

## OLIVIA: Objectoriented Logicsimulation Implementing the VITAL Standard

Josef Fleischmann, Rolf Schlagenhaf, Martin Peller, and Norbert Fröhlich  
Institute of Electronic Design Automation  
Technical University of Munich  
D–80290 Munich, Germany

### Abstract

*In a VHDL-based design flow for application specific integrated circuits, VITAL provides a uniform methodology for developing ASIC libraries for signoff simulation. The VITAL Standard includes specialized routines for describing behavior and timing of ASIC cells and integrates back-annotation via Standard Delay Format (SDF). One of the key issues of the VITAL initiative was to accelerate simulation performance at gate level by allowing only a restricted set of VHDL. In this paper, we present an efficient implementation of the VITAL-Standard in our objectoriented, event-driven logic simulation tool OLIVIA. First promising results concerning simulation performance compared to conventional VHDL-Simulators are given.*

### 1 Introduction

The specification and design of digital systems is continuously moving towards higher levels of abstraction. This is due to the increasing availability of automated or semi-automated EDA-tools and the growing acceptance of hardware description languages like VHDL or Verilog. Nevertheless, timing simulation of the final netlist at logic level is still an imperative step in the ASIC design cycle before sign-off.

In the past, there has been no uniform methodology for sign-off simulation based on VHDL. This was due to several factors:

- As a matter of fact, there has been no industry-accepted methodology for modeling asic cells in VHDL.
- The timing model provided by VHDL was not accurate enough for sign-off.
- Simulation performance at gate level was too low compared to dedicated logic simulation engines.

Because of the lack of ASIC libraries, designers using VHDL had to switch to Verilog or use specialized gate level simulators for sign-off simulation.

During the VITAL Initiative (VHDL Initiative Towards Asic Libraries), major semiconductor suppliers and EDA tool vendors worked together on the VITAL model development specification [IEE96] which has been adopted by the IEEE (IEEE 1076.4) in December 1995. The goal of this standard is to provide a uniform and efficient modeling style with sufficient timing accuracy for sign-off simulation based on VHDL. While enabling a uniform methodology for developing ASIC libraries, the focus was also on significantly improving simulation performance at logic level.

Since its standardization, the VITAL concept has been integrated into numerous commercial VHDL simulation tools. The VITAL package can be used with any VHDL simulator, as it is entirely written in VHDL. But for improving performance of the simulation, the simulation kernel itself has to be modified in order to take advantage of the VITAL restrictions. The potential for faster simulation stems from the fact that only well defined timing procedures have to be used when describing an ASIC cell and the possible modules for calculating delays and checking for timing violations have to appear in a strictly defined order.

Our approach for improving simulation performance at gate level differs from the VHDL simulator vendors approach, as we do not start from a fully-fledged VHDL simulation system and try to accelerate the VITAL subset for the simulation of netlists. The development of OLIVIA (Objectoriented Logicsimulation Implementing the VITAL Standard) was rather based on the dedicated logic level simulation tools LDSIM [KA90, Kro89] and OSIM [Sch93].

The paper is organized as follows: In section 2, we give a short introduction into the VITAL Modeling Specification for those who are not yet familiar with this new standard. In section 3 we highlight some details about the implementation of the new objectoriented logic simulation tool. First results from simulated benchmark circuits and a performance comparison to a commercial VHDL/VITAL simulator are shown in section 4. In the final section, some

concluding remarks and an overview of our future work is presented.

## 2 VITAL Modeling Style

Within a VHDL-based design flow, the VITAL Standard establishes a uniform format for the description of library cells for timing verification. In more detail, the following aspects are covered by the VITAL concept:

- Model development guidelines for ASIC cells and restriction of the applicable set of VHDL
- Timing routines for defining temporal behavior and timing checks for detecting and controlling timing violations
- Primitives for modeling functionality of cells
- Standardized generic parameters for backannotation of layout dependent delay parameters via SDF (Standard Delay Format)

