



HAL
open science

A DEVS-based Modeling Behavioral Fault Simulator for RT-Level Digital Circuits

Laurent Capocchi, Fabrice Bernardi, Dominique Federici, Paul-Antoine
Bisgambiglia

► **To cite this version:**

Laurent Capocchi, Fabrice Bernardi, Dominique Federici, Paul-Antoine Bisgambiglia. A DEVS-based Modeling Behavioral Fault Simulator for RT-Level Digital Circuits. SCS Summer Computer Simulation Conference (SCSC04), Jul 2004, San Jose, United States. pp.481-486. hal-00165460

HAL Id: hal-00165460

<https://hal.science/hal-00165460>

Submitted on 26 Jul 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A DEVS-based Modeling and Behavioral Fault Simulator for RT-Level Digital Circuits

Capocchi Laurent, Bernardi Fabrice, Federici Dominique, Bisgambiglia Paul
University of Corsica
UMR CNRS 6134, Quartier Grossetti
BP 52, 20250 Corte, France
{capocchi, bernardi, federici, bisgambi}@univ-corse.fr

ABSTRACT

The domain of fault simulation for digital circuits described at the RT-level is currently under heavy researches. The goal of these researches is to define a fast and efficient methodology for the validation of test patterns very early in the design flow. We propose in this article a new approach for the modeling and the simulation of behavioral faults for digital circuits described in the VHDL language, using a discrete event approach. This methodology, based on the DEVS formalism, is implemented in a working prototype, and experimental results show the correctness of our approach and the efficiency of our behavioral fault simulator called BFS-DEVS. The fault model used to validate our results is essentially based on the stuck-at fault model since a good simple stuck-at faults coverage rate implies a good real faults coverage rate.

Keywords: DEVS, fault simulation, testability, modeling, simulation.

INTRODUCTION

With the increasing complexity of digital circuits, test patterns generation and fault simulation have become very complex steps in the design flow. The use of structural models described at the gate level implies heavy and costly test environments, and also very expensive execution times unacceptable for the industry. In order to answer these problems, there is a need to define new methodologies that can be applied as soon as the behavioral level and that are also efficient for the test at the hardware level.

For many years, researches in the simulation domain have contributed to the development of the RT-level test. These researches concerns fault models [8, 13, 12], fault simulators [2, 6] and testability or test generators analysis [7]. The main problem of the fault simulation is the difficulty to integrate these classical algorithms inside the HDL simulators. This problem comes from the complexity and the particular goals of the HDL languages which are languages for hardware de-

scription. Even if these algorithms are very well known for many years, commercial tools do not integrate them. Our approach proposed in this paper shows that, using a VHDL behavioral description transformation in the DEVS formalism, we can easily integrate these complex simulation algorithms and simulate the digital circuits faulty behavior in an efficient and concurrent way.

Fault injection in HDL descriptions existing approaches can be based on the modification of the HDL code [2] or on the modification of the simulator or its interaction [5]. However the coding lines that are added during this modification imply an increase of the simulation time, but also an access to the source code of the simulator. On the other hand, the interactive method implies a perfect knowledge of the very complex simulation process. In our approach, the VHDL code is transformed but not modified, and no interaction with the simulation engine is needed thanks to the DEVS abstract simulator derived directly from the whole model. As described in [10], each VHDL instruction is transformed in a DEVS component able to present an healthy or faulty behavior following a given fault model. The interconnection composed by all the DEVS components is directly simulable using the BFS-DEVS simulator.

We chose the fault model used in [8, 4] based on the stuck-at fault model allowing a good correlation with the real faults. We chose also the concurrent fault simulation since this approach is fast and uses complex simulation algorithms allowing us to show how easy is the integration inside the BFS-DEVS methodology.

This article is organized as follows. Section 2 is devoted to the presentation of the DEVS discrete event modeling and simulation formalism. Section 3 presents the fault simulator with the general simulation space architecture, the transformation of VHDL descriptions into DEVS components network, and the BFS-DEVS simulator. We provide the fault model and the fault simulation approach in Section 4, some experimental results in Section 5 and we conclude and give some perspectives of work in Section 6.

THE DEVS FORMALISM

Since the seventies, some formal works have been directed in order to develop the theoretical basements for the modeling and simulation of dynamical discrete event systems [14]. DEVS (Discrete Event system Specification) has been introduced as an abstract formalism for the modeling of discrete event systems, and allows a complete independence from the simulator using the notion of abstract simulator.

