The PIF Editor -a Data Processor for the VISTA TCAD Framework

G. Rieger and S. Selberherr

Abstract

For the use in TCAD frameworks it is required to perform initial and intermediate data manipulations in order to obtain complete simulation flows. The following paper discusses an approach for an interactive and extensible data editor for the VISTA TCAD framework that may as well be used in automatic simulation flows. It can be configured and extended via an extension language interface and is able to serve as an simple ersatz for the framework shell.

1 Introduction

For simulation of semiconductor processes and devices TCAD (Technology Computer Aided Design) frameworks have become widely accepted.

Usually these frameworks allow to use a couple of different simulators, provide some data interface, have a graphical user interface, and offer a control for complex simulation sequences.

Besides the main components of such a framework like simulators, visualization, and a shell there is need for non-simulation data processing. One is to guarantee a consistent wafer-state of the simulation data after each step, which can be achieved by explicit invocation of an appropriate program [1], or implicitly by a server-like data interface [2].

Wafer-state level data processing demand arises for comfortable specification of initial simulation data, and to calculate uncritical process steps in a simple way instead of performing time consuming simulations, for example directly applying a mask instead of simulating spin–on, illumination, and development. These tasks should ideally be possible user-controlled as well as automatically within a simulation flow.

2 Concept of a Wafer-State Data Manipulator

The demands for a tool fulfilling the above mentioned jobs can be approached from two sides. One is the idea to allow arbitrary wafer-state level data manipulations interactively. The user gets the ability to "construct" a new device or to perform modifications

on an existing device structure. Restricted capabilities of sub–wafer-state changes for repairs or tricks may be desired too.

The other aspect is batch processing. Initial device structures should be generated from scratch with some parameters, and more or less simple operations should be performed on existing devices. The latter requires the processing to be robust because the structure of the device might change for different simulation runs. For example the deposition of a contact, depending on the surface's vertical position and its relief, is a non-trivial geometric operation. To make the developer independent of built-in models a comprehensive extension language is required for robust ("intelligent") solutions. "Comprehensive" means that not only some geometric and physical commands be available, but also elements for primitive calculations, variable handling, data structures like arrays and lists, and conditional and looping constructs.

The graphical properties of such a tool are related to classical visualization tasks, because comfortable editing presumes displaying of the actual device. It may therefore well be considered to extend the frameworks visualizer for these duties.

3 The PIF Editor within the VISTA Framework

The VISTA (Viennese Integrated System for TCAD Applications, [3]) framework uses a common data format, the PIF (Profile Interchange Format, [4]), that is defined with a powerful syntax and is implemented in binary form offering random access. VISTA has a graphical, X11 toolkit based user interface. The code for tool binding, the shell, the simulation flow controller [5], and even its CASE (Computer Aided Software Engineering) environment are programmed in LISP.

It allows fully automatized simulation of complete semiconductor processes and the resulting device characteristics [6].

A tool to meet the above specified requirements has been developed. Its name is "PIF Editor" (PED) because its first intention has only been interactive editing of PIF data. Due to the use of a very specialized geometric data format in the visualization tool, it has been designed as a new program.

The PIF editor is a program that allows interactive or batch mode processing of PIF data. It has a list-oriented interface to the wafer state database to enable complicated operations. In interactive mode restricted visualization and comfortable editing of the device structure are possible. All features are accessible from the integrated LISP interpreter and thus can be used in command files and macros. The user interface too is subject to LISP-programming and can be configured and extended. The ability to invoke external executables rounds off the PIF editors characteristics.

3.1 PIF Editor Structure

The different requirements are best met with a modular structure. The *kernel* of the PIF editor provides full access to the PIF data and some high-level operations. It is implemented in C, but its functions can be accessed from LISP as well. It is the only module required for pure batch processing. With adding the *view* module that projects and transforms the PIF data depending on dimension and mode (geometric, physical, profile etc.) to simple graphical data, and with a *device*-module interpreting and displaying or printing this graphical format, hardcopy capability is achieved.

