VISTA Status Report
June 1993

F. Fasching, S. Halama, C. Pichler, H. Pimingstorfer,
G. Rieger, G. Schrom, S. Selberherr, M. Stiftinger

Institute for Microelectronics
Technical University Vienna
Gusshausstrasse 27-29
A-1040 Vienna, Austria

Contents

Summary

The work of the VISTA team was largly characterized by efforts to stabilize the VISTA system. This has been archieved by some of the following contributions as well as periodic automatic tests of the complete software.

An important step towards user level consistency of the framework has been achieved by a new user interface forcing homogeneous "feeling" for all parts of VISTA, not depending on programmer, programming language, or tool.

A new material database may serve as a unique source for material properties for all adapted tools.

A general grid support has been designed and implemented, the *Unstructured Grid Support* (UGS). It defines a consistent way to handle grids that cannot be represented by tensor-products.

To avoid tedious and faulty manual language bindings an improved automatic XLISP language binding, the first part of the *Tool Abstraction Concept* (TAC), has been introduced.

The *Simulation Flow Control* module (SFC) was added to the framework which provides a mechanism for executing multi-step simulation tasks in a comfortable manner, supporting the definition of long *process flows* by means of referenced process modules.

An example of VISTA user interface and TCAD-shell utilization is presented that allows comfortable SUPREM3, SUPREM4, and MINIMOS coupling.

The PIF *Editor* (PED) is a useful PIF data front end for specifying and modifying 1-D and 2-D device geometries.

Another VISTA example utilization shows a way to sample various device characteristics with MINIMOS in a generic and comfortable way.

1 The VISTA User Interface

1.1 Structure

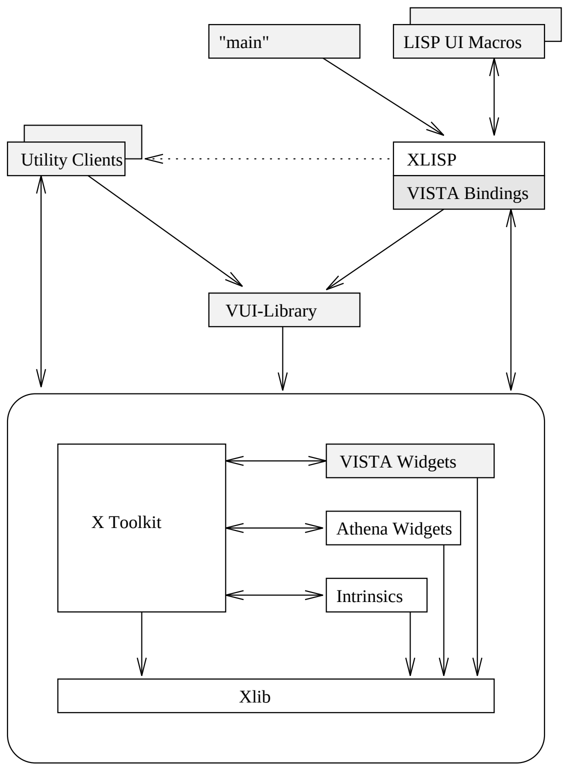


Figure 1: The structure of the VISTA user interface. Shaded boxes represent extensions to the public domain products XLISP and the MIT X Window system. The arrows indicate the sequence of function calls between different parts of the user interface.

The structure of the VISTA user interface is shown in Figure 1. The bottom layer is the X Toolkit[1], an object-oriented subroutine library, designed to simplify the development of X Window applications. The X Toolkit defines methods for creating and using widgets, which appear to the user as pop-up windows, scrollbars, text-editing areas, labels, buttons, etc. Basic functionality is provided by the generic *Athena widgets*, which are part of the MIT X11 distribution. We have decided to use this widget set rather than any other open standard, because a migration from these generic widgets to another widget set (like OSF/Motif, or Open Look) is significantly easier than vice versa.

A widget-wrapping layer has been put on top of these widgets in order to achieve some widget-set independence. All widgets are created and modified via specific functions rather than via the generic interface of the X Toolkit. This facilitates the potential migration of the entire user interface onto another X Toolkit-based platform.