DEVS Modeling

DEVS defines two kinds of models: *atomic models* and *coupled models*. An atomic model is a basic model with specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. Coupled models tell how to couple several component models together to form a new model. This kind of model can be employed as a component in a larger coupled model, thus giving rise to the construction of complex models in a hierarchical fashion [15]. As in general systems theory, a DEVS model contains a set of states and transition functions that are triggered by the simulator.

A DEVS atomic model AM with the healthy behavior is represented by the following structure :

$$AM^h = \langle X^h, Y^h, S^h, \delta_{int}^h, \delta_{ext}^h, \lambda^h, t_a^h \rangle$$

where:

- $X^h : \{(p, v) | (p \in \text{input ports}, v \in X_p^h)\}$ is the set of input ports and values for the reception of healthy external events,
- $Y^h : \{(p, v) | (p \in \text{output ports}, v \in Y_p^h)\}$ is the set of output ports and values for the emission of healthy events,
- S^h : is the set of internal sequential healthy states,
- $\delta_{int}^h : S^h \rightarrow S^h$ is the healthy internal transition function that will move the system to the next healthy state after the time returned by the time healthy advance function,
- $\delta_{ext}^h : Q \times X^h \rightarrow S^h$ is the healthy external transition function that will schedule the states changes in reaction to an input healthy event,
- $\lambda^h : S^h \rightarrow Y^h$ is the healthy output function that will generate external healthy events just before the healthy internal transition takes places,
- $t_a^h : S^h \rightarrow \mathbb{R}_\infty^+$ is the healthy time advance function, that will give the life time of the current healthy state (*returns the time to the next healthy internal transition*).

The dynamic interpretation is the following:

- $Q = \{(s, e) | s \in S^h, 0 < e < t_a^h(s)\}$ is the total state set,

- e is the elapsed time since last transition, and s the partial set of healthy states for the duration of $t_a^h(s)$ if no healthy external event occur,
- δ_{int}^h : the model being in a healthy state s at t_i , it will go into s' , $s' = \delta_{int}^h(s)$, if no healthy external events occurs before $t_i + t_a^h(s)$,
- δ_{ext}^h : when an healthy external event occurs, the model being in the healthy state s since the elapsed time e goes in s' , The next healthy state depends on the elapsed time in the present healthy state. At every healthy state change, e is reset to 0.
- λ^h : the healthy output function is executed before an healthy internal transition, before emitting an output healthy event the model remains in a transient healthy state.
- A healthy state with an infinite life time is a passive healthy state (*steady healthy state*), else, it is an active healthy state (*transient healthy state*). If the healthy state s is passive, the model can evolve only with an input healthy event occurrence.

The DEVS coupled model CM is a structure :

$$CM = \langle X, Y, D, \{M_d \in D\}, EIC, EOC, IC \rangle$$

where:

- X is the set of input ports for the reception of external events,
- Y is the set of output ports for the emission of external events,
- D is the set of components (coupled or basic models),
- M_d is the DEVS model for each $d \in D$,
- EIC is the set of input links, that connects the inputs of the coupled model to one or more of the inputs of the components that it contains,
- EOC is the set of output links, that connects the outputs of one or more of the contained components to the output of the coupled model,
- IC is the set of internal links, that connects the output ports of the components to the input ports of the components in the coupled models.

In a coupled model, an output port from a model $M_d \in D$ can be connected to the input of another $M_d \in D$ but cannot be connected directly to itself.

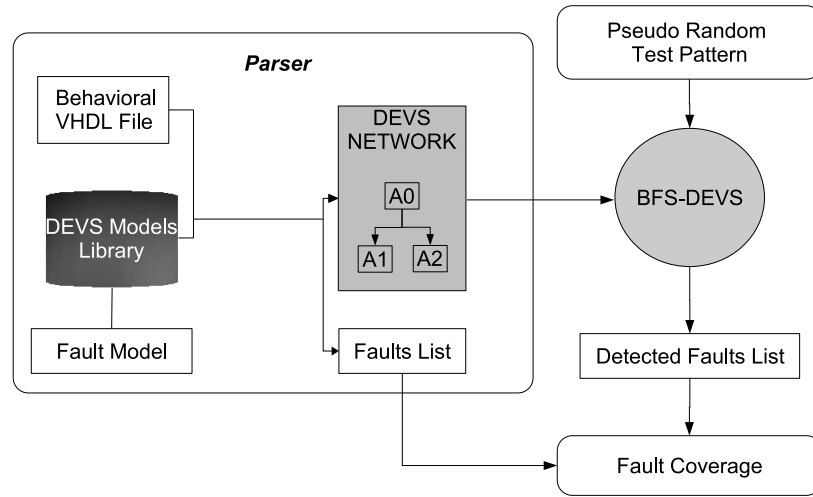


Figure 1: Data flow of the proposed approach for behavioral fault simulation.