With a window-based and extended *device* accepting and forwarding user interaction via the *view* module, and a *shell* with comfortable command elements like menus, that embeds the *device* — both modules realized for the X11 window system in the current implementation — and, last but not least, a state machine (the *control* module) checking user input and triggering appropriate reactions, the PIF editor gets full editing capabilities.

3.2 Data Processing

The PIF editor has a special functional interface to the PIF database called CLS (C list support), that allows efficient random access to the device structure. In contrast to the PAL (PIF application Layer), that supports reading and writing of arrays of points, lines etc. [7], the CLS provides lists of points, lines etc., and, e.g., the references of a face to its surrounding lines are organized as a list too. There exist functions for list processing like applying a function to each element, to find minimum or maximum or to sort lists. Each PIF object is identified by a handle which is, hidden from the application, a pointer to a record that contains data describing the object and all its dependencies and back-references.

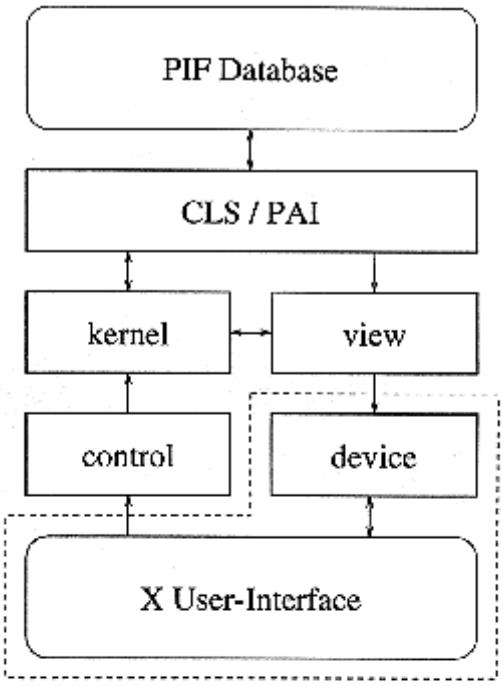

Figure 2: PIF Editor Block Diagram

These object handles and the functions for inquiring, creating, modifying, and deleting the objects are available not only in C but also in LISP allowing data processing at extension language level.

The PIF editor's kernel has a couple of high level functions, e.g. to generate solids from a set of faces, for geometry modeling, or to find exposed boundaries of a device.

3.3 External Executables

It is possible to invoke external executables from LISP via an operating system independent interface (UNIX and VMS are currently supported). This feature has been developed for the VISTA shell and the VMAKE (Vienna Make), but is very useful for the PIF editor too. The common way is to write a LISP function that accepts the arguments for the program. These may be checked and converted to text, and with a special LISP call accepting the program's name and its options and parameters the run is started. Waiting for the termination of the program or parallel continuation of the session are possible.

Especially in combination with the later described user interface those call functions can be used like conventional code parts.

A typical utilization of this feature is invocation of a grid generator. Interactively a segment the grid shall be defined on may be selected and the configuration parameters for the program can be tuned in a dialog window. The LISP code inquires the PIF identification of the segment (segment–list name and index), flushes the in–memory CLS data to disk to synchronize the file–based PIF, closes the PIF file to allow writing by the grid generator, and invokes it with command line parameters specifying the name of the PIF file, the segment, the grid's name, and some special parameters. After termination the PIF editor reopens the PIF file, searches for the grid by its name, and displays it (see Fig. 1).

