AN INTEGRATED GRAPHICAL ENVIRONMENT FOR OPERATING IC PROCESS SIMULATORS

by

Alexander S. Wong

Memorandum No. UCB/ERL M89/67

25 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN INTEGRATED GRAPHICAL ENVIRONMENT FOR OPERATING IC PROCESS SIMULATORS

by

Alexander S. Wong

Memorandum No. UCB/ERL M89/67

25 May 1989

COVER PAGE

AN INTEGRATED GRAPHICAL ENVIRONMENT FOR OPERATING IC PROCESS SIMULATORS

by

Alexander S. Wong

Memorandum No. UCB/ERL M89/67

25 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

An Integrated Graphical Environment for Operating IC Process Simulators

Alexander S. Wong [illegible]

Electronics Research Laboratory
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, California 94720

ABSTRACT

This report describes PROSE (PROcess Simulation Environment), an integrated environment for operating integrated-circuit process simulators. PROSE features a graphical user interface for displaying and editing layout and cross-sectional structures, interactive menus and dialog boxes for invoking simulation tools and entering process parameters, and a Profile Interchange Format for specifying device structure information. This environment is based on the VEM/OCT/RPC CAD system, and uses a new process manager named SIMPL-RPC (SIMulated Profiles from the Layout, Remote Procedure Call).

Special attention has been placed on the development of the Profile Interchange Format (PIF). A parser has been implemented in SIMPL-RPC for reading and writing ASCII PIF files. SIMPL-RPC also has the ability to store and manipulate binary PIF data in the OCT database with a new library of data access functions and a policy which defines how PIF data is stored in OCT.

May 23, 1988

Acknowledgements

I would like to thank Professor A. R. Neureuther for his expert guidance in research and engineering, and for his much valued advice on life in general. Professor W. G. Oldham also deserves special mention for reading this paper, for which I am greatly appreciative.

I wish to thank everyone involved in SIMPL, including the previous SIMPL-DIX crew of Simon Koh and Joe Wu, and those currently involved with SIMPL - John Camagna, Ed Scheckler, Ed Scheckler, Martha Tilmann, and Robert Wang. I mention Ed's name twice because of his numerous contributions to this project. I would like to acknowledge others outside my group who I have consulted with on countless occasions, particularly Chris Williams of BPFL fame, and those involved with VEM/OCT/RPC.

Thanks also to all the permanent residents of my cubicle - Rich Ferguson, Ed Scheckler, and Nelson Tam, as well as frequent visitors Gino Addiego, John Gamelin, Sherman Kwok, Jaime Ramirez, and Kenny Toh. I continue to enjoy their advice and friendship. All of the little cubicle amusements which they helped develop makes working with them a true pleasure.

There are just too many people to thank. I hope those who I've forgotten to thank will forgive me. "The essence of true friendship is to make allowance for another's little lapses" - David Storey.

The financial support of the Semiconductor Research Corporation is gratefully acknowledged.

I dedicate this report to Mom and Dad.

Table of Contents

Acknowledgements i
Table of Contents ii

Chapter 1: History and Overview 1
Background and Introduction 1
PROSE Overview 2
A Typical Process Step 3
Outline for the Remaining Chapters 5

Chapter 2: The Process Simulation Environment 10
Introduction 10
The VEM/OCT/RPC Framework 10
The User Interface 10
Process Flow Specification 11
Intertool Communications 12
Storage of Results 13
Display of Results 13
SIMPL Technology Library 15
Hardware and Software Requirements 15

Chapter 3: The Profile Interchange Format 16
Overview 16
The ASCII PIF 17
The Binary PIF 18

Chapter 4: Conclusion 21

References 22
Appendix A: SIMPL Technology Library Listing 23
Appendix B: SIMPL-RPC Generated PIF Example 32

Chapter 1
History And Overview

1.1. Background and Introduction

The concept of SIMPL (SIMulated Profiles from the Layout) is to automatically generate integrated-circuit (IC) device cross-sections from layout and process information through simulation. Mask information is first extracted from a specified cut-line through the layout, then process flow data is used to perform a series of simulation steps to create the final profile. SIMPL-1 [1] was the first program to generate device cross-sections using this technique. In a matter of seconds, it can take layout and process flow information and produce rectangular profiles using one-dimensional simulation models. SIMPL-2 [2] is a second-generation two-dimensional process simulator and manager which produces arbitrary-shaped polygonal cross-sections. Its internal simulator uses fast analytical models capable of describing more complicated phenomena such as bird's beak shapes in oxidation and undercut in etching. SIMPL-2 is also a process manager capable of calling the SAMPLE [3] program for more rigorous simulation of deposition processes. An X-Window based user interface for running SIMPL-2 and other simulators has also been developed. SIMPL-DIX (Design Interface in X) [4] features graphical options for displaying layout, cross-sections, and doping profiles, and a *Designer's Toolbox*, which can assist the user in examining device cross-sections, estimating worst case misalignment, and choosing appropriate locations for cut-line placement. SIMPL-DIX has also been linked to the RACPLE program for extracting resistance and capacitance information from device profiles [5].

The SIMPL programs described above are excellent for *rapidly* generating device-cross sections using both fast analytical models and a limited set of rigorous external models, but with recent advances in computing speeds, designers now are more willing to take an "accuracy at any cost" point of view in producing cross-sections using only rigorous (and computationally expensive) simulation tools. Specialized tools now exist for accurately simulating IC process steps --

some examples are SAMPLE for etching and deposition, CREEP [6] for oxidation and reflow, SUPREM [7] for diffusion and ion implantation. A system which integrates these tools together would be very powerful for effectively predicting device cross-sections.

1.2. PROSE Overview

PROSE applies the SIMPL concept of linking layout and process flow, and *commonality* to implement an integrated environment for generating cross-sections using multiple process simulators. The *commonality* refers to a common graphical user interface for input and output, common description languages for layout, device structure, and process flow information, and a common method of communication.

In PROSE, physical editing of the layout and cross-sections, and process command entry is performed through a single editor. This common user interface gives the system a uniform look and feel, making it easier to learn and use. Common description languages, such as the Profile Interchange Format (PIF) [8] for describing profiles, and eventually the Berkeley Process Flow Language (BPFL) [9] to specify process flow, are used to avoid the need for multiple translators when communicating between different programs. Finally, PROSE exchanges data using a common communications mechanism.

The first-generation PROSE is based on the VEM/OCT/RPC CAD system [10] and the SIMPL-RPC (Remote Procedure Call) process manager. VEM, OCT, and RPC provides general user interface, database, and communication functions for PROSE, respectively. SIMPL-RPC is an enhanced version of the SIMPL-2 program that runs as an RPC application under VEM. It manages the process simulators and their interactions with the VEM/OCT/RPC system, handles all PIF functions for parsing the ASCII PIF and storing profile data in the OCT database, and supplies the VEM editor with menu and prompt information. A schematic view of the current PROSE implementation is displayed in Figure 1.1.

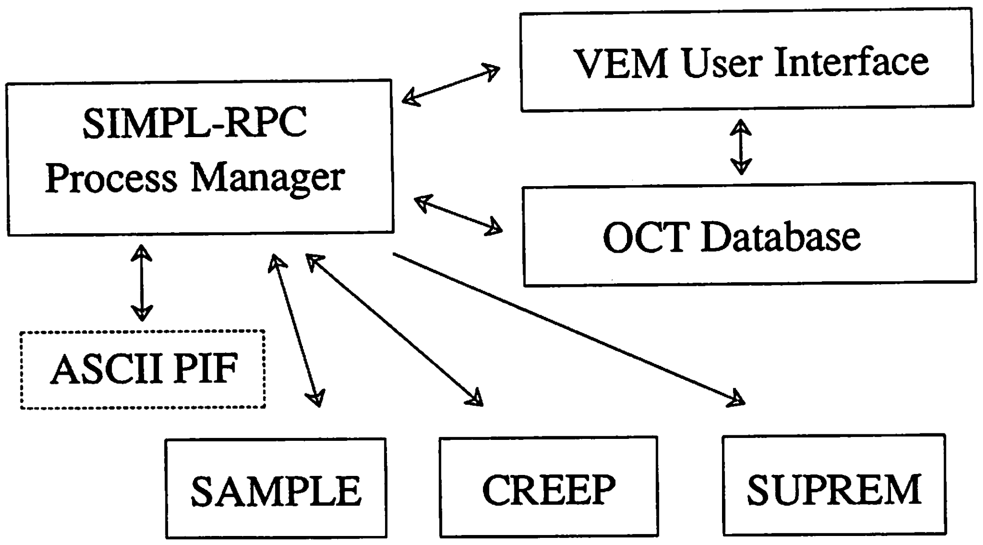


Figure 1.1 Schematic view of the PROSE environment.