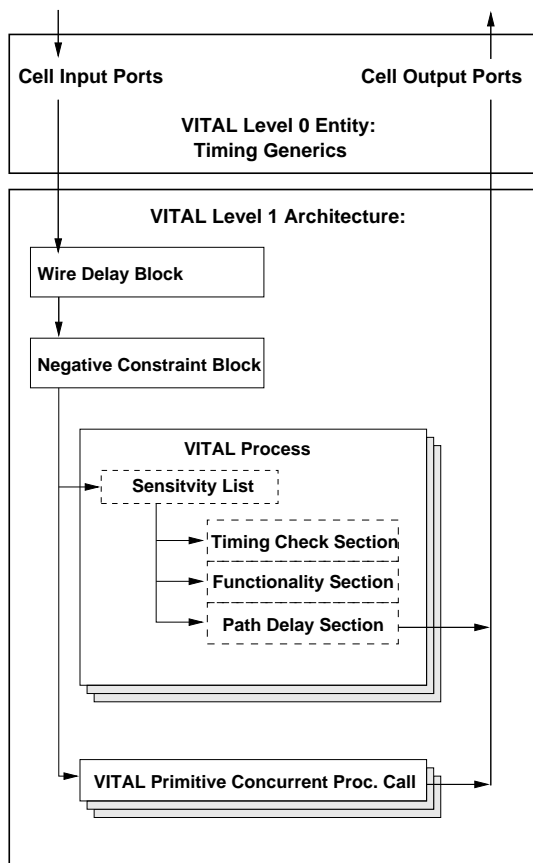


Figure 1. Structure of a VITAL-Model [IEE96]

In Figure 1 an overview is given of the building blocks of a VITAL library cell and the order in which they have to be specified. There are two levels of modeling in VITAL (a tutorial introduction to VITAL can be found in [Lev95, Sch95]): VITAL level 0 rules are restricted to the entity interface and these rules include a fixed variable naming convention for timing generics in order to enable SDF backannotation. VITAL level 1 guidelines apply to the architecture of the model: Here it is possible to model wire delays, negative constraints, timing checks, functionality and individual path delays in the depicted order using the procedures and functions from the VITAL package. The set of applicable VITAL timing modules and the logical order in which they have to be specified is strictly defined. This fact can be exploited for a more efficient implementation for the sake of speeding up gate level simulation.

## 3 Implementation

As a starting point for the new implementation, the object-oriented simulation tool OSIM [Sch93] was used. OSIM contains classes for implementing an event-driven simulator: Events, event queue administration, boolean primitives, signal models, input and output modules. It is based on a specific logic modeling specification originally presented in [Kro89].

During the development of OSIM, one requirement was to decouple the simulation kernel from the model of the circuit to be simulated. This was done by using a strict object-oriented approach, thus facilitating later reuse of the simulation kernel for various applications.

Within the event-driven logic simulator OLIVIA, there are three main groups of objects which are instantiated during the simulation:

- Objects, which represent the circuit to be simulated
- Event objects, which propagate information during the simulation run
- Objects, which implement the simulation cycle

Due to the object-orientation of OSIM it was possible to implement OLIVIA in a relatively short period of time. A more detailed description of the different logic modeling styles in OSIM and VITAL and implementational details are given in [Pel96]. OSIM and OLIVIA are very good examples for software reusability and code readability when strictly adhering to an object-oriented programming paradigm.

### Objects for Representation of the Circuit

During simulation, the circuit is represented by a number of various communicating simulation objects. These

