

Built-in Fault Injectors – The Logical Continuation of BIST?

Andreas Steininger, Babak Rahbaran, Thomas Handl

Embedded Computing Systems Group,
Vienna University of Technology, Vienna, Austria
{steininger,rahbaran,handl}@ecs.tuwien.ac.at

Abstract — *With the increasing number of embedded computer systems being used in safety critical applications the testing and assessment of a system's fault tolerance properties become a crucial issue. Experimental fault injection plays an important role in the process of fault tolerance assessment. The restricted accessibility of nodes within a chip or system-on-a-chip, however, make such experiments extremely difficult. The idea we want to promote in this paper is to add a fault injector on to the silicon that can perform fault injection automatically on demand during several stages of design. In a first step we discuss the usefulness of such an infrastructure IP module. Then, starting from an FPGA-based fault-injection toolset that we have already developed, we investigate the feasibility of designing such a module under the given tight economic constraints.*

1 Introduction

Fault-tolerant computer systems are becoming more and more popular, as computers are being employed in a growing number of safety-critical applications. For the designer of a fault-tolerant system it is not sufficient to prove the proper function of the system during normal (i.e. fault-free) operating conditions, the system's fault-tolerance property must be assessed and proven as well. While testing the "normal" functional integrity of a (potentially complex and distributed) system is already an extremely challenging task, the fault-tolerance assessment adds another dimension of complexity: All feasible modes of system operation must be investigated under all imaginable (or anticipated) fault conditions.

Most often modelling approaches are used to assess the fault-tolerance behavior of a system. These models, however, can only describe the system behavior on a relatively high level of abstraction. No approach has been found so far that allows to model the complex interaction between "real-world" physical faults and the computer hardware in a general manner. So physical fault-injection experiments have become the method of choice for getting the input parameters required for the high-level models. Section 2 will briefly present the related techniques. Independent from the particular technique that is finally employed the degree of accessibility of locations within the circuit is of central importance, especially in context with complex chip architectures such as systems on a chip:

Register values and/or signal levels must be manipulated in a well controlled way and the respective system reaction be observed with a high level of detail. These requirements will be discussed in Section 3, and the advantages of on-chip logic dedicated to this purpose will be outlined. In analogy to the built-in self test we will consequently propose to integrate dedicated fault-injection logic into a chip. Potential reuse during various phases of the system life cycle will be identified. In this sense the proposed on-chip logic can be regarded as an infrastructure IP-module that does not add to the functionality of the chip from the application point of view, but serves an important purpose during manufacturing and/or system integration [1] such that the overhead incurred by it will eventually amortize. Section 4 will introduce FIDYCO, a fault-injection framework we have developed for dependability evaluation. In its current state FIDYCO is targeted for an FPGA platform and used during the development stage. In this paper we will analyze what needs to be done to make FIDYCO applicable as an on-chip module for custom chips as well. In particular we will try to find a reasonable tradeoff between cost and flexibility. Finally Section 5 will draw the conclusion and give an outlook on future work.

2 Fault Injection

2.1 Objectives of Fault Injection

In general fault injection can be viewed as a kind of "time lapse" for the occurrence of faults (*fault acceleration*). With respect to the fault model, however, two fundamental aims of fault-injection experiments can be distinguished:

- **Fault removal:** As already mentioned the test of a fault-tolerant system not only covers its functional properties alone, but must also include its ability to detect, handle and tolerate faults. The easiest way to do this is to inject faults that are intended to be covered by a fault tolerance mechanism and observe whether this fault tolerance mechanisms performs properly. A so called white box model of the target is employed and usually a relatively small number of well chosen faults are injected. One important requirement on the experimental tool-set is the ability to inject faults in a well-controlled and reproducible manner (ideally) at any desired location. In order to be able to identify the appropriate corrective actions it is furthermore necessary to have a good diagnostic resolution.
- **Fault forecasting:** The aim of fault forecasting is to estimate reliability figures for the operation of the target system in the field. Ideally the system is exposed to the same fault scenario as anticipated in the field - though accelerated - and statistical ratings are derived for proportion of covered faults and detection latency. The system behavior observed throughout the experiments is then projected to the anticipated behavior during field operation. Obviously the key point here is to create a fault scenario that matches the anticipated field conditions as closely as possible, otherwise the estimation will be biased. Among other things this requires that the means for fault injection and data collection employed during the experiments should not change the actual target in any way (i.e. they should be *non-intrusive*). Moreover, the set of faults to be injected should be representative for the field environment. Therefore the experimental tool-set must be flexible and powerful enough to em-

ulate a (potentially) large variety of faults. Since *statistical* measures are derived, a large number of faults must be injected to attain a reasonably high confidence in the results. This translates into the demand for an automated tool-set that is able to inject faults in sequence rapidly.

