An Enhanced Debugger for Real-Time Fault Injection

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

Abstract

This paper presents a case study on the reuse of the on chip debug infrastructures, present in most recent microprocessors, to execute real time fault injection campaigns. It is based on a debugger customized for fault injection and designed for maximum performance and flexibility. The developed methodology can be applied on the verification of dependable systems.

1. Introduction

As electronic systems increase in complexity and decrease in size their correct operating behavior is becoming harder to guarantee. Fault injection is a possible solution to test fault effects and tolerance in order to evaluate the target system dependability. Specific instruments and tools must be used to induce faults and monitor their effects and in the case of microprocessor systems, access to the internal resources is of utmost importance. Many of today's microprocessors provide such access through dedicated built-in debug circuitry, often designated as on chip debug (OCD). This paper describes recent research on real time fault injection (i.e. without halting application execution) using such devices, based on the development and use of a debugger optimized for fault injection.

A major problem with OCD is the lack of a consistent set of capabilities and communications interface. An industry consortium has been working on the establishment of a standard for OCD, which is designated as "IEEE-ISTO 5001, The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface" [1]. The feature set that this standard proposes provides a useful set of tools for real time fault injection in the form real time access to memory and on-the-fly program and data trace. Experiments with commercial compliant devices, namely the MPC565 microprocessor and compatible debuggers, show that the available features and performance are somewhat limited for real time fault injection [2].

2. Case Study

The debugger used for the presented case study was developed as a customized solution for real time fault injection on NEXUS compliant devices. The fault injection environment used consists of a target microprocessor with a compliant OCD, the debugger running the fault injection campaigns and the host machine being used for set up and data analysis. This is represented in Figure 1 .

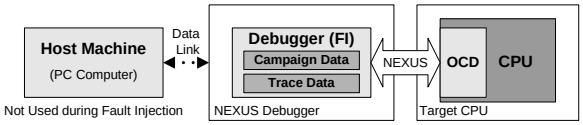

Figure 1 – Fault Injection Environment

The customized debugger consists of a controller core connected to two memory banks and to a NEXUS debug port, as represented in Figure 2.

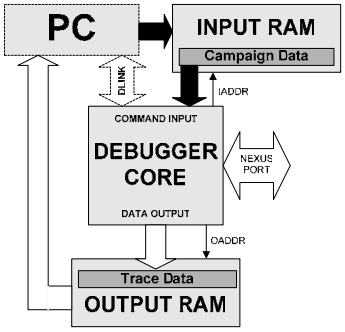

Figure 2 - Debugger

All elements were designed to optimize the fault injection process, with emphasis on execution speed. The debugger core is a simple processor type device that (1) fetches commands from the input memory; (2) stores trace data on the output memory; (3) controls execution and communications and (4) manages possible error conditions. The fault injection experiments were structured into campaigns, each one defining a set of operations where a specific trigger condition (instruction address) and fault coordinates (location x value) were selected. Each campaign is

described as a script with the necessary commands stored into the debugger memory. After the initial set up, the debugger waits for the triggering condition to be met (signaled by a watchpoint hit) before sending a message to the OCD instructing it to write into the target memory position the intended faulty value. During the entire operation the output memory stores the trace messages that are sent by the OCD (via debugger), to allow a subsequent fault effect analysis.

3. Experimental Results

A single 32 bit RISC CPU core and three different OCD (and compatible debugger) configurations were used, as summarized in Table 1, where the MPC565 is included for comparison.

Table 1 - Target System Configurations

Configuration	*CLK (MHz)*	*MDI (bits)*	*MDO (bits)*
Normal	*25*	*2 bits*	*8 bits*
Extended	*25*	*4 bits*	*8 bits*
Parallel	*25*	*Parallel Link (72 bits)*	
MPC565	*40*	*2 bits*	*8 bits*

The *Normal* and *Extended* configurations vary in terms of NEXUS port width and on the size of the internal message buffers, with MDI being the Message Data In bus and MDO the Message Data Out bus. *Normal* represents a configuration equivalent to the best available for the MPC565 microprocessor and *Extended* represents an improved configuration for faster memory writing. The *Parallel* configuration replaces the NEXUS communication elements present on the OCD by synchronous access to the OCD input signals and requires a special version of the debugger.

The simulation of about 100 fault campaigns, targeting memory space and triggered by the running application, repeated for each configuration, returned the results presented in Table 2. In this table inconclusive results represent experiments that had to be discarded due to the running application interfering with the fault injection process, and fault injection delay represents the time interval between the meeting of the trigger condition and the actual insertion of the faulty value. The synthesis results for the main components are also presented.

Table 2 – Fault Injection and Synthesis Results

Configuration		***Normal***	***Extended***	***Parallel***
Inconclusive Results		4%	3%	0%
Fault Injection Delay (In Clock Cycles)		38	21	3
Equivalent Logic Gates	*CPU Core*	53717		
	OCD	17601	18801	15211
	Debugger	992	1079	820

The following analysis is also possible at this stage:

- It wouldn't be possible to execute the same fault campaigns (on real time) on a system using an MPC565 and a commercial controller as the reaction delay would be too high.
- The width of the communication bus between the debugger and the OCD clearly affects the performance of the fault injection process, with the use of larger buses reducing the occurrence of inconclusive results.
- The synthesis results confirm that the debugger (tasked only with fault campaign management and results storage) requires comparatively little space on a programmable device.

4. Conclusions

Dependability evaluation efforts sometimes neglect the possibilities of powerful OCD infrastructures, present on the target device, even knowing that their use as a mean to execute non-intrusive fault injection campaigns is often a good solution in terms of performance and capabilities. Our case study shows that the use of an optimized debugger and an OCD with real time access capabilities allows the execution of real time fault campaigns on the target memory space. The main advantage of this fault injection solution is the debugger capability to manage the entire fault injection process. Although the host machine is responsible for downloading the campaign data to the debugger and uploading the trace data after the campaign execution, the entire process is executed autonomously by the debugger. The possibilities in terms of fault triggering and fault injection delay are dependent on the OCD capabilities, with communication speed being a key factor. The use of larger communications ports allows faster operation and therefore minimizes the risk of the running application interfering with the process. The standardization of OCD capabilities and access ports would also benefit the reusability of this fault injection approach.

5. References

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

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