"OCD-FI, On-Chip Debugging and Fault Injection for validating microprocessor based dependable systems"

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

This paper proposes a set of modifications to the on-chip debugging infrastructures present in many actual microprocessor cores, with the objective of supporting the validation and verification steps of fault-tolerant mechanisms through fault injection campaigns. A synthesisable microprocessor core for programmable components was used as a target system an. a debugging infrastructure compliant with the NEXUS 5001 proposed standard for on-chip debugging was implemented on this target. To improve the process of real-time memory fault injection, an upgraded infrastructure designated as On-Chip Debugging and Fault Injection (OCD-FI) was developed. The complete system was analysed in terms of area overhead and fault injection capabilities and performance. All elements were designed as synthesizable VHDL modules and evaluated in simulation.

1. Introduction

In recent years, there has been a rapid increase in the use of microprocessor-based systems in critical areas where failures imply risks to human lives, the environment or expensive equipment. One solution for avoiding a possible disaster lays in the use of dependable systems, able to tolerate and eventually correct faults, requiring high quality validation & verification in their development cycle.

Most recent microprocessors include an on-chip debugging (OCD) infrastructure to facilitate common debug operations. Although these vary considerably from case to case, they usually include a similar set of features like memory read and write, instruction single-stepping, program trace and some type of breakpoint support.

The motivation behind the work described in this paper is the belief that it is possible to develop a efficient fault injection methodology using a modified debugging infrastructure. The objective is to provide additional capabilities and an increase in performance of the fault injection process.

2. State of the Art

In safety critical computer based applications dependability is of utmost importance. Dependable systems are designed to detect errors that originate from software or hardware faults and recover from them maintaining acceptable operating conditions. The verification and validation of these systems is an important and hard to handle problem, although benefiting from some proposed solutions such as: analytical modeling, experimental techniques and fault injection [1]. Fault injection is recognized as a powerful solution, particularly to measure the effectiveness of the error detection mechanisms. It consists of injecting faults in the system components, while functional applications are being executed, and then observing the system response.

The hardest part of this approach is the methodology for actually injecting the fault, namely how to access those elements of the microprocessor where faults are more probable, generally the memory elements and communication buses.

The efficiency of a fault injection technique depends on the controllability and observability level of each microprocessor. Nowadays, almost every microprocessor comes with a debug & test infrastructure which provides a reasonable mean to access its core. However, such infrastructures are generally based on different architectures and access ports, normally requiring specific hardware and often with proprietary parts.

IEEE-ISTO 5001-1999, the NEXUS 5001 Forum Standard for a Global Embedded Processor Debug Interface [2] is an open industry standard that provides a general-purpose interface for the software development and debug of embedded processors. This proposed standard is an interesting possibility for the development of a common fault injection methodology for the verification and validation of critical microprocessor-based systems.

Most fault injection techniques that use on-chip debugging infrastructures rely on halting the processor, either by the use of control signals or using breakpoints, and subsequently modify the targeted registers or memory locations to emulate a fault. The usual approach involves the use of a host

machine running the fault injection campaign and a debugger to access the target infrastructure.

The NEXUS proposed standard offers some interesting possibilities, as it allows real time access to memory and trace information. Work on this area has confirmed the importance of these capabilities [3] for fault injection purposes, and allowed the identification of some shortcomings in synchronization and performance.

As all the available NEXUS compliant debuggers communicate with the outside world through either Ethernet or USB cables, this imposes a critical limitation on real time fault injection in high performance systems. Depending on the memory position targeted, it may prove difficult or even impossible to read its contents and write back a modified value before it is actually written by the running application.

Fault Triggering is also a problem as even using a debugger [4] that outputs trace data without halting the processor, this information is not readily available as it must reach the host machine before it can be acted upon, and this delay can be measured in milliseconds or more, which effectively prevents its use as a triggering solution.

3. Target System

The choice of target system was made with some key features in mind. It had to be freely available as a fully synthesizable VHDL (or Verilog) model. The possibility of using different processor configurations and the ease of modification were important selection factors. The actual implementations of the device and the available documentation were also considered.

The final choice fell on a publicly available *opencores* [5] project, namely the *cpugenerator* [6] building tool. This tool allows for the automatic creation of 4, 8, 16 or 32 bit microprocessor cores.

In *cpugenerator*, data and address buses can be configured and several RAM and ROM types can be used, including the possibility of implementing a single RAM block for data and program storage. The instruction set is pre-defined, but it is possible to add new instructions. Interrupt support and stack configuration are also configurable.

A compiler tool is also provided to translate assembler (text) files into object files, readable by VHDL memory simulation modules. The tool output is a set of VHDL files that compose the microprocessor core and some additional (non-synthesizable) files that allow RAM and ROM emulation. It is also possible to use synthesizable memory definitions, but in this case the memory contents must be created manually.