Both objectives require an understanding of the effects of real faults and thus of the related behavior of the target system that shall be attained by the experiments. And in both cases the efficiency of the fault tolerance mechanisms is being evaluated, with the main difference being that fault removal provides a *qualitative* feedback about potential design flaws while fault forecasting produces *quantitative* results that can be used as input e.g. for dependability models.

2.2 Basic Fault-Injection Techniques

Within the numerous approaches that have been proposed throughout many years of research in this field three main classes can be distinguished:

2.2.1 Hardware-implemented fault injection

In hardware-implemented fault injection a physical (prototype) system is subjected to physical faults. Examples are applying noise to the power supply, artificially creating EMI surges or submitting a portion of the chip area to heavy ion radiation or a laser beam. Furthermore, external drivers can be used to override ("*force*") the state of a signal line, and additional logic gates can be inserted to manipulate the logic value of a signal ("*insertion*"). Another approach is to freeze the system operation at some point and introduce modified flip-flop and register contents over the scan chain. Examples for hardware fault-injection tools are *RIFLE* [2] and *FOCUS* [3].

The use of a physical prototype yields a perfect representation of the target system (including implementation faults), which yields a good representation of "real-world faults". Most hardware fault-injection methods are non-intrusive (from a macroscopic point of view at least) and allow the experiments to be performed in real time, which facilitates collecting a large number of results within a reasonable period.

Forcing and insertion are quite easy to apply and therefore often used, but they are confined to the signal traces on the printed circuit board, which limits the applicable fault set and the diagnostic resolution.

Another drawback is that the tools employed for hardware fault injection are most often very sophisticated and problem specific (with respect to target system, fault model etc.), hence portability is very poor. In addition, it is comparatively difficult to automate hardware fault injection.

2.2.2 Software-implemented fault injection (SWIFI)

In software-implemented fault injection memory and register contents on a physical prototype system are manipulated to inject a fault. This can be done either before system operation is started (*off-line injection*) causing the system to operate on these incorrect data and/or instructions at runtime or during system operation by means of interrupts or traps (*runtime injection*) that alter register or memory contents when activated during runtime.

Among others the following tools have been already developed: *FIAT* [4], *XCEPTION* [5], *DOCTOR* [6].

In contrast to hardware-implemented fault injection SWIFI does not require any sophisticated, problem-specific hardware for fault injection and is hence quite cheap easy to apply. It is also very easy to automate SWIFI.

According to [7] the main drawback of SWIFI is its restriction to memory and register contents accessible by software for injecting faults. A mapping of "real world faults" that are typically assumed to occur on a low level of abstraction to these high level faults is extremely tedious [8]. Moreover, runtime SWIFI tends to cause an interference with the temporal behavior of the system, which rules out its use in real-time environments.

2.2.3 Simulation-based fault injection

In simulated fault injection a simulation model of the system is used instead of a physical prototype and faults are introduced by means of changing the model. While this method provides the ultimate flexibility with respect to the types of faults that can be injected, it has some significant drawbacks: Since the simulation tends to cause considerable computational efforts, one second of real time may well take several hours of simulation time. Therefore the method cannot be applied to complex systems and it cannot be performed in a real-time environment. Moreover, it is difficult to map real-world faults to the respective changes of the simulation model. The following tools have been already developed: *VERIFY* [9]; *MEFISTO-C* [10].

2.2.4 Hybrid Fault Injection

A hybrid approach combines two or more of the above fault-injection techniques with the intention to maximize the benefits while minimizing the drawbacks. The mixed-mode fault-injection technique proposed in [11], e.g., allows the advantages of both SWIFI and simulation based fault injection to be utilized, i.e. the actual target system may be executed at full speed except during the injection of a fault when a simulator providing detailed access to the target system is used instead.

The LIVE tool proposed in [12] integrates fault injection and software testing techniques to achieve an accurate and non-intrusive analysis of a system prototype. It uses pin-level forcing or generates interrupts to activate software fault injection procedures.