In addition specialized VISTA widgets have been developed on top of the widget-wrapping layer for supporting TCAD-related information flow. The VISTA widgets are also created and accessed via specific functions, so that they can more easily be replaced by other widgets, should the need arise.

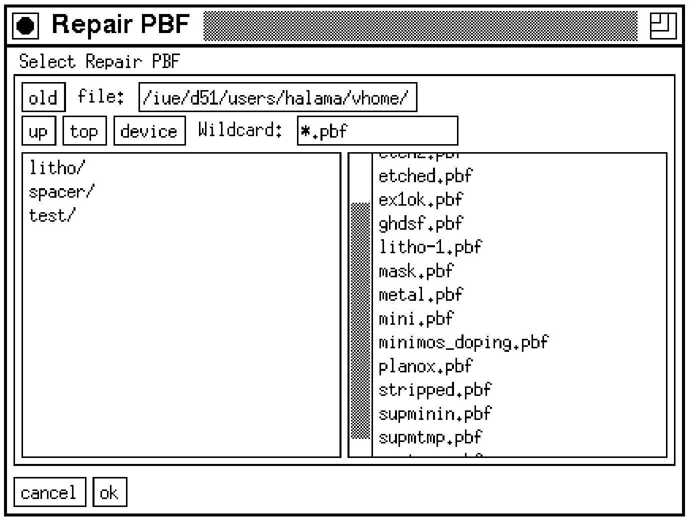
The top layer, the VUI (VISTA *User Interface*) library serves two purposes. It provides some often needed higher-level operations and it simultaneously contains most of the policy which is shared among VISTA applications. In other words, the VUI library takes care that different parts of VISTA look alike and behave similar. Interactive applications (like visualization clients or the device editor) have their own VUI-based user interface, whereas applications requiring no user interaction (like simulators or converters) are provided with a front-end user interface which is executed by the XLISP interpreter.

1.2 New Components

Many parts of the user interface that have previously been implemented in LISP have been redesigned and re-implemented in C to be accessible by C applications. The LISP environment, however, has proven to be very well suited for prototyping. The LISP-bindings for this C-coded parts are generated automatically, so that a homogeneous programming interface can be guaranteed.

1.2.1 File Selection

As it is not provided with the Athena widget set, we have implemented an advanced file selection widget (see Figure 2), which allows operating system transparent specification of files (including a *GNU Emacs* like filename completion) using a string subwidget and operating system independent traversal of the directory tree and selection of existing files using list subwidgets.



file: /iue/d51/users/halama/vhome/
Wildcard: *.pbf

```
litho/
spacer/
test/
```

```
etched.pbf
ex1ok.pbf
ghdsf.pbf
litho-1.pbf
mask.pbf
metal.pbf
mini.pbf
minimos_doping.pbf
planox.pbf
stripped.pbf
supminin.pbf
supmtmp.pbf
```

Figure 2: The VISTA File Selection Widget

1.2.2 VUI Library

The VUI library contains functions which create often-used combinations of several widgets in one step, arrange them and set up all required connections and callback functions. These widget *bouquets* behave as if they were single composite widgets and are indistinguishable from the user's point of view. This approach is similar to the OSF/Motif "Convenience Function" concept[2], and helps to maintain a unified appearance for different VISTA applications.

References

[1] P.J. Asente and R.R. Swick. *X Window System Toolkit, The Complete Programmer's Guide and Specification.* Digital Press, 1990.

[2] *OSF/Motif Programmer's Guide, Release 1.1*, 1991.