1.3. A Typical PROSE Process Step

A typical session using PROSE to perform a process step is summarized in Figure 1.2. The session begins after VEM is invoked from the operating system. From VEM, the user opens a window to display and edit the mask layout. SIMPL-RPC is then invoked from a separate cross-section window, which automatically prompts the user for the carrier type and doping concentration of the starting substrate. A separate set of SIMPL-RPC menus is enabled, allowing the user to interactively select the process steps to be performed. At this point, all initializations are complete.

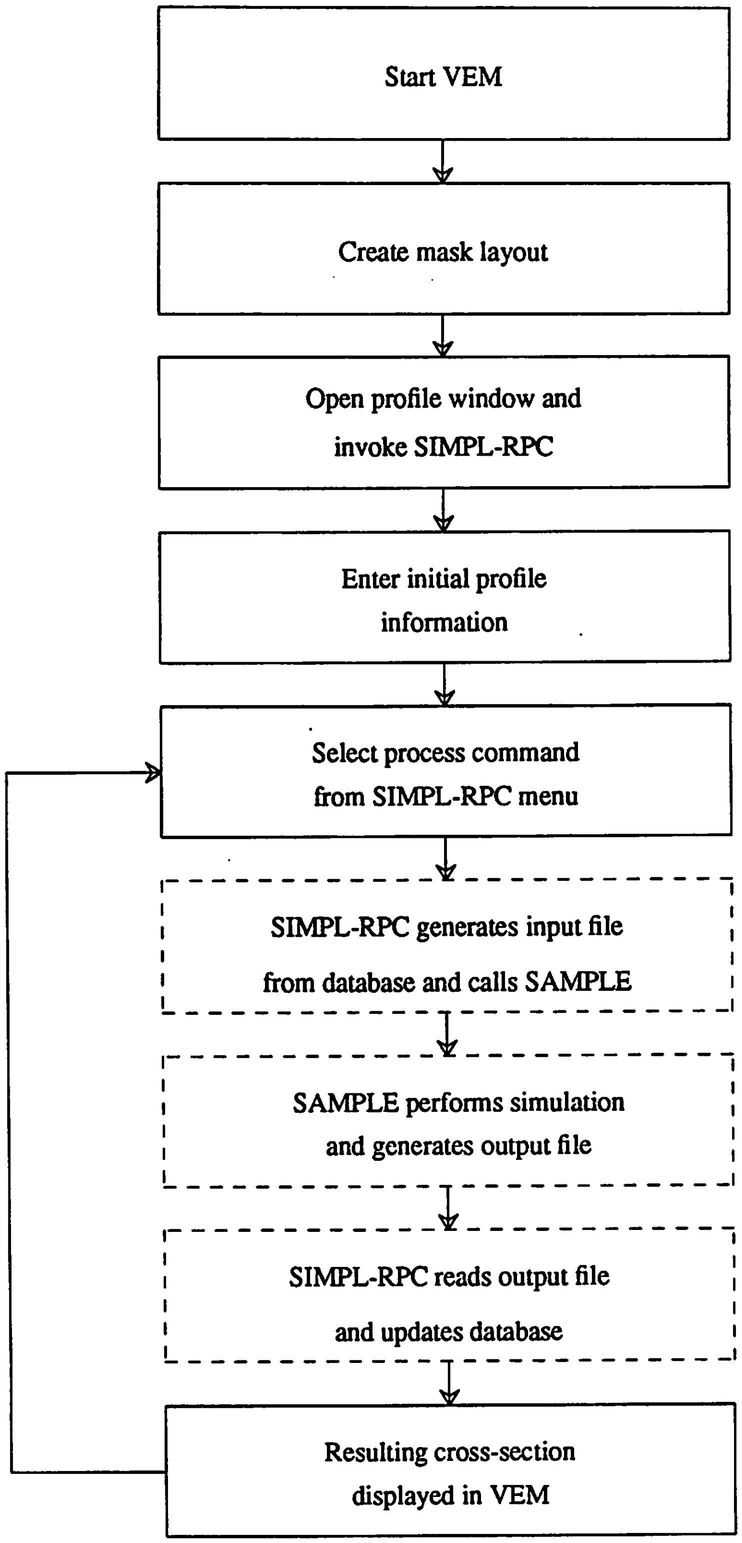
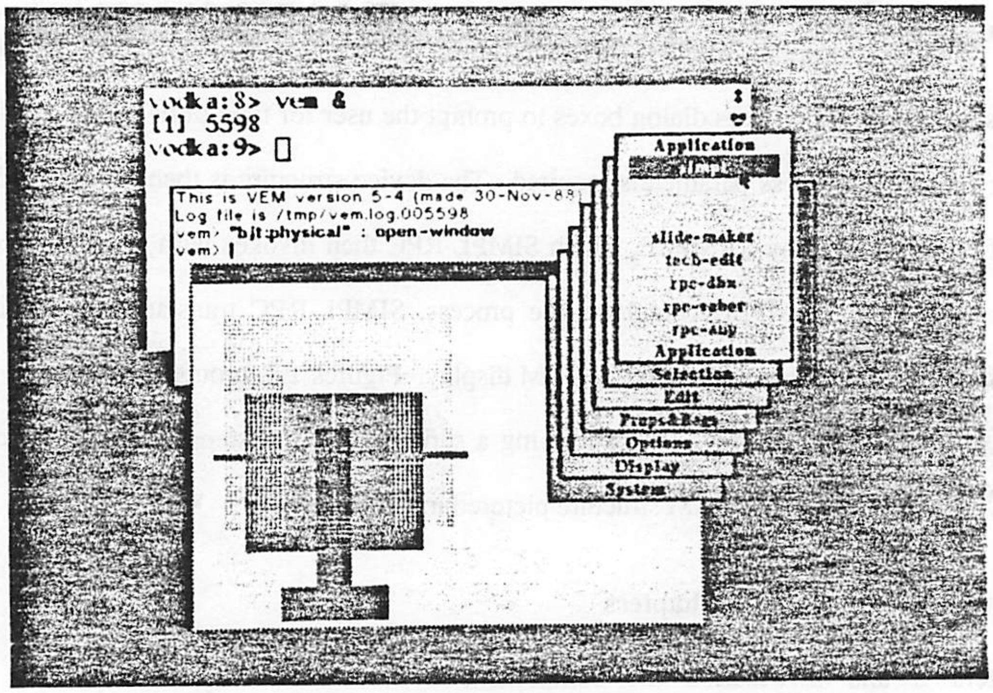

Start VEM
Create mask layout
Open profile window and invoke SIMPL-RPC
Enter initial profile information
Select process command from SIMPL-RPC menu
SIMPL-RPC generates input file from database and calls SAMPLE
SAMPLE performs simulation and generates output file
SIMPL-RPC reads output file and updates database
Resulting cross-section displayed in VEM

Figure 1.2 A typical PROSE process step.

VEM waits for user input, which it parses and sends back to SIMPL-RPC. For a SAMPLE deposition step, SIMPL-RPC uses dialog boxes to prompt the user for type of deposition desired and for any additional process parameters required. The device structure is then translated from PIF into a format readable by SAMPLE, which SIMPL-RPC then invokes with the proper command sequence. After SAMPLE completes the process, SIMPL-RPC translates the resulting profile back into PIF format and updates the VEM display. Figures 1.3 through 1.8 illustrate this process as seen by a PROSE user. By performing a series of process steps, a complete cross-section can generated, like the DRAM structure pictured in Figure 1.9.

