are performed by the OCD infrastructure. As this is already required for debug purposes, the added FI module provides considerable advantages with a very low logic overhead. It should be easy to add to most devices, and with eventual modifications it is a lightweight solution for most microprocessor architectures. As an added feature, the debugger may be included into the same programmable device as the target system, therefore ensuring best performance and reducing the necessary resources and associated costs, the only limitation being the availability of memory for data storage. As a downside, an adequate OCD infrastructure is needed and both the OCD and the target CPU must be available in the form of an HDL model. If injecting faults on a physical device, an external debugger is also required along with an adequate communications channel. The ongoing work addresses the applicability of this fault injection approach to different scenarios and fault tolerant architectures. Different target systems are also being considered, the LEON [9] being a good prospective target due to its possible use in space missions, where dependability is of utmost importance.

6. References

[1] "Basic concepts and taxonomy of dependable and secure computing"; A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr; IEEE Transactions on Dependable and Secure Computing, Volume 1, Issue 1; Jan 2004.

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

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

[4] www.isystem.com

[5] www.lauterbach.com

[6] "CPUGEN 2.00", Giovanni Ferrante, 2003.

[7] www.opencores.org

[8] "How to characterize the problem of SEU in processors & representative errors observed on flight"; R. Velazco, R. Ecoffet, F. Faure; 11th IEEE International On-Line Testing Symposium; Saint Raphael, France; July 2005.

[9] www.gaisler.com