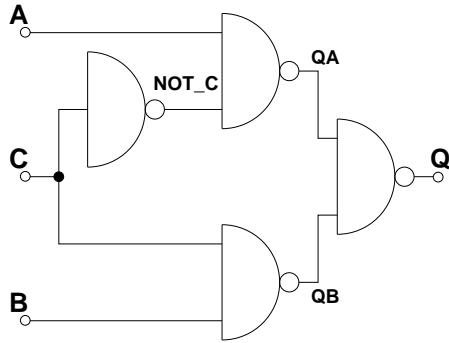


Figure 2. Example circuit

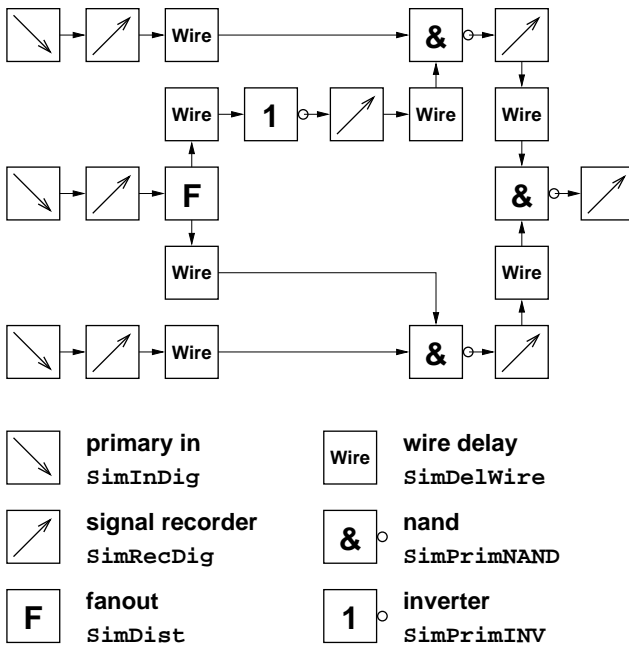


Figure 3. Internal Representation in the Simulator

are dynamically created and connected to each other at the beginning of a simulation run. Unfortunately, the classes for circuit representation of OSIM could not be used for OLIVIA, because the element and signal modeling of OSIM and VITAL differ. Nevertheless, several base classes could be used as a starting point for refinements, thus reducing the required time for developing the new tool.

All components of the circuit model are instances of different classes derived from the uppermost base class SimObj. Therefore, they can all be mutually connected corresponding to the circuit netlist. By these connections, signal changes can be propagated to succeeding

objects. This way, component evaluations are triggered by preceding circuit elements and data is exchanged. Thus, there is quite a similarity between the natural signal flow in a circuit and its realization in the simulator. There are special objects which represent timing related behavior of the circuit. They can generate events, which are scheduled by the event administration and executed by the SimObj itself later. By the data hiding mechanism of the object-orientation paradigm, the function of a component is totally hidden within itself. Therefore, the objects can have very different functions without mutual interference. In OLIVIA for example, there exist components for wire delays (SimDelWire), gates (e.g. SimPrimAND) and state tables (SimPrimStateTable) besides others. Different tasks are realized within the components, for example setup and hold time check (VitalSetupHoldCheck) and appropriate selection of path delays (VitalPathDelay01).

Figures 2 and 3 show, how a circuit is represented by a set of objects within the simulator. Each rectangle is an instance of a SimObj. Besides the already mentioned components for wire delays and gates, objects for stimulation of primary input signals, recording of signals and fanout representation are depicted.

### Event Objects

Corresponding to the discrete event simulation paradigm, each non-immediate signal change within the circuit must be coded in an event object. They are all dynamically generated during the simulation run, then executed and finally deleted. Due to the significantly different signal modeling concept in VITAL, new event classes became necessary. They were implemented by inheritance from existing base classes for events. Therefore it was possible to reuse big parts of the complex event administration of OSIM for OLIVIA.

Concerning event handling there is a difference between OLIVIA and other VHDL simulators: The normal preemption mechanism of VHDL is substituted by an alternative which is more suitable for future parallelization of the simulator. In OLIVIA, invalidated events are not preempted immediately when the most recent event is inserted in the event administration. Instead, they are deleted just before their originally scheduled execution. Normal preemption would destroy the lookahead [LL90] of the simulation model, which is a basic condition for the parallelization method TIME WARP [Jef85]. The alternative method realized in OLIVIA keeps the lookahead, but does not have any impact on the simulation output. In order to implement this method, the normal VHDL event had to be extended by two values:

- Generation tick of the event  $t_{Gen}$

- event type (INERTIAL or TRANSPORT)

The distinction between INERTIAL and TRANSPORT events is necessary only for special situations during glitch detection. Normally, a component generates either one or the other type of events during the whole simulation run corresponding to its VITAL description.

$t_{Gen}$  is the more important value for correct event handling. The topology component, which creates an event, has a tick counter. It is increased at every VHDL delta cycle. When generating a new event, the topology component sets  $t_{Gen}$  of the new event to the same value as its own tick counter. This value is used later when the event is scheduled. Then we decide whether this event is still valid and must be executed or if it has to be ignored because of causality reasons. The decision is done by comparing the event tick with a second reference tick  $t_{RefExe}$  within the topology object. If the event is valid and executed,  $t_{RefExe}$  is set to the tick value of the event.