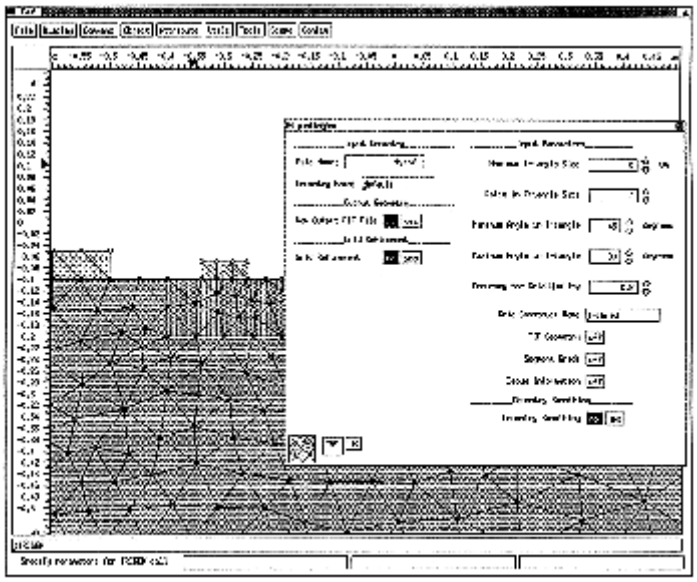

Figure 1. PIF Editor with Panel for TRIGEN Invocation and Resulting Grid

3.4 User Interface

In interactive mode the PIF editor activates an X11 window system interface, consisting of some menus, a drawing area, a command line, mode information la-

bels, and temporary dialog windows. X11 events triggered by the user generally cause invocation of callback functions. Sources of user events are especially menu selection, command line entering, mouse pointer movement, mouse button press, or finishing a dialog window. In the first evaluation stage a C function is called. This causes the LISP interpreter to evaluate a function referenced in a certain place. Some data characterizing the event are passed, e.g. mouse pointer position, or so called client–data that has been stored with the LISP function.

Evaluation of such a function may directly perform some action if no further data has to be entered by the user, or it can generate and send a token to the infinite state machine [8].

This state machine is configured with a couple of rules. Each rule specifies a valid sequence of token types. When a token matches the expectations of the rule, its data is stored on a stack. After accepting the last token for a rule, an appropriate function is invoked which has access to the data on the stack.

A rule may specify other rules to be applied instead of requiring some token. In this case the result of such a subrule is used in place of a token's data. The result of a rule is the result of the function called on its termination. This nesting mechanism may be exploited to unlimited depth.

In many cases the state machine has to select one of many different subrules depending on the type of the arriving token.

The processing advance of the state machine's current rule determines the mode of the PIF editors user interface, which consists of the functions and data stored for evaluation on mouse button press, the mode title and mouse button help texts, the cursor appearance, the temporary mouse-position dependent drawing, and popping up or down of dialog shells.

The rules of the state machine are specified in LISP and may be supplied during startup as well as runtime.

3.5 Extensions

All user interface setup and control is programmed in LISP. Generation of menu entries, specification of state machine rules, help texts, reactions to mouse movement etc. are open to LISP programming. In combination with the data processing features it is possible to implement complete new functions and editing modes to meet special requirements without recompiling the program.

It is possible to configure the PIF editor for a special purpose, e.g. to generate device structures for use by the device simulator MINIMOS [9]. When all specialized functions, mode specifications etc. are placed in one file, this can be seen as a "style file". A collection of such style files allows the user to load only the currently required style(s) preventing a confusing overload of the user interface. Even the generation of personal styles is possible.

The speed of interpreted LISP is sufficient for configuration and control purposes. CPU-intensive operations like searching or numerical transformations are implemented in C. Binding C functions to the LISP interpreter is supported by a highly automatized mechanism in the developing environment.

3.6 Three–Dimensional Approach

The PIF editor supports one–, two–, and three–dimensional device geometries. While the properties of the default mode restrict geometric operations to manipulation of simple elements like points, lines or faces with respect to their environment, it is straightforward to develop a mode for objects of a higher semantic level. In this mode some basic object types can be specified in a textual way and the resulting device can be displayed immediately. This is especially useful for three–dimensional applications where the mouse pointer position defined by only two coordinates is not unique, and parts of the device are hidden and cannot directly be accessed.

3.7 Programming Example

Fig. 4 shows how a simple function **contact** for applying a rectangular contact can be programmed. It

expects parameters for position, size, material, and voltage for later device simulation. Then it creates a rectangular face of the given material. In the example two contacts are generated using the new function.

Fig. 5 shows the interactive utilization of the above function. To simplify the example the material is hard-coded to "Al". A rule "`Al-contact`" for the state machine is defined that allows (but does not force) the user to specify the position of the contact with the mouse. Then the state machine expects a second coordinate pair to yield the size of the rectangle. In this phase mouse movement causes temporary drawing of the expected shape by automatically invoking the function `ped::elastic-contact`, Fig. 3.