Since it is comparatively easy to apply (even in a real-time application), software-implemented fault injection has gained particular interest. However, recent investigations [7] have led to the result, that this method needs to be complemented by hardware-implemented fault injection, e.g., to achieve complete and convincing results. As already mentioned above the main problems with hardware-implemented fault injection are that (a) it often requires a highly specialized and expensive experimental setup, and that (b) the accessibility of chip-internal circuit elements and nodes is very limited. Good accessibility is mandatory to perform well controlled and reproducible experiments, which, in turn, is a prerequisite for attaining high diagnostic resolution.

3 The Need for On-chip Fault-Injection Support

As already mentioned the quality of a fault-injection setup is largely determined by the accessibility of the locations (nodes and registers e.g.) in the system or on the chip, respectively. There are two important reasons for this:

1. In context with fault forecasting it is crucial for the representativeness of the fault-injection experiments to use a fault model that matches the anticipated field conditions as closely as possible. This means that all parameters of an injected fault must be chosen and varied strictly according to the fault model and that the faults must hence be injected in a well controlled manner. In context with fault removal it is furthermore desirable to align fault injection with certain (e.g. worst case) conditions, which requires some sort of trigger mechanism. Well-controlled triggered fault injection is also mandatory, if reproducibility of the experiments is desired (for certification or to verify design improvements).
2. In order to assess the system's reaction to a fault injection it is necessary to trace its behavior. For debugging complex chips it is usually not sufficient to observe the signal values on the pins, a more detailed look into internal signal and register values is often required to attain the desired level of diagnostic resolution and to judge on the existence of residual dormant faults at the end of the experiment.

As outlined in the previous section hardware-implemented fault injection has several very attractive properties. At the same time accessibility is very limited for this technique. Therefore it appears reasonable to spend some additional hardware elements on the chip for the purpose of fault injection: Placing "fault injectors" to well chosen locations in the circuit provides easy access and keeps the experimental setup simple. While it is extremely difficult and tedious to control and observe internal signals and register values from outside – and in fact a very target specific experimental setup is required – an on-chip fault-injection controller has direct access to any desired location. A generic IP module can be designed that only needs some minor tuning (number of lines to be observed/controlled, trigger conditions etc.) to be customized to a particular target chip. The principle is very similar to the built-in self-test (BIST) approach, and in fact it solves a similar problem. Even the argument of reusability holds for built-in fault injection as well: In the course of fault-removal error detection mechanisms have to be checked on the chip after fabrication. Later on, during system design and system integration the fault tolerance properties have to be assessed and verified on the system level preferably by means of fault injection. And finally it is well known that an in-depth test of a fault-tolerant system is mandatory upon start-up (or sometimes even during mission). The coverage of this test is crucial to the dependability. Even for maintenance and repair a built-in fault-injector can do a good job. During all of these phases a built-in logic supporting the fault injection can be extremely valuable, especially if the experiment procedure is automated. Depending on the needs of the particular phase SWIFI can be employed as a completion.

From an economic point of view the question still needs to be answered, whether these benefits justify the overheads involved. Therefore we will investigate the overheads of a potential implementation of our approach in the following.

4 The Fault-Injection Toolset FIDYCO

4.1 Current Status

We have already developed a toolset called FIDYCO (*Flexible on-chip fault Injector for run-time Dependability validation with target specific COmmand language*, [13]): FIDYCO is originally targeted for use in FPGA-based platforms. In some sense FIDYCO is a combination of hardware-implemented and simulation-based fault injection (a so-called "hybrid" approach).

FIDYCO comprises two main parts, namely a hardware part written in VHDL and a software part running on a host PC. The software is responsible for experiment configuration, FPGA download, experiment control through direct user intervention (optional), readout data collection, processing and visualization. The FIDYCO hardware is downloaded to an FPGA together with the target design that shall be exposed to fault injection (*device under test*) and optionally a second, identical target design that serves as a reference (*golden node*). Figure 1 shows the basic setup.