DEVS Simulation

The DEVS abstract simulator is derived directly from the model. A simulator is associated with each atomic model and a coordinator is associated with each coupled model. In this approach, simulators allow to control the behavior of each model, and coordinators allow the global synchronization between each of them. The communication between all these elements is performed using four kinds of messages. The initialization messages (i, t) are used to achieve an initial temporal synchronization between all actors. The internal transition messages $(*, t)$ allow the processing of an internal event, while the external transition messages (x, t) allow the processing of an external event. Finally, the output messages (y, t) allow the transportation of the output values to the parent elements and is the result of an $(*, t)$ message.

THE BFS-DEVS BEHAVIORAL FAULT SIMULATOR

General Architecture

For the validation of our approach proposed in this article, we developed a simulation environment shown in Figure 1 and composed by:

- A *VHDL description to DEVS components transformation module* achieved using a parser that transforms each VHDL instruction (describing the healthy behavior of the digital circuit) in a DEVS representation, store these components in a models library and generates the list of all faults that could happen inside the circuit. The resulting network of components is an easy-handling

oriented instruction graph. The behavioral fault model will be later integrated in the models specifications.

- A *pseudo-random test pattern generator* providing the stimuli that will be applied to the circuit following the knowledge of the faults to be simulated. In order to validate our approach, we chose to use a pseudo-random test pattern generator without any knowledge of the faults to be tested.
- A *DEVS-based behavioral fault simulator* called *BFS-DEVS* based on one of the main advantages of the DEVS formalism that is its ability to generate automatically the simulator from the models. Our simulator needs the DEVS models from the transformation but also input data from the pseudo-random test pattern generator. Since the fault model is integrated inside each atomic model, the faulty behavior of the whole system is simulable. This fault simulation is performed in order to define the quality of a test pattern. This quality is described by a coverage rate which is the number of detected faults by the number of simulated ones.

VHDL into DEVS Transformation

The first step for the fault simulation is the transformation of each VHDL instruction into a DEVS model in order to define a directly simulable network. This transformation presents many advantages:

- DEVS allows to model and simulate structural languages;
- We can easily insert a behavioral fault model in an atomic model;

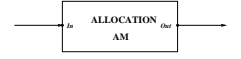
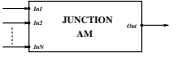

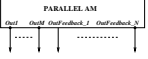
VHDL Instruction	Allocation	Selection	Conditional	Process
Atomic Model				

Table 1: Mapping of the VHDL instructions with DEVS components

- We obtain the fault simulator directly from the DEVS model;
- DEVS allows a hierarchical and component-oriented modeling for a better reusability of the fault models.

The selected approach for the transformation of VHDL behavioral descriptions with sequential instructions relies on the association of each VHDL instruction with a DEVS atomic model and each VHDL process with a coupled model. We define four kinds of models described in Table 1: The *Allocation atomic model* is mapped to the VHDL allocation instruction, the *Junction atomic model* to the selection instructions (end if, end case, end loop...) and allowing to make a junction between atomic models, the *Conditional atomic model* to the if, for, case... instructions, and the *Parallel atomic model* to the management of the processes.

In the simple example provided in Figure 2, we can see how the sequential instruction flow is transformed in an interconnection of atomic models encapsulated in a coupled model MC representing the following process:

```

process2: process(sensitive signal list)
begin
if (conditional statement 1)
allocation statement 1
elsif (conditional statement 2)
allocation statement 2
else (conditional statement 3)
allocation statement 3
end if;
end process;

```

The behavioral fault simulation using a fault list propagation will happen in this network. The Generator atomic model is used only for the activation events generation for the process.

