RAPID SEMICONDUCTOR PROCESS DESIGN WITHIN THE VISTA FRAMEWORK: INTEGRATION OF SIMULATION TOOLS

CH. PICHLER AND S. SELBERHERR

Institute for Microelectronics, Technical University of Vienna
Gußhausstraße 27–29, A-1040 Vienna, Austria

1 Introduction

Long fabrication times in a wafer fab make the simulation of semiconductor fabrication processes and device behaviour an indispensable aid for experimenting with device designs and process parameters. When a first prototype device leaves the fabrication facility, it is likely to be rejected due to a couple of possible flaws. Any deviation of the device parameters from the desired device specification causes modifications to be made to the device design and starts a new iteration of the outer design fabrication loop (ref. Fig. 1). Any difference exceeding the specified tolerances between the device design and the device actually produced or simulated calls for a modification of the process design and triggers the inner design loop anew.

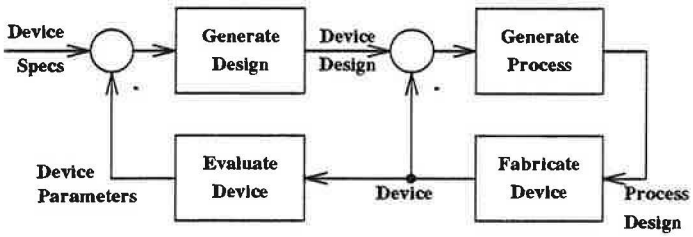

Figure 1: VLSI Design–Fabrication Loop

The time spent on the fabrication of the device plays a key role in timing consideration as manufacturing resides at the heart of both loops. Technology CAD (TCAD) has successfully used process simulation tools to replace 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, a number of TCAD frameworks have been developed, e.g., [4], [5], [6], [7], [11], [12]. Drawing on experience gained from these undertakings, we took on devising and implementing a *simulation flow control* module for the Vienna Integrated System for TCAD Applications (VISTA) [9], [10], putting special emphasis on an open concept which allows for the integration of arbitrary simulators, on supporting multi-platform environments, 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.

The following section deals with the simulation of semiconductor fabrication processes, concentrating on the integration of simulation tools. Section 3 presents the simulation flow control module.

2 Semiconductor Process Simulation

Consider a set of process simulation tools such as PROMIS [3] or SAMPLE [1], each capable of performing the numerical simulation of one or more VLSI fabrication processes, e.g. an ion implantation process, an etch process, etc. Typically, the user invokes a simulation step 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, or a user interface of any kind (graphical or text-based, e.g.). Additionally, it has to be taken care of providing initial geometry and dopant distribution data 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, such as the fabrication of a complete MOS transistor, 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 writing 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, e.g. the diffusion tool has to be able to read the boron concentration written by an ion implantation module, and the device simulator has to understand the doping profile generated by a diffusion module.

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. Before starting the execution of a simulation program, we have to check for any invalid parameter combinations which might lead to invalid results or to the abortion of the computation.

At the *task level*, the user wants to specify her simulation intent as intuitively as possible, concentrating on design parameters rather than on tool invocation subtleties. For instance, if we wish to examine the variation of the threshhold voltage of a MOS transistor with the length of the gate , we have to apply the same process sequence to a set of varied initial geometries. The series of intermediate calculation steps appears

as a black box delivering the result sought. Furthermore, 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, but we should be able to specify the changes we want to make to certain parameter values, i.e., predefined process sequences should remain unchanged even if minor alterations are to be made. Using process modules previously written to build up larger process flows greatly simplifies the process flow design.

Providing the user with a comfortable means for automatically executing a large series of simulation steps helps with reducing the cycle time in the design iteration loop. The following section presents our approach towards this goal, concentrating on the aspects of a generic tool integration concept.

3 Tool Integration in the VISTA Framework

The Vienna Integrated System for TCAD Applications (VISTA) represents a framework for the integration of semiconductor process and device simulation. Integration at the data level is achieved by using the Profile Interchange Format (PIF) [2], [8], coupling of the simulation tools to the TCAD shell is accomplished by using XLISP as an extension language.

Integration and control issues mentioned in the previous section are subsumed under the *simulation flow control* module, which is described in the follwing.

3.1 The Simulation Flow Control Module

Intended to liberate the user from a couple of tedious tasks when attempting to investigate the design domain by reiterating VLSI process, device, and circuit simulation sequences, the simulation flow control module represents a hands-on approach to tool integration. At present, an intermediate stage of development has been reached, which will be followed by a consolidation phase to draw from gained experience and to update specifications. Essentially, the simulation flow control module consist of three parts, the XLISP bindings of the simulator modules, the tool control module, and the task control module. This subdivision may sometimes not be reflected in the actual implementation.

