# Exploiting the IEEE 1149.1 Standard for Software Reliability Evaluation in Space Applications

M. Zenha-Rela
*University of Coimbra, Coimbra, Portugal*

J.C. Cunha & L.E. Santos & M. Gameiro & P. Gonçalves
*Instituto Superior de Engenharia de Coimbra, Coimbra, Portugal*

G. Alves & A. Fidalgo & P. Fortuna
*Instituto Superior de Engenharia do Porto, Porto, Portugal*

R. Maia & L. Henriques & D. Costa
*Critical Software S.A., Coimbra, Portugal*

ABSTRACT: The IEEE 1149.1 standard (boundary-scan) was originally developed as a technology to provide in-circuit testing of digital devices. Its effectiveness lead to unanticipated successes such as its extension to support on-line monitoring and in-circuit emulation. Meanwhile, its applicability for fault-injection had already been demonstrated by academic prototypes. In this paper we describe the first commercial tool, the BSCAN4FI plug in for Xception®, that provides support for software reliability evaluation for aeronautics and space applications using the boundary-scan technology as a means for controlled fault-injection. This tool allows transparent integration testing without any modification to the original system to be deployed and was developed specifically for the SPARC V.7 TSC695f space processor. Besides extended fault models and test features only made possible through this technology, in-system non-intrusive monitoring capabilities are also made possible.

## 1 INTRODUCTION

For many years fault-injection has been regarded as a major player in dependability assurance. It is commonly used for deriving coverage probabilities, assessing the effectiveness and validating fault and error handling mechanisms. Fault-injection addresses a topic that can't be adequately handled by the more traditional testing techniques, because rather than focus on the correct functioning of systems, it addresses the point of anticipating anomalous behavior under unexpected circumstances.

Many fault-injectors have been developed and described in the literature, with very different approaches on how to disturb the target system. Physical (Arlat et al. 1989, Karlsson et al. 1994), software (Carreira et al. 1995) and simulated (Jenn et al. 1994) fault-injection methodologies have been devised for this purpose. A comprehensive description of fault-injection methods and tools is given in Hsueh (1997).

The point is that different approaches to fault-injection can lead to different results, since the different system abstractions considered are seldom comparable (Karlsson et al. 1995). Thus different tools provide complementary rather than alternative approaches. For example, while it is true that software implemented fault-injection SWIFI provides a level of control and efficiency hardly achievable by other techniques, it is also true that pin-level fault-injection may induce situations impossible to emulate through software (e.g. during a system restart) or when the use of SWIFI is cumbersome or simply can't be applied (e.g. in communication ASICS).

Nevertheless, pin-level fault-injection has been almost abandoned due to several reasons: its target dependency prevents tool reusability, the technical constraints posed by the back-driving effects may lead to target damage (burn-out) and the increasingly high working frequencies makes a controlled disturbance a technically complex goal. So, despite the perception that, at the physical level no other technique can provide the same insight on unpredictable failure modes and anomalous system behavior, pin-level fault-injection is restricted to a niche in automotive electronics and some chip-makers of proprietary communication devices (e.g. Cisco Systems).

In this paper we present an extension to the capabilities of the Xception fault-injection tool by using the IEEE Standard 1149.1 (Boundary-Scan), to cover the low-level abstraction layers in an integrated fault-injection framework. This technology allows extended hardware fault-injection support as well as emulation of software errors.

The paper is organized as follows: in the next section we present an overview of IEEE 1149.1 and a critical analysis on the use of boundary-scan as a support for fault-injection. In Section 3 we present the BSCAN4FI plug-in and how it fully exploits the potentials of using boundary-scan for fault-injection purposes. In the section that follows a case-study il-

lustrating the use of such plug-in is presented. Section 5 concludes the paper with an overview of the tool and the next steps that will be followed.

## 2 USING IEEE 1149.1 FOR FAULT-INJECTION

The Boundary-Scan technology became a standard in 1990. Since then it received three revisions, being the last one made in 2001 and known as IEEE Std 1149.1-2001—*Standard Test Access Port and Boundary-Scan Architecture,* IEEE (2001). The driving force behind this standard were two continuing trends having significant adverse impact on the cost of testing printed circuit boards in the 80's: increasing complexity and greater miniaturization. Complexity makes functional testing longer due to the need to propagate test data through the complex ICs while applying tests to other chips on the board. Miniaturization limits physical probing, a method through which structural tests were performed.

