System Dependability Analysis using VHDL Models with Integrated Fault Descriptions[1]

Volkmar Sieh, Oliver Tschäche, Frank Balbach

Institut für Mathematische Maschinen und Datenverarbeitung (IMMD) III,
Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, Germany
{vrsieh,ortschae,balbach}@informatik.uni-erlangen.de

Abstract - *When injecting faults in digital systems, most approaches measures only the probability of a fault leading to a failure. This paper addresses the necessity of considering also the fault rates to get reasonable results from the fault injection experiment. For solving this problem we developed a VHDL-based fault injection tool, which allows the correct fault description and which is able to maintain consistency between the hardware model and the fault model.*

Keywords - Fault injection, VHDL

1. Introduction

During the last decades fault injection has become an indispensable technique for evaluating dependable digital systems. A central question coming along with this technique is the adequate description of faults for the system under consideration. Whereas several approaches try to describe the systems and its faults at an abstract level, e.g by considering only completely failed processors or links in a massively parallel system [4][7], other approaches try to consider faults which are very close to the physical implementation of the digital system.

Fault injection at the physical level may either be done by stressing the hardware with environmental parameters, e.g. heavy-ion radiation [5], or using pin-level fault injectors [10]. As the current tendency of integrating more and more components (Cache, MMU, etc.) on-chip, the problem of an adequate fault description increases especially for the latter technique.

The second possibility of injecting faults in existing hardware is software-implemented fault injection, where a process' memory image and/or internal registers of the processor will be corrupted during run-time [1][6][8]. A basic assumption of these tools is, that it is sufficient to inject faults at the interfaces of non-accessible internal components (e.g. registers of an adder) for evaluation of its dependability features.

Simulation-based fault injection is the third group of fault injection techniques. It allows a comparison of design alternatives without the need of the physical hardware. For this purpose, a hardware model of the system will be developed and faults will be injected during simulation time [2][3]. In order to reduce the gap between the hardware model and the physical system, several research groups use a hardware description language [9][11].

All mentioned approaches for fault injection evaluate the probability of a failure of the system (or service) for the faults which has been injected. So far, the methods for describing the faults do not consider the fault rate, i.e. the mean time of occurrence for each fault. This paper addresses the need of considering both aspects to correctly evaluate the dependability features

1. This work is supported by the Deutsche Forschungsgemeinschaft as part of SFB 182 and Project number Da365/2-1

of a given digital system. In addition to this, we will show that the assumption of neglecting implementation details of the physical system by injecting faults only at abstract level may lead to completely wrong results regarding system's dependability. Given two implementations of the same function (e.g. an adder or a FIFO) the estimation of their reliability may produce complementary results when comparing them by only considering the probability of a fault leading to a failure or, by additionally taking into account the probability of the occurrences of the faults.

Therefore, we developed the fault injection tool VERIFY (**V**HDL-based **E**valuation of **R**eliability by **I**njecting **F**aults efficientl**Y**) which allows a correct description and injection of faults in digital systems together with an estimation of its probability of failing. In addition to this, VERIFY ensures the consistency between implementation details described by the hardware model and the fault model by using the hardware description language VHDL.

The rest of this paper is organized as follows. In the following Section 2, we will discuss more in detail the need of correct fault descriptions. In Section 3 the basic ideas of the VERIFY-tool will be described.

2. Correct Fault Descriptions

The motivation for setting up a fault injection experiment for a digital system is to get a deeper knowledge of the system's behavior in the presence of faults. For this purpose the designer is interested in measuring the system's failure rate or the probability of a system's failure in a given time interval. Therefore, fault injection experiments are used by designers of software and hardware to compare alternative implementations regarding the system's dependability (e.g. evaluating error detection mechanisms or software for system diagnosis). The failure rate of a system is defined as follows:

$$\mu = \sum_{f = f_1}^{f_N} \lambda_f p_f$$

where N is the number of different types of faults, λ_f is the rate of occurrence of fault *f*, and p_f is the probability of fault *f* leading to a failure in the system.

Given this definition, the probability of a failure of the system within time *T* is defined by

$$P = 1 - e^{-\mu T}$$

An assumption made by almost all fault injection tool is the equality of all λ_f. Therefore, experiments for fault injection are performed without specifying these parameters. For the faults, which will be injected, the probability of the fault leading to a failure of the system will be measured by the experiment. We will show with the following example, that ignoring the λ_f or the time factor when estimating the probability of a system failure within time *T* will lead to complementary results of the fault injection experiments.