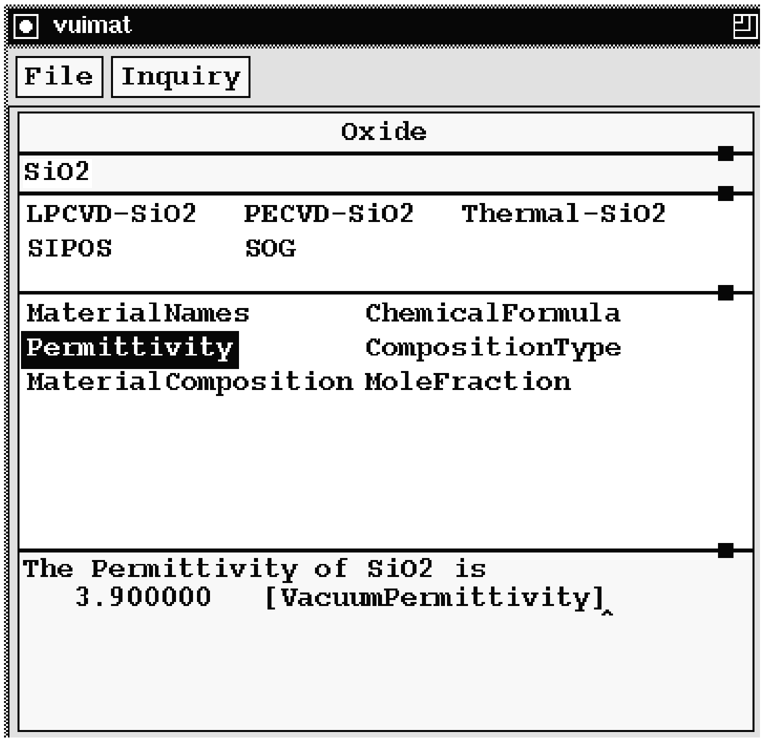
2 The VISTA Material Server

As a part of VISTA, a material database system, the so-called VISTA material server has been designed and implemented. It consists of one or more material databases and a set of functions for accessing and maintaining the data.

A simulator usually needs to know the properties of the segments belonging to the simulation domain. Such data are either stored directly in the PIF–file or are "referenced" by a material, i. e., the PIF–file contains just the name of the material. The simulator can then use the VISTA material server to inquire the material's properties or just identify it to apply some built-in model.

The intention of the VISTA material server is to provide a semantic buffer layer between the different tools which must deal with materials on different levels of abstraction and to free the tool programmers from the tedious task of case-checking and exception-handling. This eases the development and maintenance of simulation tools and at the same time adds a lot of flexibility and maintainability to the whole TCAD system.

For example, a process simulator can write specific material names such as "BPSG" or "Si3N4" onto its output-file and a device simulator can then use the VISTA material server to recognize the material as an insulator and inquire its permittivity. Furthermore, combinations of two or more materials can also be handled. This allows the characterization of the dopants in a semiconductor as well as the storage of process-related material properties when several materials are involved in a process step.



```
The Permittivity of SiO2 is
   3.900000   [VacuumPermittivity]
```

Figure 3: The Interactive Material Database Program

3 Unstructured Grid Support

3.1 Overview

The *Unstructured Grid Support* (UGS) module has been introduced into VISTA to cope with the multitude of non-tensor-product grids used in today's TCAD tools. It is designed to provide generic storage and retrieval functions for any unstructured grid in any geometric dimension through a flexible extension mechanism.

3.2 Grid Representations

Figure 4 shows that an unstructured grid is stored in the PIF through a surrounding `grid` construct, as is the case for tensor product `orthoProduct` grids. But in contrast to the latter, the points the grid consists of are stored in a `valueList` construct inside a `pointList`. Then a `faceList` construct is used to group single points together into a grid element. The number of points per element depends on the element type, and need not be the same for each element, thus grids consisting of multiple different elements can be stored in the PIF.

To determine the element type of each face of the grid, an additional `attribute` construct inside the `grid` is used. This `attribute` has the `attributeType ElementType`, and a `valueType` of `asciiString`. If all elements of the grid are of the same type, then one string entry in the `attribute` specifying the element type is sufficient. However, if there are more than one element type in the grid, a string for each element describing its type has to be specified.

3.3 Extension Mechanism

Element type strings conform to their names which are listed in a configuration file, where new element types can be made easily available to the framework, by just specifying a decomposition and an interpolation function. The decomposition function receives the element and a callback function as arguments, and has to use this callback function in decomposing the specific element into simplex elements of their respective dimension, i. e. triangles in two and tetrahedrons in three geometric dimensions. The interpolation function receives the element nodes, attribute values defined on them, and a point lying inside the element as arguments, and has to return the interpolated attribute value on this point.

