

# VERIFY: Zuverlässigkeitsanalyse unter Verwendung von VHDL-Modellen mit integrierter Fehlerbeschreibung<sup>1</sup>

V. Sieh, F. Balbach, O. Tschäche

Institut für Mathematische Maschinen und Datenverarbeitung (IMMD) III,  
Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, Germany  
E-Mail: Volkmar.Sieh@informatik.uni-erlangen.de  
Telefon: +49 (0)9131 85 7007, Fax: +49 (0)9131 85 7239

## Kurzfassung

*Die Intention von Modellierungssprachen wie z.B. VHDL oder VERILOG ist es, eine durchgängige Beschreibungssprache vom Entwurf über den Test bis zur Implementierung zur Verfügung zu stellen. Dem Benutzer werden dafür Sprachkonstrukte zur Verfügung gestellt, die es erlauben, die Funktionalität und den zeitlichen Ablauf der Komponentenreaktion eindeutig und vollständig zu beschreiben. Ziel unserer Arbeit war es, die Beschreibung der möglichen Fehler eines Systems (mittlere Rate, mittlere Dauer und Auswirkung) mit in die normale Spezifikation des fehlerfreien Verhaltens aufzunehmen. Das Resultat ist ein **einheitliches Modell** auch für die **Zuverlässigkeitsanalyse von Systemen**. Dieser Ansatz wurde in einer Prototyp-Implementierung einer Modellierungsumgebung mit dem Namen VERIFY (VHDL-based Evaluation of Reliability by Injecting Faults efficiently) umgesetzt. Erste Experimente haben die Nützlichkeit des Ansatzes gezeigt.*

## Einleitung

Es ist eine Vielzahl von Fehlerinjektionswerkzeugen spezifiziert und implementiert worden, um die Zuverlässigkeit verschiedener Systeme zu messen. Dazu wird das zu untersuchende System (Modell oder reale Hardware) mit Software oder Hardware instrumentiert, um temporäre oder auch permanente Fehler injizieren und ihre Auswirkungen protokollieren zu können. Ziel ist es, Schwachstellen der Systeme zu erkennen und gegebenenfalls zu beseitigen. Eine Übersicht über diese Versuche geben z.B. Iyer [10] und Clark [4]. Diese Ansätze können in Hardware-basierte (z.B. von Gunneflo und Karlsson [7], [13], Arlat [1] und Madeira [15]), Software-implementierte (z.B. Echtle [5] (EFA), Carreira [3] (Xception), Kanawati [12] (FERRARI), Segall [18] (FIAT), Sieh [16] und Han [8] (DOCTOR)) sowie Simulationsbasierte Verfahren unterteilt werden.

Während beim Hardware-basierten und Software-implementierten Ansatz die reale Hardware zur Fehlerinjektion und Beobachtung instrumentiert wird, werden im simulationsbasierten Ansatz nur Beschreibungen der eigentlichen Hardware zur Fehlerinjektion verwendet. Diese Beschreibungen werden vor den Experimenten erweitert bzw. modifiziert, um auch die möglichen Fehler des Systems zu spezifizieren. Simulatoren werten dann diese so erweiterten Modelle aus, um Aussagen über die Zuverlässigkeit, Fehlerüberdeckung u.ä. machen zu können. Unter diesen Modellierungsumgebungen befinden sich z.B. DEPEND [6], ADEPT [14], MEFISTO [11].

Ein Nachteil all dieser Verfahren ist, daß sie für die Fehlerinjektion mehrere verschiedene,

---

1. Diese Arbeit wurde unterstützt durch die Deutsche Forschungsgemeinschaft als Teil des SFB-182.

nur lose zusammenhängende Modelle verwenden. Es existiert meist ein Hardware-Modell bzw. reale Hardware, ein davon getrenntes Fehlermodell (implementiert im Simulator bzw. Fehlerinjektor) und eine Steuerdatei, die angibt, welcher Fehler wo zu welchem Zeitpunkt während der Simulation aktiviert werden soll. Dieser Nachteil hat mehrere Konsequenzen:

- Der Modellierer muß mehrere Modellierungs-/Programmiersprachen lernen.
- Das normale und das fehlerhafte Verhalten einer Komponente ist in verschiedenen Dateien in verschiedenen Formaten beschrieben (Konsistenzproblem).

Ziel dieser Arbeit war es, diese verschiedenen Darstellungsarten zu vereinheitlichen und damit die beschriebenen Nachteile zu beseitigen.

## **Erweiterung von VHDL zur Beschreibung von Fehlern**

Da das Zeitverhalten integraler Bestandteil der Sprache VHDL ist, sollte auch die Erweiterung zur Beschreibung des Fehlerverhaltens eingebetteter Bestandteil der Sprache VHDL werden. Zudem war ein Compiler und Simulator für die Fehlersimulation zu schreiben, um verschiedene neue, speziell die Fehlersimulation beschleunigende Verfahren testen zu können (siehe Güthoff und Sieh [8]), so daß eine Syntaxerweiterung der Sprache nicht ins Gewicht fiel.

