VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions[1]

Volkmar Sieh, Oliver Tschäche, Frank Balbach

Department of Computer Science III
University of Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
email: {vrsieh, ortschae, balbach}@informatik.uni-erlangen.de

Abstract

*A new technique for reliability evaluation of digital systems will be presented by demonstrating the functionality and usage of the simulation based fault injector VERIFY (**V**HDL-based **E**valuation of **R**eliability by **I**njecting **F**aults efficientl**Y**). This software tool introduces a new way for describing the behavior of hardware components in case of faults by extending the VHDL language with fault injection signals together with their rate of occurrence. The accuracy of the results is obtained by using the same VHDL-models which have been developed during conventional phases of hardware design. For demonstrating the capabilities of VERIFY, a VHDL-model of a simple 32-bit processor (DP32) will be used as an example to illustrate the several steps of reliability evaluation.*

1 Introduction

During the hardware design of safety-critical digital systems it is indispensable not only to tune the functional and temporal behavior of the system to the demands of its specification, but also to estimate the basic numbers of dependability, e.g. the reliability or the mean time to failure of the system. In order to support the dependability evaluation during all phases of hardware design we developed VERIFY. VERIFY is able to cover all levels of abstraction by the fact that it is based on VHDL [2] which itself is widely used to develop hardware starting from system level, via the register-transfer level down to the gate level. Therefore, the systems designer can use a single language for testing the functional and temporal behavior as well as determining the reliability of the system he has designed (uniform model). Because there is no longer the need to use an external tool and to develop a model which again describes the hardware and its failure modes, this approach is very efficient for hardware design. Due to the fact, that VERIFY uses an integrated fault model, the dependability evaluation is very close to that of the actual hardware. Using simulation based fault injection, full observability and controllability of all components of the hardware is guaranteed.

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

2 Related Research

The impacts of faults have been investigated by several researchers. Several approaches towards this goal can be distinguished. Injecting faults at the physical level has been done by either stressing the hardware with environmental parameters or by modification of the pin-level values. The first method has been used by Karlsson et al. by inducing soft errors with heavy-ion radiation [15]. The second approach to inject faults at the physical level is the use of pin-level fault injectors, where the signal values of pins of an IC are under control of external devices which determine the time and duration of injection (MESSALINE [1] and RIFLE [19]). Whereas the latter method enables reproducibility of the results by the ability to control all fault parameters inducing soft errors corresponds more to the real physical nature of the faults. In addition, the current trend of integrating more and more components on-chip makes it difficult for pin-level fault injection to cover the internal faults adequately. Both approaches for the injection at the physical level tend to have a high overhead in hardware.

Several research groups have developed powerful tools to inject faults by software. Barton et al. made one of the early approaches with a tool called FIAT [3], where the task's memory image can be corrupted during run-time. FERRARI, which was developed by Kanawati et al. [17] allows transient fault injection by corruption of a process's memory image and by insertion of software trap instructions. Kao et al. proposed a tool named FINE which is able to inject faults by using a software monitor to trace the control flow [16]. In addition, several examples for software implemented fault injection into existing systems

are DOCTOR by Han et al. [13], Xception by Carreira et al. [5] and EFA by Echtle et al. [9]. All of these approaches need access to at least a prototype of the hardware for which the effects of faults have to be examined. Another drawback of software implemented fault injection is the fact, that the implementation of faults like erroneous exception handling which affect the cache subsystem is very costly.