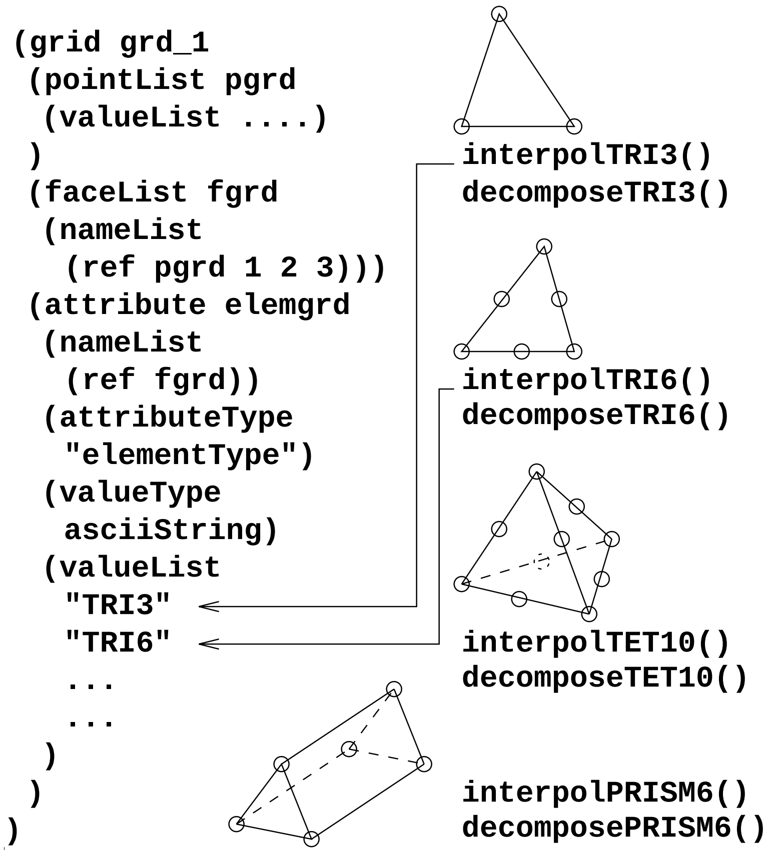
By providing these two functions, adding the new element type to the configuration file and recompiling VISTA, support for this new element type is immediately available to the framework and all applications.

3.4 Functional Interface

The UGS functional interface provides initialization, reading and writing operations on unstructured grids. Interfaces exist for C applications and — through automatically generated bindings — to FORTRAN applications.

To write an unstructured grid, the initialization routine has to be called. After that, a UGS-specific routine to write the grid `pointList` has to be called. Then for each grid element a write function has to be applied, to write the element node indices to the `faceList`.

Reading an unstructured grid is also straightforward: After calling the initialization routine, the standard PAI function to read the `pointList` can be used. In a loop the element read function can be used to read the element type and its nodes from the grid's `faceList`. After the last element has been read, a cleanup function has to be called, to free internal data structures.



```
(grid grd_1
 (pointList pgrd
  (valueList ....)
 )
 (faceList fgrd
  (nameList
   (ref pgrd 1 2 3)))
 (attribute elemgrd
  (nameList
   (ref fgrd))
  (attributeType
   "elementType")
  (valueType
   asciiString)
  (valueList
   "TRI3"
   "TRI6"
   ...
   ...
  )
 )
)
```

Figure 4: Unstructured Grid Representations

4 The Tool Abstraction Concept

The aim of the *Tool Abstraction Concept* (TAC) is to provide a unified and homogeneous method for the *formal* description of *tools* (which are right now mostly functions written in an implementation language). From this unified tool description, several utilities are generating helpful things like documentation, language bindings (which would otherwise require a lot of tedious programming work) and parts of the user interface.

The previously existing prototype implementation of the TAC has been redesigned and joined with the LISP-based *vmake* utility. During the build process of the VISTA distribution special comments in the source code are read and from these formal descriptions, bindings to the LISP interpreter are generated automatically. This code would otherwise have to be written manually, which is tedious work and prone to errors.