Die Tatsache, ob ein Fehler zur Zeit aktiv ist, läßt sich am einfachsten durch ein Signal beschreiben. Das Auftreten bzw. das Verschwinden eines Fehlers kann dann durch einen Event am Signal dem Prozeß mitgeteilt werden. Dieses Signal wird jedoch nicht wie die normalen Signale in VHDL durch andere Prozesse gesetzt. Es ist als Zufallsvariable zu verstehen, die unabhängig vom System ihren Wert ändert. Um die Zufallsvariable genauer zu spezifizieren, sind Parameter für ihre Auftrittsverteilung und Verteilung der Dauer des Auftretens anzugeben. Da in diesem Ansatz alle Fehler als unabhängig voneinander angesehen werden sollten, reicht für die Angabe der Auftrittsverteilung ein Parameter für die Exponentialverteilung (mittlere Auftrittsrate). Ähnlich wurde ein Wert für die mittlere Dauer des Fehlers angenommen.

Dieser Ansatz gibt den Hardware-Herstellern die Möglichkeit, ihr Wissen über mögliche Fehler in einzelnen Komponenten in ihre Komponentenbibliothek zu integrieren. Durch die Erweiterung der Modelle um ihr Fehlerverhalten kann ein Simulator schon in der frühen Design-Phase eingesetzt werden, um die Fehlertoleranzeigenschaften eines Systems zu bestimmen, die Fehlerausbreitung zu untersuchen und den Aufwand und Nutzen von verschiedenen Fehlertoleranzmaßnahmen zu vergleichen.

Abbildung 1 zeigt ein Beispiel für die Modellierung von verschiedenen Fehlern in einem NOT-Gatter mit Hilfe der neuen Sprachkonstrukte. Die normale Beschreibung des Gatters ist fett gedruckt (die Entity-Definition ändert sich nicht). Modelliert wurde im Beispiel ein Stuck-At-ähnliches Fehlermodell.

## **Beispiel**

Um die Nützlichkeit dieses Ansatzes zu demonstrieren, wurde ein vollständiges Modell des DP32-Risc-Prozessors [2] mitsamt Speicher und Kontroll-Logik aufgebaut und exemplarisch ausgewertet.

Zu schreiben war nur eine ausreichende Komponentenbibliothek mit integriertem Fehlerverhalten. Durch ein Synthese-Tool wurde automatisch aus dem gegebenen Register-Transfer-Level-Modell des DP32 ein Gatter-Modells generiert. Die Modellauswertung konnte durch VERIFY ohne Benutzerhilfe selbständig durchgeführt werden.

```

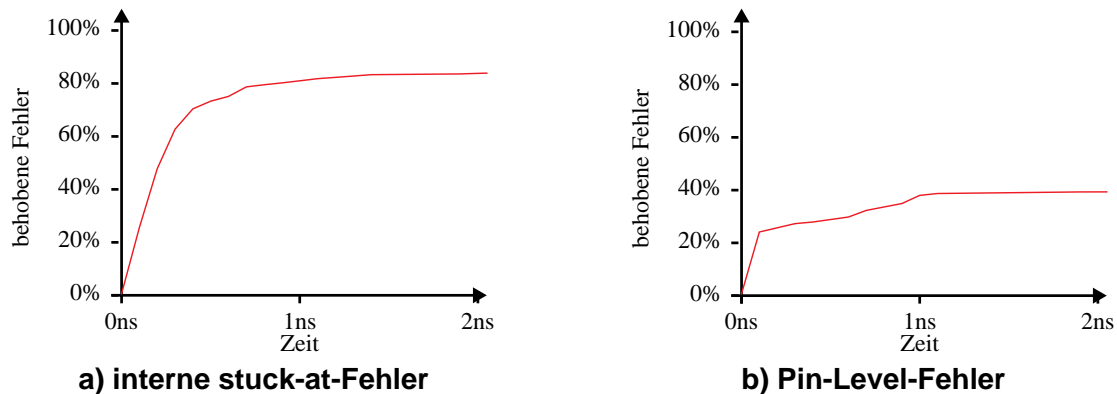
ENTITY not_gate IS
    PORT (    input:    IN    bit;
           output:    OUT  bit);
END not_gate;

ARCHITECTURE behaviour OF not_gate IS
    SIGNAL i_sa_0: BOOLEAN INTERVAL 1000 h DURATION 5 ns;
    SIGNAL i_sa_1: BOOLEAN INTERVAL 1500 h DURATION 5 ns;
    SIGNAL o_sa_0: BOOLEAN INTERVAL 2000 h DURATION 5 ns;
    SIGNAL o_sa_1: BOOLEAN INTERVAL 3000 h DURATION 5 ns;
BEGIN
    PROCESS (input, i_sa_0, i_sa_1, o_sa_0, o_sa_1) BEGIN
        IF i_sa_0 OR o_sa_1 THEN          output <= '1';
        ELSIF i_sa_1 OR o_sa_0 THEN      output <= '0';
        ELSE                             output <= NOT input;
        END IF;
    END PROCESS;
END behaviour;

```

**Abb 1: NOT-Gatter mit Erweiterungen zur Fehlerbeschreibung**

Abb. 2 zeigt beispielhaft, welche Fehlertoleranzdaten automatisch zu gewinnen sind. Die Grafik gibt an, wieviele der injizierten Fehler nach einer angegebenen Zeit noch Auswirkungen im System haben.