The third group of tools developed for fault injection covers the simulation based approach. The advantages of this approach are the observability of all components which have been modeled, and the ability to obtain the values for reliability already in the design phase of the system. One of the early approaches of run-time injection has been presented by Czeck and Siewiorek [8] who examined error propagation by injecting faults in a VERILOG-model of an IBM RT PC. Choi et al. injected transient faults in a model of a jet-engine controller by using the mixed-mode simulator SPLICE [4]. Another mixed-mode fault simulation approach has been presented by Cha et al. [6], where transient gate-level faults have been injected by using a combination of a timing fault simulator (TIFAS) and a zero-delay parallel fault simulator TPROOVES to speed up the simulation time. ADEPT [18], REACT [7], DEPEND [10] and SIMPAR [12] are tools which allow building and evaluating models of dependable computing systems at system level. Because of the big overhead of the simulation engines used, these tools are unmanageable for evaluating models at gate level. Rimen et al. proposed a general approach to fault injection in VHDL models [20]. They are using a *saboteur*-based technique, a *mutant*-based technique and built-in commands of the VHDL simulator for fault injection. The research group developed a tool named MEFISTO, which covers the fault injection techniques of these categories [14]. The drawback of using mutants is a huge overhead for system evaluation as these mutants are static and the model has to be recompiled for each of the experiments.

Figure 1 contains a comparison of many existing dependability evaluation tools.

	system level	RTL level	hardware level	uniform model	integrated fault model	model similar to hardware	usable for design	hierarchical models	observability	controllability	event signaling using signals	efficient evaluation
Petri-Nets	+ +	- -	- -	+ +	+ +	- -	+ +	-	+ +	+ +	+	+
ADEPT	+ +	+	-	-	+	-	+ +	+ +	+ +	+ +	+	+/-
DEPEND	+ +	+	- -	+ +	+	- -	+ +	+	+ +	+ +	-	-
REACT,	+ +	- -	- -	- -	-	- -	+ +	- -	+ +	+ +	- -	-
SIMPAR	+ +	+	- -	+ +	+	- -	+ +	+ +	+ +	+ +	-	-
MEFISTO	+ +	+ +	+ +	- -	- -	+	+ +	+ +	+ +	+ +	+ +	- -
FERRARI	- -	+ +	- -	- -	- -	+	- -	- -	-	+	-	++
FIAT	- -	+ +	- -	- -	- -	+	- -	- -	-	+	-	+ +
DOCTOR	- -	+ +	- -	- -	- -	+	- -	- -	-	+	-	+ +
XCEPTION	- -	+ +	- -	- -	- -	+	- -	- -	-	+	-	+ +
EFA	- -	+ +	- -	- -	- -	+	- -	- -	-	+	-	+ +
FINE	- -	+ +	- -	- -	- -	+	- -	- -	-	+	-	+ +
Heavy Ion	- -	- -	+ +	-	-	- -	- -	- -	- -	- -		+ +
Power Dist.	- -	- -	+ +	-	-	- -	- -	- -	- -	- -		+ +
MESSALINE	- -	- -	+ +	- -	- -	- -	- -	- -	-	+ +		+ +
RIFLE	- -	- -	+ +	- -	- -	- -	- -	- -	-	+ +		+ +
VERIFY	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+

+ + is well provided - is poorly provided
+ is provided - - is not provided

Figure 1: Comparison of Evaluation Tools

3 Fault Injection using VERIFY

3.1 Model Description

VHDL has been established as one of the most important hardware description languages for integrated digital circuits. So far, it has not been foreseen in VHDL to directly support the checking of the reliability parameters of the system during the design phase. The development of dependable systems does not only require the validation of temporal and functional behavior but also the ability to validate the dependability features. For this purpose we came to the conclusion that the mean time between faults, their duration and their effects should be an integral part of the description of each behavioral component of the model [21]. In order to demonstrate the feasibility of this approach, we developed the VERIFY tool, which allows the integrated description of the fault free behavior as well as the component's behavior after one of the faults correlated with this component has been activated. The natural way in VHDL of exchanging information with a component is the use of signals. Therefore, we used the concept of signals to be able to describe the faults correlated to a component and at the same time have an interface for the simulator to activate the fault for a given time. Each of the possible faults known for the component can therefore be described by a separated signal. The behavioral description of the component has to be extended by the actions correlated to the faults.

