Edited by S. Selberherr, H. Stippel, E. Strasser – September 1993

Process Flow Representation within the VISTA Framework

Ch. Pichler and S. Selberherr

Institute for Microelectronics, TU Vienna
Gußhausstraße 27-29, A-1040 Wien, AUSTRIA

Abstract

The execution of multi-step simulation sequences involving a number of independent simulation tools is taken care of by the VISTA simulation flow control module which allows for the definition of a simulation task by means of the simulation flow description, a representation of the process flow using symbolic names to call simulation tools. Large process flows can be modeled by using predefined sequences, where the calculation results of all intermediate simulation steps remain available for analysis at a later time. Data level integration is based on the PIF data exchange format which is used to create a wafer state description containing all wafer geometry and material data after each simulation step.

1. Introduction

Due to long fabrication times in a wafer fab the simulation of semiconductor fabrication processes and device behavior are an indispensable aid for experimenting with device designs and process parameters. Technology CAD (TCAD) has successfully used process simulation tools to model production process steps during the design stages of a device. A wide variety of such tools exist, each being more or less specialized to perform a specific task. In order to integrate simulation tools and to present them to the user in a uniform way, TCAD frameworks have been developed, e.g. [1], [2], [3]. Drawing on experience gained from these undertakings, we took on devising and implementing a *simulation flow control* module for the Viennese Integrated System for Technology–CAD Applications (VISTA) [4], [5], putting special emphasis on an open concept which allows for the integration of arbitrary simulators, and on a simple, extendible representation of simulation sequences, enabling the process engineer to quickly apply changes to the process design, investigate the results, and optimize device performance.

2. Multistep Process Simulation

Consider a set of process simulation tools such as PROMIS [6] or SAMPLE [7], each capable of performing the numerical simulation of one or more VLSI fabrication

process steps. Typically, the user invokes a simulation by specifying the values of the tool's input parameters via an input deck, i.e., a text file containing directives for setting these parameters, which is read by the tool upon execution. Additionally, initial geometry and dopant distribution data have to be provided in a format readable by the tool; after the successful completion of a simulation run, the output data are written to a file.

Should the user intend to simulate a series of process steps, the appropriate simulation tools have to be called sequentially, with the output data of a tool run being used as the input data for the next one. Therefore, a tool must be able to understand the data generated by its predecessor in the process sequence, i.e. the tools have to share a common data format or be able to translate from a foreign format to their own. Calling the tools one after the other in a UNIX environment is usually accomplished by a shell script which generates an input deck and calls a simulator for each simulation step. As the number of input parameters for a single tool is of the order of ten to hundred, modifying the script in the case of a change of a parameter value or of the process sequence is a tedious and error-prone task.

If we look at a process simulation sequence from different levels of abstraction we are able to identify a number of problem domains. At the *data level*, we have to ensure that the results generated by a tool are understood by the next one. At the *tool control level*, the simulator has to be provided with a set of key values encoding the parameters for a particular process step. Depending on the tool, various methods have to be provided for passing the process settings, e.g., generating an input deck. At the *task level*, we want to specify our simulation intent as intuitively as possible, concentrating on design parameters rather than on tool invocation subtleties. If we are not satisfied with the final outcome of a simulation sequence and want to modify our design, it should not be necessary to completely rewrite the process flow description, i.e., predefined process sequences should remain unchanged even if minor alterations are to be made. Using such *process modules* previously written to build up larger process flows greatly simplifies the process flow design.

Offering the user a comfortable means for automatically executing a large series of simulation steps reduces the cycle time in the design iteration loop. The following section presents our approach towards this goal, where we concentrate on the aspects of a generic tool integration concept.

3. Tool Integration and Process Flow Representation within VISTA

The Viennese Integrated System for Technology–CAD Applications (VISTA) uses the Profile Interchange Format (PIF) [8] as a common data format to exchange wafer data between simulation tools. Simulators which do not adhere to this policy have to get a wrapping program which establishes a PIF interface. Tool integration and task definition issues with respect to process flow simulation are subsumed under the VISTA *simulation flow control* module. Essentially, this module consists of three parts, the XLISP binding functions for the simulation tools, the tool control module, and the task control module.