1.4. Outline for the Remaining Chapters

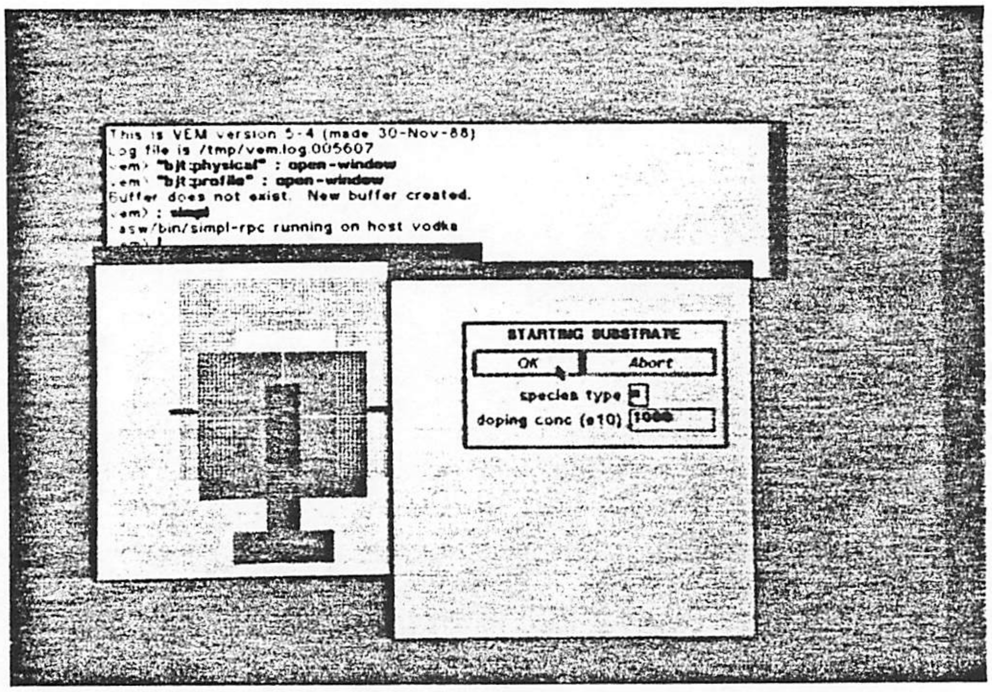
Chapters 2 and 3 describe the components in PROSE. Chapter 2 reviews the VEM/OCT/RPC framework and all PROSE components other than the PIF, which is the focus of Chapter 3. Chapter 4 concludes this report.



```
vodka:8> vem &
[1] 5598
vodka:9> 
```

```
This is VEM version 5-4 (made 30-Nov-88)
Log file is /tmp/vem.log.005598
vem> "bjt:physical" : open-window
vem> |
```

Figure 1.3 Starting up the VEM editor, creating the mask layout, and invoking SIMPL-RPC. The layout and horizontal cut-line across is mask is entered graphically.



```
This is VEM version 5-4 (made 30-Nov-88)
Log file is /tmp/vem.log.005607
vem> "bjt:physical" : open-window
vem> "bjt:profile" : open-window
Buffer does not exist.  New buffer created.
vem> : simpl
asw/bin/simpl-rpc running on host vodka
```

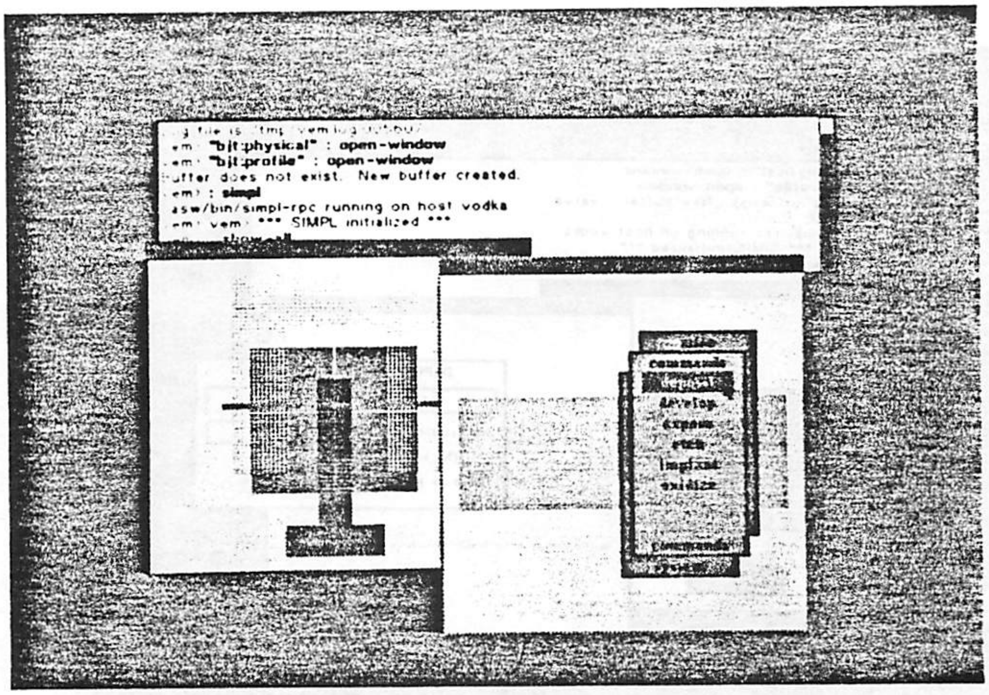
STARTING SUBSTRATE

OK Abort

species type [illegible]

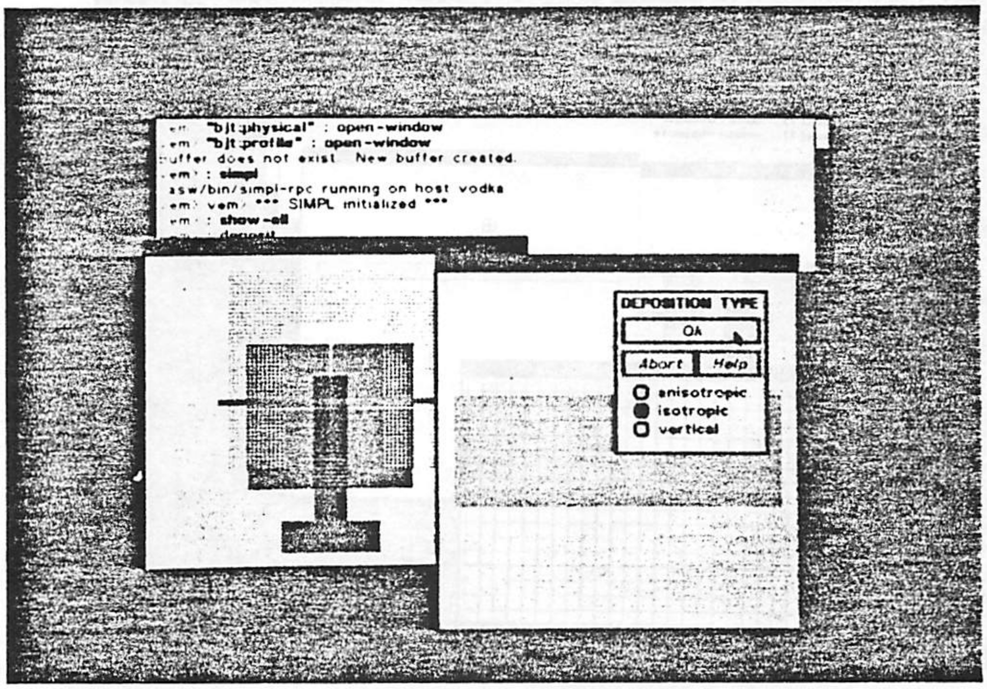
doping conc (e10) 1000

Figure 1.4 Entering parameters into a dialog box for the starting silicon substrate.