To overcome these constraints IEEE 1149.1 provides a logical test-access mechanism into the ICs themselves. The boundary-scan path consists of a series of Boundary Scan Cells (BSC) added at every digital I/O pin on a component. The resulting Boundary Scan Register and other test features are accessed through a standard JTAG (*Joint Test Action Group*) interface — the TAP (*Test Access Port*). Through it, test instructions and test data are shifted in, and test data results are shifted-out. Boundary scan cells allow to control, observe, or take both actions on system signals.

Figure 1 presents the overall test logic architecture. As shown, the boundary-scan register is just one of the test data registers present. All of them, as well as the Instruction Register (IR), share the same generic structure: a shift-register matched with an optional latch. The entire logic is synchronously accessed through the four mandatory pins of the TAP: TMS (mode select), TCK (clock), TDI (data-in) and TDO (data-out).

The first two pins drive the TAP controller — a state machine that defines the register type (data/instruction) present between the last two pins (TDI, TDO) and the sequence of operations performed on it: *capture*, *shift* and *update*. *Capture* loads the shift

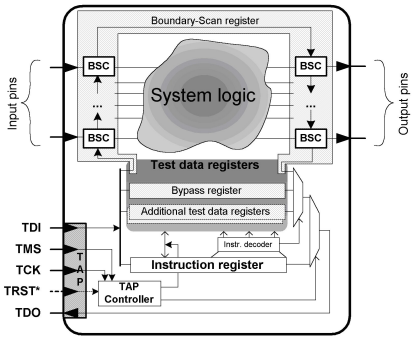



Figure 1. The boundary scan architecture.

register with data present on its parallel inputs. The *Shift* operation allows the captured data to be examined on TDO and new input data to be entered on TDI. Update makes the new data to become active, i.e. loaded on the latch and present on the parallel outputs of the selected register. The optional pin TRST may be used to asynchronously reset the test logic.

Through this infrastructure we can access and modify the boundary-scan register. Thus we have a direct access to the IC pins. However, in many modern ICs like complex ASIC, DSP and microprocessors, most operations are performed inside the chip. This fact, sustained by advanced techniques such as internal memories, prefetching and speculative execution, reduces the effectiveness of acting only on the boundary of such circuits. At the same time such complex ICs are actually the more common targets in need of fault-injection activity. Fortunately, several vendors enrich the complex ICs they produce with an extensive set of additional test data registers integrated into the IEEE 1149.1 infrastructure. According to the standard, these registers may be provided to allow access to any test-support features embedded in the design. These features might include scan-test, self-test registers, test data registers that cover the cache, the general purpose registers file (including the floating-point unit), the status and debugging registers and even access to the pipeline registers. Thus, the boundary-scan infrastructure provides a means to access not only the IC pins, but also internal structures, some of which are not accessible even through the target instruction set. So, while the use of boundary-scan to perform fault-injection was suggested since the standard's inception (Sedmak et al. 1994), this extended reachability disclosed its full potential.

In the next section we present the BSCAN4FI project which was inspired by the work on GOOFI presented in (Folkesson et al. 1998, Folkesson 1999, Aidemark et al. 2001).

## 3 THE BSCAN4FI APPROACH

This project extends the pure SWIFI approach in which Xception was originally based by providing support for boundary-scan technology. The TSC695F single chip implementation of the SPARC v7 compatible ERC32 open-source processor was chosen. Its development was supported by the European Space Agency (ESA) being currently used in Europe, China and Brazil for aeronautical and space related applications.

We shall now describe how the BSCAN4FI plug-in supports a typical fault-injection campaign. The first step is to define the trigger conditions, which may be *temporal* (e.g. time since an event), *spatial* (e.g. access to a given memory address) or *event-driven* conditions (e.g. interruption). Next, the fault must be

injected. Its *locality* (e.g. which state element(s)), *dynamics* (for how long) and *type* (e.g. *bit-flip*) fully characterize the process. Finally, its impact must be observed. This step can also be triggered by *temporal* (experiment timeout), *spatial* or *event-driven* (e.g error detected) conditions.