In order to make executable modules available to the TCAD shell, they have to have a representation in the shell's extension language, XLISP. These *binding functions* enable the user to invoke a simulation step just like any other XLISP function. All calling details for the execution of a simulation module are hidden from the user, and so are all data manipulations necessary to feed wafer data into the simulator

and to get the calculation's results back. All parameter values, file names, and other control data for a tool run are passed as LISP *key* arguments to avoid errors due to a wrong argument order. These control data are translated into the simulator's control argument format, e.g., an input file is generated, or a command line is synthesized, or simulator specific PIF objects are added to the PIF input file. While the PIF defines a syntax for wafer description, it does not enforce a semantically strict representation of wafer data. In generating an input data file, ambiguities brought about by this semantic liberty are resolved by the binding function. For instance, we have to insure that all symbolic names used in a PIF file to describe material types are understood by the tool. If a material is not recognized, its name is replaced by an appropriate alias. Similarly, simulators do not agree upon geometry orientations (clockwise or counter clockwise), or they might require the presence of certain PIF objects which a preceding tool has not supported. In order to establish a standard data interface for plugging in simulation tools, the XLISP binding combines wafer data from before and after the simulation run to generate the current *wafer state*, i.e., a complete description of the wafer geometry and impurity concentrations reflecting the current state of the wafer.

Simulators usually require a large number of input parameters which alter some aspects of the computation and are not changed anymore once a tool is tuned to optimally satisfy the user's simulation requests. The *tool control* part passes a complete set of arguments to the simulator, merging user defined values and default values so that one is only obliged to enter those parameters one's concerned about. By switching at run-time between directories with the files containing the default values for the tools, the simulator behavior may be modified without affecting user settings.

In this context, we call a *task* any sequence of computations carried out on related data, e.g. a series of process simulation steps working on a common wafer, or an iteration loop which performs the same sequence of calculations on a set of initial geometries which might be automatically generated from a prototype geometry description, and the like. The *task control* module is the only one interacting directly with the user. A task is defined by writing a *simulation flow description* in LISP syntax, where symbolic names are used to specify a call to one of the *registered* simulation tools or to execute some control commands for data handling during the execution of a task. For instance, if the user wants to call a predefined process sequence, the process reference keyword PROCESS followed by the name of the file containig the process sequence is used. An optional override mechanism allows any parameter in any subprocess to be modified. The process reference and parameter override mechanisms work recursively.

The following example shows a small part of a wafer fab run traveller as it appears in the simulation flow description.

```
(
  (start-with :phys-pif-infile "InitGeom.pbf")
  (monte-carlo-implant :elem "BORON" :dose  1e13 :energy  30.)
  (anneal :temp 900 :time (35 "min"))
  (isotropic-deposition :time 225. :material ("SiO2" 0.0015))
  (anisotropic-etch :time 68. :material-default (0. 0.0001)
                    :material ("SiO2" 0 0.005))
  (monte-carlo-implant :elem "BORON" :dose 1e15 :energy 45.)
  (anneal :temp 900 :time (20 "min"))
)
```

The sequence shown above defines the process steps necessary to simulate the fabrication of an LDD (lightly-doped drain) structure of a p-channel MOS transistor. The PIF file `InitGeom.pbf` contains a PIF model of the wafer to be processed, basically a chunk of silicon partially covered by a nitride layer defining the gate location.

4. Conclusions

The simulation flow control module provides a comfortable means for defining, executing and modifying multistep simulation tasks. Existing process sequences can be easily modified to optimize device characteristics, large process flows can be built up from process modules. The XLISP tool binding functions establish an interface which allows for the integration of a large class of simulation tools. These tools are available to the user as plug-in modules for his simulation tasks. If the user chooses so, all intermediate calculation results remain available for analysis at a later time, simplifying error recovery as well as a detailed examination of process steps.

Acknowledgements

Our work is significantly supported by Digital Equipment Corporation at Hudson, USA; and Siemens Corporation at Munich, Germany.

References

[1] D. S. Boning, *Semiconductor Process Design: Representations, Tools, and Methodologies*, PhD Thesis, Massachusetts Institute of Technology, January 1991.

[2] A. S. Wong, *Technology Computer-Aided Design Frameworks and the PROSE Implementation*, PhD Thesis, University of California, Berkeley, 1992.

[3] E. W. Scheckler et al., *A Utility-Based Integrated System for Process Simulation*, IEEE Trans. Comp. Aided Design, Vol. 11, No. 7, pp. 911-920, 1992.

[4] H. Pimingstorfer et al., *A Technology CAD Shell*, SISDEP IV, pp. 409-416, 1991.

[5] S. Halama et al., *Consistent User Interface and Task Level Architecture of a TCAD System*, NUPAD IV, pp. 237-242, 1992.

[6] G. Hobler et al., *RTA-Simulation with the 2D Process Simulator PROMIS*, NUPAD III, pp. 13-14, 1990

[7] W. G. Oldham et al., *A General Simulator for VLSI Lithography and Etching Processes: Part II-Application to Deposition and Etching*, IEEE Trans. Electron Devices, Vol. ED-27, No. 8, pp. 1455-1459, 1980.

[8] St. G. Duvall, *An Interchange Format for Process and Device Simulation*, IEEE Trans. Comp. Aided Design, Vol. 7 No. 7 pp. 741-754, 1988.