SIMULATION OF SEMICONDUCTOR DEVICES AND PROCESSES Vol. 4
Edited by W. Fichtner, D. Aemmer - Zurich (Switzerland) September 12-14, 1991 - Hartung-Gorre

A Technology CAD Shell

Hubert Pimingstorfer, Stefan Halama,
Siegfried Selberherr, Karl Wimmer, Peter Verhás
Institute for Microelectronics, Technical University of Vienna, Austria

Abstract

A new TCAD shell is presented which is capable of performing complex development tasks by using LISP as interaction and programming language. Data level integration of simulation tools and capabilities required in process and device development into a homogeneous environment is based on a binary implementation of the Profile Interchange Format.

1 Introduction

The use of CAD tools for analysis and prediction of IC technology is generally a substitute for physical experimentation to save time, efforts and money, and to provide additional insight. A recent trend is to integrate the tools into a TCAD environment [1]-[3] to meet demands that range from simple simulator coupling over process and device characterization to technology optimization.

The aim of our TCAD shell is to ease and automate this in a way that the user is allowed to concentrate on performing complex development tasks rather than on supervising single simulator runs.

2 Demands on a TCAD Shell Language

Firstly, the TCAD shell language is the *command language* with which the user interacts with the TCAD system; so it has to be interpreted. In addition, it must be able to run time consuming tasks as background processes or in batch mode.

Our first consideration to use one of the various shells under UNIX or DCL under VMS has been dropped due to their inconvenient (or total lack of) control structures like branches, loops, and subprograms, mechanisms for defining new variables and controlling their scope and visibility, and performing mathematical operations. Another major design goal was to be independent from the operating system (not restricted to UNIX, like other efforts in this field are).

The second task of the TCAD shell language is to serve as an *extension language*, in which new functionality is added, customizations are specified, and macros or just shortcuts for often used shell command sequences are defined.

The design of a special-purpose language can be considered a tremendous and yet unnecessary task, as existing interpreted languages are sufficient (cf. [4]). We have chosen LISP as the base of the TCAD shell because of the flexibility of this programming language.

Among the candidates of publicly available LISP interpreters we picked XLISP [5] in its current version 2.1. This small interpreter combines features of Common LISP (which will

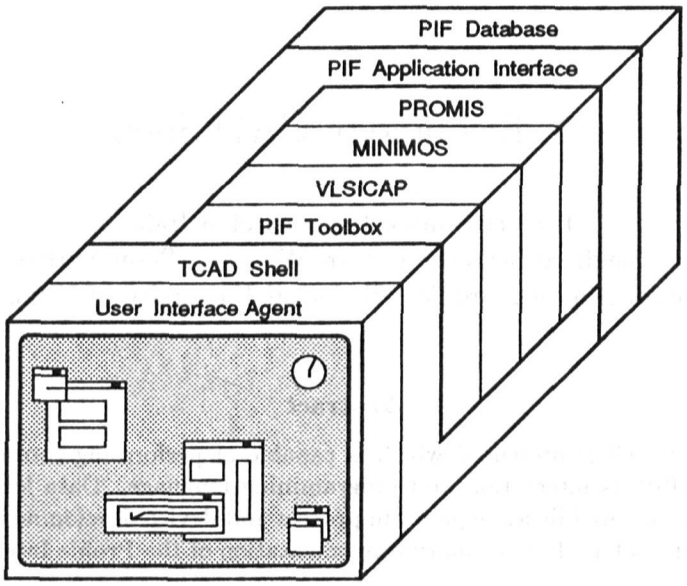

Figure 1: VISTA System Overview

probably evolve to a kind of standard among the various LISP dialects) with object-oriented capabilities. XLISP is written in portable C with modularized design and exhibits a clear C-to-LISP interface. The source code availability meets the need to implement TCAD shell functions in C, which are linked together with the original interpreter and are further handled like built-in functions.

Beside the basic needs, using LISP allows - as it makes no distinction between program and data structures - *process flow representations* (like in [6]) or *simulation flow representations* to be either executed directly or manipulated as data.

Excellent and well-known examples for the successful application of LISP as an extension language in other software engineering domains are the text editor Emacs [7] or the CAD program AutoCAD [8].

3 Integration into a TCAD Environment

The TCAD shell is part of the VISTA project [9], [10] to build a homogeneous TCAD environment; cf. Fig. 1. All kinds of simulators (process, device, interconnect), grid manipulators, discretizers, solvers, measurement data translators, tools for automatic device characterization, one- and multi-dimensional optimizers, graphical editors, previewers, etc., can be included as directly callable shell functions. The TCAD shell serves as a command interpreter and as extension language as already indicated in the previous section.