### 3.1 *Fault-Trigger*

Let's assume that the trigger condition for a given fault-injection experiment is the presence of a specific system input combination (*spatial trigger*). To handle this requirement we use the chip's internal OCD (on-chip debugging) mechanisms by programming the breakpoint registers available through the JTAG interface. This is actually the same OCD mechanism originally proposed for fault-injection back in 1995 by the Xception tool (Carreira et al. 1995).

#### 3.1.1 *Injecting the fault*

When the breakpoint condition occurs, execution of the workload stops and fault-injection takes place. This is performed by reading the relevant scan-chain, flipping the bits stated in the fault model and writing back the modified chain into the target. Fault-injection may also be accomplished by resident code in the target. Compared with SWIFI, the use of the IEEE 1149.1 infrastructure brings two significant advantages: it allows to disturb locations inaccessible to the software and avoids changes on the workload. This opens new possibilities e.g. in the aerospace applications where '*test what you fly/fly what you test*' is a mandatory requirement. Using such approach we can make final system testing without the need to leave dead, potentially dangerous, code in the application. However, an important drawback is also present: the temporal intrusiveness imposed is quite high and doesn't scale with target speed improvements.

BSCAN4FI presents a subtle difference in relation to GOOFI: we consider digital lines as fault targets while GOOFI faults only state elements with transient bit-flips only. With this later approach single-step operation is unnecessary after the injection. Nevertheless, single stepping, a standard OCD feature, is actually used by the both approaches to study in detail the error propagation. After a fault is injected the system operation is resumed by writing to the appropriate chain.

### 3.2 *Observation*

Monitoring the system behavior after the fault-injection is a trivial process easily achieved by sampling the target through the different scan chains. If specific instants/locations are of interest a procedure similar to triggering the fault can be used. The Xception default infrastructure supports collecting the system output to a database to be analyzed off line. In this phase the BSCAN4FI plug-in goes beyond the more traditional approaches, providing access to the target even if it hangs.

### 3.3 *Target Dependencies*

Although accessing the target system through the IEEE 1149.1 standard interface, the boundary-scan based fault-injection poses some portability constraints. These constraints have been thoroughly discussed in (Santos et al. 2003). First, trigger support and the test/system logic synchronization challenges are not solved directly through IEEE 1149.1 standard mechanisms, but rather by the IEEE 1149.1 based OCD support. This scheme avoids the most problematic constraints of the boundary-scan infrastructure for fault-injection purposes but are only available on processor-based targets. Second, even being a processor, it is also needed that the OCD mechanism provided is accessible through the IEEE 1149.1 (JTAG) interface. This is actually not very restrictive as many vendors like ARM, IBM, NEC, Motorola and Intel adopt JTAG based OCD strategies in most products. However, despite using a standard access interface, the OCD approaches are considerably distinct. Another fault-injection tool, FlexFI (Benso et al. 1999), also explores the OCD mechanisms of the Motorola MC68332 to perform fault-injection. However it uses BDM, a scheme that imposes a proprietary serial debug interface developed by Motorola. Moreover, it relies on instruction-made breakpoints, rather than hardware breakpoints. As reported in their paper, this strategy and the half clock frequency imposed by the BDM mode, imposes a runtime slowdown by a factor of 70× to 90×.

Finally, by relying on built-in mechanisms, the fault-injection tool needs a way to be notified that the trigger condition has been reached and the target is frozen. There is no foreseen way in the standard that allows the tested component to notify the tester about some event. Two approaches can be taken: by using an additional TRACE signal pin, as was done in GOOFI, or by continuously polling the target state through the scan-chain. Although ingenious, the former approach violates the requirement of using only the IEEE 1149.1 standard interface. In practice, this is not a real problem, since the surveyed targets have usually extra status pins, including one reserved to target-driven notifications.

The later approach conforms to the standard, however it imposes a non-negligible latency from the occurrence of the trigger until the injector is notified. The BSCAN4FI plug-in is fully configurable so it is possible to select the preferred approach. Polling is the default.

Interestingly, the most problematic factor is a non-technical issue: the information about private

IEEE 1149.1 instructions and additional test data registers is proprietary. Usually, it is available only to emulator vendors under a Non Disclosure Agreement (NDA), nevertheless a practice that fully conforms with the IEEE 1149.1 standard.