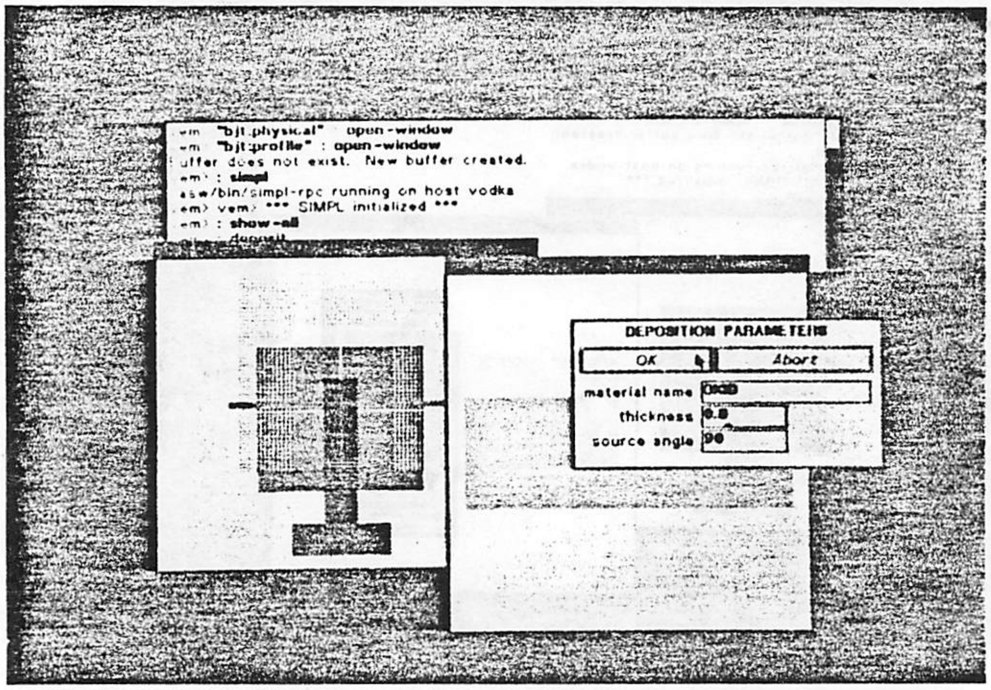
```
g file is  tmp  vem log
em   "bjt:physical" : open-window
em   "bjt:profile" : open-window
uffer does not exist.  New buffer created.
em  : simpl
asw/bin/simpl-rpc running on host vodka
em  vem  *** SIMPL initialized ***
```

Figure 1.5 Using the mouse to select the deposition process step from the SIMPL-RPC deck-of-cards menu.



```
em   "bjt:physical" : open-window
em   "bjt:profile" : open-window
uffer does not exist.  New buffer created.
em  : simpl
asw/bin/simpl-rpc running on host vodka
em  vem  *** SIMPL initialized ***
em  : show-all
```

Figure 1.6 Entering the type of deposition process desired.



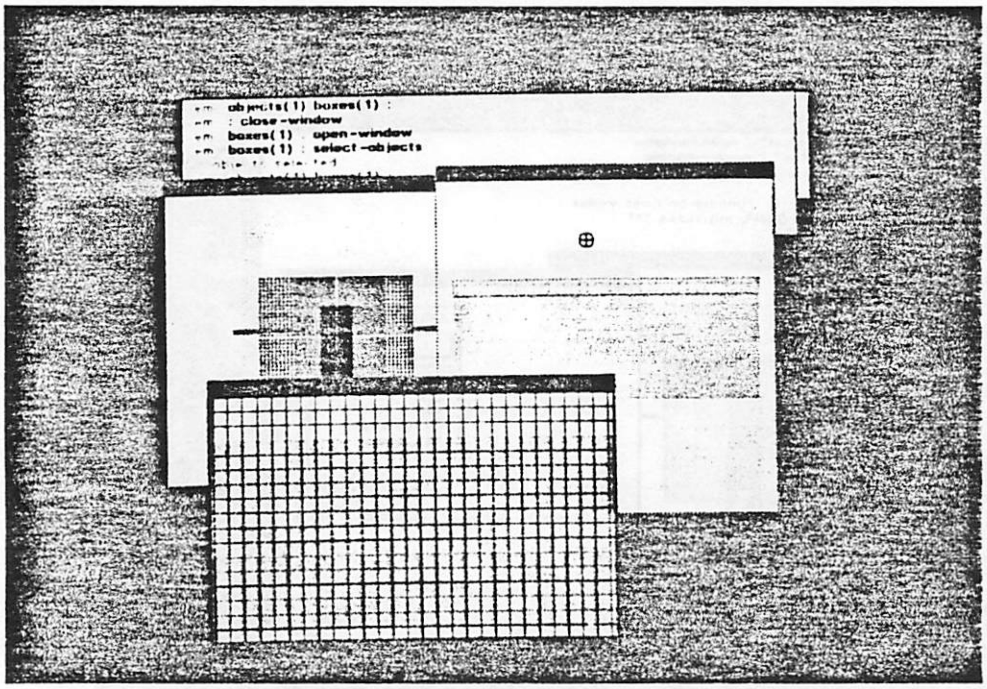
```
vm  "bjt.physical"  open-window
vm  "bjt:profile" : open-window
uffer does not exist.  New buffer created.
em> : simpl
sw/bin/simpl-rpc running on host vodka
em> vem> *** SIMPL initialized ***
em> : show-all
```

DEPOSITION PARAMETERS

OK | Abort

material name	[illegible]
thickness	0.8
source angle	90

Figure 1.7 Entering the necessary deposition parameters.



```
vm  objects(1) boxes(1) :
vm  : close-window
vm  boxes(1) : open-window
vm  boxes(1) : select-objects
```

Figure 1.8 The result, which has been highlighted, is immediately reflected in the VEM display. A second window shows a zoomed in view of the cross-section displayed on a user-definable grid.

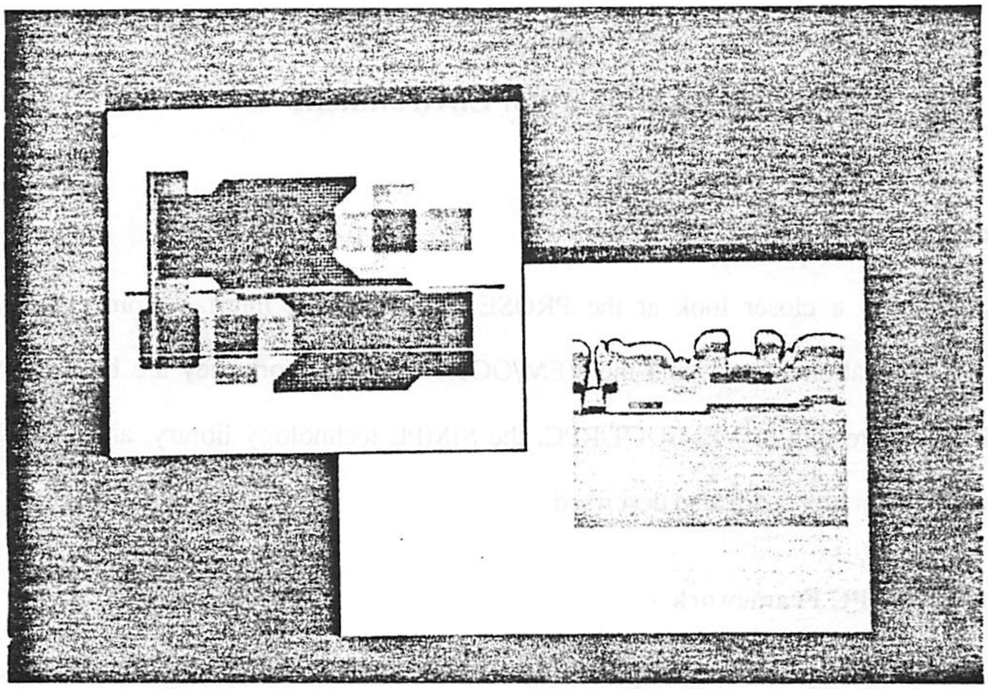


Figure 1.9 DRAM layout and cross-section structure generated using PROSE.

Chapter 2

The Process Simulation Environment

2.1. Introduction

This chapter takes a closer look at the PROSE graphical user interface, process flow specification, communication format, and the VEM/OCT/RPC framework they are built upon. Storage and display of results in VEM/OCT/RPC, the SIMPL technology library, and PROSE hardware/software requirements are also described.

2.2. The VEM/OCT/RPC Framework

The VEM/OCT/RPC system was adopted as the PROSE framework for several reasons. An important one is the freedom it gives the user to view <u>and</u> edit layouts and cross-sections simultaneously, a feature not available in SIMPL-DIX. VEM/OCT/RPC also allows programs within PROSE to access a whole family of logic synthesis and circuit simulation tools and data.

In addition to providing a general framework, the VEM/OCT/RPC system also has several specific capabilities which are well suited for PROSE. VEM offers a user-friendly interface for mouse and keyboard input, and the typical graphical display and editing capabilities of a mask layout editor. Its ability to work with non-manhattan geometries makes it particularly useful for viewing and editing device cross-sections or other polygonal structures. The generality of OCT, in combination with its access function library, has provided a working database on which PIF is being developed. The RPC communications package used by VEM and OCT integrates SIMPL-RPC into this CAD system and allows PROSE to operate in a distributed computing environment.