The last missing component is the contact voltage. A dialog window prompts the user for a real number, and the mouse buttons are configured to the most common values. When all input is provided the function `contact` is invoked and undertakes the creation of the contact with permanent drawing.

Generation of a menu entry and hooking the new rule to the main state make the example complete.

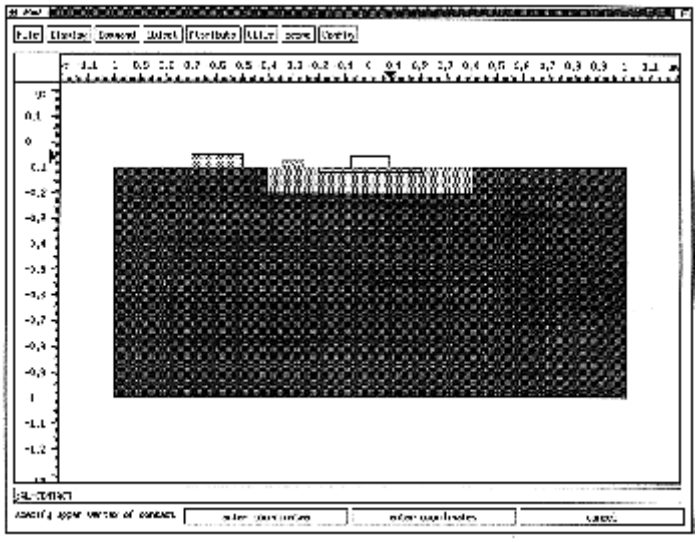

Figure 3: PIF Editor during Interactive Creation of a Contact

```
(defun contact
  (ybase xmid width height mat voltage
   &aux point1 point2 point3 point4
        segment1 segdesc1 mattype1 contvolt1)

  (setq point1 (point (- xmid (/ width 2)) ybase))
  (setq point2 (point (+ xmid (/ width 2)) ybase))
  (setq point3 (point (+ xmid (/ width 2)) (+ ybase height)))
  (setq point4 (point (- xmid (/ width 2)) (+ ybase height)))

  (setq segment1
        (segment (face (list
                        (line point1 point2)
                        (line point2 point3)
                        (line point3 point4)
                        (line point4 point1)))))

  (setq segdesc1  (attribute NIL "SegmentDescription"  segment1))
  (attribute segdesc1 "MaterialType"   segment1 mat)
  (attribute segdesc1 "ContactVoltage" segment1 voltage
             :unit "V"))

(contact 0.0  -1.0  0.1  0.05 "Al" 0.0)
(contact 0.0   1.0  0.1  0.05 "Al" 3.0)
```

Figure 4: Example for a Data Processing Function

```
;;; This function call generates and stores a new rule "Al-contact".
(newrule
 "Al-contact"   ; the nonterminal to be extended
 NIL            ; we have a constant terminal as first element, see below
 '(:AL-CONTACT
   (mode "specify middle of contact surface"    ; title of mode
         ("enter coordinates" "%G\n")           ; left mouse button
         ("" "")                                ; middle mouse button
         ("cancel" ":CANCEL\n")                 ; right mouse button
         NIL NIL NIL NIL)                       ; no dialog-shell
   "coords"     ; get a point - by any of the rules specified for "coords"
   (mode "specify upper vertex of contact"      ; title of mode
         ("enter coordinates" "%G\n")           ; left mouse button
         ("enter coordinates" "%G\n")           ; middle mouse button
         ("cancel" ":CANCEL\n")                 ; right mouse button
         #'ped::elastic-contact #'ped::elastic-contact  ; draw/undraw-elastic-rectangle
         ($ 3)                                  ; mid coordinates as client-data
         NIL)                                   ; no dialog-shell
   "coords"            ; get a second coordinate pair as data item 3
   (mode "specify contact voltage"              ; title of mode
         ("ground"   "0.0\n")                   ; left mouse button
         ("5.0 Volt" "5.0\n")                   ; middle mouse button
         ("cancel"   ":CANCEL\n")               ; right mouse button
         NIL NIL NIL
         ped::contvolt-dialog)                  ; hand from contactVoltage
   "float"                                      ; voltage
   (contact (cadr ($ 3))                        ; extract y-base
            (car ($ 3))                         ; x-mid
            (* 2 (abs (- (car ($ 5)) (car ($ 3)))))     ; width
            (- (cadr ($ 5)) (cadr ($ 3)))       ; height
            "Al" ($ 7))))                       ; material, voltage

;;; generate a menu entry to trigger the "contact" rule
(ped::add-to-menu
 "Object"               ; the menu where the new entry comes
 "Al-contact"           ; the name of the new entry
 #'ped::rt-append-text  ; the function to be invoked on selection
 ":AL-CONTACT\n")
```