Module	description	N_f	N_c	binding	code size
`xvw`	Extended Widget Set	105	19	115 kB	531 kB
`vui`	User Interface Library	83	3	96 kB	254 kB
`ve`	Global Error System	14	37	20 kB	165 kB
`svg`	Graphics Library	22	10	21 kB	73 kB
`ptb`	PIF Toolbox	8	29	26 kB	190 kB
`ver`	Version Control	0	7	1 kB	0 kB
`vos`	OS Interface	(63)	12	(2) kB	214 kB
`pai`	PIF Application Interface	(20)	(117)	(54) kB	2256 kB
	total	315	234	335 kB	3683 kB

Table 1: Number of functions (N_f), number of constants (N_c), size of code for the XLISP interface, and module size of every module that is linked with the XLISP interpreter. Numbers in parentheses indicate manual binding, all other code is generated automatically.

Table 1 gives an overview about the modules which are currently bound to the XLISP interpreter using the TAC. It is planned to extend the TAC in the future to handle the FORTRAN binding generation as well.

5 The VISTA Simulation Flow Control Module

5.1 Overview

The VISTA *Simulation Flow Control* module (SFC) provides a mechanism for defining, editing, and executing simulation tasks consisting of an arbitrary number of simulation steps carried out on an initial wafer model. *Process flows* can be defined by using symbolic names for simulation tools, and long sequences can be build up from smaller, predefined ones.

5.2 Defining a Task—Process Flow Representation

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.

In addition to tool calls, a couple of control commands and keywords (ref. table 2) may appear in the simulation flow description.

Keyword	Description
`start-with`	Loads a PIF file to start subsequent calculations
`save-state`	Enables saving of intermediate results
`dont-save-state`	Disables 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

Table 2: SFC Control Commands and Keywords

If the user wants to call a predefined process sequence, the process reference keyword `PROCESS` followed by the name of the file containing 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.

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.