2.3. The User Interface

The user interface provides the main link between the user and the simulation tools. It enables users to input data, invoke simulators, and edit simulator output effectively from a single environment. The PROSE user interface is based on the VEM layout editor and RPC user interface package. VEM contains graphical capabilities for entering, viewing, and editing mask layout and device cross-sections. The RPC package allows SIMPL-RPC to run as an application program inside VEM, and provides dialog boxes and other mechanisms to obtain data from the user.

Inputs are interactively entered through the PROSE user interface using the keyboard and/or mouse. Graphical entry is well suited for entering physical mask layout, cut-line, and cross-section information. Deck-of-cards menus are used for invoking process steps or other options are naturally presented in menu format. Commands and parameters may also be typed directly from the keyboard for the benefit of experienced users (or fast typists). Some inputs can be entered using a combination of the mouse and keyboard input, such as a query for the type of deposition and associated deposition parameters in a process step. These are treated interactively through RPC *dialog boxes*. Typical examples of command and process information entry in PROSE can be found in Figures 1.3 through 1.8.

2.4. Process Flow Specification

PROSE uses the same process flow input format as SIMPL-2, which is adequate for the process steps currently supported by SIMPL-RPC. In the interest of commonality, a more general process flow specification, such as BPFL, is planned for future inclusion in PROSE. Although BPFL is designed for use in the Computer Integrated Manufacturing (CIM) Environment, there is promising evidence that it can also be adapted to work in PROSE by treating simulators like machines, with dial and knob settings replaced by simulation parameters. Previous work which translated BPFL into SIMPL-2 process flow format has demonstrated the feasibility of this adaptation, as well as the limitations of the current SIMPL-2 format [11].

2.5. Intertool Communications

The PROSE uses both Remote Procedure Call and standard ASCII file transfer procedures for communicating between components. SIMPL-RPC uses the classical method to communicate with rigorous simulation programs such as SAMPLE and CREEP. As represented in Figure 2.1, this method employs UNIX system calls to invoke the simulator, and exchanges ASCII data through pre-determined files.

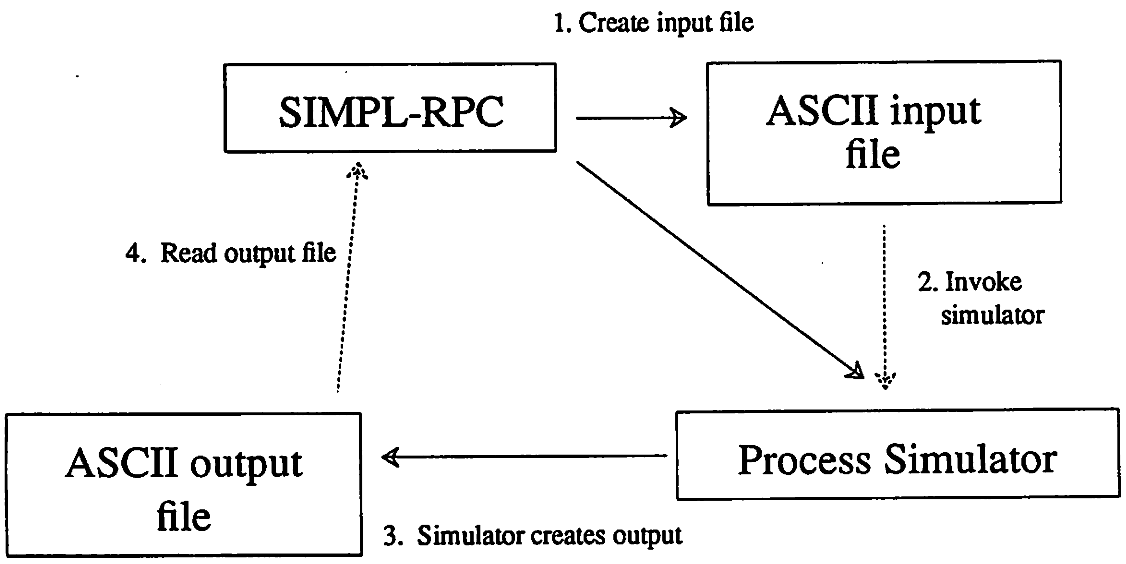


Figure 2.1 Communications using the classical ASCII file method.

As its name suggests, SIMPL-RPC relies heavily on the RPC package. For example, RPC communications are used when SIMPL-RPC is invoked from inside VEM, when process and command data (ie. menu, dialog box, or input parameter data) is exchanged between SIMPL-RPC and VEM, and when any calls made between SIMPL-RPC and the OCT database for PIF information. The RPC package supports an asynchronous binary communications format designed to run on multiple machines in a distributed computing environment. Binary function calls made by programs on different machines are handled by RPC "interpreters" residing on each machine. The duty of each interpreter is to translate any incoming binary data into a format that its machine can understand, and to perform a similar translation for any outgoing data.

The classical ASCII method is slowly being phased out in favor of the more flexible RPC. Since different tools have different computing requirements, one clear advantage for using a distributed computing environment is the ability to match tools with machines. For instance, because of RPC, the VEM editor can run on the user's workstation, the OCT database on a central file server, and rigorous process simulators on a mainframe, as shown in Figure 2.2.

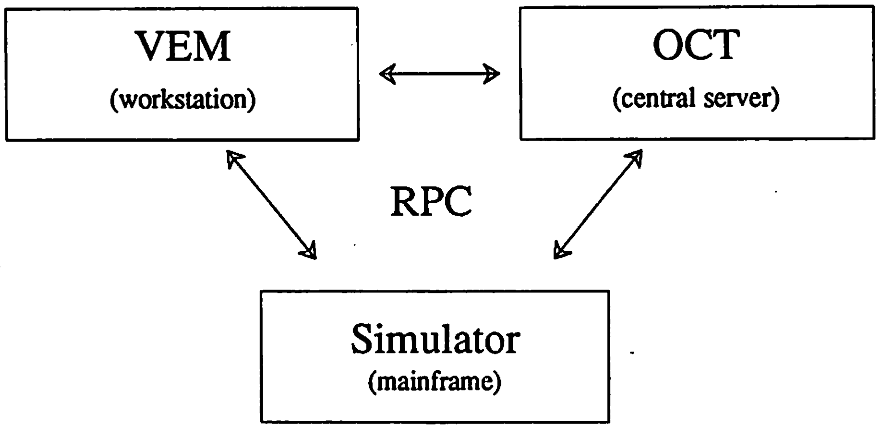


Figure 2.2 RPC allows programs to reside on different machines.

2.6. Storage of Results

The PROSE uses the Profile Interchange Format to store simulation results. (PIF specifics can be found in Chapter 3). All PIF data is stored in binary format in the OCT database and is updated each time a process step is performed. For example, a process step which uses a SIMPL analytical model obtains the current profile from OCT, performs the simulation, writes the resulting profile into OCT using binary function calls. Programs which do not have a direct interface to OCT employ the ASCII communications method described above to obtain the necessary data. In this case, SIMPL-RPC acts as the communications operator and even a translator, if necessary.

2.7. Display of Results

Cross-section geometries can be easily displayed by VEM, but there are other forms of data which can not be directly exhibited. This is because PIF data, although it is stored in OCT, does not always conform exactly to OCT physical layout policy. In these cases, SIMPL-RPC provides interpretation facilities for translating data between PIF policy and OCT physical policy. For instance, substrate doping grid data stored in arrays need to be translated into VEM displayable geometries. This problem is solved by a contour generator in SIMPL-RPC, which takes array data, calculates a set of contours of equal potential, and writes them into the PIF facet as geometric paths solely for VEM display purposes, as shown in Figure 2.3. SIMPL-RPC also contains a function to convert cut-line information, represented geometrically in OCT physical policy by a line across the mask, into PIF policy, which saves both the geometric line and the coordinates where the mask intersects the cut-line. Future graphical display functions, such as line and contour plotters, will require a more extensive translator between PIF and OCT physical policies.