3.1.1 XLISP Bindings of Simulator Modules

In order to make arbitrary executable modules available to the calling context (TCAD shell), these modules or programs have to be wrapped in XLISP functions representing the control level interface between tools and framework. Therefore, a LISP function is defined for each simulator to be integrated, so the user can call the tool like any other XLISP function. All parameter values and file names are passed as key parameters to avoid errors due to a wrong argument order. The main action performed by such a function is to start the respective simulator as a background job and to notify the calling shell upon completion. For this purpose a callback concept is used, i.e. a LISP function is called upon return from a system job to resume execution of LISP code (ref. Fig. 2).

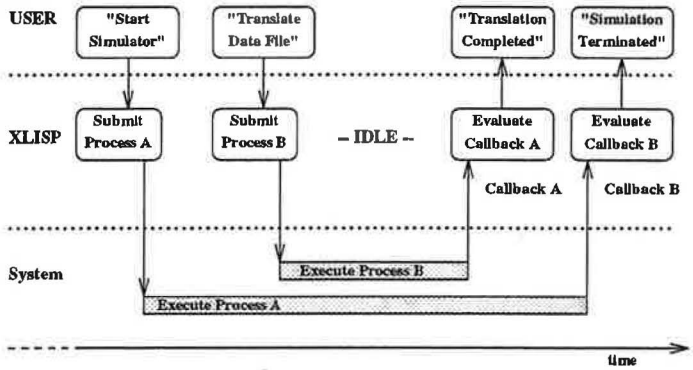

Figure 2: The Callback Mechanism

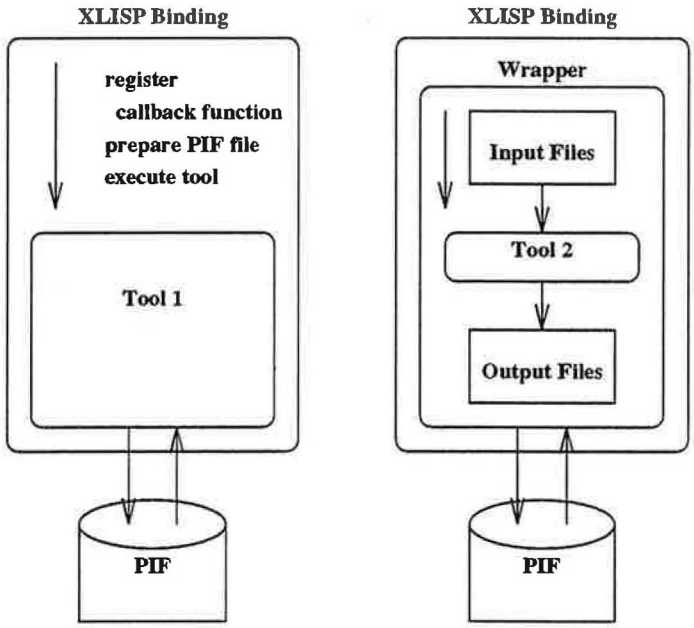

Figure 3: XLISP Binding of simulation tools

Furthermore, it is taken care for providing the simulator with all required input files and command line arguments. In our case, simulators are assumed to read initial wafer data from a PIF file generated by the preceding tool and to write their results to a PIF file. Accepting PIF input and writing PIF output is considered prerequisite for an integration into VISTA, thus tools which don't adhere to this regulations should get a wrapping function establishing a PIF interface (ref. Fig. 3).

While the PIF defines a syntax for wafer description, it does not enforce a semantically strict representation of wafer data. In generating an input file for a simulator, ambiguities brought about by this semantic liberty are resolved by the binding function. To begin with, a simulator might recognize the material silicon, e.g., by the material name `Si`, whereas another tool might exclusively accept `SILICON`. The binding function checks the PIF input file for all occurrences of material names and replaces them as appropriate. In the second place, simulators do not agree upon geometry orientation when reading or writing PIF data. Therefore, the XLISP binding has to transform the geometry to the correct orientation (clockwise or counter clockwise) for the tool in question. Thirdly, some simulators require the presence of certain PIF objects in

their input file which a preceding simulator might have discarded due to utter ignorance of its fellow tool's needs. The binding functions check for the presence of required objects, try to restore missing items, and solicit user interaction if they fail to do so.

In order to establish a standard interface for plugging in simulation tools, the XLISP binding combines wafer data from before and after a simulation run to generate the current *wafer state*, i.e. a complete description of wafer geometry and impurity concentration data reflecting the current state of the wafer after each simulation step.

