

## Object Orientation in Time Warp Simulation

N. Fröhlich, R. Schlagenhaft, A. Ganz, and J. Fleischmann

Institute of Electronic Design Automation,

Technical University of Munich, 80290 Munich, Germany

Email: {froehlich,schlagenhaft,ganz,fleischmann}@regent.e-technik.tu-muenchen.de

**Abstract** *In this paper different simulation systems are presented, which have been developed with special emphasis on an object oriented programming paradigm. A set of classes for parallel Time Warp simulation was created. Using these classes, a sequential and based on this a parallel logic simulation system has been built. Moreover, a parallel simulation system with a specific application interface has been implemented. The most recent design using this set of classes is a sequential system for VHDL simulation on logic level according to the VITAL standard. All of these systems share classes and show the successful reuse of software in different applications. Object orientation was also very useful to achieve a clear software structure, which eases understanding and maintainance.*

**Keywords:** object orientation, sequential & parallel simulation, Time Warp

### 1 Introduction

There are many huge and complex simulation problems requesting lots of memory and computation power. One way to overcome pending resource bottlenecks is parallelization. There are already a lot of parallel simulation applications in the research community, but there is only little acceptance in industry yet. Usually parallel programs have considerable synchronization overhead compared to the corresponding sequential program, thus becoming relatively complex software packages. This is one reason for the limited use of parallel and distributed computation in industry. Furthermore, parallel programming requires certain

restrictions as data consistency or I/O management. Keeping large software packages understandable and maintainable is a major advantage of object orientation.

Motivation for the work presented in this paper was a parallel system for simulating circuits on logic level developed at our institute. Logic simulators are still the most important validation tools in the computer aided VLSI design process. Increasing circuit sizes and more precise element modeling continuously requires more computing resources. Therefore, a parallel system was developed, which was the first to achieve good speedups for logic simulation on a cluster of workstations [1]. But during the long lasting development process the software became very complex and difficult to maintain. Getting a clearer software structure, which eases understanding and maintainance, was one reason for the use of object oriented techniques. Furthermore, the valuable experience made in parallel simulation should be transferred to other applications than logic simulation. Therefore, exploiting modularity for using the same classes in different simulation systems was another reason for applying object orientation.

The first step towards object orientation was to completely redesign the sequential logic simulator. On this base the much more complex parallel logic simulation system was also redesigned. At the same time a Time Warp simulation system with a special interface to the application model was developed. The interface allows Time Warp simulation of arbitrary applications which comply with the interface specifications. The main difference to the logic

simulation system is, that the model of the application is not coded within the simulator. It is coded separately by the user of the simulation tool. Recently another simulator has been developed, which supports the VITAL standard. VITAL is an extension to the standard hardware description language VHDL for performing simulations with precise timing information on logic level. Up to now the VITAL simulator is only available in a sequential version, but a parallel version is under development.

The next section gives a short description of Time Warp. Then the sequential logic simulator OSIM and its extension the parallel logic simulation system OPSIM are presented. Further simulation packages using object orientation are the general parallel simulation system LANTW and the sequential VITAL simulation system OLIVIA.

## 2 Time Warp

For logic simulation with a precise timing analysis basically two different methods are known. The first one, called *compiler driven*, simulates the circuit in fixed discrete time steps. Since at each time step all the elements are evaluated, no scheduling mechanism for element evaluation is necessary. Choosing the interval size of the time steps is critical. Large intervals yield a poor time resolution. Small intervals cause many computations. Experiments showed that there are numerous time steps where no element is active. Even at those steps with activity there are only few elements active, but always all elements are evaluated. Therefore, it is more efficient to generate an event for each activity in the circuit. In *event driven* simulation, every event is related to only one element and has a time stamp marking the time it has to be processed. As it is possible to create events with arbitrarily small time distance, this results in a high time resolution during phases of high activity and during phases of no activity there are no computations. Event processing means, that only the active elements are

evaluated. A drawback of event driven simulation is the overhead necessary for event management.