Simulation in a distributed computing environment will be addressed, where graphical user interaction is done on a workstation and computationally expensive modules run on fast floating point machines like vector computers or massively parallel systems.

As shown in Fig. 2, three levels in building an integrated TCAD environment can be identified: the data level, the tool level, and the task level.

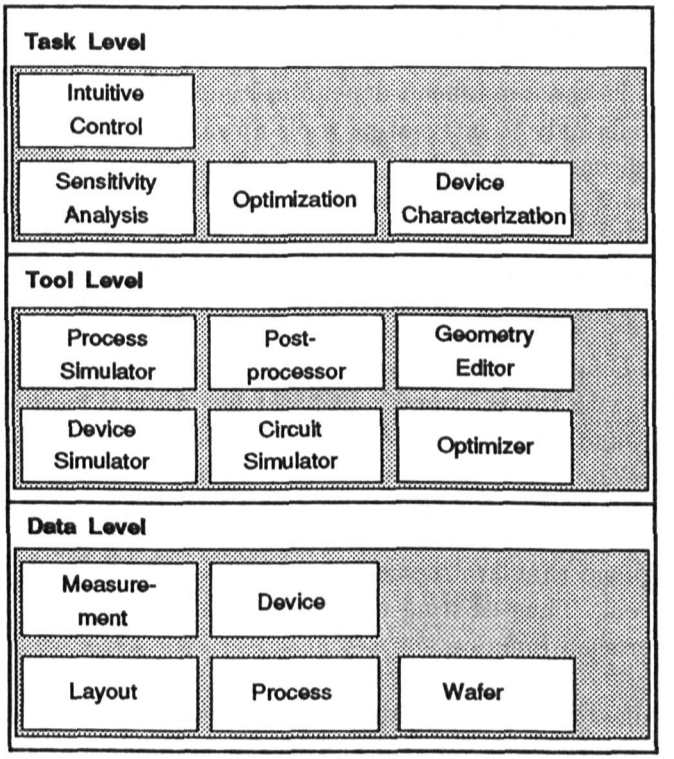

Figure 2: Three Levels in TCAD Engineering

4 Data Level - The PIF Implementation

The database is a binary implementation of the Profile Interchange Format (PIF) [11], which has been modified and extended to fulfill the needs of an integrated TCAD system.

The TCAD database, consisting of so-called binary PIF files, is accessed from programs with an application interface. Our implementation of this interface [12] is strictly layered, thus conforming to modern software engineering style. In contrast to other approaches (e.g. [13]), we designed even the low-level database structure specifically for TCAD purposes, resulting in considerable performance improvements compared to implementations built on top of commercial database systems.

The interface itself is implemented in C, but FORTRAN and LISP applications have been taken into account with the support of appropriate language bindings to the interface's C functions. The application interface for LISP provides routines which create, delete, read or write entire PIF objects. Thereby access for all shell programs and even interactive PIF data manipulation is provided.

The method how a new simulator is adapted to obey the PIF object storage convention [14] depends on whether its source code is subject to manipulation or not. In the first case, the input and output functions are replaced with corresponding application layer functions to read and write PIF. Even in the latter case in which conventional translators are necessary, the advantage gained from a unique data exchange format is obvious: Only n translators for coupling n tools are needed and not $n \cdot (n-1)/2$ for all possible tool to tool connections.

5 Tool Level – The Workhorses

Tools can be integrated in three manners, depending on the language they are programmed in (note that these items refer only to integrating tools upwards into the task level; the integration on data level has been discussed above):

- LISP tools just have to be loaded and executed by the TCAD shell. This is useful for high-level optimization loops or module sequencers, which consume only small amounts of computation time compared to other tools probably called.
- Tools in form of a C function just have to get a small C-to-LISP interface. Then they can be linked together with the shell and called just like normal built-in shell functions. This is useful for small and frequently needed tools which consume some computation time. They could as well be called as separate executables with the system call, but linking them to the shell eliminates the operating system overhead.
- Tools in any language that are separate executables can be called with a shell built-in system call function. Thus existing simulators, most are coded in FORTRAN, can be used like any other shell function.

Full flexibility can be gained from splitting the simulator into reasonably sized modules (e.g. separating grid generation, discretization, solver and physical models) and from combining the new modules with existing TCAD tools into task level programs almost arbitrarily. To do so, the modules must have a small extension language interface to make them callable from LISP, and they have to adhere to the PIF object storage convention [14].