5.2.1 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 *process module* 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)))
)
```

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.

5.3 The XLISP Tool Bindings

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 .

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 .

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.

6 An Example of VISTA User Interface and TCAD-Shell Programming

6.1 Motivation

It is an often done design task to run the sequence of a process simulation, device simulation and parameter extraction in order to see the changes in the device parameters due to a change or variations in one or more process parameters. In this application SUPREM3 and SUPREM4 are used as (1-D and 2-D) process simulators. The results are combined to a MOS doping profile and fed into MINIMOS which performs the device simulations. The result in the form of a I-V characteristic is the input for a parameter extraction program which automatically determines the parameters of the MOS device.

6.2 Overview

Formerly this task was performed step by step with the help of various shell scripts. Therefore the device engineer had to wait until the previous task had finished. The input necessary was partially redundant and the engineer had to care for consistency of all input of the various tasks. The program which puts the 1-D or 2-D process output together to a doping profile in the MINIMOS format requires a small input deck which had to be manually written and consistent with the process simulations. A 1-D MOS doping profile can be put together from two or three SUPREM3 simulations. It can be symmetric, asymmetric or have a lightly doped drain (LDD). A 2-D profile can be generated only from a SUPREM4 simulation or from one or two SUPREM4 simulations and one SUPREM3 simulation. The doping profile conversion program is able to handle six different cases.

Another aspect is the utilization of a distributed computing environment. If more than one SUPREM task has to be performed they can be run in parallel. Also the MINIMOS runs for all bias points are independent from each other. VISTA allows distributing parallel tasks in a highly configurable manner. Also synchronizing jobs (e. g. the parameter extraction can be started only after all MINIMOS runs have finished) is automatically provided.

6.3 User Interface

The user interface functions are easily callable from XLISP, the TCAD-shell programming language. The user interface provides special fields for each type of required input (booleans, integers, reals, strings, choices, files). Therefore very user-friendly and self explanatory interfaces can easily be generated which help minimize possible input errors.

6.4 Panel

The user interface panel (window) is divided into five small sections. Their order equals the sequence of the simulation steps. The first section determines the way in which the MOS doping profile is put together (mode). The second section contains the input-fields for SUPREM, the third those for the profile conversion program, the fourth those for MINIMOS and the fifth those for the parameter extraction program.

6.4.1 Mode Section

This section selects one of the above six cases in which the SUPREM doping files are put together to a 1-D or 2-D MINIMOS doping profile. The choice made in this section determines how many and which SUPREM runs have to be performed.

6.4.2 SUPREM Section

This section consists of only one button for a pop-up menu. This pop-up menu contains fields for the SUPREM files. The layout of this SUPREM panel is adapted according to the choice made in the mode section. So the number of files to be entered and the description of the required input are always consistent. There is the possibility to choose both SUPREM input decks and result files. If a result file is chosen (the criterion is the file extension) no SUPREM run is done for this file. File selection can also be done very easy using the VISTA file selection widget. Therefore also if the choice made is confirmed this panel is popped down again.

6.4.3 Conversion Program Section

In this section some input concerning things like offsets for LDD inplants are determined. From these values and the choice made in the mode section the small input deck for the conversion program is written automatically by the TCAD-shell.

6.4.4 MINIMOS Section