Figure 5: Example for Interactive Binding of a Function

4 Conclusion

The PIF editor is a flexible and comfortable data manipulation tool in the VISTA framework. It may play an important role within simulation flows, like initial data creation. It may be configured or programmed with LISP for a wide range of purposes. It gives access to the PIF database in 1, 2, and 3 dimensions, and allows invocation of other tools. In interactive mode it provides a comfortable user interface and supports the engineer in many ways.

Acknowledgements

This work has been sponsored as part of ADEQUAT (JESSI project BT1B), ESPRIT project 7236; and ADEQUAT II (JESSI project BT11), ESPRIT project 8002.

We are also very grateful to S. Halama who contributed to the basic ideas of the PIF editor, and to the many users that helped with their suggestions to develop it towards meeting realistic requirements.

References

[1] S. Halama. *The Viennese Integrated System for Technology CAD Applications–Architecture and Critical Software Components.* Dissertation, Technische Universität Wien, 1994.

[2] SWR Working Group of the CFI TCAD TSC. Semiconductor Wafer Representation Procedural Interface V1.0. Technical Report CFI Document TCAD-91-G-2, CAD Framework Initiative, Austin, Texas, July 1992.

[3] S. Halama, F. Fasching, C. Fischer, H. Kosina, E. Leitner, Ch. Pichler, H. Pimingstorfer, H. Puchner, G. Rieger, G. Schrom, T. Simlinger, M. Stiftinger, H. Stippel, E. Strasser, W. Tuppa, K. Wimmer, and S. Selberherr. The Viennese Integrated System for Technology CAD Applications. In F. Fasching, S. Halama, and S. Selberherr, editors, *Technology CAD Systems*, pages 197–236. Springer, 1993.

[4] F. Fasching, C. Fischer, S. Selberherr, H. Stippel, W. Tuppa, and H. Read. A PIF Implementation for TCAD Purposes. In W. Fichtner and D. Aemmer, editors, *Simulation of Semiconductor Devices and Processes*, volume 4, pages 477–482, Konstanz, 1991. Hartung-Gorre.

[5] Ch. Pichler and S. Selberherr. Process Flow Representation within the VISTA Framework. In S. Selberherr, H. Stippel, and E. Strasser, editors, *Simulation of Semiconductor Devices and Processes*, volume 5, pages 25–28. Springer, 1993.

[6] G. Schrom, D. Liu, Ch. Pichler, Ch. Svensson, and S. Selberherr. Analysis of Ultra-Low-Power CMOS with Process and Device Simulation. In C. Hill and P. Ashburn, editors, *24th European Solid State Device Research Conference - ESSDERC'94*, pages 679–682, Gif-sur-Yvette Cedex, France, 1994. Edition Frontieres.

[7] F. Fasching, W. Tuppa, and S. Selberherr. VISTA-The Data Level. *IEEE Trans. Computer-Aided Design*, 13(1):72–81, 1994.

[8] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers*. Addison-Wesley, 1986.

[9] C. Fischer, P. Habaš, O. Heinreichsberger, H. Kosina, Ph. Lindorfer, P. Pichler, H. Pötzl, C. Sala, A. Schütz, S. Selberherr, M. Stiftinger, and M. Thurner. *MINIMOS 6.0 User's Guide*. Institute for Microelectronics, Technical University Vienna, March 1994.