Until now the device simulator MINIMOS, e.g. [15], the process simulator PROMIS, e.g. [16], and the interconnect capacitance simulator VLSICAP, e.g. [17], have been integrated into the TCAD system.

As an example, Fig. 3 shows how the most recent versions of the simulators MINIMOS and PROMIS fit into the TCAD environment. PROMIS is split into four completely independent executables, called from the shell through a small LISP interface. Every module reads and writes from/into the PIF database. There are four interface routines to the shell, called `promis-analytic-implant`, `promis-mc-implant`, `promis-diffuse` and `promis-oxidize`. The function `run-promis` in the TCAD shell example is just a sequence of PROMIS functions simulating a complete process. MINIMOS modules are coupled internally; PIF input/output is done by two specialized ones. All modules are controlled by a stack-driven sequencer, callable as `run-minimos`.

The major advantage when building a new simulator is that it is no longer necessary to provide a specific grid generator, solver, etc., since these tools are readily available on the shell level. Therefore, simulator designers are able to concentrate on the specialized parts of simulator construction.

The executable modules are usually small and, for appropriate simulation problems, can be run (in parallel) on different machines under control of the TCAD shell, thus yielding considerable speed improvement. When modularized properly, the most time consuming parts (e.g. linear solvers) can be executed on a supercomputer communicating with the TCAD shell running on a workstation.

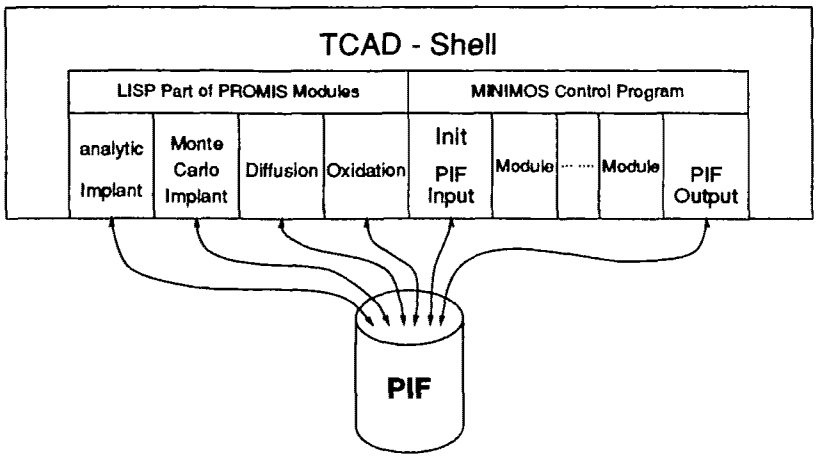

Figure 3: MINIMOS and PROMIS in the TCAD Environment

6 Task Level – TCAD Shell Functions

From the viewpoint of the TCAD shell, a simulator is a function that takes input and returns output in two possible formats – as normal LISP expression and/or as a handle to an opaque PIF object, accessible from the shell level via the PIF application layer.

Shell functions specialized on MOS transistors have been written which compute, e.g., the threshold voltage and drain and/or bulk current by invoking MINIMOS and returning the value as a LISP expression. The relative transconductance or the gate swing, e.g., are computed by invoking the device simulator twice for different gate voltages.

These functions combined with a one-dimensional optimizer are used, for instance, to find the maximum of the bulk current or of the relative transconductance in the linear regime. Combined with looping constructs, the shell functions are tailored to compute I/V characteristics or any other variation of an output quantity versus any allowed input key applying a constant or an adaptive step size.

For each simulator run, the user is relieved from adapting an input deck with an editor, starting the simulator on the command line and getting the required values from the output.

With few lines of TCAD shell code a new shell function, tailored to the very specific needs of the user, can be written as a combination of any tool callable at shell level and normal LISP code. The TCAD shell allows arbitrarily complex tasks to be performed, ranging from simply calling a single module interactively over coupling simulators to running whole optimization loops as background processes.

7 User Interface

The TCAD shell serves as a textual user interface to the TCAD system in cases where terminal capability is required to be enough. For higher convenience, the User Interface Agent (UIA) has been designed which allows graphical control of the TCAD system. This includes editing, manipulating and viewing geometries or simulation results, a visual programming interface to symbolic simulation flow representations and postprocessing.

An interface to the X11R4 window system has been implemented into the LISP interpreter, based on X Toolkit and the Athena widgets to address the portability issue between workstations

from different vendors. In principle, a witget callbacks cause LISP expressions to be evaluated. Thus a certain shell function can be selected by the mouse cursor and a simulator started by a button press.