## 4 CASE STUDY

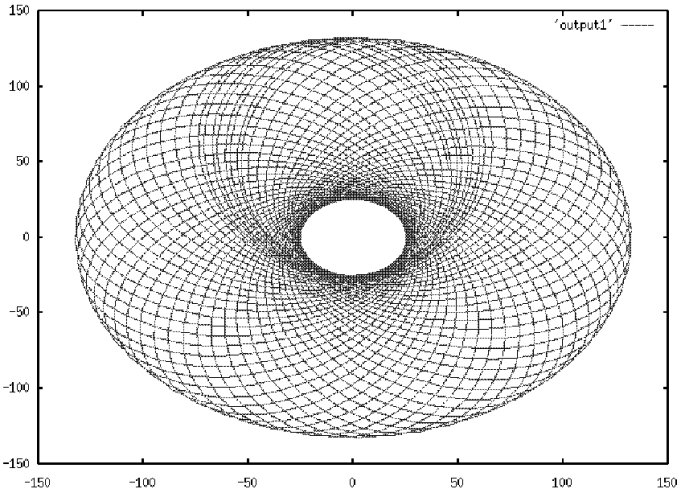
To evaluate the capabilities of the BSCAN4FI plug-in we shall now present some fault-injection experiments. The target hardware is an Atmel eVAB-695 evaluation board with a TSC695F processor, 512K boot flash memory, and 2 banks of SRAM (2Mbyte each).

The workload `gravity` is running on top of the RTEMS 4.5 real time kernel (RTEMS 2005). `gravity` is an application that calculates the trajectory of a mass (e.g. a satellite) attracted by a bigger mass (e.g. a planet) using Newton's gravity law. The workload outputs the successive satellite positions, in x-y coordinates. The faults target specific application variables such that its impact can be visualized by disturbances in the orbit performed by the low-mass object. The utility `gnuplot` generates a graphical representation of the results (Fig. 2(a)).

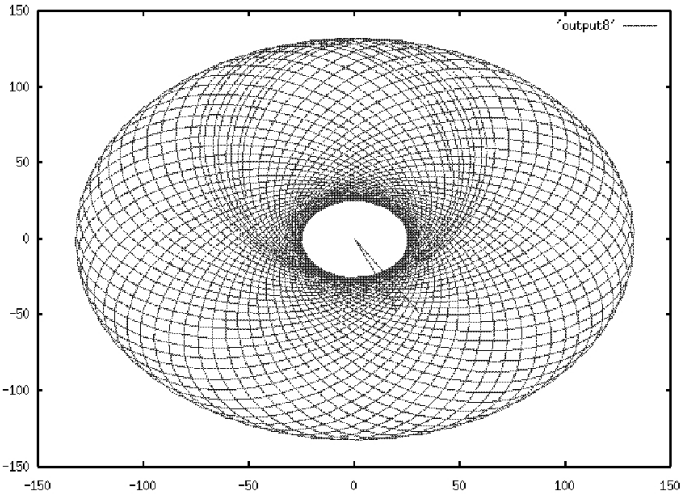
### 4.1 *Manually set Faults*

The goal of the first experiments is to show the accuracy of the tool. The definition of faults to be injected are derived from `gravity`'s source code developed in the programming language C. These are somehow 'well behaved' faults, in that they do not affect the target kernel structures so the application keeps running despite the presence of these controlled disturbances (i.e. leading to pure data errors). The TSC695F processor has built-in parity protection for most state elements, namely the internal registers, so in this series of experiments the parity protection is circumvented by generating faults that flip an even number of bits. If a single bit-flip was performed using (e.g.) a SWIFI fault injector, such error would never be detected by this built-in capability, since a correct parity would be automatically generated. Now we shall present the effects of some faults that are injected in the application according to this model.

In the core of the trajectory calculation algorithm, the `s.Planet.sPosition.dY` variable holds the 'y' coordinate of points updated in each iteration. This value is loaded into register `%o0`, so a fault injected in this register leads to a transient error in this particular value. We set a spatial trigger on the relevant instruction address to corrupt the register `%o0` with an even bit-flip fault to be activated at the 200th occurrence. The faulted bits are chosen so that the error is easily observable.