For parallel event driven logic simulation several algorithms have been proposed. Using a cluster of workstations connected by a network, e.g. Ethernet, yields a parallel platform with high computational power at the processors and low communication performance between them. This favors a MIMD (multiple instruction on multiple data) parallel algorithm, i.e. the data is divided into parts and at each processor the same program is working on one part. Thus, for logic simulation the circuit has to be partitioned. As there are always cut signals between the divided parts, the individual simulators have to be synchronized.

Conservative approaches [2] enforce a strong synchronization between the individual simulators, i.e. before processing the next time step each simulator waits for all the other simulators to finish the current time step. This might lead to undesirable deadlock situations and reduces an efficient exploitation of the parallelism available in the application.

Optimistic approaches [3] allow each individual simulator to advance in its own virtual time under the assumption that there is no message from another simulator, which has a smaller time stamp. If there is a message causing a signal change in the past of the receiving simulator, this simulator has to roll back and restore the corresponding state in the past. To be able to restore past states these have to be saved during simulation. The event with the smallest time stamp in the whole simulation system marks the smallest possible time to roll back to. This time is called global virtual time (GVT). Therefore, all states for points in time, which are smaller than this time, can be deleted. For low memory consumption it is crucial to keep the GVT as actual as possible [4]. The optimistic approach has a considerable overhead for state saving and rollback handling. Much research has been done to reduce this overhead. Efficiency for logic simulation has been demonstrated [1][5][6][7].

## 3 The Simulators

### 3.1 Sequential Logic Simulator – OSIM

The first step towards object orientation was the implementation of the sequential simulator OSIM (object oriented **s**imulator) for logic circuits. The modeling of the circuit's logic and timing behavior is the same as in LDSIM (logic design **s**imulator), which was presented in [8]. In contrast to LDSIM, in OSIM the simulator code itself and the source code for the simulation model are decoupled as far as possible by means of object orientation (data hiding, polymorphism, inheritance). This is the base for a more flexible modeling in the future.

The basic ideas of discrete event simulation are realized in certain classes and abstract base classes. The main tasks of the simulator are independent of the model, which has to be simulated. The simulator has to build the model from a file description, invoke the model components, maintain events and input stimuli and record simulation results. Therefore the abstract base classes for model components have to provide methods for building a certain topology by mutually connecting them and virtual methods for model execution. The base class for events must enable the simulator to enqueue, maintain, schedule and start execution of them. Input and output is a minor problem in sequential simulation and is still realized by normal streams in OSIM.

Different models can be implemented by using the mentioned abstract base classes. The virtual member functions for model evaluation and event execution are overloaded by the behavior of the model to be implemented. In the case of OSIM, the circuit model used in LDSIM is fully coded in classes, which are derived from the virtual base classes for model components (**SimObj** in Figure 2, non-shaded area) and events (**SimEv**, without figure).

The presented classes can be split in three categories (compare with section 3.4). The first type is classes, which are used for representing the static topology of the model to be simu-

lated. They are all derived from **SimObj**. The second type is event classes. These are derived from **SimEv** and are used for the dynamic behavior of any model component. The third category contains all classes, which belong to the simulator kernel itself and are not related to the model (e.g. event administration).

In Figure 1 an example for the representation of a small logic circuit by objects within the simulator is given. There is an obvious similarity between the physical circuit and its model in the simulator. Also the way objects invoke methods of their successors during simulation (arcs in Figure 1) corresponds to the natural signal flow of the circuit. These similarities are typical for good object oriented modeling of a problem. The classes used in Figure 1 can all be found in the class hierarchy in Figure 2.

### 3.2 Parallel Logic Simulation System – OPSIM

OPSIM (object oriented **p**arallel **s**imulator) is based on the sequential object oriented logic simulator OSIM. As an advantage of object orientation, all classes modeling the topology of a logic circuit and its timing behavior as well as the simulator kernel can be reused. Only a few classes have to be extended by applying inheritance. In the following, we first describe both classes for the simulation model and the simulator kernel. Second, the additional classes for parallel Time Warp simulation are introduced.