8 Example

As an example, the bulk current of an n-channel MOS transistor has to be minimized by varying the dose of the lightly doped drain (LDD) implant. Therefore process and device simulation are coupled within an optimization loop.

The doping profiles simulated by PROMIS are characterized by several MINIMOS runs, computing the threshold voltage, the saturation current, and the maxima of the relative transconductance in the linear regime and of the bulk current, for which an optimizer is used to adjust the gate voltage. The ratio of bulk to drain current for a constant bias condition is the value to drive the optimization loop for the LDD implant dose.

The one-dimensional optimizer applied, is an implementation of the well-known golden section algorithm. It needs 8 iterations to explore a dose range from $1e10$ to $1e16cm^{-2}$ with a resulting tolerance factor less than 2. That means, 8 times PROMIS and due to the extensive device characterization about 100 times MINIMOS is run automatically under control of the TCAD shell.

The TCAD shell program for this example is shown in Fig. 4 and the corresponding control flow diagram in Fig. 5. As the resulting diagram of I_b/I_d versus LDD implant dose in Fig. 6 shows, an explicit minimum exists and the device can be readily improved.

```
;; optimize LDD implant dose for minimal ib/id[2.5/5.0]
;; run PROMIS and characterize the profile running MINIMOS for U_th,
;; Id_saturation, gm_max, Ib_max, and Ib/Id for each optimizer iteration
;;
(defun minimize-ib/id
  (PR-INPUT MM-INPUT DIRECTIVE OCCURENCE KEY MIN MAX TOL
    &key (PR-BASENAME TCAD-PR-TFN) (LOG NIL)
    &aux PROFILE  RESULT-LIST  Ib/Id-VALUE)
  (golden-section      ;1d-optimizer
    #'(lambda (VALUE)
      (setq PROFILE    ;new profile name
        (new-profile-name PR-BASENAME DIRECTIVE OCCURENCE KEY VALUE))
      ;;modify PROMIS input deck
      (set-pr-key PR-INPUT DIRECTIVE KEY VALUE OCCURENCE)
      ;;run PROMIS
      (run-promis PR-INPUT :EXEC-MODE "i")
      ;; run MINIMOS several times
      (setq RESULT-LIST (append RESULT-LIST
        (list (list VALUE
          (u-th     MM-INPUT :PROFILE PROFILE)
          (id[bias] MM-INPUT :UG 5.0 :UD 5.0 :PROFILE PROFILE)
          (gm-max   MM-INPUT 1.0 3.0 0.2 :UD 0.1 :PROFILE PROFILE)
          (ib-max   MM-INPUT 1.0 3.0 0.2 :UD 5.0 :PROFILE PROFILE)
          (setq Ib/Id-VALUE
            (ib/id[bias] MM-INPUT :UG 2.5 :UD 5.0 :PROFILE PROFILE))))))
      Ib/Id-VALUE); end lambda
    MIN MAX TOL :LOG LOG); end golden-section
  RESULT-LIST            ; return result list
)
```