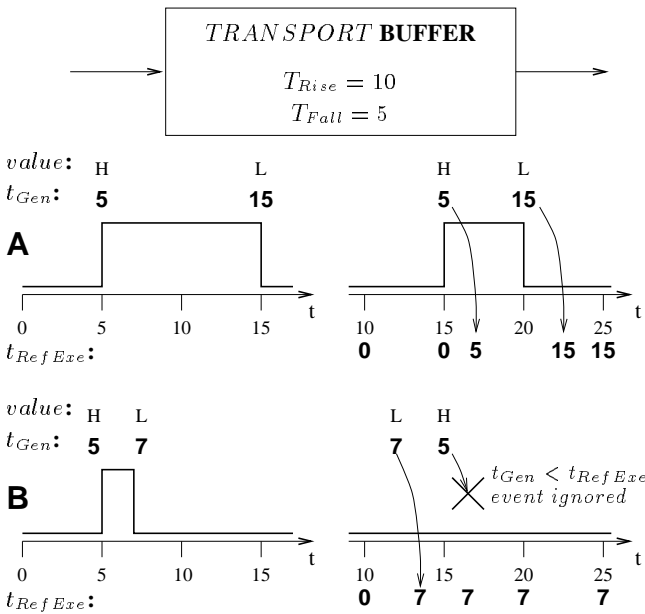


Figure 4. Event Cancellation

Figure 4 shows an example using a simple buffer with the delays  $T_{Rise} = 10$  and  $T_{Fall} = 5$ . In case **A** the event ordering ('1','0') during generation and execution is the same and no preemption or correction is necessary. In case **B**, the event ordering changes. In VHDL event '1' is deleted by preemption, when event '0' is generated. In our approach, event '1' is cancelled just before its execution, because  $t_{Gen}$  of the event is smaller than  $t_{RefExe}$  of the buffer object.

## Objects Implementing the Simulation Cycle

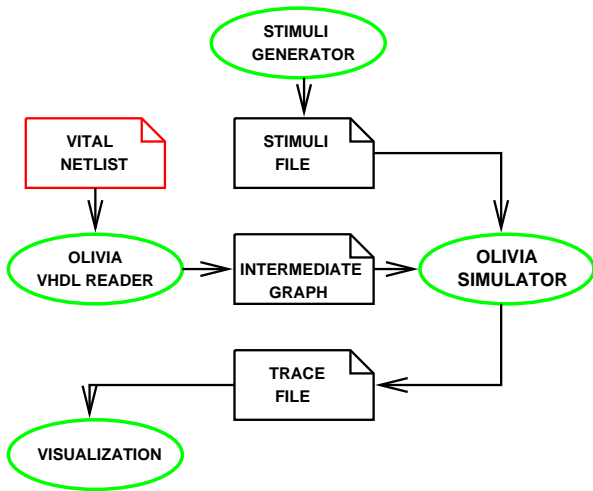
Additionally, there are some objects which implement the simulation cycle itself. The most important ones to mention are the object which builds and manages the circuit topology, and the object for event administration (event queue). There is only one instantiation of each of these objects and their lifetime lasts over the whole simulation run. This set of objects represents the basic functionality of the simulator kernel. Although a slightly different simulation cycle had to be implemented for simulating VHDL, software reuse of major parts formerly used in OSIM was possible.

## 4 Results

In order to verify the correct functionality of the new simulator OLIVIA and to be able to compare its performance to commercial VHDL simulation systems, we simulated different benchmark circuits. For this purpose, we used three combinational circuits from the LGSYNTH91 [Yan91] benchmark set (C17, ALU2 and C6288) and a locally developed sequential circuit (ADD4). All simulation experiments described in this section were conducted on a single-processor DEC ALPHA 250 4/266 running DIGITAL UNIX V3.2. Our simulator OLIVIA was compiled with GNU G++ and all experiments with commercial simulators were done in compiled mode.

An overview of the experimental simulation environment is shown in Figure 5. The VHDL circuit netlist is read by a VHDL reader, which has been developed using the GNU BISON and FLEX tools. The reader converts the circuit description into an intermediate directed graph containing signal and component nodes. This intermediate graph structure is used by OLIVIA to build all necessary simulation objects as shown for the small example in Figure 3. For generating stimuli, we used a simple program for generating random vectors. A primitive backend is employed for processing the outputfile produced by OLIVIA and visualize the simulation results. A graphical user interface has not been implemented yet.