In the shaded region of Figure 2, additional topology classes for parallel simulation are shown. One main requirement for topology elements is to restore circuit states of earlier simulation times. The reason is a rollback situation as described in Section 2. Therefore, a new abstract base class called **SimStateObj** is introduced. This class contains properties and virtual methods for storing and retrieving information of previous simulation times. We apply incremental state saving in order to keep memory requirements within reasonable ranges. Since changes of signal values can be stored within events, the events of already

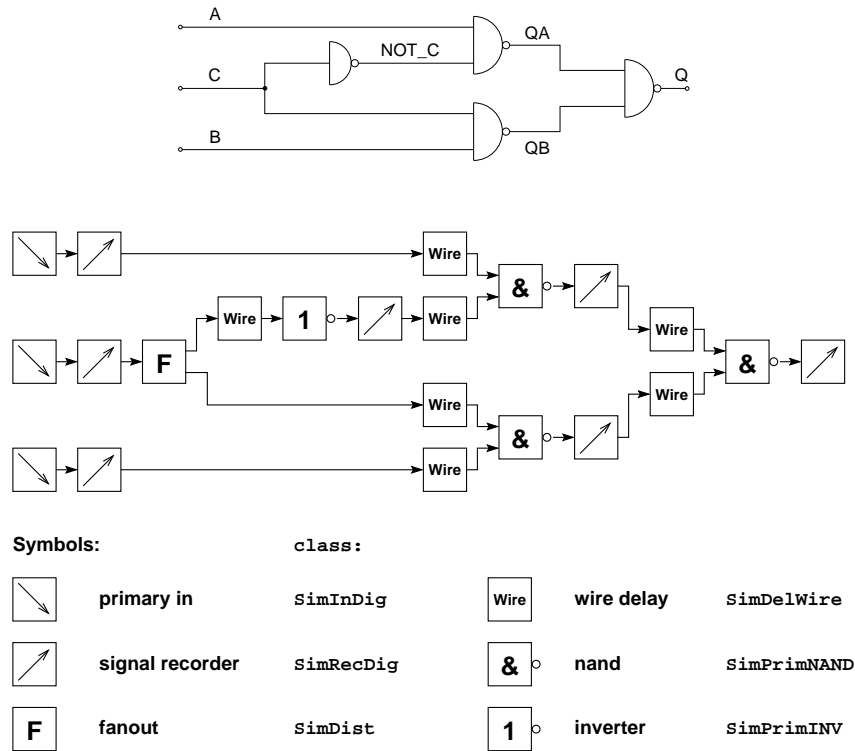


Figure 1: Example circuit and its representation in the simulator

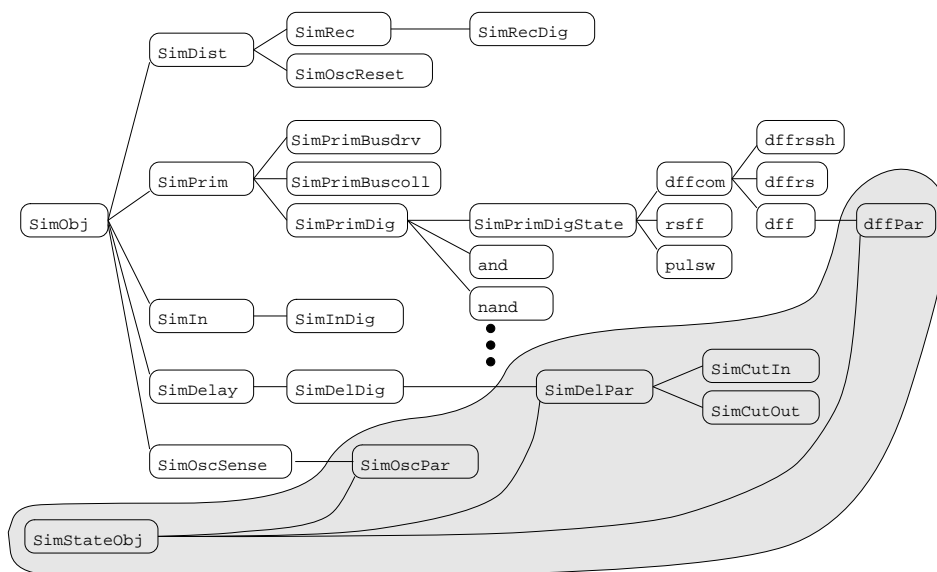


Figure 2: Hierarchy of topological classes

processed simulation times are not deleted but applied for state saving. This function-

ality is maintained by introducing the class `SimDelPar`, which is derived from the sequen-

tial delay class `SimDelDig` and the basic state saving class `SimStateObj`.

Accordingly, a new event class including state saving is also derived. These events additionally contain old signal values, which have to be restored in case of a rollback behind the time stamp of the event. Furthermore, new events are introduced solely for saving states of both flipflop objects and objects for oscillation detection. These two topological classes are derived from the ones of the sequential simulator `OSIM`, as shown in Figure 2. Events, which are not generated and executed in the same circuit partition, have to be sent as messages. The reason are cut signals as already mentioned in Section 2. Since old states also have to be saved for cut signals, the topological classes for sending (`SimCutOut`) and receiving (`SimCutIn`) events are derived from `SimDelPar`.