**Abb 2: Recovery-Zeit-Verteilungsdiagramm**

Zu sehen ist, daß interne Fehler im Durchschnitt schneller behoben werden als Fehler an den Ein- und Ausgangspins der CPU. Auch werden prozentual mehr interne Stuck-At-Fehler behoben als Pin-Level-Fehler (ca. 80% gegenüber ca. 40%).

## Zusammenfassung

Wie sich gezeigt hat, können mit dem Tool VERIFY sehr schnell und einfach Modelle zur Zuverlässigkeitsanalyse eines Systems aufgebaut und vollautomatisch ausgewertet werden. Durch den verwendeten Modellierungsansatzes ist es sehr gut möglich, die Konsistenz von Hardware- und Fehlermodell auch bei häufigen Modelländerungen zu gewährleisten.

Durch die gewonnenen Daten erfährt der Designer, welche Komponenten besonders fehleranfällig sind und in welchen Komponenten Fehler mit der größten Wahrscheinlichkeit zu einem Ausfall des Gesamtsystems führen.

## Literatur

- 1 J. Arlat, Y. Crouzet, J.-C. Laprie, "Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems", FTCS-19, Chicago Illinois, 1989, IEEE, S. 348-355.
- 2 P. J. Ashenden, "The VHDL Cookbook", University of Adelaide, South Australia, Tech. Report, 1990.
- 3 J. Carreira, H. Madeira, J. G. Silva. "Xception: Software Fault Injection and Monitoring in Processor Functional Units", DCCA-5, Urbana Champaign, USA, 1995, S. 135-149.
- 4 J. A. Clark, D. K. Pradhan, "Fault Injection — A Method for Validating Computer-System Dependability" in "Computer Innovative technology for computer professionals", IEEE Computer Society, Vol. 28, No. 6, S. 47-56.
- 5 K. Echtele, M. Leu, "The EFA Fault Injector for Fault Tolerant Distributed System Testing", Workshop on Fault-Tolerant Parallel and Distributed Systems, Amherst, USA, S. 28-35, 1992.
- 6 K. K. Goswami, R. K. Iyer, "DEPEND: A Design Environment for Prediction and Evaluation of System Dependability", 9th Digital Avionics Systems Conference, October 1990.
- 7 U. Gunneflo, J. Karlsson, and J. Torin: "Evaluation of error detection schemes using fault injection by heavy-ion radiation", FTCS-19, Chicago, USA, 1989, S. 340-347
- 8 J. Güthoff, V. Sieh, "Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method", FTCS-25, Pasadena, USA, 1995, S. 196-206.
- 9 S. Han, H. A. Rosenberg, K. G. Shin: „DOCTOR: An Integrated Software Fault InjeCTiOn EnviRonment“, Technical Report University of Michigan, 1993.
- 10 R. K. Iyer, "Experimental Analysis of Computer System Dependability", Fault-Tolerant Computing, Second Edition, D. K. Pradhan, Ed., Prentice Hall, 1994.
- 11 E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson: "Fault Injection into VHDL Models: The MEFISTO Tool", FTCS-24, Austin, USA, S. 66-75, 1994
- 12 G. Kanawati, N. Kanawati, J. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties", FTCS-12, Boston, USA, 1992, S. 336-344.
- 13 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 (Editor), in Reihe "Dependable Computing and Fault-Tolerant Systems", Vol. 4, Springer-Verlag Wien-New York, 1991, S. 197-212.
- 14 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, S. 114-123, 1995.
- 15 H. Madeira, F. Moreira, P. Furtado, M. Rela, J. G. Silva, "Pin-level Fault Injection: Some Research Results at the University of Coimbra", Workshop Fault and Error Injection for Dependability Validation, Gothenburg, 1993.
- 16 V. Sieh, A. Pataricza, B. Sallay, W. Hohl, J. Hönig, B. Benyo, "Fault Injection Based Validation of Fault-Tolerant Multiprocessors", 8th Symposium on Microcomputer and Microprocessor Applications, Budapest, Hungary, 1994, S. 85-94.
- 17 V. Sieh, O. Tschäche, F. Balbach, "Comparing Different Fault Models Using VERIFY", erscheint in "Sixth IFIP International Working Conference on Dependable Computing for Critical Applications", Greinau, Germany, 1997.
- 18 Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, T. Lin, "FIAT — Fault Injection Based Automated Testing Environment", FTCS-18, Tokyo, Japan, S. 102-107.