E2.

AN INTEGRATED TECHNOLOGY CAD ENVIRONMENT

F. Fasching, C. Fischer, S. Halama, H. Pimingstorfer, H. Read†,
S. Selberherr, H. Stippel, P. Verhas, K. Wimmer

Institute for Microelectronics, Technical University of Vienna, Austria
†ECE Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract — A new TCAD system is presented, capable of performing complex development tasks by means of a powerful interaction language and an efficient database system. The integration of tools is supported through a comfortable layered application program interface.

1 Introduction

The demands on Technology CAD (TCAD) range from simple simulator coupling to process, e.g. [1], and device technology optimization. Our system as shown in Fig. 1 is controlled through an interaction language based on a LISP interpreter — called the TCAD shell — providing homogeneous integration of the data, tool and task levels. The available TCAD tools, be they simulators or PIF ToolBox utilities, and their data are used by the shell which in turn can be controlled through the User Interface Agent enabling graphical representations and intuitive usage of the system. The data format upon which the system is built is an enhanced and extended intertool mode of a widely used profile interchange format (PIF) proposed in [2]. This PIF database is accessible from the tools and the shell through the application interface, and toolbox functions for conversion to textual (ASCII) representation for intersite data exchange, e.g. via eMail, are available.

2 Task Level – The TCAD Shell

To perform complex simulation tasks, the process and device engineer needs an *extension language* with which he can write programs manipulating simulator input, *invoking TCAD tools* and feeding back the results of a simulator call into the program in order to perform optimizations. Additionally, this "meta language" has to be *interpretative* to form a shell in which the engineer works during a TCAD session, but the ability to run task level programs as *background processes* or in batch mode has to be supported too. Another issue of major importance is *portability*, which means that the language interpreter's

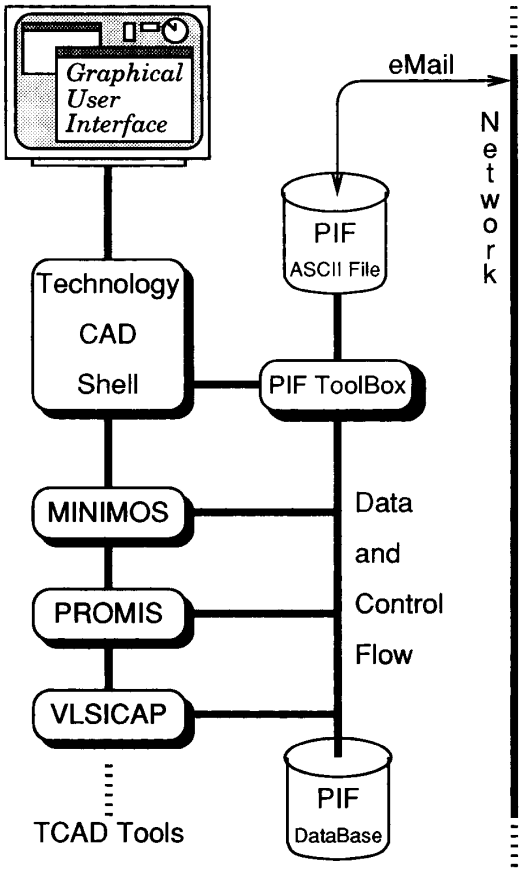

Figure 1: TCAD System Overview

sourcecode is publicly available and written in a portable language. The language has to provide constructs powerful enough to ease the task of writing shell programs (a simple command processor like the various shells under UNIX or DCL under VMS is definitely not sufficient).

The language interpreter itself has to be *extendable* with builtin commands, i.e. linking of self-written functions to

the interpreter must be possible, implying the availability of the sourcecode. And finally, *integrating PIF* should be an easy task which is best guaranteed with a LISP-like syntax of the extension language.

Among the few candidates left we chose a small LISP interpreter, implementing a subset of Common LISP. A system call is provided to run separately executable TCAD tools. By redirecting input/output from/to a file, the shell can be run in the background. The interpreter itself is written in portable C and implements a wide range of control structures and functions to ease the development of task level programs. Extendability is ensured through modularized design and a clear C-to-LISP interface. Since the PIF syntax is very close to LISP, it is easy to manipulate PIF data in the TCAD shell. Thus all necessary requirements on a TCAD extension language are fulfilled ideally.

Modules or tools are directly callable as shell functions of the extension language, thus enabling programs written in this language to call these tools. This structure allows arbitrarily complex tasks to be performed, ranging from simply calling a single module interactively over coupling simulators via a shell function to running whole optimization loops as background processes. Starting tools on different machines is also possible (distributed processing), although system dependent.