In the following, general classes for parallel simulation with the Time Warp method are introduced. In order to be flexible for future extensions, for all of these general classes there exist abstract base classes with virtual methods from which the specific ones can be derived. One example showing the advantages of object orientation is the communication object. The corresponding class provides virtual methods for blocking and non-blocking communication between simulators. A first version of our parallel simulator used P4 [9] for message passing. For switching to PVM [10], only a new PVM communication class had to be derived from the abstract communication base class.

The global control process is responsible for reading the circuit description and the stimuli, coordination of the simulators, as well as for producing the output of the simulation. Since these tasks are common for many parallel simulation applications, basic classes are introduced for all of this mentioned tasks, e.g. a communication class, a class for I/O-management, and so on. From base classes, those for the logic simulator `OPSIM` are derived. They are for example capable of reading and writing a format required by logic simulation. A central task in Time Warp simulation is the calculation of the

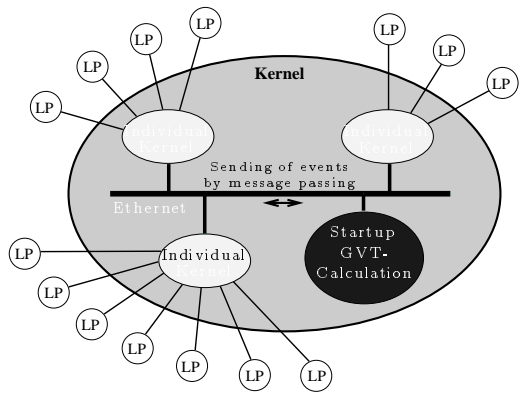
GVT. Two classes, a first for collecting necessary data and a second for performing the actual calculation were developed.

The speedups computed over a series of benchmark circuits proved the efficiency of our object oriented Time Warp simulator `OPSIM`. Accelerations of about 5 were achieved when running on 10 simulators. This is approximately the same result as reported in [4].

It can be summarized that for the extension of the sequential to a parallel simulator all classes for modeling the topology and the simulator kernel could be reused. A couple of new classes have been developed for parallel Time Warp simulation. The modularity of object orientation was used when applying some of those, e.g. GVT-classes and communication class, to another parallel simulator, which is presented in the following section.