Only the input keys for MINIMOS which are most likely to be changed in the application like device type, gate length, and some more can directly be changed from this section. There is nevertheless a button which allows the user to pop up the interactive MINIMOS input deck editor to have the possibility to change the other input keys. Different default values for n-channel and p-channel devices are automatically provided.

6.4.5 Parameter Extraction Section

In this section the model used for the parameter extraction is chosen. Because of the very specific problem for fitting standard MOS devices the parameters which are to be optimized are predefined and cannot be changed. The same is true for the start values.

6.5 Implementation

If the OK-button in the main panel has been pressed, the SUPREM runs are performed in parallel (if at least one input file in the SUPREM panel has been chosen). After the last SUPREM task has finished, the doping profile conversion program is called with the automatically generated input deck. Thereafter MINIMOS is invoked. The MINIMOS result files are appended in the proper order after the last MINIMOS run has finished. As last task the parameter extraction is done.

6.6 Conclusion

This example implementation of an often done design task in VISTA provides a user friendly interface. After all input has been provided the tasks are run without further user interaction, and a distributed computer environment allows much shorter computation times. The fact that all input is provided from one panel at the very beginning of the whole task allows also consistency checking for all input.

7 The PIF Editor

7.1 VISTA Front End

An important part of a TCAD system is a comfortable to use and comprehensive data front end. If the visualization is the users eye, this part might be called his hand. In the VISTA system this tool is the PIF *Editor* (PED). It allows the user to directly access the PIF data.

The PED is the ersatz for former typing long lists of numbers with a text editor. It helps to input geometrical and physical data in a user friendly manner. Due to the graphical input and view many errors can be detected immediately or totally be avoided.

The PED has been designed for use in the PIF VISTA-environment thus fitting well and consistently into the framework.

The user interface of the PED has been developed with intense contact with and reaction to users of VISTA, therefore being both simple to use and powerful.

7.2 PED Window

The PIF editor window as it is presented to the user consists of the following main parts:

- A row of menu buttons.
- The graphical (drawing) area.
- The text command line.
- A row of status displays.

Temporary pop-up-windows for confirmation or typing in numbers or names complement the image.

7.3 Main Features

The PED can be used to create, view, modify, or delete one- and two-dimensional device geometries together with their most important physical properties.

It can read data from PIF files and store data to PIF files. It supports the hierarchic structure of PIF geometries by automatically merging identical points, lines etc. during input.

7.4 User Interface

The user interface of the PED is largely built on the VISTA user interface. Only the PED-specific parts that have no equivalent in other VISTA parts are designed especially for PED, particularly the drawing area with the rulers due to its very extensive interactive requirements.

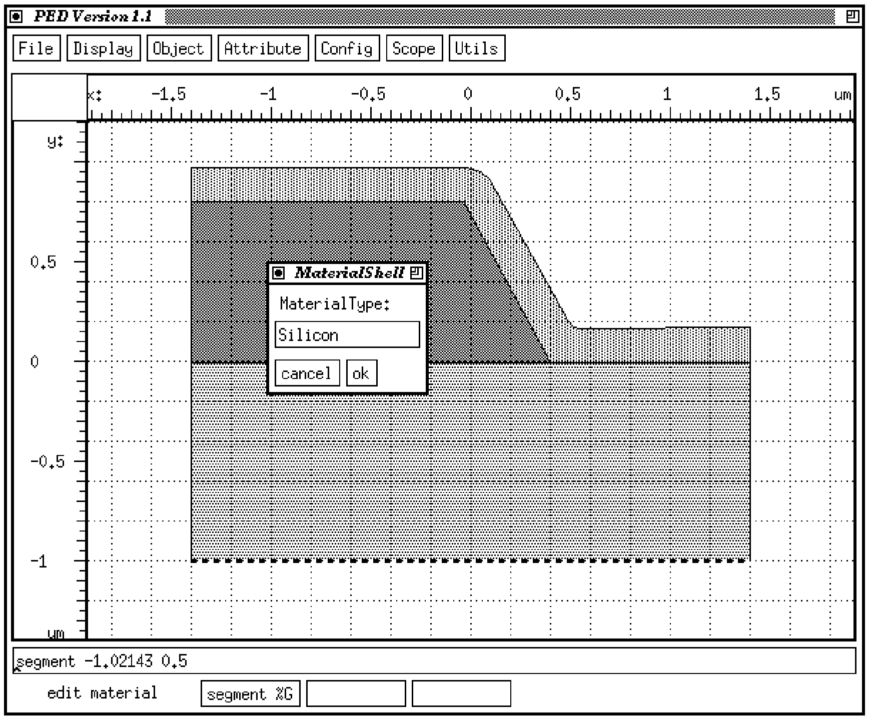


Figure 5: PIF Editor Window

7.5 Display Control

A set of functions allows the user to setup the display: The size of the image can be made smaller or larger with the zoom function or automatically be scaled to the window size by auto scaling. The position of the image may be shifted with the pan function or by auto centering. The horizontal and vertical axes can be assigned to x and y of the PIF data in any order and orientation. Many minor properties like tick distance and length of the scales can be configured by X resources.

Viewing and constructing data can be supported by a background grid and a "magnet"-function that rounds the mouse coordinates to even numbers.

7.6 PIF Conformance

The modes for entering device structures are tightly coupled to the PIF data structure: Modes for point, polyline, face, boundary, and segment input are provided. Polyline and face mode automatically create all required points and/or lines. The other modes build on selecting underlying objects corresponding to the hierarchic PIF definition. Polylines are broken automatically to simple lines limited by two points each.