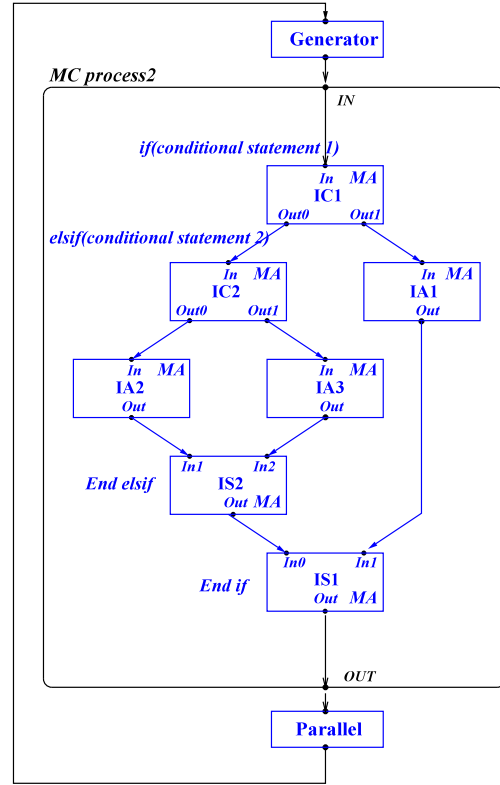


Figure 2: Example of a VHDL to DEVS transformation

BFS-DEVS Simulation

The basic principles of our approach can be sum up in some steps:

- The Generator model build the initial fault list composed by the faults relative to the sensitive signals of the processes;
- This list is propagated inside the network of all atomic models. The list is modified by the nature of the atomic model (and so, by the nature of the VHDL instruction) and the nature of the traversed path;
- When the fault simulation is achieved, the Parallel component analyzes the faults observability. If a fault is observable, it is added to the detected fault list and is no longer propagated in order to speed up the simulation process.

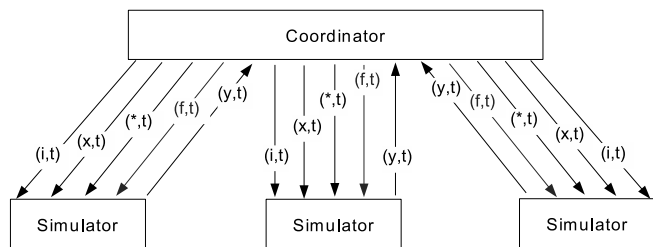


Figure 3: BFS-DEVS simulator architecture

The modifications we performed inside the DEVS simulation engine allow us to concurrently simulate the digital circuits described using the DEVS formalism. These modifications are the following: The integration of a fault model inside a DEVS atomic model is performed through the modification of the δ_{ext} and δ_{int} functions, and/or the λ function. The faulty behavior of an atomic model is managed using a new external faulty transition function $\delta_{fault} : S \times \mathcal{R}_0^+ \times X_f \rightarrow S$ as proposed in [9] activated by the reception of a new kind of messages called (f, t) .

The faulty behavior of a DEVS atomic model is activated by a (f, t) message associated with a faulty external event as shown in Figure 3. The coordinator, using its X input event set, is then able to send two kinds of external events: an (x, t) message for an healthy behavior, and a (f, t) for a faulty behavior and the simulation. Using this distinction, we are able to achieve concurrently the healthy and faulty simulations. Indeed, the fault simulation can only be performed if the healthy simulation has been already performed. In this case, the healthy values are known and can be used as a reference for the fault simulation.

The simulator associated with the DEVS atomic model receiving an external event x (and supposed to present a faulty behavior) receives two messages: an (x, t) message (for the healthy simulation) followed by a (f, t) message (for the faulty simulation). If the DEVS model receives a faulty external event x_f the associated simulator will only receive a (f, t) message from its parent coordinator.

FAULT MODEL AND FAULT SIMULATION APPROACH

The fault model used in order to validate our approach is inspired by the works presented in [11, 8]. We consider only the permanent faults acting on the logical function of the circuit. We defined three kinds of faults which can affect the behavior of the DEVS models:

- The F1 fault type: these are stuck-at faults on signals

and variables corresponding to the fault type number 1 described in [8]. The signal (or the variable) get a forced value until its next variation. Following the Table 1, this kind of faults can appear inside the four kinds of models.

- The F2 fault type: These are branch stuck-at faults on control structures corresponding to the fault type number 2 described in [8]. The result of the conditional instruction is forced to a boolean value during the simulation cycle. Following the Table 1, this kind of faults can appear in conditional models.
- The F3 fault type: These are the jump faults on instructions corresponding to a part of the fault type number 5 described in [8]. The affectation instruction is not evaluated if the signal value must be updated. Following the Table 1, this kind of faults can appear in allocation models.

We can note that these faults are behavioral ones, and their effects are taken into account in the atomic models behaviors. If the chosen fault model presents some structural faults, their effects will only be taken into account in the specification of the coupled models.