In the upper part of Table 1, λ_f, p_f and the mission time *T* for a given service are defined for three systems, which have to be compared regarding their dependability. System A denotes the original system without any mechanisms of fault tolerance, system B is extended with fault tolerance at hardware-level and system C uses fault tolerance at software-level. Due to the mech-

anisms for fault tolerance added in system B and C, the probability of a fault leading to a failure (which is measured by fault injection) is lower than in the original system A. The additional hardware needed in system B increases the rate of faults because the number of gates increases. The additional software needed in system C (e.g. by using application-based-fault-tolerance) causes a time overhead and increases the mission time for a given service.

Table 1 Dependability evaluation of an example systems

	System A (original system)	System B (hardware overhead)	System C (time overhead)
prob. of fault leading to failure (p)	2.0 %	1.0 %	1.5 %
fault rate (λ)	4 / year	10 / year	4 / year
mission time (T)	0.5 years	0.5 years	1.0 years
Dependability Values			
coverage (C)	98%	99%	98.5%
failure rate (μ)	0.08 / year	0.10 / year	0.06 / year
prob. of mission failure (P)	3.9 %	4.9 %	5.8 %

In the lower part of the table, the results of the experiment are presented. When ignoring the fault rates for evaluating the results of a fault injection experiment, the coverage (C) of the system would be $C = 1 - p$ with p as the measured probability of a fault leading to a failure. Assuming the numbers given in the upper part of the table, system B would have the best coverage of all three systems. When taking into account the fault rates λ, system C with its software implemented fault tolerance would have the best failure rate of all three systems. When also looking at the probability of a mission failure, system A, i.e. the system without any additional mechanisms for fault tolerance should be chosen.

As it can be seen from this example, fault injection experiments have to consider the fault rate and the mission time for the service the system has to deliver, in order to be able to correctly evaluate the dependability of the system. It is not enough just to inject faults without the knowledge of the rate of occurrence and to measure the probability of the fault leading to a failure in the system.

3. Injecting Faults with VERIFY

As we have shown in the last section, evaluating the dependability of a digital system needs the consideration of the fault rates. Therefore, we developed the simulation-based fault injection tool VERIFY, which allows the correct fault description. By extending the widely used hardware description language VHDL [13] with so called fault injection signals (FIS), we also considered the problem of the consistency between the hardware model and the fault model. We already showed in our previous work [12], that it is also indispensable to describe the faults at a very detailed level of abstraction, i.e. the gate-level, in order to get reasonable results for the failure rate of a system. Therefore, it is not necessary to inject faults at higher levels than gate-level which reduces the need of additional work during hardware development. Once the hardware manufacturer described the faults for all gates in the cell-library he provides, the hard-

ware developers can develop the system as usual and as soon as a synthesis tool (e.g. SYNOPSIS^{TM}) converts the register transfer level description (RTL) to gate-level components (cells of the library), the system's dependability can be evaluated.

In Figure 1 we give a simple example of a description of a NOT-gate, which has been extended by its fault descriptions. The bold typed lines denote the standard VHDL description of the gate, whereas the normal weighted lines show one possibility of extending the description of the gate with its fault behavior. It can be seen, that the interface of the NOT-gate, i.e. the ENTITY declaration, does not differ from the fault free model of the gate. In this example, we used the widely accepted stuck-at fault model but it should be noted, that any other fault behavior can also be described using this technique. In the ARCHITECTURE description of the gate, we inserted four FIS where, for example, o_sa0 stands for a stuck-at-0 fault at the output. The mean time between the occurrence of this fault is given as 20.000 hours and its mean duration is 5 nanoseconds. It can be seen that all parts of the extended VHDL-language which correspond to the fault description and the fault behavior are shielded from the environment of the gate-level component. This ensures the easy usage of VERIFY and allows the hardware developer to build the hardware without the need of additional description effort for the fault model.

```
ENTITY not_gate IS
      PORT  (     input:          IN          bit;
                  output:         OUT         bit);
END not_gate;


ARCHITECTURE behaviour OF not_gate IS
      SIGNAL i_sa0: BOOLEAN INTERVAL 10000 h DURATION 5 ns;
      SIGNAL i_sa1: BOOLEAN INTERVAL 15000 h DURATION 5 ns;
      SIGNAL o_sa0: BOOLEAN INTERVAL 20000 h DURATION 5 ns;
      SIGNAL o_sa1: BOOLEAN INTERVAL 30000 h DURATION 5 ns;
BEGIN
      PROCESS (input, i_sa0, i_sa1, o_sa0, o_sa1)
      BEGIN
            IF i_sa0 OR o_sa1 THEN
                  output <= '1';
            ELSIF i_sa1 OR o_sa0 THEN
                  output <= '0';
            ELSE
                  output <= NOT input AFTER 10 ns;
            END IF;
      END PROCESS;
END behaviour;
```