Besides of the basic needs stated above, a major influence in the decision to use LISP as the extension language has been that there is no distinction between program and data structures. This allows, for example, a process flow representation to be either executed directly in the shell as a program, or to be stored in the database as data.

A powerful extension is the User Interface Agent (UIA) which allows graphical control of the TCAD system including editing, manipulating and viewing geometries, simulation results and symbolic process flow representations. In addition, the experienced user can directly use the shell language to create new functions or modify existing ones. The environment does not depend on the graphical interface which is inherently system dependent. It could as well be used without it (terminal capability is enough), although it is more convenient to use the UIA.

An example of a task level program for minimizing the bulk current of a device by means of modifying the LDD implant dose is presented in Fig. 2. The shell function `minimize-i-b` is given an implant dose and a handle to a binary PIF file with a device geometry desription. The process simulator PROMIS, started with an initial LDD implant dose, adds doping profiles to the PIF file. The device simulator MINIMOS is run on this file, calculating the actual bulk current. Then a loop is run on these two simulators, which performs an update of `LDDimpl-dose` with `compute-new-dose` and breaks when the test criterion of a minimal bulk current is met.

```
;;; sample TCAD shell task level function
(defun minimize-i-b (LDDimpl-dose obj-hdl)
   (run-promis LDDimpl-dose obj-hdl)
   (run-minimos obj-hdl)
   (setq i-b-act
      (extract obj-hdl "BulkCurrent"))
   (do ((test-criterion i-b-act
         LDDimpl-dose obj-hdl))
      (setq LDDimpl_dose
         (compute-new-dose i-b-act
          LDDimpl-dose obj-hdl))
      (run-promis LDDimpl-dose obj-hdl)
      (run-minimos obj-hdl)
      (setq i-b-act
         (extract obj-hdl "BulkCurrent"))
   )
   LDDimpl-dose
)

;;; sample usage
; edit a geometry with the uia PIF editor
(setq obj-handle (uia-ped))

; call the function
(minimize-i-bulk 5e12 obj-handle)

; the output: LDDimpl-dose for minimum i-b
2.01371e13
```

Figure 2: Sample TCAD function

3 Tool Level – The Workhorses

Modules callable at the tool level include all kinds of simulators (process, device, circuit), grid manipulators, discretizers, solvers, measurement data translators, optimizers, graphical editors, previewers, etc., some of which are incorporated in a PIF ToolBox which also provides for intersite to intertool format conversion and vice versa. Until now the device simulator MINIMOS, e.g. [4], the process simulator PROMIS, e.g. [5], and the interconnect capacitance simulator VLSICAP, e.g. [6], have been integrated. Note that Fig. 1 shows the simulators explicitely, but in fact they are just tools like those in the toolbox from the shell point of view.

A new tool can be integrated in three manners, depending on the language it is programmed in (note that these items refer only to integrating tools upwards into the task level

alias the TCAD shell; integrating them downwards into the data level is discussed below):

- LISP tools just have to be loaded and executed from the TCAD shell. This is useful for high-level optimization loops or module sequenzers, which implement mainly logic and consume only small amounts of computation time compared to other tools probably called.
- C 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 builtin 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 program load time.
- Tools in any language that are separate executables can be called with a shell-builtin system call function. Conventional simulators thus can be used like any other shell function, if a (very small) LISP interface is added.

New simulators can be added very easily by replacing input and output functions with corresponding application layer functions (see below). This small change yet allows data level integration into the TCAD system.

Additional flexibility can be gained by splitting the simulator (e.g. separating grid generation, discretization and solver parts) and by 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 a PIF object storage convention [7]. As an example, Fig. 3 shows how the modularized 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. MINIMOS modules are coupled internally; PIF input/output is done by two specialized modules. All modules are controlled by a stack-driven sequenzer written in FORTRAN or LISP. In case of PROMIS there are four interface routines to the shell, called `promis-analytic-implant`, `promis-mc-implant`, `promis-diffuse` and `promis-oxidize`, whereas MINIMOS is callable simply with `run-minimos`. The shell function `run-promis` in the TCAD function example is just a sequence of PROMIS functions simulating a complete process.