The target applications used were two specifically designed programs. The first, designated as *iset_test*, consists of a test program that executes all instructions present in the original *cpugenerator* instruction set. A second program is the *matrix_addFT* which is a matrix adding application with fault tolerant characteristics. These consist of duplicating each arithmetic operation and then comparing the results, with any difference triggering an error detection routine. Although not as powerful as hardware fault tolerance, this solution allows for some degree of fault tolerance without modifications to the hardware, at the cost of some ROM memory space.

4. Debugging Infrastructure

The implemented debugging infrastructure was designed from the beginning to be compliant with the NEXUS 5001 proposed standard. This option was taken with so that the subsequent modifications could be applied to all compliant infrastructures.

As there is no mandatory implementation, the OCD design was based on the infrastructure present on the MPC565 microcontroller, which is a well documented NEXUS compliant device. The objective was to implement a Class-2 compliant infrastructure as configurable as the target system. It should be compatible with the different CPU configurations and also automatically generated.

The NEXUS proposed standard defines the minimum set of features and the communication port to be used on a compliant infrastructure. For a class 2 compliant device the required features are:

- Read/Write User Registers in Debug Mode
- Read/Write User Memory in Real Time
- Enter a Debug Mode from Reset / User Mode
- Exit Debug Mode to User Mode
- Single step instruction in User Mode and re-enter Debug Mode
- Stop Program execution on instruction/data breakpoint and enter Debug Mode (minimum 2 breakpoints)
- Ability to send out an event occurrence when watchpoint matches
- Ability to set breakpoint or watchpoint conditions
- Device identification Message

The communication protocol is defined in the standard and is message based. Each message consists of a code defining the message type and additional data packets if required. The infrastructure accepts command messages in this form and outputs result messages

As to the NEXUS port, it must include the following signals:

- Two clocks for messaging input and output control.
- Message Data Buses (Output and Input) for data communication. Depending upon bandwidth

requirements, one, two, four, eight, or more pins may be implemented for each bus.

- Message Start/End (Output and Input) indicate when a message on the respective data bus has started, when a variable-length packet has ended, and when the message has ended.
- Two event pins (Output and Input) with the input pin allowing halting the processor and the output pin indicating exact timing for a single breakpoint status indication.
- A Reset pin for resetting the Nexus infrastructure.

The implemented OCD infrastructure is divided in three modules as seen on Figure 1.

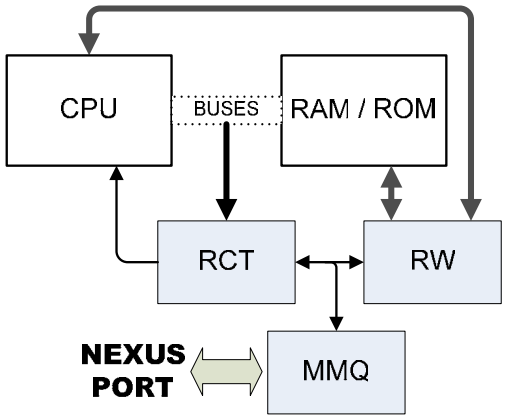

Figure 1- OCD Infrastructure

The **RCT** (Run Control & Trace) module is responsible for CPU run control and bus snooping. It receives commands both from the MSG module and the RW module and outputs trace data and watchpoint hit signals.

The **RW** (Read & Write) module is used both to access debug specific registers and CPU resources (memory and registers). This module uses a special register (RAW) where data and access information is stored so that a single triggering signal may order the execution of a single read/write operation.

The **MMQ** (Message Management and Queuing) module is the NEXUS message handler that translates all debugging operations into messages and vice versa and manages the message queues.

Some additional logic is required to deal with signal multiplexing, collision handling and timing issues.

All required NEXUS features were implemented, being possible to insert up to two program and one data breakpoint. Both types of breakpoints can be activated at the N occurrence of their trigger condition. Additionally a watchpoint may be generated in the same manner as either type of breakpoint. Program trace is performed by sending out messages at program branch or exception occurrences. If enabled, data trace can be performed by messaging out data values and addresses at memory write instants.

5. On-Chip Fault Injection

Any NEXUS compliant OCD infrastructure already has both triggering and data access capabilities, in the form of watchpoint support and read/write access to microprocessor registers and memory.

On-Chip Fault Injection (OCD-FI) can be described as a hardware module implemented on the microprocessor chip that uses the available debugging functions to automatically inject faults.

The proposed solution was developed with three objectives: Simplicity, adaptability and efficiency. It has to be simple to imply the least logic overhead, it should be adaptable so that it can be configured for different microprocessor architectures and it has to be efficient to justify its use.

OCD-FI consists off adding an additional Fault Injection (FI) module to a (NEXUS compliant) OCD infrastructure. This module, when enabled, monitors the watchpoint signal(s) so that it can activate a memory write operation to inject a bit-flip fault on a given address.