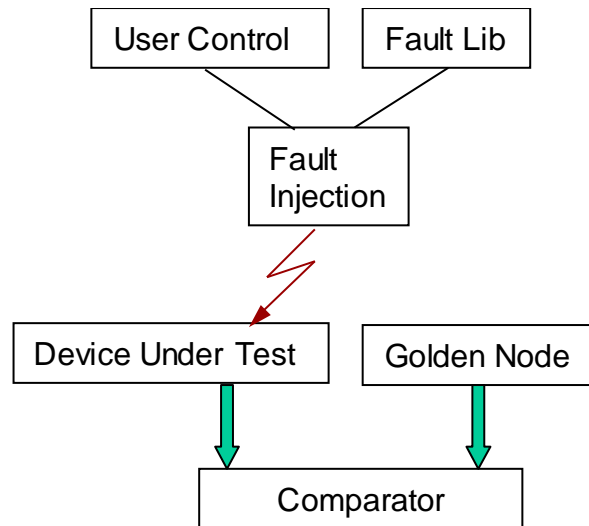


Figure 1: Fault Injection

4.1.1 Structure of FIDYCO

Having fault injector, device under test and golden node on the same chip considerably extends the applicable fault models. The fault injector being immediately connected to the target allows to react extremely fast to trigger events and apply fault-injection methods like insertion of logic gates to force signal values without the risk of causing damage to the circuit.

As the block diagram in figure 2 shows, the FIDYCO hardware is highly modular to allow for an easy adaptation to different targets. It is composed of three main blocks: the host interface(Interface RS232), the control block and the data collector.

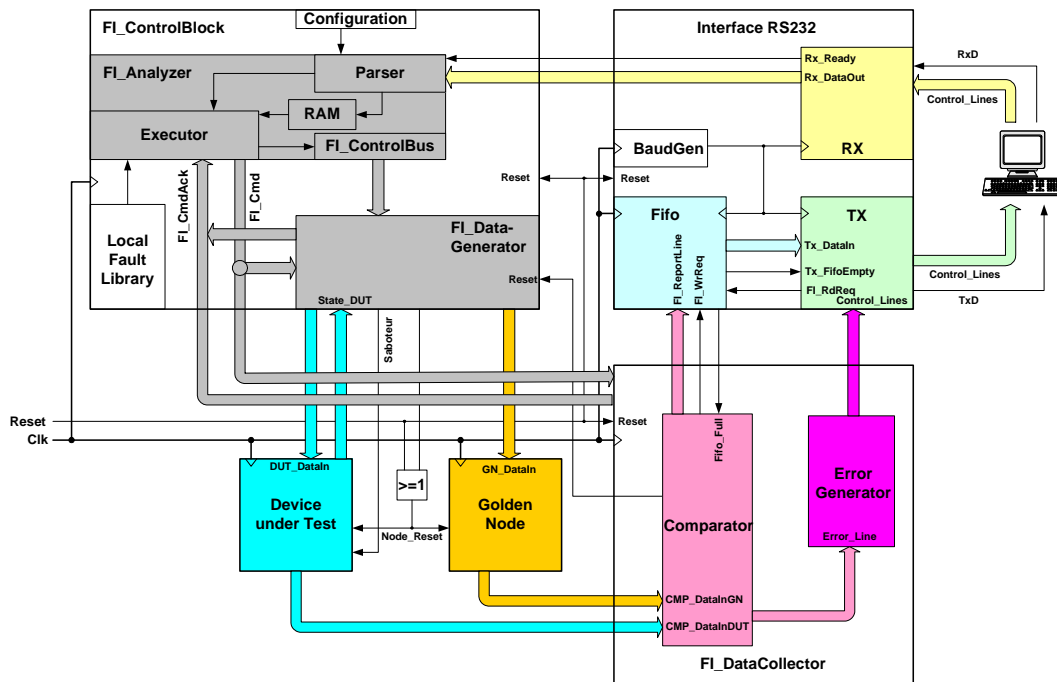


Figure 2: FIDYCO Block Diagram

The **host interface** is responsible for receiving setup and configuration information and direct fault-injection commands from the host PC. Furthermore it is used for sending the readouts to the host for further processing. The actual type of this interface is not critical, currently it is implemented as an RS232 interface.

The **control block** can be operated in two different modes:

- In the *interactive mode* the control block receives commands from the user via the host interface during runtime. A built-in parser analyzes the received commands and stores them in a RAM. After this phase of analysis control is handed over to the executor. This unit sequentially fetches commands from the RAM and forwards them to the data generator. The vector of stimuli provided by the data generator is finally applied to the device under test as soon as the trigger condition has been reached. To assess the reaction of the device under test a user definable set of signals can be compared to the corresponding reference signals of the golden node.
- In the *automatic mode* the executor sequentially fetches commands from a local fault library (instead of the RAM). The local fault library has been configured during the experiment setup phase, so no user intervention is required. Apart from this fact the rest of the experiment flow is the same as in the interactive mode. In addition the executor has to perform a kind of *Power On Self Test*.