During simulation we applied randomly generated test vectors to the circuit inputs. In each experiment, we traced a set of internal signals and all output signals and we got the same simulation results on OLIVIA and the commercial VHDL/VITAL simulator. The performance figures shown in Figure 6 and 7 are mean execution times obtained by five independent simulation runs. The measured cpu time in experiments with OLIVIA include the time for parsing the VHDL circuit description and the translation to an intermediate graph, the time to read the stimuli file, the time for circuit simulation and for writing the trace file.



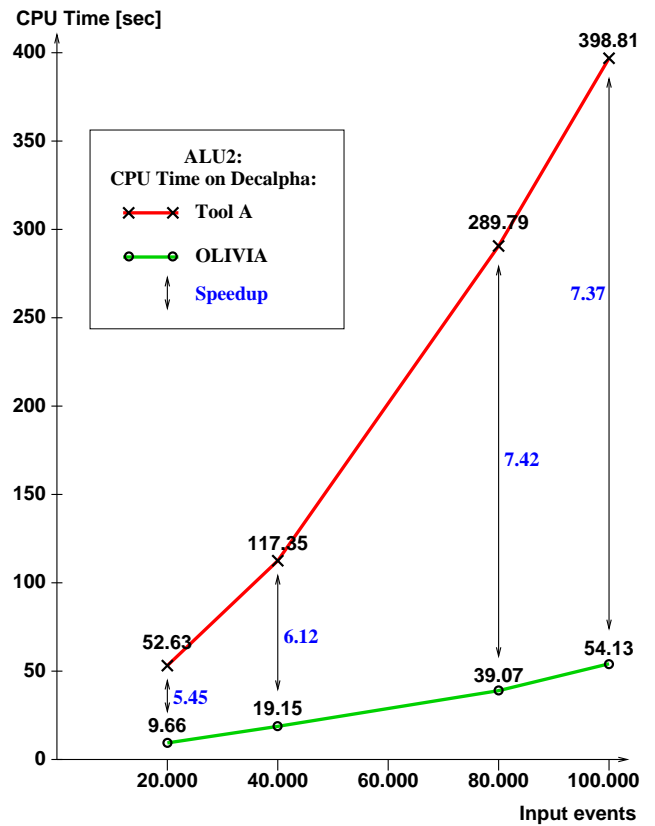
**Figure 5. Experimental Simulation Environment**

First, we did some investigations on the relation between simulation runtime and number of input vectors. In Figure 6 it can be seen, that for our example ALU2 there is a near linear correlation between execution time and number of vectors for both the commercial tool and OLIVIA. In our experiments we observed that the nonlinear tendency increases with the size of the circuit and the number of test vectors. This effect may most likely result from sideeffects like the large size (> 20 Mb) of the simulation output files.

In Figure 6 you can also see the significant difference in performance between our new simulation tool OLIVIA and the commercial state-of-the-art simulator TOOL A<sup>1</sup>: When simulating the combinational circuit ALU2 we experienced an increase in simulation speed by a factor between five and seven depending on the number of input vectors.

An overview of our first performance results with the benchmarks simulated up to now is depicted in Figure 7: We obtained a really significant speedup for all circuits simulated yet. Even more interesting is, that the speedup obtained by OLIVIA seems to increase with larger circuit sizes and larger sets of input vectors. For a more detailed performance investigation we are now working on simulating more and larger benchmark circuits. Furthermore, we are currently conducting experiments with other commercial VITAL optimized simulators. Our preliminary results indicate that simulation tool OLIVIA outperforms TOOL B – which is said to be one of the fastest VHDL/VITAL simulators at the moment – by a factor between two and three.

<sup>1</sup>For reasons of license agreements, we do not explicitly name any of the employed commercial simulation tools in this paper.