(a) Correct output of `gravity`



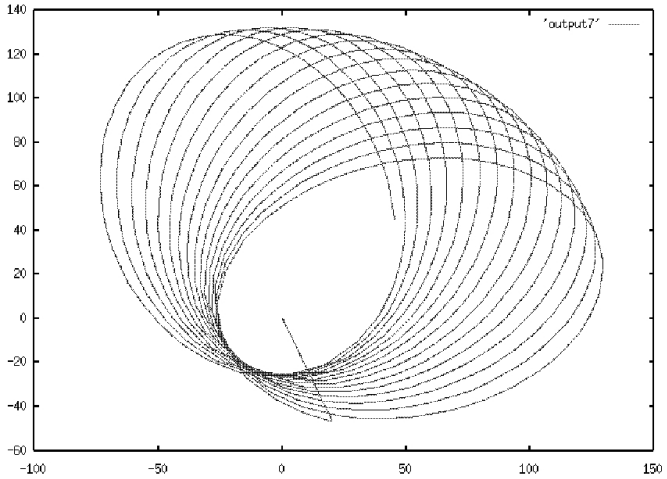

(b) Disturbance on `dY`

Figure 2. Effect of a transient fault in the output.

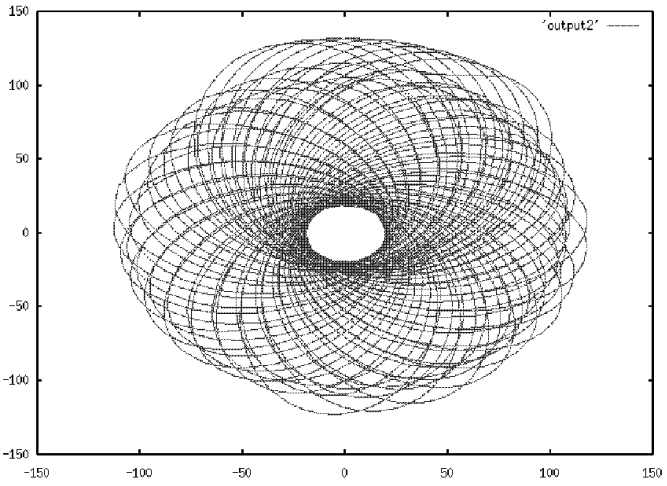
After this fault is injected we get the output depicted in Figure 2(b) where the presence of the disturbed point can be seen. Note that since no other value is affected, the computation proceeds normally afterwards.

Now, we shall perform a disturbance with stronger impact. The initial velocity value, loaded into register `%l4`, is set to `8.0` (default is `10.0`). The output failure can be observed in 3(a). In this fault the effect is not transient since due to the lower initial velocity the trajectory is unstable but the application exits normally.

As a final illustration of the tool accuracy the initial velocity is now set to 30.0. The effect of this new fault is depicted in figure 3(b). Again, the effect is not transient since due to the higher initial velocity the trajectory diverges and the application eventually aborts. Nevertheless the system exits normally as we could observe from the internal state collected through the scan chains.




(a) Initial velocity set to 8.0




(b) Initial velocity set to 30.0

Figure 3. Effect of different permanent errors in the output.

### 4.2 *Automatically Generated Faults*

The above experiments show that using the BSCAN4FI plug-in the Xception fault-injection environment provides the same accuracy offered by SWIFI. However, the BSCAN4FI plug-in can also perform automatically generated faults according to a given fault model. In Table 2 we present the fail-silent violations for automatically generated faults according to the setup listed in Table 1.

Table 1: Fault setup used on automatic generation of faults

| |
|---|
| 50% faults targeted at kernel. |
| 50% of faults targeted the application. |
| 50% of faults are single bit-flips. |
| 50% of faults are double bit-flips. |
| 1223 runs (approx. 6 hours). |

As we have seen, only a fraction of graceful terminations (the application exits normally) involve the generation of correct outputs. This represented a total of about 5.7% fail-silent violations, a figure which agrees with previous research. The remaining outcomes lead to wrong outputs and/or system crash.

These simple observations are only intended to show the nature of data that can be collected using the BSCAN4FI plug-in. However the Xception has a comprehensive suite of fault-injection analysis tools whose description is outside the scope of this paper.