The major advantage when building a new simulator is that it is no longer necessary to provide a specific grid

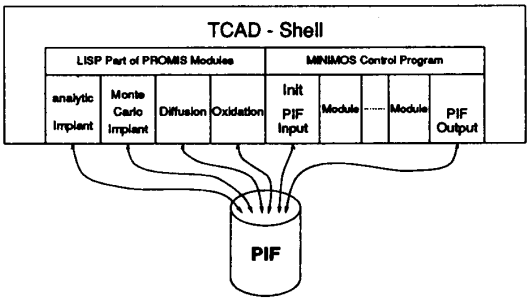

Figure 3: MINIMOS and PROMIS in the TCAD environment

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 can be run (in parallel) on different machines under control of the TCAD shell, thus yielding considerable speed improvement. When modularized appropriately, the most time consuming parts (e.g. linear solvers) can be executed on a supercomputer communicating with the TCAD shell running on a graphics workstation using our PIF linear solver communications protocol [7].

4 Data Level – The Database System

The database is a binary implementation of a textual PIF [2] which has been modified and extended to fulfill the needs of an integrated TCAD system. As an example, Fig. 4 shows part of a textual PIF describing the attributes of a physical device segment (but not its geometry) consisting of $Ga_{0.7}Al_{0.3}As$ at 937 K. An attribute `MySegmentDescription` is defined over the segment `MySegment`. The compound material `MyMaterial` is specified in the subattribute `MaterialType`. A `MaterialComposition` subattribute (named `MyComposition`) and a `Temperature` subattribute (named `MyTemperature`) complete the `SegmentDescription`. Note that it depends on the simulator's capabilities, whether these attributes are recognized or not, but the textual PIF provides a consistent and unique way to specify them.

The TCAD database, consisting of so-called binary PIF files, is accessed from programs with the help of an application interface (Fig. 5). Our implementation of this interface is strictly layered, thus conforming to the most recent software engineering standards. The interface itself

```
(attribute MySegmentDescription
  (attributeType "SegmentDescription")
  (nameList MySegment)
  (attribute MyMaterial
    (attributeType "MaterialType")
    (valueType asciiString)
    (valueList "Gallium" "Aluminum" "Arsenic")
  )
  (attribute MyComposition
    (attributeType "MaterialComposition")
    (valueType real)
    (units "1")
    (valueList 0.7 0.3 1.0)
  )
  (attribute MyTemperature
    (attributeType "Temperature")
    (valueType real)
    (units "K")
    (valueList 937.0)
  )
)
```

Figure 4: PIF Example describing GaAlAs Segment

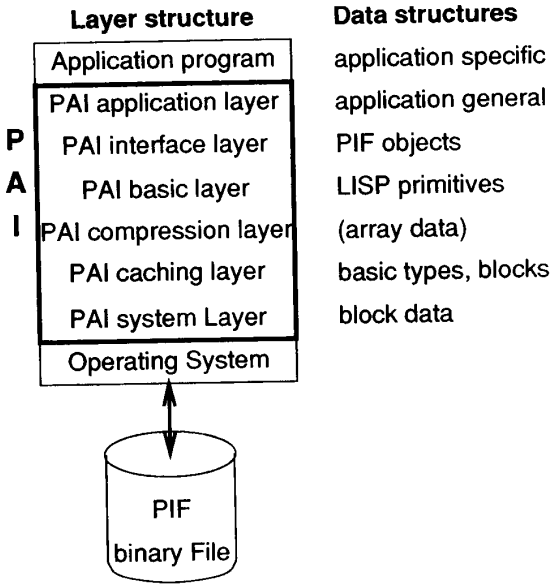

Figure 5: Application Interface Layer Structure

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. In contrast to other approaches (e.g. [3]) we designed even the low-level database structure specifically for TCAD purposes, resulting in considerable performance improvements.

A *system layer* at the very bottom is used to hide system specifics from the rest of the application interface. The interface is open to all operating systems and not restricted to e.g. UNIX, since only a very basic functionality like random read/write on a file is needed.

A *caching layer* implementing a segment-buffer caching algorithm sits on top of the system layer and enhances significantly access speed and memory utilization. A highly (partly at runtime) configurable combination of a split-address method together with a least-recently-used algorithm and variable segment sizes ensure the necessary versatility for the layers above. Various caching degrees are available, including completely uncached files for enhanced data security to fully cached (memory) files for temporary scratch data.

The *basic layer* then is used to access primitive objects which resemble LISP's atoms and lists on a file. Additionally (even compressed) arrays of those low-level objects are available, decreasing the considerable space consumption of e.g. distributed attributes on large and dense grids.