This approach enables the hardware manufacturers which provide the design libraries, i.e. the AND-gates, OR-gates and other basic behavior-described components to express their knowledge of fault occurrences in these

components. By extending the models by their fault behavior, a simulator can be used in an early design phase to evaluate the reliability of dependable systems, investigate the manifestation of faults and to compare several design alternatives regarding the overhead and benefit of fault tolerance mechanisms. It should be noted that the parameters of the faults, i.e. frequency and duration, can easily be adjusted according to the environment the dependable system will be used in (e.g. space-mission systems, controllers for nuclear power plants, etc.), by exchanging the design library modules. No structural components needs to be changed.

Figure 2 gives an example of a VHDL description of a NOT-gate which has been extended by an exemplary fault description. The bold typed lines show the standard VHDL

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


ARCHITECTURE behaviour OF not_gate IS
  SIGNAL sa0: BOOLEAN INTERVAL 20000 h DURATION 5 ns;
  SIGNAL sa1: BOOLEAN INTERVAL 30000 h DURATION 5 ns;
BEGIN
  PROCESS (input, sa0, sa1) BEGIN
     IF sa1 THEN       output <= '1';
     ELSIF sa0 THEN    output <= '0';
     ELSE              output <= NOT input;
     END IF;
  END PROCESS;
END behaviour;
```

Figure 2: Example Code of a NOT-Gate

description of the gate and the normal weighted lines one possibility of extending the description of the NOT-gate with its fault behavior and the corresponding parameters. As it can be seen by this example, the entity declaration and therefore the interface with other components need not be modified for describing the faults. For this demonstration, 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. As it can be seen from the behavioral description the signal named *sa0* denotes the case, where a stuck-at-0 fault occurs at the output of the NOT-gate. The mean time between the occurrences of this fault type is given as 20000 hours and its mean duration is 5 ns.

Even if the exact values of the fault rates, fault duration times etc. are not known it is of great value to document the assumptions which are made while evaluating the reliability of a system. Writing down fault attributes and effects will clarify many questions about the experiments because the model contains all information needed to perform or to reproduce the experiments.

Using this approach it is not necessary to recompile the model to inject another fault, to have any control-file for the simulator or to build up a new fault injector for a new model. This is the case because the model itself contains all information necessary for simulation and not the simulator or it's control-file as in many other tools for dependability evaluation. This approach will help to preserve consistency between hardware and fault model during the desing phase.

3.2 Simulation, Multi-Threaded Fault Injection

For the VERIFY-tool we developed a compiler which is able to handle the described extensions to VHDL and a simulator for running the fault injection experiments. The fault injecting signals (FIS) are extracted automatically by the compiler and supplied for the simulator which is linked to the executable. Using this technique, the simulator has access to all fault parameters described for the system. Figure 3 gives an overview of the different phases and modules of VERIFY.

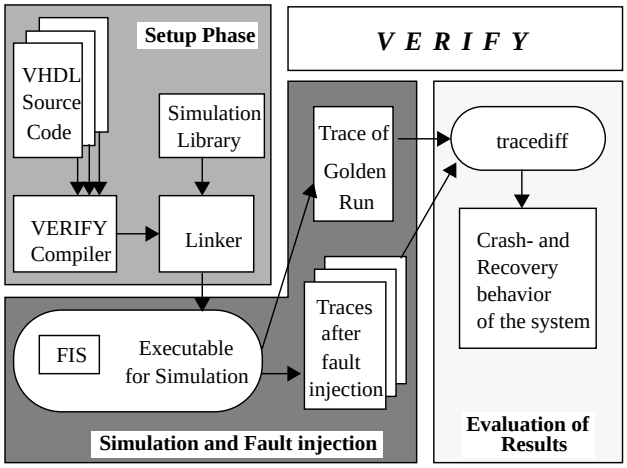

Figure 3: Overview of the VERIFY Fault Injection Tool