7.7 Other Editing Facilities

In addition to the constructing modes as mentioned above there is a mode to move points to other positions, and to delete any objects.

7.8 Attributes

Only the most important PIF attributes are supported up to now. These are the `MaterialType` and the `BoundaryConditionType` attributes. Both have text string as values and are related to segment and boundary resp.

8 Generic Device Characteristics

8.1 Overview

The VISTA TCAD shell user interface now supports generic device characteristics which are MINIMOS scalar output values as a function of an input deck key with another input deck key as parameter. Any input deck key of data type `REAL` (this limitation is caused by the graphical user interface and the plot procedures) of any input deck directive can be selected as abscissa or parameter. This is the generalization of the input deck directive `STEP`, which is applicable for bias conditions only.

8.2 User Interface

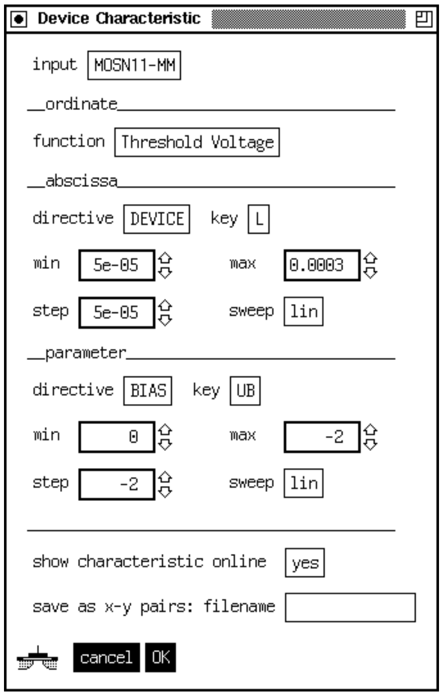
The panel to compute and display online general device characteristics is shown in Figure 6. The first line shows the input deck name. The remainder of the panel is made up of three main parts: the ordinate, which sets the function to compute, the abscissa, and the parameter of the characteristic. The functions to choose are run-time configurable. Currently available are:

Function	Symbol	Unit
Threshold Voltage	Vth	V
Drain Current	Id	A
Bulk Current	Ib	A
Bulk Current Maximum	Ib_max	A
Transconductance	gm	A/V
Gate Swing	S	V/decade
Saturated Vth	VTF	V
Maximum Drain Current	IDS	A
Ib_max for Ud=Udd, Ub=0	ISUB	A
Breakdown Drain Voltage	BVDS	V

In the abscissa section first the directive can be selected in form of a pull-down menu and then a key of the selected directive is chosen. The information about available MINIMOS directives and corresponding keys is taken from a LISP list at run-time, so modifications (e. g. new keys) are taken into account here automatically. Minimum and maximum value for the abscissa are entered next. For the computation of the abscissa data points, one can toggle the sweep mode between linear and logarithmic.

The parameter section is composed identically to the abscissa part. In this context, the term *parameter* refers to the parameter in an x–y plot, for which a separate curve or line is drawn and the value is shown in the legend of the plot.

In the bottom section of the panel one can select whether to show the device characteristic as x-y plot graphically and one can enter a file name to store the result formatted as x-y value pairs.


Device Characteristic

input MOSN11-MM

ordinate

function Threshold Voltage

abscissa

directive DEVICE key L

min 5e-05 max 0.0003

step 5e-05 sweep lin

parameter

directive BIAS key UB

min 0 max -2

step -2 sweep lin

show characteristic online yes

save as x-y pairs: filename

Figure 6: Device Characteristics Panel

As the computations of data points are independent of each other and can be run concurrently, a separate MINIMOS run is scheduled for each point. This allows maximum utilization of a local workstation cluster or of multiprocessor machines.

8.3 Online Graphics

If the entry "show characteristic online" is set to "yes", the characteristic will pop up after the first two successfully computed data points. After each additional termination of MINIMOS, the resulting data point is added to the graph and the total graph is rescaled and redrawn.

The resulting graph for the default panel settings is shown in Figure 7. This diagram, the threshold voltage versus the channel length for two bulk bias conditions depicts clearly one of the short channel effects, the threshold voltage lowering for smaller devices. This effect is somewhat also called short channel roll-off.

8.4 Conclusion

For a device characteristic MINIMOS is run typically 10 to 50 times. The tasks performed automatically without further user interaction once the `OK`–button is pressed include input deck generation, simulator run scheduling on a workstation cluster, extraction of the desired output values, and plot generation.

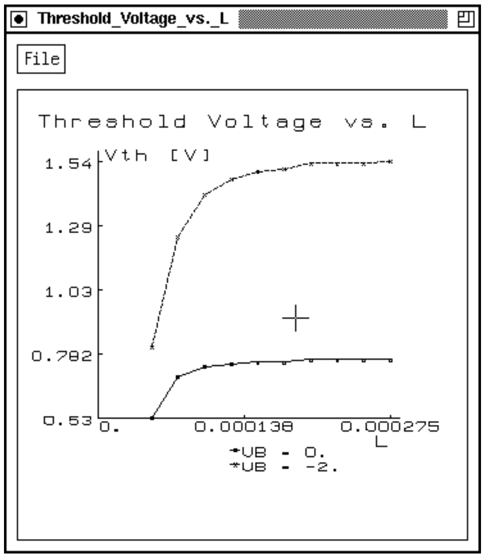


Figure 7: Device Characteristic: Threshold Voltage vs. Channel Length