The **data collector** preprocesses the experimental readouts (by performing a comparison between the corresponding signals on the device under test and the golden node, for instance) and sends the results to the host.

4.1.2 Experiment flow using FIDYCO

FIDYCO is configured and controlled by three text-files that provide information about the layout and functionality of the system: The *command file* allows the user to define specific commands which he can issue in the interactive mode. A preprocessor checks these commands before they are converted into a binary format that can be used in the FPGA. In addition FIDYCO has numerous built-in commands. Each of these built-in commands is assigned a unique reference number in the *look-up file*. Finally, all constants relevant for the parameterization and layout of the system are contained in the *configuration file*.

From these files a compiler generates the information required for the synthesis of FIDYCO. Together with the device under test and the golden node FIDYCO is then downloaded to the FPGA platform. There the experiments can be conducted, either using the local fault library (automatic mode) or based on user commands (interactive mode).

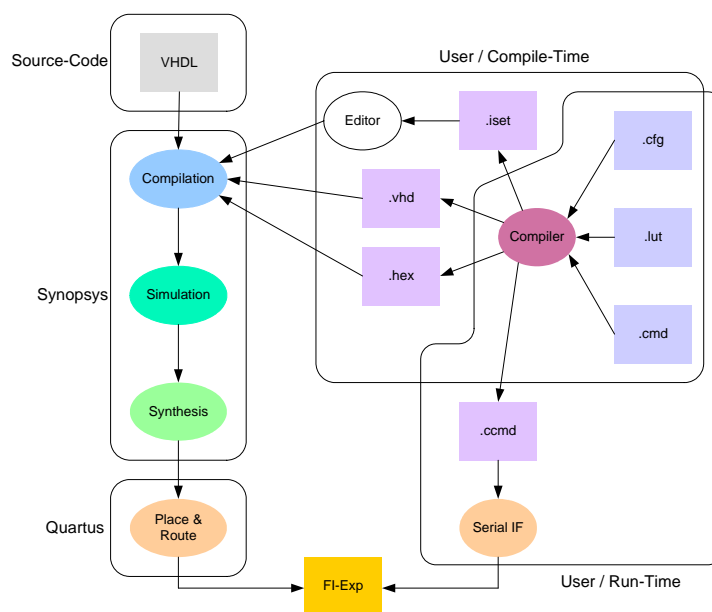


Figure 3: Experiment Flow

The chosen approach combines the advantages of both hardware implemented fault injection and simulation based fault injection: Physical faults are injected into a physical prototype (that is, however, still a model of the final VLSI chip). The programmable logic of the FPGA allows direct access to virtually any point in the chip, thus providing easy control and observability. The experiments can be performed in real-time, which allows to inject a high number of faults in a realistic real-world environment. The main drawback of the method is that the low-level hardware structure of the FPGA prototype implementation is different from the final product.

4.2 Required Modifications

One interesting feature we want to stress here is that in the automatic mode the hardware portion of FIDYCO is able to perform fault injection autonomously and thus requires

only a minimal host interface for control. By sacrificing the flexibility of FIDYCO and tailoring it to the specific needs of a particular custom chip it should be possible to reduce the overheads incurred by the fault-injector to a minimum and thus make its integration economically attractive. Our vision is to have a fault-injection logic available in the shape of an "infrastructure IP module" [1] that can easily be adapted to any target design.

The following restrictions of FIDYCO are conceivable:

- We do not support the interactive mode, therefore we do not need the parser. In the current implementation of FIDYCO this is the largest submodule.
- We use a pre-configured module, hence we do not need a host for configuration. We will still need, however, some controller to start the automatic fault injection.
- Usually we cannot afford duplication of the design, hence we do not use a golden node, but a golden run instead. This means that we do not have a "live" reference, but we have to know the correct behavior of our target system in advance. This reference behavior can either be derived from the specification or from an observation during a run without fault injection (golden run).
- If we perform the check of correct system operation on a higher system level (e.g. check whether the application still produces correct output) rather than observing the correct function of the chip or node, we do not need the comparator and the result analyzer. This implies a tradeoff with respect to the diagnostic resolution, however.
- We can skip the complete communication interface, the only thing that needs to remain is some means to start the fault injector. This could potentially be done by means of the Test Access Port (TAP) [14] that is available in many designs anyway. The communication interface is the second largest submodule in the current design.