The new technique implemented by VERIFY enables a completely automated evaluation of system dependability features. The execution of all fault-injection experiments is performed by the simulator without any need of interaction with the user of the tool. The time and the location for injecting the next fault is determined according to the weighting of the described fault intervals.

To speed up simulation, the experiments will be carried out by a technique called *multi-threaded fault injection* described by Güthoff and Sieh in [11] (see figure 4). Only one golden run is performed (long arrow). During this simulation checkpoints are saved at some points in time (T_1, T_2, …, T_N; dotted arrows). These checkpoints are used for fault injection by N faulty runs (F_1, …, F_N; short arrows). The state of each faulty run

is compared dynamically against the state of the golden run simulated in parallel. Once the faulty run has reached the same state as the golden run (fault has been recovered from) or the simulation interval of the faulty run has reached a given limit (timeout) the simulation of the faulty run is aborted.

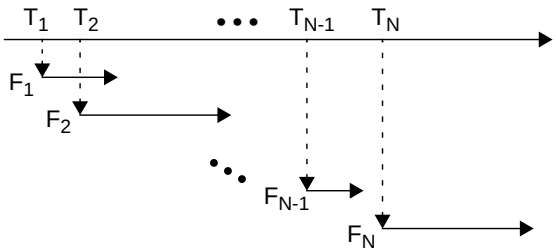

Figure 4: Multi-Threaded Fault Injection

Using this approach simulation overhead due to fault injection is very low. The golden run has to be simulated in any way to test the given model. All faulty runs are as short as possible. They start at the occurrence of the fault and end at the point of successful recovery termination (or after a given timeout).

3.3 Evaluation of Experiment Results

During the experiments of fault injection a trace of all signal values will be logged for the golden run and for each single fault injection experiment.

According to the complete observability any desirable analysis of the experiment may be performed. Each trace contains data describing one fault injection experiment:

- location, type, time and duration of fault
- exact system behavior over time due to the injected fault (trace of all signals)

So it is possible to evaluate the rates of system crashes and recoveries and the recovery time distribution *related to* the location, type, date and duration of the faults injected. Subcomponents which faults often lead to system failure or result in long recovery times may be detected and their design can be improved.

VERIFY determines the recovery time of each injected fault by the following algorithm (see figure 5). Before injecting the fault (during phase A) all corresponding signals will have the same value. During (small black rectangle) and after fault injection values of signals of the golden run and the faulty run will differ (phase B). If fault recovery is successful all signals will return to legal values (phase C). The overhead introduced by the error recovery, it is the recovery time, is the difference of the length in time of phase B of the faulty run and phase B of the golden run.

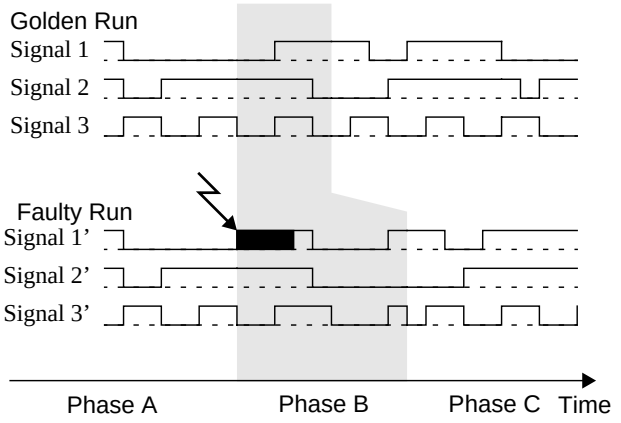

Figure 5: Recovery Time Determination

Because of the fact that some signals may have a different value after error recovery (e.g. error counter) VERIFY allows to exclude these signals when comparing the state of the golden run and the state of the faulty run.

Since most of all faults lead to only short system disturbances which can be masked even without any special hardware or software redundency as it was shown by injecting faults into a DP32 risc processor system using VERIFY [21] this method seems to be adequate to estimate the recovery time distribution of the system. This approach has the advantage that it can be applied to any system without any special knowlegde about the system itself.

VERIFY is used in a common project of the University of Jena and the University of Erlangen-Nürnberg (3D-Smart-Pixels) to compare the efficiency of different approaches to make an opto-electronic system fault tolerant. The electronical part of the system will be modelled as described by [21]. To simulate the optical part of the system the extensions to VHDL introduced by VERIFY will be used to model even optical faults, e.g. crosstalk.