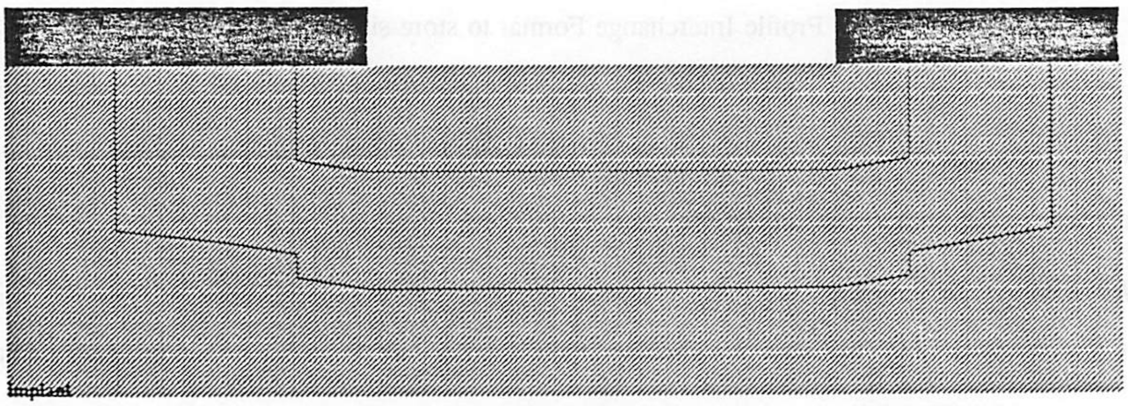


Figure 2.3 A PROSE cross-section with doping contours.

2.8. SIMPL Technology Library

A small but important piece of PROSE is a library for storing technology information required by VEM. Currently in the **simpl** technology library, the layer names and "looks" (colors and stipple patterns) of all standard SIMPL mask and material names are stored for VEM display purposes using the physical editing style. A listing of this technology can be found in Appendix A.

2.9. Hardware and Software Requirements

PROSE is designed to run in a workstation environment. The current implementation uses a DEC Micro-VAX GPX-II workstation running Ultrix V2.2. PROSE consumes approximately 1Mb of memory, of which 800Kb is used by VEM/OCT/RPC 2.0 and 200Kb by SIMPL-RPC. SIMPL-RPC consists of about 10,000 lines of source code, and is written in the C language. A BSD compatible UNIX workstation running X-Windows is required; a color display is highly recommended.

Chapter 3
The Profile Interchange Format

3.1. Overview

Historically, each process simulator uses a different input and output format, leaving the task of data exchange between programs up to specialized translators. In an environment which uses many different tools, a reasonable approach to limit the number of translators that are needed is to use a common interchange format. At worst, each new program would then need only one pair of interpreters -- one to write to the common format, and another to read it in. PROSE uses this common language approach for specifying profile information.

Developing a common format for profile specification is particularly difficult due to the varied and sometimes conflicting needs of different simulation tools. One common specification that has been proposed is PIF, which is formalized in a paper by Duvall [8]. This PIF specification is general enough to specify arbitrary profiles, and is designed to eventually be incorporated into EDIF 2.0.0 [12]. With recursion, instantiation, and hierarchical description features, it provides a very elegant way to specify profiles.

Unfortunately, these same features make it too complex to realize a full implementation of the specification exactly as described. To help identify the general strengths and weaknesses of PIF, a skeleton parser capable of specifying critical geometry, grid, and attributes, is used in the current PROSE. This implementation confirms the usefulness of PIF in describing cross-section structures. At the same time, it also identifies several areas of concern in PIF, such as the lack of a formal specification for representing program specific attributes, and its overall lengthiness.

Specifically, even though cut-line and mask information is technically not part of the profile, it is used by many process simulators such as SIMPL-2 and SAMPLE, and should be definable as a program specific attribute. Also, the length of PIF is of concern because it directly influences the amount of time needed to read and write the file (as well as the amount of paper used in printing it out). The PIF specification for a simple MOS transistor is about 230 lines in length, while a more complicated CMOS structure consumes over 1300 lines. A simple way to shorten PIF would be to compress common information, such long lists of carrier concentrations of a constant value, without endangering either its generality or human readability.

Both the *intersite* and *intertool* modes of PIF are used in PROSE. The *intersite* or ASCII PIF (APIF) uses ASCII representation for sending data between sites, while the *intertool* or binary PIF (BPIF) transfers data between tools in binary format. In the latter case, a new, database independent BPIF function library is being defined to to preserve the commonality feature in the BPIF even when different databases are involved.

3.2. The ASCII PIF

PIF is an EDIF compatible ASCII specification for representing profile information. A PIF *snapshot* consists of a collection of *geometries*, *grids*, and *attributes*. Briefly, *geometries* are described in a hierarchical manner, beginning with points. Lines are built by linking points together, faces built by linking lines, solids built by linking faces, and so on, to specify 1D, 2D, 3D, (and even 4D!) structures. PIF supports rectangular, triangular, tetrahedral, and other *grid* descriptions. *Attributes* specify the characteristics of an object, such as doping concentrations and layer types and can be defined on both geometries and grids. A more detailed description of PIF can be found in [RR]; a SIMPL-RPC generated example of PIF for a simple structure can be found in Appendix B.

The PROSE PIF parser, which is built into SIMPL-RPC, translates the minimum amount of information necessary to read and write SIMPL cross sections. The writer writes out cross-sections in acceptable PIF and is guaranteed readable by the reader. The reader uses a keyword search and parenthesis counting approach for parsing the input, and performs only primitive syntax checking. At each parenthesis level, the reader expects to find keywords valid only for that level. If one is found, the parser continues, otherwise, it will issue an error message and stop. A list of keywords recognized by the PIF parser is listed in Table 3.1.

Table 3.1 Keywords recognized by the ASCII PIF parser

Recognized APIF keywords	
Standard	PIF, attribute, attributeName, coordinates, data, lineList, faceList, geometry, grid, pointList, segmentList, segmentNameList, substrateDoping, snapshot
New	cutlineInfo, maskInfo, windowInfo

Two modifications have been made to the original PIF specification. Three additional attributes, **maskInfo, cutlineInfo,** and **windowInfo** have been added to store cut-line and mask information. Substrate doping information has also been compressed into the format (# of points, concentration value) for grid points which remain constant in doping over several grid lines. Since much of the silicon is at the same background doping concentration, this compression has reduced the PIF length by about 25% for variable rectangular grids, which is the only type supported at this time.

3.3. The Binary PIF

The binary PIF is used for transferring binary profile data between programs sharing a common database. Like the ASCII PIF representation, either specialized or generalized approaches may be used. PROSE uses a general PIF method described below.

Since it is recognized that different sites will probably want to use different databases, a database independent PIF function library is proposed to be built on top of each database. This is analgous to the emerging X-Window System standard for graphics displays. The X-Window library provides a standard set of function calls for drawing graphics independently of the display hardware. Similarly, the PIF library provides a standard set of function calls to access PIF data independent of the database.

Currently, a small set of PIF access functions to write between the internal SIMPL-2 format and OCT database have been implemented. These OCT-based PIF functions contain the necessary functions to read, write and replace a SIMPL generated profile, and are listed in Table 3.2.

Table 3.2 List of Binary PIF OCT functions available in SIMPL-RPC

BPIF Functions in SIMPL-RPC	
Name	Function
DeleteContour	Delete doping contour geometries
Read	Read in whole profile
ReadGrid	Read in rectangular grid
ReadProp	Read in all properties
ReplaceGrid	Replace grid
ReplaceLayers	Replace layers in current profile
Write	Write out entire profile
WriteContour	Write out doping contour geometries
WriteGrid	Write out grid
WriteLayers	Write out layers in current profile
WriteMask	Write out layout mask information
WritePoly	Write a polygon

A simple tri-level binary PIF *policy*, shown in Figure 3.1, defines how PIF data is stored OCT. At the root of the PIF policy tree is the PIF snapshot. Level two contains attributes with grid, cut-line, mask, and layer name information. The leaves of this PIF tree are the geometries, stored in **OCT_POLYGON** format. At the moment, this PIF policy is mainly for geometry specification, and is very similar to OCT physical policy [RR]. This is because cross-sections are displayed in the VEM physical editing style, and must therefore be compatible with OCT physical format.

Development of a full PIF policy parallels the current work in defining and implementing a formal PIF library of access functions.