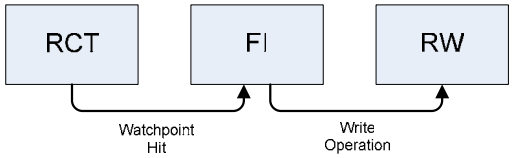

Figure 2 – Fault Injection Module

This approach requires that both the data value to be written and the respective address be determined beforehand and preloaded in the RAW register. To do this it is necessary a previous analysis of the running application to determine the target memory position contents at the injection instant. 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.

The required data must be downloaded to the OCD infrastructure prior to the watchpoint occurrence, and the RW module must not be used until the actual fault activation.

Once the fault is inserted the FI module disables itself and the OCD resources can then be used normally. It should be noted that the trace features are not affected by this and operate normally before, during and after the fault injection process, reacting exactly as if a “real” fault was injected.

As the NEXUS proposed standard has support for additional (configurable) messages, it is possible to use these to control the fault injection module. In fact, the watchpoint configuration messages are already used for debugging purposes and it is only required to add support for messages to enable and disable the FI module and for the setting up of the address and data values for the actual fault injection. In this manner it should be very simple to add fault

injection support to any NEXUS compliant debugging system.

The fault model considered for this case derives from the most common fault scenarios for critical systems [7] and is defined as:

Fault Type	Bit-Flip
Fault Location	RAM memory space
Fault Trigger	Instruction Execution
Fault Duration	Transient

Table 1 – Fault Model

The fault type and duration were chosen to mirror the expected effects of radiation on the target system. The fault location is limited to RAM memory because it is in this area that the OCD-FI is most advantageous as fault injection can be performed in real-time. The fault trigger can be any instruction occurrence of the running application, covering the entire execution time.

A fault campaign defines a set of fault injection runs where in each case a specific fault location and trigger is selected. In each such fault operation the processor is reset and the application runs from the beginning. The FI module can be programmed prior to this or even in runtime. The actual faulty value to be written depends on the target memory value at the moment of the injection. To determine this value beforehand it is possible to either use the knowledge of the running application code or perform a prior faultless application execution up to the fault triggering instant and use the OCD to read the relevant memory contents. The fault trigger is also selected beforehand from the executed application code. A complete fault campaign is generated externally and stored in memory as a sequence of commands for the OCD-FI infrastructure. Fault campaign management is performed using specific VHDL modules that use RAM memory blocks for reading the fault injection campaign data and storing the relevant results. This allows the use of a single programmable device (FPGA) for the entire fault injection process.

6. Results

The target system, the debugger, the fault injection module and the different memories were designed as VHDL models using the Xilinx ISE 7.1i [8] development environment and simulated using the Modelsim 6.0a simulation engine. The number of equivalent gate count for each module is given in Table 2 for two different CPU configurations.

Fault campaigns were executed on both configurations and using both target applications. Each campaign was performed twice, one using the initial OCD version and other using also the FI module. Table 3 presents the timing for each fault injection operation in clock cycles. Set up represents the delay due to fault injection related activities performed before each fault injection run is started and writing represents the time interval between the fault activation condition being met and the actual insertion of the faulty value.

Module	8 bit CPU		32 bit CPU	
	# Equivalent Gates	%	# Equivalent Gates	%
CPU core	9166	N/A	53717	N/A
RCT	2391	34	5113	27
RW	369	5	643	3
MMQ	4225	60	13045	69
FI	75	1,1	75	0,4
OCD-FI	**7060**	**100**	**18876**	**100**
Debugger (except RAM)	766	N/A	1079	N/A

Table 2 – Synthesis Results

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

Table 3 – Fault Injection Timings

The obtained results show that the FI module allows the injection of bit-flip faults with minimum delay and requiring a very low logic overhead. As such, the proposed OCD-FI infrastructure should provide an efficient fault injection mechanism in terms of reusability, coverage, performance and cost. The downside is the need of an adequate OCD infrastructure and the required availability of both the OCD and the target CPU in some type of HDL description.

Actually, different applications and fault injection scenarios are being used to further validate the OCD-FI infrastructure. Its applicability to other microprocessor and OCD architectures is also being studied.

References

[1] Ghani A. Kanawati et al, “FERRARI: A Flexible Software-Based Fault and Error Injection System”, IEEE transactions on computers 44, 1995.

[2] IEEE-ISTO 5001 [www.nexus5001.org], “The Nexus 5001 Forum Standard for a Global Embedded Processor Interface version 2.0”, 2003.

[3] Pedro Yuste, David de Andrés, Lenin Lemus, Juan J. Serrano, Pedro J. Gil; “INERTE: Integrated NExus-Based Real-Time Fault Injection Tool for Embedded Systems”; DSN 2003

[4] www.isystem.com/Products/Emulators/iC3000/

[5] www.opencores.org

[6] Giovanni Ferrante, “CPUGEN 2.00”, 2003

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

[8] www.xilinx.com