4 References

[1] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, D. Powell: "Fault injection for dependability validation: a methodology and some applications", *Trans. on Soft. Eng.*, Vol. 16, No. 2, 1990, pp. 166-182

[2] P. Ashenden: "The VHDL-cookbook", Univ. of Adelaide, South Australia, Technical Report 1990

[3] J. Barton, E. Czeck, Z. Segall, D. Siewiorek: "Fault Injection Experiments using FIAT", *Trans. on Comp.*, Vol. 39, No. 4, 1990, pp. 575-582

[4] G. Choi, R. Iyer, V. Carreno: "Simulated fault injection: A methodology to evaluate fault tolerant microprocessor architectures", *IEEE Trans. on Reliability*, Vol. 39, No. 4, 1990, pp. 486-490

[5] J. Carreira, H. Madeira, J. G. Silva: "Xception: Software Fault Injection and Monitoring in Processor Functional Units" *Preprints of Conf. on Dependable Computing for Critical Applications (DCCA-5)*, Urbana Champaign , USA, 1995, pp. 135-149

[6] H. Cha, E. Rudnick, G. Choi, J. Patel, R. Iyer, "A Fast and Accurate Gate-Level Transient Fault Simulation Environment", *Proc. 23rd Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France, 1993, pp. 310-319

[7] 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 Dependable Systems*, G. M. Koob, C. G. Lau (ed.), Kluwer, 1994, pp. 169-192

[8] 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, 1990, pp. 236-243

[9] K. Echtle, M. Leu "The EFA Fault Injector for Fault Tolerant Distributed System Testing", *Proc. Workshop on Fault-Tolerant Parallel and Distributed Systems*, Amherst, USA, 1992, pp. 28-35

[10] K. K. Goswami, R. K. Iyer, L. Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis", *Trans. on Computers*, Vol. 46, 1997, pp. 60-74

[11] J. Güthoff, V. Sieh: "Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method", *Proc. 25th Symp. on Fault-Tolerant Computing (FTCS-25)*, Pasadena, California, 1995, pp. 196-206

[12] A. Hein, K. K. Goswami, "Combined Performance and Dependability Evaluation with Conjoint Simulation", *Proc. of 7th European Simulation Symposium*, Erlangen, Germany, 1995, pp. 365-369

[13] S. Han, K. G. Shin, H. A. Rosenberg, „DOCTOR: An IntegrateD SOftware Fault InjeCTiOn EnviRonment for Distributed Real-Time Systems", *Proc. Int. Computer Performance and Dependability Symp. (IPDS'95)*, Erlangen, Germany, 1995, pp. 204-213

[14] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson: "Fault Injection into VHDL Models: The MEFISTO Tool". *Proc. 24th Symp. on Fault Tolerant Computing, (FTCS-24)*, Austin, Texas, 1994, pp. 66-75

[15] J. Karlsson, U. Gunneflo, J. Torin: "Use of Heavy-Ion Radiation from Californium-252 for Fault Injection Experiments" in *Dependable Computing for Critical Applications*, A. Avizienis, J.-C. Laprie (eds.), Vol. 4, Springer, 1991, pp. 197-212

[16] W. Kao, R. K. Iyer, D. Tang: "FINE: A Fault Injection and Monitor Environment for Tracing the UNIX System Behavior under Faults", *Trans. on Soft. Eng.*, Vol. 19, No. 11, 1993, pp. 1105-1118

[17] 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, 1992, pp. 336-344

[18] S. Kumar, R. H. Klenke, J. H. Aylor, B. W. Johnson, R. D. Williams, R. Waxman, "ADEPT: A Unified System Level Modeling Design Environment", *Proc. 1st Annual RASSP Conference*, Arlington, Virginia, 1995, pp. 114-123

[19] 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, 1994, pp. 199-216

[20] 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, 1993, pp. 461-483

[21] V. Sieh, O. Tschäche, F. Balbach, "Evaluation of Dependable Systems using VERIFY", *Preprints 6th Conference on Dependable Computing for Critical Applications (DCCA-6)*, Grainau, Germany, 1997, pp. 59-76