Figure 4: Example TCAD Shell Program

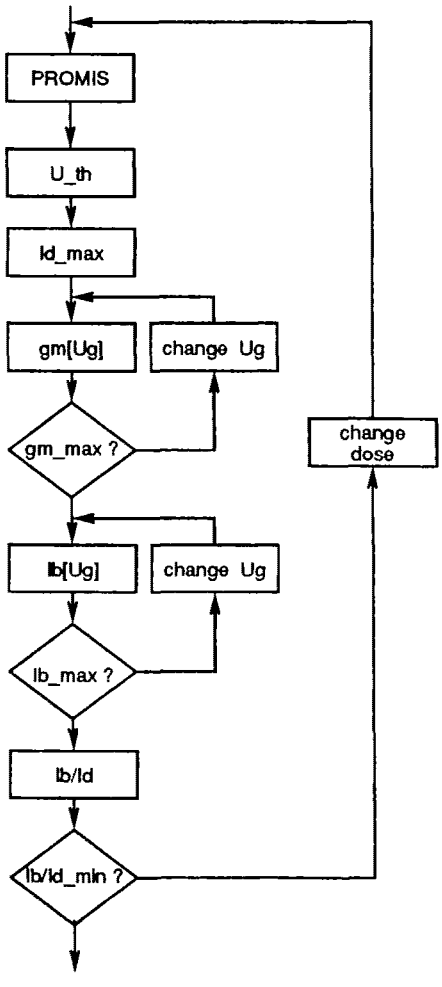

Figure 5: Control Flow Diagram

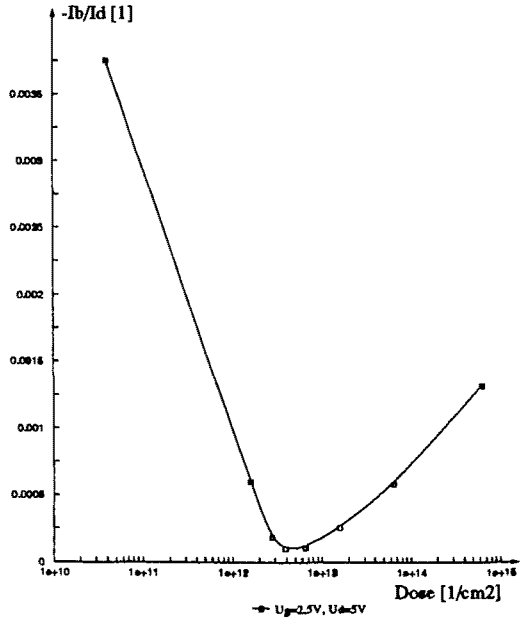

Figure 6: I_b/I_d vs. LDD Implant Dose

9 Conclusion

The TCAD shell uses LISP as command and extension language. A PIF interface enables data level integration of simulation tools. The graphical user interface based on X11 allows intuitive control of the TCAD system. Due to its possibility to easily combine different tools with the powerful shell language, our TCAD shell provides an ideal environment for technology characterization, sensitivity analysis, and process and device optimization. It is highly extensible, customizable and operating system independent.

Acknowledgements

This project is supported by the research laboratories of: AUSTRIAN INDUSTRIES - AMS Int. at Unterpremstätten, Austria; DIGITAL EQUIPMENT Corp. at Hudson, USA; SIEMENS Corp. at Munich, FRG; and SONY Corp. at Atsugi, Japan.

References

[1] D. S. Harrison, A. R. Newton, R. L. Spickelmier, and T. J. Barnes, *Electronic CAD Frameworks*, Proceedings of IEEE, Vol. 78, No. 2, pp. 393-417, 1990.

[2] E. W. Scheckler *et al.*, *A Utility-Based Integrated Process Simulation System*, Symp. on VLSI Technology, pp. 97-98, 1990.

[3] P. Lloyd, H. K. Dirks, E. J. Prendergast, and K. Singhal, *Technology CAD for Competitive Products*, IEEE Trans. Computer-Aided Design, Vol. 9, No. 11, pp. 1209-1216, 1990.

[4] CFI Extension Language Sub-Committee, *CFI Extension Language Selection Document*, CAD Framework Initiative, April 1990.

[5] D. M. Betz, *XLISP: An Object-oriented Lisp*, Version 2.0, Peterborough, NH, Febr. 1988.

[6] M. B. McIlrath, D. E. Troxel, D. S. Boning, M. L. Heytens, P. Penfield Jr., and R. Jayavant, *CAFE: A Computer-Aided Fabrication Environment*, Proc. International Electronics Manufacturing Technology Symposium, Washington D.C., Oct. 1990.

[7] R. Stallman, *GNU Emacs Manual, sixth ed. Emacs Version 18*, Free Software Foundation, March 1987.

[8] N. Johnson, *AutoCAD: The Complete Reference*, Berkeley, CA: Osbourne McGraw-Hill, 1989.

[9] S. Selberherr *et al.*, *The Viennese TCAD System*, Proc. Int. Workshop on VLSI Process and Device Modeling, Oiso, 1991

[10] F. Fasching *et al.*, *An Integrated Technology CAD Environment*, Proc. Int. Symp. on VLSI Technology, Systems and Applications, Taipei, Taiwan, 1991.

[11] S. Duvall, *An Interchange Format for Process and Device Simulation*, IEEE Trans. Computer-Aided Design, Vol. 7, pp. 489–500, 1988.

[12] F. Fasching *et al.*, *A PIF Implementation for TCAD Purposes*, this volume.

[13] A. Wong *et al.*, *The Intertool Profile Interchange Format*, Proc. NUPAD III, pp. 61–62, 1990.

[14] F. Fasching *et al.*, *Viennese Integrated System for TCAD Applications*, Institute for Microelectronics, Technical University Vienna, Austria, 1990.

[15] S. Selberherr, *Three Dimensional Device Modeling with MINIMOS 5*, Proc. Int. Workshop on VLSI Process and Device Modeling, pp. 40-41, 1989.

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

[17] F. Straker *et al.*, *Capacitance Computation for VLSI Structures*, Proc. EUROCON, pp. 602-608, 1986.