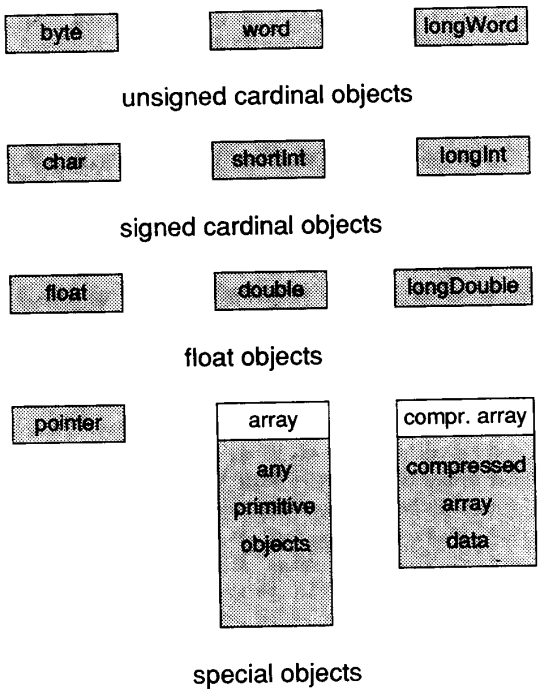

Figure 6: PIF primitive data objects

Fig. 6 shows the available data types. All objects can be linked together into lists; an additional pointer object is used to incorporate lists in another list. A symbol hash table maintained in the basic layer stores PIF object names for quicker object retrieval. It should be noted that the syntactical equality of TCAD shell language and binary PIF, introduced already on this low level, increases significantly the versatility of the system. Thus it is possible, for example, to bijectively transform any datum or program into the binary PIF and the TCAD shell workspace, as long as it is expressible in LISP notation. This implies that even layout data based on this notation (e.g. [8]) can be stored and manipulated with basic layer functions — a precondition to incorporate layout information on higher levels.

The *interface layer* deals with PIF objects which are made up of primitive objects. Designed to work with C applications specifically created for PIF, it performs functions like reading and writing specific object slots or selectively searching objects. However, these functions will work on data structures differing somewhat from conventionally designed simulators.

To provide a convenient interface for existing C and FORTRAN applications, an *application layer* has been designed which deals with all TCAD objects based on PIF. Working on common simulator-internal data structures, it provides powerful routines that create, delete, read or write whole PIF objects while performing unit conversions and coordinate transformations specified by the simulator before reading or writing in order to accomodate the data to the PIF object storage convention [7]. Some rules have to be obeyed, concerning the naming, arrangement and hierarchy of objects written to the database, which are partly guaranteed by the application layer and partly by TCAD tools checking PIF data for consistency. These aspects ensure that adding any new TCAD tool (simulator, measurement interpreter, correlator, etc.) is a simple and straightforward task.

It should be noted that the application layer cannot provide routines suitable for any imaginable existing FORTRAN or C application. However, we tried to generalize the routines as much as possible in such a way, that conventional applications using common data structures like arrays of point coordinates or arrays of point indices representing lines etc. can easily use the application interface. For special applications it may be necessary to add a few routines to the application interface. But this is no big deal since the interface layer provides all the necessary functionality for the application layer routines.

5 Conclusion

The integration of TCAD tools by means of a LISP shell and a binary PIF implementation provides a homogeneous environment for all tasks of advanced device design. With the syntax similarity of TCAD shell extension language and database format close tool cooperation is possible. A specifically customized database binary format together with a layered application program interface ensures high-preformance data storage and retrieval while providing different levels of functionality. Establishing a link to horizontal layout design will make the TCAD framework complete, allowing all activities to be performed in a homogeneous and expandable environment for the process, device and circuit engineer.

Acknowledgements

This project is supported by the research laboratories of: Austria Mikro Systems at Unterpremstätten, AUSTRIA; DIGITAL EQUIPMENT Corp. at Hudson, USA; SIEMENS Corp. at Munich, FRG; and SONY Corp. at Atsugi, Japan.

References

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

[2] S. Duvall, *An Interchange Format for Process and Device Simulation*, IEEE Trans. CAD, Vol. 7, pp. 489–500, 1988.

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

[4] S. Selberherr, *Three Dimensional Device Modeling with MINIMOS 5*, Proc. VLSI Workshop, pp. 40–41, 1989.

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

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

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

[8] *EDIF – Electronic Design Interchange Format Version 2 0 0*, Electronics Industries Association, Washington D.C., 1987.