### 3.3 General Parallel Simulation System – LANTW

Much research has been done to improve the Time Warp method to be efficient for parallel logic simulation. For showing that Time Warp is not only efficient for logic simulation, a transfer of the valuable experience to other applications was desired. Our cooperation partner at the University of Calgary investigates in Time Warp on a shared memory architecture. In the logic simulator the modeling of the application is hard coded in the simulator. In Calgary a strong separation between the application model and the simulation functions with a specific interface in between was proposed. Besides a clearer software structure this has the advantage of the possibility to exchange application models and simulator kernels. Thus we decided to use our Time Warp knowledge to develop a simulation system supporting the same interface, but to run on a cluster of workstations instead of a shared memory architecture as in Calgary. We call this simulation system for general applications LANTW (**L**ocal **A**rea **N**etwork **T**ime **W**arp). As an application we use models of communication networks developed by the University of Calgary and our sec-



dustrial acceptance. The reason for this situation was the lack of a widely accepted standard for modeling of ASIC cells for timing simulation at gate level. Though more and more designers have been using VHDL as a hardware description language in recent years, there has been no uniform methodology for sign-off simulation of digital circuits based on VHDL. This was caused by the fact that the timing model provided by VHDL was not accurate enough for reliable timing verification. Furthermore, VHDL simulation performance at the gate level was too low compared to dedicated logic simulation engines.

To tackle this problem, major semiconductor suppliers and EDA tool vendors came together in the VITAL Initiative (VHDL Initiative Towards Asic Libraries) and the resulting VITAL model development specification [15] has been adopted as IEEE Standard 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. In VITAL, a fixed set of applicable timing modules and the order in which they have to be specified when creating a library model is strictly defined. The imposed restrictions can be exploited for a more efficient implementation of a simulator. This results in considerable performance gains. A tutorial introduction to VITAL can be found in [16].

In order to implement a VITAL-compliant simulator both the definition of the simulation cycle in VHDL [17] and the modelling rules and timing routines of the VITAL specification have to be considered. As a starting point, we used the object-oriented logic simulator OSIM which is described in section 3.1. As written earlier, OSIM is based on a previously developed simulation and timing model [8] which is significantly different from the VHDL/VITAL simulation concept. For example, OSIM employs a two-phased simulation method which is based on the superposition of signal edge values

and allows for a min-max delay model, whereas OLIVIA is based on the VHDL simulation cycle and the 9-valued logic system `std_ulogic` (IEEE 1164).

Nevertheless, it was possible to implement the sequential simulator OLIVIA in a relatively short period of time. When developing OSIM, big emphasis was laid on adhering to a strict object-oriented programming paradigm and on separating the simulation kernel from the application model. Therefore, major parts of OSIM could be reused for the new VITAL simulator OLIVIA. Within the event-driven simulator OLIVIA, objects can be classified into three main groups:

1. **Objects, which implement the simulation cycle**

These objects implement the basic functions of the simulation kernel. There is an object, which builds and manages the structure of the circuit to be simulated. Another object is used for event administration during the whole simulation run. Although a slightly different simulation cycle had to be implemented for simulating VHDL, only very little modifications to the classes formerly implemented in OSIM had to be made.

2. **Objects, which represent the simulated circuit**

The simulated circuit is mapped to a set of communicating objects, which are dynamically created at the beginning of a simulation run. 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. Due to the different element modelling in VITAL, new classes for circuit components became necessary. But these new classes are all derived from the formerly designed class `SimObj`. Furthermore the required time for these changes was drastically reduced, as the former implementation was used as a skeleton for

the newly implemented simulation primitives.

### 3. Event Objects

In a discrete event simulator, all dynamic behavior of the simulated application model is represented by events which are triggered and evaluated by individual components of the model. For VITAL simulation, a new signal modelling concept had to be implemented. Therefore, a new class for events (*e.g.* for correct handling of path delays and preemption) became necessary. But for event administration and management of event queues all objects of the previous simulator OSIM could be reused.

Currently, a sequential version of a VITAL simulator is available and very promising performance results compared to commercial VHDL/VITAL simulators [14] have been obtained. Right now, the parallel version is under development. The concept for the parallel simulator is heavily based on OPSIM which has been presented in Section 3.2.