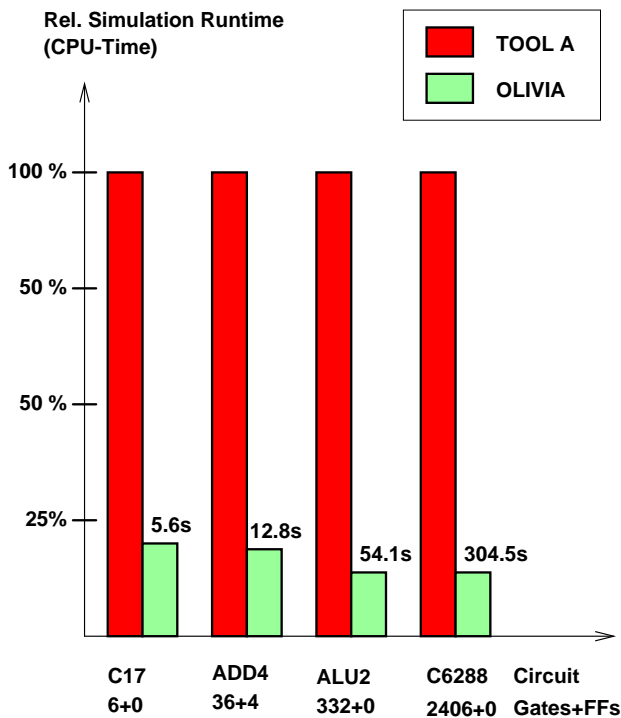
**Figure 6. Simulation Runtimes for Different Sets of Input Vectors**

## 5 Conclusion and Future Work

In this paper we presented a new objectoriented event-driven simulation tool for timing simulation at gate level implementing the VITAL Standard. The tool OLIVIA is currently under development and was originally designed to be used in a VLSI design course. Therefore, OLIVIA is not yet capable of parsing an arbitrary library of VITAL simulation models, but it works on a restricted design library consisting of a number of combinational and sequential primitives. Although there has been no fine-tuning of the code for performance yet, the simulation tool shows significant speedups compared to conventional VHDL simulation tools at logic level while producing the same simulation results.

The focus of current and future work is on the following topics:

- Further performance evaluation and comparison to commercial VITAL optimized simulators while simulating larger benchmark circuits



**Figure 7. Normalized Simulation Execution Times**

- Enhance capabilities of the frontend VHDL reader in order to be able to process hierarchical structural VHDL designs
- Implementation of a mechanism for backannotation of layout dependent delay information via SDF

During the conceptualization and implementation phase of OLIVIA special emphasis has been laid on the ability to port the simulator to a parallel environment later. There are already plans for a parallel VITAL simulator operating on a workstation cluster. This endeavor will be guided by our experience in parallel gate level simulation [SRSB95, FW95]

## References

- [FW95] Josef Fleischmann and Philip A. Wilsey. Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators. In *ACM/SCS/IEEE Workshop on Parallel and Distributed Simulation (PADS)*, pages 50–58, Lake Placid, New York, June 14-16 1995.
- [IEE94] IEEE Standards Board. *VHDL Language Reference Manual*, 1994. ANSI/IEEE 1076-1993.
- [IEE96] IEEE Standards Board. *IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification*, 1996. IEEE 1076.4-1995.
- [Jef85] D. R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.
- [KA90] H. T. Krodel and K. J. Antreich. An Accurate Model for Ambiguity Delay Simulation. In *European Conference on Design Automation (EDAC)*, pages 563–567, Glasgow, 1990.
- [Kro89] Hans Thomas Krodel. *Verfahren zur Logiksimulation komplexer digitaler Schaltungen mit flexibler Modellierung*. PhD thesis, Institute of Electronic Design Automation, Technical University of Munich, 1989.
- [Lev95] Oz Levia. Introduction to ASIC Cells Modeling with VITAL. In *VHDL Forum for CAD in Europe*, April 1995. Tutorial.
- [LL90] Yi-Bing Lin and Edward D. Lazowska. Exploiting Lookahead in Parallel Simulation. *IEEE Transactions on Parallel and Distributed Systems*, 1(4):457–469, 1990.
- [Pel96] Martin Peller. *Logiksimulation mit VITAL*. Master's thesis, Institute of Electronic Design Automation, Technical University of Munich, 1996.
- [Sch93] Rolf Schlagenhaft. *Objektorientierter Simulator für Logikschaltungen*. Master's thesis, Institute of Electronic Design Automation, Technical University of Munich, 1993.
- [Sch95] Steven E. Schulz. Introduction to VITAL '95. In *Mentor User's Group Meeting*, 1995. Tutorial Presentation.
- [SRSB95] Rolf Schlagenhaft, Martin K. Ruhwandl, Christian Sporrer, and Herbert Bauer. Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations. In *ACM/SCS/IEEE Workshop on Parallel and Distributed Simulation (PADS)*, pages 175–180, Lake Placid, New York, June 14-16 1995.
- [Yan91] Saeyang Yang. *Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0*. MCNC, Research Triangle Park, N.C. 27709, 1991.