Table 2. Experimental observations generated automatically.

| Code coverage | RTEMS | | `gravity` | |
|---|---|---|---|---|
| Faults targeted to | 605 | | 628 | |
| Effectively injected | 38% (127) | | 90% (568) | |
| | Single bit | Double bit | Single bit | Double bit |
| Effectively injected | 110 | 117 | 248 | 320 |
| Correct outcome | 26% | 40% | 19% | 48% |
| Graceful termination | 27% | 47% | 20% | 53% |

## 5 CONCLUSION AND FUTURE WORK

As a standalone technique, the boundary-scan infrastructure, as dictated by the IEEE 1149.1 standard, can't cope with the requirements of a fault-injection campaign. However, as demonstrated by previous works, using this standard to access the on-chip debugging (OCD) functions allows to achieve significant results. OCD brings support for the initial and the final phases of a fault-injection experiment. In fact, defining the injection instant is easily accomplished by programming the breakpoint registers through the JTAG interface. Additionally, while observing the fault impact, capabilities like single-stepping stresses the value of boundary-scan based OCD integration. Time intrusiveness, a major constraint of this approach, has to be traded against workload independence.

The main advantage of using the standard boundary-scan infrastructure derives from its ability to access all the desired abstraction levels present on the target, orthogonally to the purely functional aspects. Additionally, the high accuracy of this technique provides the necessary support to emulate software faults.

The next step in the BSCAN4FI project is to extend the boundary-scan interface to support the NEXUS standard as has been proposed by (Yuste et al. 2003). This technology standardizes the access to the OCD implementations, while providing IEEE 1149.1 compatibility at the basic levels (level 1). This promises to be a major step to minimize target dependency.

## 6 REFERENCES


Aidemark, J. et al. GOOFI: Generic Object-Oriented Fault Injection tool. *Proc. Intl. Symp. DSN-2001*. 83-93.

Arlat, J. et al. 1989. Fault injection for dependability validation of fault-tolerant computing systems. *Proc. Intl. Symp. FTCS-19*, Chicago-IL. 348-355.

Benso, A. 1999. FlexFI: A flexible fault injection environment for microprocessor-based systems. *Proc. Intl. Symp. SAFECOMP'99*, Toulouse-France, 323-335. Springer Verlag.

Carreira, J. et al. 1995. Xception: Software fault injection and monitoring in processor functional units. *In DCCA-5,* Urbana, Champaign-USA. 135-149.

Folkesson, P. et al. 1998. A comparison of simulation based and scan chain implemented fault injection. *Proc. Intl. Symp. FTCS-28:* 284-293.

Folkesson, P. 1999. *Assessment and Comparison of Physical Fault Injection Techniques.* Phd thesis, Chalmers University of Technology.

Hsueh, M.C. et al. 1997. Fault injection techniques and tools. *IEEE Computer*, 30(4):75-82.

IEEE Std 1149.1-2001. *IEEE Standard Test Access Port and Boundary-Scan Architecture.* New York: IEEE. 2001.

Jenn, E. et al. 1994. Fault injection into VHDL models: The MEFISTO tool. *Proc. Intl. Symp. FTCS-24*, Austin-USA. 66-75.

Karlsson, J. et al. 1994. Using heavy-ion radiation to validate fault-handling mechanisms. *IEEE Micro*, 14(1):8-11, 13-23.

Karlsson, J. 1995. Application of three physical fault injection techniques to the experimental assessment of the MARS architecture. *IFIP Working Conf. on Dependable Computing for Critical Applications*. 150-161.

RTEMS - Real-Time Executive for Multiprocessor Systems, http://www.rtems.org

Santos, L. et al. 2003. Constraints on the Use of boundary scan for fault injection, *Proc. Intl. Symp. LADC2003*, São Paulo-Brazil, LNCS 2847, 39-55. Springer-Verlag.

Sedmak, R. 1994. Boundary-scan: Beyond production test. *IEEE VLSI Test Symposium:* 415-420.

Yuste, P. et al. 2003. INERTE: Integrated Nexus-Based Real-Time Fault Injection Tool for Embedded Systems. *Proc. Intl. Symp. DSN'03.*