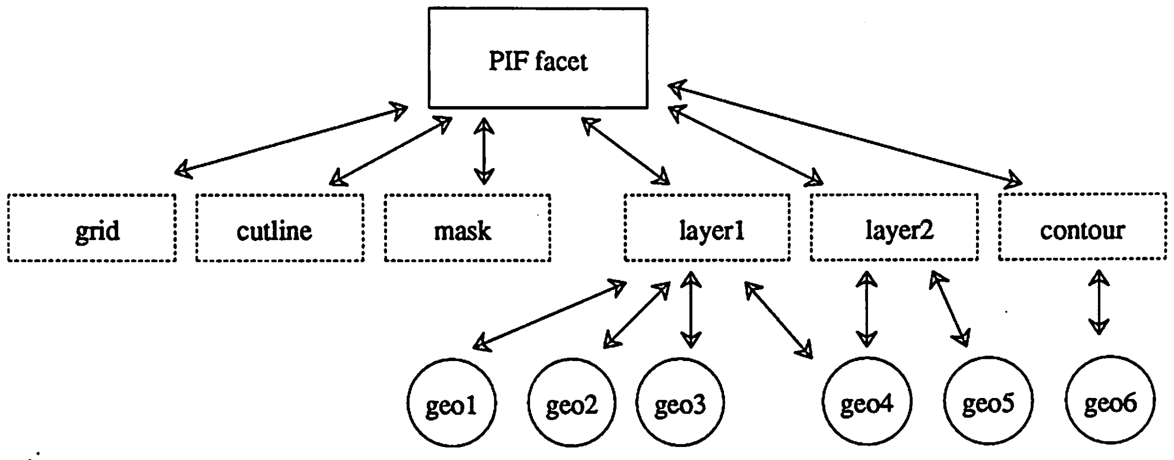


Figure 3.1 Schematic description of PIF data stored in the OCT database.

Chapter 4
Conclusion

The "accuracy at any cost" point of view in producing cross-sections with the rigorous simulation tools has defined a need for an integrated environment for operating process simulators. PROSE implements one such environment using the concept of *commonality* -- commonality in the user interface, specification languages, and communications method. PROSE is based upon the OCT/VEM/RPC CAD system and a new version of SIMPL named SIMPL-RPC. Several new user-friendly capabilities have been introduced, including the ability to graphically edit both layout and cross-sectional views, the addition of menu and dialog box systems in the user interface, and the support of new graphics additions in the VEM editor for cut-line and doping contour plotting.

Advances have also been made in using a common format for profile specification. A simple parser for PIF has been implemented to demonstrate the feasibility of a common specification for device structures, and to explore ways to improve the original specification. Binary PIF information can also be stored in the OCT database and manipulated using a set of PIF access functions.

The implementation of a common process flow language, a complete PIF, and links to additional process and device simulations are high on the priority list for future PROSE work.

References

[1] M.A. Grimm, K. Lee and A.R. Neureuther, "SIMPL-1 (SIMulated Profiles from the Layout - Version 1)," IEDM Technical Digest, pp. 255-258, December 1983.

[2] K. Lee and A.R. Neureuther, "SIMPL-2 (SIMulated Profiles from the Layout - Version 2)," 1985 Symposium on VLSI Technology, Kobe, Japan, Digest of Technical Papers, pp. 64-65, May 1985.

[3] W. G. Oldham, A. R. Neureuther, C. Sung, J. L. Reynolds, and S. N. Nandgaonkar, "A General Simulator for VLSI Lithography and Etching Process: Part I - Application to Projection Lithography," *IEEE Trans. Electron. Devices*, Vol. ED-26, No. 4, pp. 717-722, April 1979.

[4] H.C. Wu, A.S. Wong, Y.L. Koh, E.W. Scheckler, and A.R. Neureuther, "SIMulated Profiles from the Layout - Design Interface in X (SIMPL-DIX)," IEDM Technical Digest, pp. 328-331, December 1988.

[5] E.W. Scheckler, *Extraction of Topography Dependent Electrical Characteristics from Process Simulation Using SIMPL, with Application to Planarization and Dense Interconnect Technologies*, Electronics Research Laboratory, University of California, Berkeley, March 1989.

[6] P. Sutardja, Y. Shacham-Diamand and W.G. Oldham, "Two-Dimensional Simulation of Glass Reflow and Silicon Oxidation," 1986 Symposium on VLSI Technology Technical Digest, pp.39-40, May 1986.

[7] C.P. Ho, J.D. Plummer, S.E. Hansen, and R.W. Dutton, "VLSI Process Modeling: SUPREM III," *IEEE Trans. Electron Devices*, vol. ED-30, no. 11, pp. 1438-1453, November 1983.

[8] S. Duvall, "An Interchange Format for Process and Device Simulation," *IEEE Trans. Computer-Aided Design*, vol. CAD-7, no. 7, pp. 741-754, July 1988.

[9] C.B. Williams, *The Berkeley Process-Flow Language: Reference Document*, Electronics Research Laboratory, University of California, Berkeley, October 1987

[10] A.R. Newton, A. Sangiovanni-Vincentelli et. al, *OCT Tools Distribution 2.0*. Electronics Research Laboratory, University of California, Berkeley, November 1987.

[11] C.B. Williams, Personal Communication, 1988.

[12] *EDIF - Electronic Design Interchange Format Version 2 0 0*, EDIF Steering Commitee, Electronic Industries Association, 1987.

Appendix A

SIMPL Technology Library

```
SIMPL TECHNOLOGY LIBRARY:  May 1989 ASW
---------------------------------------

Technology root directory:  .
Specific technology directory:  facet contains own technology
Layer "PWEL" looks:
  For a "GENERIC-COLOR" device:
      Priority: -15  nColors: 2    Fill: solid      Border: empty
      Colors: (60901,53619,45013) (58915,46999,40380)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0     nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1000000001000000001000000001000000001000000001000000001000000001"
            strokes:  (135.0 - 8)
      Border pattern: 1x1  "0"

Layer "NWEL" looks:
  For a "GENERIC-COLOR" device:
      Priority: -15  nColors: 2    Fill: solid      Border: empty
      Colors: (65535,35746,0) (58915,28464,10591)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0     nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "0000000100000010000001000000100000010000001000000100000010000000"
            strokes:  (45.0 - 8)
      Border pattern: 1x1  "0"

Layer "COM2" looks:
  For a "GENERIC-COLOR" device:
      Priority: 50   nColors: 1    Fill: solid      Border: empty
      Colors: (0,0,0)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0     nColors: 0    Fill: solid      Border: empty
      Colors:
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

Layer "NDIF" looks:
  For a "GENERIC-COLOR" device:
      Priority: -10  nColors: 2    Fill: solid      Border: empty
      Colors: (0,65535,0) (17873,52295,22506)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0     nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1100000001100000000110000000011000000001100000110000011000001100000"
>>>fill pattern can't be represented with strokesLayer "PDIF" looks:
  For a "GENERIC-COLOR" device:
      Priority: -5   nColors: 2   Fill: solid    Border: empty
      Colors: (45675,23168,15887) (33098,27140,21845)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0    nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1000000011000000101100011001101100001110000000100000000000000000"
>>>fill pattern can't be represented with strokesLayer "POLY" looks:
  For a "GENERIC-COLOR" device:
      Priority: 10   nColors: 2    Fill: stippled  Border: empty
      Colors: (65535,6619,6619) (58253,9929,13239)
      Fill pattern: 4x4  "0111111111110111"
            strokes:  (0.0 - 2) (45.0 - 2) (90.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0    nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1111111110000001100000011000000110000001100000011000000111111111"
            strokes:  (0.0 - 4) (26.6 - 8) (63.4 - 8) (90.0 - 4) (116.6 - 8) (153.4 - 8)
      Border pattern: 1x1  "0"

Layer "COPS" looks:
  For a "GENERIC-COLOR" device:
      Priority: 50   nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,0)
      Fill pattern: 8x8  "1111111110111101101111011111111111111111101111011011110111111111"
            strokes:  (0.0 - 2) (45.0 - 4) (90.0 - 2) (135.0 - 4)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0    nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1111111110111101101111011111111111111111101111011011110111111111"
            strokes:  (0.0 - 2) (45.0 - 4) (90.0 - 2) (135.0 - 4)
      Border pattern: 1x1  "0"

Layer "CONS" looks:
  For a "GENERIC-COLOR" device:
      Priority: 50   nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,0)
      Fill pattern: 8x8  "1110011111000011100110010011110000111100100110011100001111100111"
>>>fill pattern can't be represented with strokes   For a "PostScript-BW" device:
      Priority: 0    nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1110011111000011100110010011110000111100100110011100001111100111"
>>>fill pattern can't be represented with strokesLayer "COPO" looks:
  For a "GENERIC-COLOR" device:
      Priority: 50   nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,0)
      Fill pattern: 8x8  "1111111110011001111111111001100110011001111111111001100111111111"
            strokes:  (0.0 - 2) (36.9 - 8) (90.0 - 2) (143.1 - 8)
      Border pattern: 1x1  "0"

  For a "PostScript-BW" device:
      Priority: 0    nColors: 0    Fill: stippled  Border: empty
      Colors:
      Fill pattern: 8x8  "1111111110011001111111111001100110011001111111111001100111111111"
            strokes:  (0.0 - 2) (36.9 - 8) (90.0 - 2) (143.1 - 8)
```