3.1.2 Tool Control

With the simulation tools being properly wrapped in the XLISP binding functions, the execution of a simulator can be initiated by calling the respective function. While existing simulation tools often require a large number of input parameters, only a few are of any concern to the user, the remaining ones being set once to tune the tool's performance. Therefore, the user should only be obliged to supply those parameters she explicitly wants to assume values other than the default. To achieve this, user-defined values and default values are merged upon invocation of a tool to from a complete list of arguments. As even default values might vary with different classes of tasks, all default values are stored in files residing in a *default values directory*. By switching between such directories at run-time, simulator behaviour may be modified without affecting user settings.

3.1.3 Task Control

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. Each line consists of a leading symbolic name, followed by a list of arguments. If a line defines a tool call, the symbolic name references the XLISP binding function for the respective tool. In this case, the user-set parameters are passed as arguments. All available tool call functions and their symbolic names are contained in a list which is generated by *registering* the XLISP tool binding functions at load-time.

In addition to tool calls, a couple of control commands and keywords (ref. Table 1) may appear in the simulation flow description. They are used to reference predefined process sequences, to switch between directories for the default values files, to enable or disable keeping of intermediate simulation results, to save a particular intermediate result for later retrieval, to set the host where subsequent computations are to be performed, to start simulation with a initial file, and to call for user interaction.

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.

Table 1: Control Commands and Keywords

Keyword	Description
`start-with`	Loads a PIF file to start subsequent calculations
`use-machine`	Defines the host for subsequent computations
`save-state`	Enables saving of intermediate results
`dont-save-state`	Disable saving of intermediate results
`copy-file`	Writes current wafer state to a file
`PROCESS`	References a predefined process sequence
`PROCESS-DIR`	Sets the directory with predefined process sequences
`DEFAULT`	Sets the directory with default values files

As simulation tools are executed in the background, several tasks can be performed simultaneously. The simulation flow control module keeps track of started system processes and switches to the task the process did originate from to continue with the next command of this task when a system process returns. To avoid conflicts due to fixed file names used by certain tools, a locking mechanism prevents the user from starting more than one task at a time in a given directory.

3.1.4 A Short Example

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. If this sequence resides in a file `spacer`, it can be referenced by using the `PROCESS` keyword. The following example shows how the user can define an override value for any parameter value of a referenced process module. In the following case, the temperature for both annealing steps is set to `875`.

```
(
  (start-with :phys-pif-infile "InitGeom.pbf")

  (PROCESS spacer :setvalue (anneal (temp 875)))
)
```

If we create two short process modules **spacer_implant1** and **spacer_implant2** containing the two implantation-annealing pairs from the first example, and a third process module **make_spacer** containing the deposition and etch steps, we can rewrite the process module **spacer** as follows.

```
(
  (PROCESS spacer_implant1)
  (PROCESS make_spacer)
  (PROCESS spacer_implant2)
)
```

Now we can override the two annealing temperature parameters in the implantation modules independently from each other:

```
(
  (start-with :phys-pif-infile "InitGeom.pbf")
  (PROCESS spacer :setvalue
    (PROCESS spacer_implant1
      :setvalue (anneal (temp 890))
    (PROCESS spacer_implant2
      :setvalue (anneal (temp 875))))
  )
)
```

The mechanism shown above is used to automatically iterate over any number of values for a parameter in a process sequence. It can be applied, e.g., to the optimization of the behaviour at high drain-to-source voltages to minimize hot-carrier effects.

The executing of a process sequence produces a PIF file which contains a complete description of all wafer state transitions in terms of the resulting logical PIF files after each simulation step. If the **save-state** mode is selected, the PIF file resulting from the **spacer** process sequence contains seven wafer states reflecting the effects of the respective treatments. If the **dont-save-state** mode is active, only the final result is kept.

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.

REFERENCES

[1] 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, August 1980.

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

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

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

[5] J. St. Wenstrand, *An Object-Oriented Model for Specification, Simulation, and Design of Semiconductor Fabrication Processes*, PhD Thesis, Stanford University, March 1991.

[6] Ch. J. Hegarty, *Process-Flow Specification and Dynamic Run Modification for Semiconductor Manufacturing*, PhD Thesis, University of California, Berkeley, April 1991.

[7] J. Daniell and St. W. Director, *An Object-Oriented Approach to CAD Tool Control*, IEEE Trans. Comp. Aided Design, Vol. 10, No. 6, pp. 698-713, June 1991.

[8] F. Fasching et al., *A PIF Implementation for TCAD Purposes*, SISDEP IV, pp. 477-482, September 1991.

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

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

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

[12] 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, July 1992.