## 4 Conclusion

In this paper four simulation systems are presented, which benefit from the advantages of object oriented software development. All principles of object orientation as decomposition, abstraction and hierarchy have been applied. Decomposing the Time Warp simulation system by strictly using objects maintains Time Warp's abstract view of the real system. The interacting model elements are reflected in the software as cooperating objects, which eases understanding the software for the human way of thinking. This is also indicated by the fact, that researchers who are new to the software, get a thorough understanding of the simulation software obviously faster than it was the case with the former procedure oriented system.

The strong modularity of objects is very useful for software maintainance. For exam-

ple, when switching to another message passing system only a new communication class had to be derived from the communication base class. Experimenting with different algorithms for the event lists resulted in several new classes and now changing algorithms just needs changing the class type in the class instantiation call.

After the first simulation system OSIM was developed the other systems could take advantage of software reuse. Especially the parallel logic simulation system OPSIM and the VITAL logic simulator OLIVIA benefited from software modules of the sequential logic simulator. But even the general simulation system LANTW with its different modeling could reuse several classes from the previous systems.

To sum up, changing to object oriented software development techniques was a great benefit. Otherwise it would not have been possible to create the four presented simulation systems in the same relatively short time.

## References

- [1] Herbert Bauer and Christian Sporrer. Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving. In *SCS/IEEE Annual Simulation Symposium (ASS)*, 1993.
- [2] K. M. Chandy, J. Misra, and L. M. Haas. Distributed Deadlock Detection. *ACM Transactions on Computer Systems*, 1(2):144–156, 1983.
- [3] D. R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.
- [4] Herbert Bauer and Christian Sporrer. Distributed Logic Simulation and an Approach to Asynchronous GVT-Calculation. In *ACM/SCS/IEEE Workshop on Parallel and Distributed Simulation (PADS)*, pages 205–209, 1992.

- [5] Naraig Manjikian and Wayne Louks. High Performance Parallel Logic Simulation on a Network of Workstations. In *ACM/SCS/IEEE Workshop on Parallel and Distributed Simulation (PADS)*, pages 76–84, San Diego, 1993.
- [6] Dirk Baezner, Greg Lomov, and Brian Unger. A Parallel Simulation Environment Based on Time Warp. *International Journal in Computer Simulation*, 4(2):183–207, 1994.
- [7] Christopher D. Carothers and Richard M. Fujimoto. A Case Study in Simulating PCS Networks Using Time Warp. In *ACM/SCS/IEEE Workshop on Parallel and Distributed Simulation (PADS)*, pages 87–94, 1995.
- [8] 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.
- [9] Ralph M. Butler and Ewing L. Lusk. User's Guide to the p4 Parallel Programming System. Technical Report ANL-92/17, Argonne National Laboratory, Mathematics and Computer Science Division, 1992.
- [10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM 3.0 User's Guide and Reference Manual. *Oak Ridge National Laboratory*, 1993. Tennessee.
- [11] Jade Simulations International Corporation. *SimKit User Interface and Implementation Notes*, 1994.
- [12] Richard M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, pages 30–53, October 1990.
- [13] Norbert Fröhlich. Anwendungsunabhängiger paralleler Simulatorkern für verteilte Architekturen. Master's thesis, Institute of Electronic Design Automation, Technical University of Munich, 1994.
- [14] Josef Fleischmann, Rolf Schlagenhaft, Martin Peller, and Norbert Fröhlich. OLIVIA: Objectoriented Logicsimulation Implementing the VITAL Standard. In *Great Lakes Symposium on VLSI (GLS-VLSI)*, pages 51–56, March 1997.
- [15] IEEE Standards Board. *IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification*, 1996. IEEE 1076.4-1995.
- [16] Oz Levia. Introduction to ASIC Cells Modeling with VITAL. In *VHDL Forum for CAD in Europe*, April 1995. Tutorial.
- [17] IEEE Standards Board. *VHDL Language Reference Manual*, 1994. ANSI/IEEE 1076-1993.