Figure 1 Example code of a NOT-gate

The compiler of VERIFY extracts the fault descriptions of all gates in the developed system and generates a list of all FIS with its parameters. After linking the VERIFY simulation library module, the simulator can use the FIS to inject the faults according to their mean time of occurrence which will be adjusted to the total simulation-time. The trace, i.e. the values of all signals in the system, after injecting a fault will be compared with the trace of the fault free run. This comparison shows whether the system recover to a fault free state in the given observation time or whether it fails.

Therefore, the faults will be injected according to their fault rates and the probability of the faults leading to a failure of the system will be evaluated by VERIFY. After that, the system's failure rate can be computed as shown in Section 2.

References

[1] J. Carreira, H. Madeira, J. G. Silva: "Xception: Software Fault Injection and Monitoring in Processor Functional Units" *Preprints of the DCCA-5, Working Conference on Dependable Computing for Critical Applications*, Urbana Champaign , USA, Beckman Institute, September 27-29, 1995, pp. 135-149.

[2] J. A. Clark, D. K. Pradhan, “REACT: An Integrated Tool for the Design of Dependable Computer Systems”, in “Foundations of Dependable Computing, Models and Frameworks for Dpendable Systems”, G. M. Koob, C. G. Lau (ed.), Kluwer, pp. 169-192, 1994.

[3] E. Czeck, D. Siewiorek "Effects of Transient Gate-Level Faults on Program Behavior", *Proc. 20th Symp. on Fault Tolerant Computing (FTCS-20)*, Newcastle Upon Tyne, June 1990, pp. 236-243.

[4] Goswami, K.; Iyer, R.K.: "*DEPEND: A Simulation-Based Environment for System-Level Dependability Analysis"*. Center for Reliable and High-Performance Computing (CRHC), Univ. of Illinois at Urbana-Champaign, 1992.

[5] U. Gunneflo, J. Karlsson, and J. Torin: "Evaluation of error detection schemes using fault injection by heavy-ion radiation.": *Proc. 19th Symp. on Fault-Tolerant Computing (FTCS-19)*, Chicago, Illinois, 21-23, June 1989, pp 340-347.

[6] S. Han, H. A. Rosenberg, K. G. Shin: „*DOCTOR: An IntegrateD SOftware Fault InjeCTiOn EnviRonment*“, Technical Report Univ. of Michigan, December 1993

[7] A. Hein, K. K. Goswami, “Combined Performance and Dependability Evaluation with Conjoint Simulation”, *Proc. of 7th European Simulation Symposium*, Erlangen-Nuremberg, Oct. 26-28, 1995, pp. 365-369.

[8] G. A. Kanawati, N. A. Kanawati, J. A. Abraham: "FERRARI: A Tool for the Validation of System Dependability Properties", *Proc. 22th Symp. on Fault-Tolerant Computing (FTCS-22)*, Boston, Massachusetts, July 8-10, 1992, pp. 336-344.

[9] S. Kumar, R. H. Klenke, J. H. Aylor, B. W. Johnson, R. D. Williams, R. Waxman, “ADEPT: A Unified System Level Modeling Design Environment”, Proceedings of the 1st Annual RASSP Conference, Arlington, Virginia, pp. 114-123, August 15-18, 1994.

[10] H. Madeira, M. Rela, J. G. Silva: "RIFLE: A General Purpose Pin-Level Fault Injector", *Proc. First European Dependable Computing Conference (EDCC-1)*, Berlin, Germany, Springer Verlag, October 4-6, 1994, pp. 199-216.

[11] M. Rimén, J. Ohlsson, J. Karlsson, E. Jenn, J. Arlat: "Design Guidelines of a VHDL-based Simulation Tool for the Validation of Fault Tolerance", *Proc. 1st ESPRIT Basic Research Project PDCS-2 Open Workshop*, LAAS-CNRS, Toulouse, France, September 1993, pp. 461-483.

[12] V. Sieh, O. Tschäche, F. Balbach: “Comparing Different Fault Models using VERIFY”, *Proc. 6th Conference on Dependable Computing for Critical Applications (DCCA-6)*, Grainau, Germany, pp. 59-76, March 1997.

[13] "*IEEE Standard VHDL Language Reference Manual*", ANSI/IEEE Std 1076-1993, IEEE Inc., 1993.