What finally remains is the following: A Fault library that is restricted to a selection of most relevant fault types, a data generator that transforms the entries of the fault library into stimuli for the device under test, and finally the fault-injection controller, again with a reduced functionality. The proposed savings clearly compromise the flexibility and versatility of FIDYCO, the applicable fault set and the diagnostic resolution. By careful tuning of the fault injector, however, the implementation overheads can be minimized while the functionality desired in different phases of the system design and operation can be ensured.

5 Conclusion

We have substantiated the need for fault-injection experiments in general and hardware-implemented fault injection in particular in the process of fault tolerance assessment and pointed out the problems related to the limited accessibility of chip-internal circuit nodes in context with hardware-implemented fault injection. This observation has led us to propose the integration of dedicated fault-injection logic on to a chip to support experimental dependability assessment. We have pointed out the benefits of such an approach. In order to improve its economic feasibility we have tried to reduce our existing fault-injection tool-set FIDYCO to the absolute minimum and have found a high potential for savings, if we are willing to sacrifice flexibility and diagnostic resolution.

Since BIST has evolved to one of the most important test methods currently in use, we are convinced of the high potential in using the built-in fault-injectors analogously in systems that require dependability assessment through fault injection.

References

- [1] Y. Zorian. IP for SOC Infrastructure. *IEEE Design and Test*, 2002.
- [2] H. Madeira, M. Rela, F. Moreira, and J. G. Silva. *RIFLE: A General Purpose Pin-level Fault Injector Purpose Pin-level Fault Injector*”,. Springer-Verlag, 1994.
- [3] Choi, Gwan S., and Ravishankar K. Iyer. FOCUS: An Experimental Environment for Fault Sensitivity Analysis. *IEEE Transactions on Computers*, 41:pp. 1515–1526, December 1992.
- [4] D. Siewiorek Z. Segall, D. Vrsalovic. FIAT - Fault Injection Based Automated Testing Environment. *Symp. on Fault-Tolerant Computing (FTCS-18)*, pages 102–107, 1998.
- [5] J. V. Carreira, D. Costa, and J. G. Silva. Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 24(2), 1998.
- [6] S. Han, H. Rosenberg, and K. Shin. DOCTOR: an Integrated Software Fault Injection Environment. Technical Report CSE-TR-192-93, University of Michigan, 1993.
- [7] P. Gil , et al. Fault Representativeness. Technical report, Deliverable of the DBENCH project (EC), June 21 2002.
- [8] Joakim Ohlsson, Marcus Rimen, and Ulf Gunneflo. A Study of the Effects of Transient Fault Injection into a 32-bit RISC with Built-in Watchdog. *FTCS-22.*, pages 316 –325, 1992.
- [9] F. Balbach V. Sieh, O. Tschche. VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions Using VHDL-Models with Embedded Fault Descriptions””,. *Fault-Tolerant Computing (FTCS-27)*, pages 32–36, 1997.
- [10] E.Jenn, J.Arlat, M.Rimen, and J.Ohlsson. Fault Injection in to VHDL Models: The MEFISTO Tool. In *Proceedings of the 24th International Symposium on Fault-Tolerant Computing*, pages 66–75.
- [11] Guthoff, J., and V. Sieh. Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method. *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, pages 196–206, 1995.
- [12] A.Amendola, P.Marmo, and F.Poli. Experimental Evaluation of Computer-Based Railway Control Systems. *Symp. on Fault-Tolerant Computing (FTCS-27)*, pages 380–384, June 1997.
- [13] M. Delvai, W. Huber, B. Rahbaran, and A. Steininger. An FPGA-Based Development Platform for the Virtual Real-Time Processor Component Spear. In *Proceeding of IEEE Design and Diagnostics of Electronic circuit and Systems Workshop-DDECS*, pages 98–105. Brno, Czech Republic, 2002.
- [14] M. Colin Maunder and Rodham E. Tulloss. *The Test Access Port and Boundary Scan Architecture*. IEEE Computer Society Press, 1992.