The selected simulation approach is concurrent. Even if an expensive memory space is needed, this approach presents many advantages compared to the parallel, deductive or serial approaches:

- we can apply this approach at the behavioral level, and then we can integrate it easily in the DEVS formalism;
- this approach is fast. The fault propagation process inside the DEVS component allows the determination of multiple faults in a same time. There is no need to perform a simulation for each fault. Moreover, the propagated lists are reduced during the simulation;
- We can perform concurrently the simulation of the healthy and the faulty circuit. This propriety is completed by the use of the DEVS formalism allowing the parallelization of these two kinds of simulations.

EXPERIMENTAL RESULTS

The implementation of our approach has been performed using the PythonDEVS simulator proposed by [3]. The validation has been performed using the VHDL ITC'99 benchmarks [1]. Table 2 presents our results. The total number faults are simulated using a pseudo-random test pattern generator that generates 500 vectors. The high coverage rates obtained using a simple pseudo random test bench show that we are able to predict the random testability of a circuit very early in the design process.

	Total Faults	Test Sequence Length	Detected Faults	Coverage (%)
B01	79	54	73	92.40
B02	48	56	41	85.45
B03	134	81	125	93.28
B04	105	72	95	90.47
B05	145	102	131	90.34
B06	95	172	82	86.31
B07	76	63	63	82.89
B08	64	70	53	82.81
B09	68	57	60	88.23
B10	163	110	150	92.02

Table 2: Fault Simulation Results for ITC'99 benchmarks

CONCLUSIONS AND PERSPECTIVES

This article proposes a new behavioral faults modeling and simulation approach of digital circuits described in the VHDL language. This approach is based on the DEVS formalism for the modeling and the simulation of complex discrete event systems. The presented BFS-DEVS simulator allows to simulate the circuits in their healthy configuration, but also with behavioral faults thanks to a concurrent simulation without any modification to the VHDL code. The experimental results performed on the ITC'99 benchmarks show the validity of our approach.

We have three main perspectives to this work. We want to implement a whole test environment with the integration of an automatic test pattern generator, to add the management of structural faults and to generalize this approach to any kind of complex discrete event systems.

References

- [1] Benchmarks included in itc'99 suite, 1999. <http://www.cad.polito.it/tools/bench>.
- [2] F. Franco A. Fin. A vhdl error simulator for functional test generation. pages 390–395, 2000.
- [3] J.S. Bolduc and H. Vangheluwe. A modeling and simulation package for classic hierarchical devts. Technical report, 2002.
- [4] F. Buonanno, F. Ferrandi, and D. Fummi. How an evolving fault model improves the behavioral test generation, 1997.
- [5] F. Corno, G. Cumani, M. Reorda, and G. Squillero. Rt-level fault simulation techniques based on simulation command scripts, 2000.
- [6] Farzan Fallah, Srinivas Devadas, and Kurt Keutzer. OCCOM: Efficient computation of observability-based code coverage metrics for functional verification. In *Design Automation Conference*, pages 152–157, 1998. citeseer.nj.nec.com/fallah98occom.html.
- [7] F. Ferrandi, F. Fummi, and D. Sciuto. Implicit test generation for behavioral VHDL models. pages 587–596. citeseer.nj.nec.com/ferrandi98implicit.html.
- [8] S. Ghosh and T.J. Chakraborty. On behavior fault modeling for digital designs. In *Journal of Electronic Testing : Theory and Applications*, pages 135–151, 1991.
- [9] E. Kofman, N. Giambiasi, and S. Junco. Fdevs: A general devts-based formalism for fault modeling and simulation. In *Proceedings of the ESS 2000*, 2000. Hamburg, Germany.
- [10] F. Bernardi D. Federici L. Capocchi, P.A. Bisgamiglia. Transformation of vhdl description into devts models for fault modeling. In *IEEE International Conference on Systems, Man and Cybernetics*, 2003.
- [11] A.L. Courbis N. Giambiasi, J.F. Santucci. Test pattern generation for behavioral description in vhdl. In *Euro-VHDL' 91*, pages 228–235, 1991.
- [12] T. Riesgo and J. Uceda. A fault model for vhdl descriptions at the register transfer level, 1996. citeseer.nj.nec.com/riesgo96fault.html.
- [13] K. Keutzer S. Devadas, A. Ghosh. An observability-based code coverage metric for functional simulation. 1996.
- [14] B.P. Zeigler. *Theory of Modeling and Simulation*. Academic Press, 1976.
- [15] B.P. Zeigler, H. Praehofer, and T.G. Kim. *Theory of the Modeling and Simulation, 2nde Edition*. Academic Press, 2000.