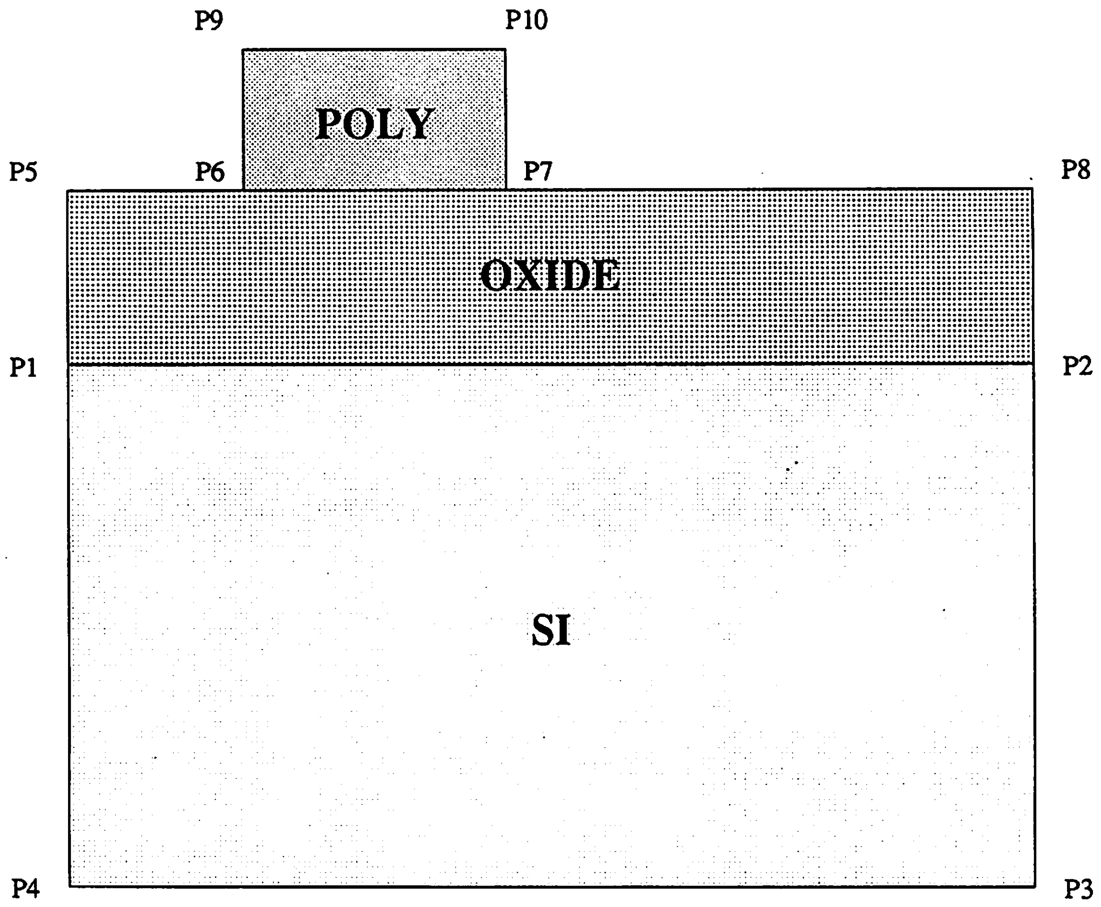
```
        Border pattern: 1x1  "0"

Layer "COND" looks:
   For a "GENERIC-COLOR" device:
        Priority: 50   nColors: 1    Fill: stippled  Border: empty
        Colors: (0,0,0)
        Fill pattern: 8x8  "1111111110111101110110111110011111100111110110111011110111111111"
                strokes:  (0.0 - 4) (45.0 - 3) (90.0 - 4) (135.0 - 3)
        Border pattern: 1x1  "0"

   For a "PostScript-BW" device:
        Priority: 0     nColors: 0    Fill: stippled  Border: empty
        Colors:
        Fill pattern: 8x8  "1111111110111101110110111110011111100111110110111011110111111111"
                strokes:  (0.0 - 4) (45.0 - 3) (90.0 - 4) (135.0 - 3)
        Border pattern: 1x1  "0"

Layer "COPD" looks:
   For a "GENERIC-COLOR" device:
        Priority: 50   nColors: 1    Fill: stippled  Border: empty
        Colors: (0,0,0)
        Fill pattern: 8x8  "1001100110111101111001111111111111111111111111001111011110110011001"
                strokes:  (0.0 - 4) (45.0 - 8) (90.0 - 4) (135.0 - 8)
        Border pattern: 1x1  "0"

   For a "PostScript-BW" device:
        Priority: 0     nColors: 0    Fill: stippled  Border: empty
        Colors:
        Fill pattern: 8x8  "1001100110111101111001111111111111111111111111001111011110110011001"
                strokes:  (0.0 - 4) (45.0 - 8) (90.0 - 4) (135.0 - 8)
        Border pattern: 1x1  "0"

Layer "MET1" looks:
   For a "GENERIC-COLOR" device:
        Priority: 20   nColors: 2    Fill: stippled  Border: empty
        Colors: (0,0,65535) (20521,20521,54281)
        Fill pattern: 4x4  "1010101001010101"
                strokes:  (63.4 - 2) (116.6 - 2)
        Border pattern: 1x1  "0"

   For a "PostScript-BW" device:
        Priority: 0     nColors: 0    Fill: stippled  Border: empty
        Colors:
        Fill pattern: 8x8  "110000011110000000111000000011100000001110000000111000000011110000011"
                strokes:  (126.9 - 8) (135.0 - 3) (143.1 - 8)
        Border pattern: 1x1  "0"

Layer "MET2" looks:
   For a "GENERIC-COLOR" device:
        Priority: 30   nColors: 2    Fill: stippled  Border: empty
        Colors: (65535,15225,65535) (56267,0,56929)
        Fill pattern: 4x4  "1100000110000011"
                strokes:  (26.6 - 4)
        Border pattern: 1x1  "0"

   For a "PostScript-BW" device:
        Priority: 0     nColors: 0    Fill: stippled  Border: empty
        Colors:
        Fill pattern: 8x8  "1000001100000111000011100001110000111000011100001110000011000001"
                strokes:  (36.9 - 8) (45.0 - 3) (53.1 - 8)
        Border pattern: 1x1  "0"

Layer "PLACE" looks:
  For a "GENERIC-COLOR" device:
       Priority: 70   nColors: 1    Fill: empty    Border: stippled
       Colors: (65535,65535,12577)
       Fill pattern: 1x1  "0"
              strokes:
       Border pattern: 2x1  "10"

  For a "PostScript-BW" device:
       Priority: 0     nColors: 0    Fill: empty    Border: solid
       Colors:
       Fill pattern: 1x1  "0"
              strokes:
       Border pattern: 1x1  "1"

Layer "GLAS" looks:
  For a "GENERIC-COLOR" device:
       Priority: 60   nColors: 1    Fill: stippled  Border: solid
       Colors: (11253,28464,11915)
       Fill pattern: 8x8  "0000000000000000000000000001100000011000000000000000000000000000"
>>>fill pattern can't be represented with strokes   For a "PostScript-BW" device:
       Priority: 0     nColors: 0    Fill: empty    Border: solid
       Colors:
       Fill pattern: 1x1  "0"
              strokes:
       Border pattern: 1x1  "1"

Layer "SI" looks:
  For a "XDEV_QDSS" device:
       Priority: 99   nColors: 1    Fill: stippled  Border: empty
       Colors: (19859,24492,31112)
       Fill pattern: 4x4  "1010000010100000"
>>>fill pattern can't be represented with strokesLayer "METL" looks:
  For a "XDEV_QDSS" device:
       Priority: 0     nColors: 1    Fill: stippled  Border: empty
       Colors: (17873,13901,53619)
       Fill pattern: 4x4  "1010010010100000"
>>>fill pattern can't be represented with strokesLayer "ACTV" looks:
  For a "XDEV_QDSS" device:
       Priority: 0     nColors: 1    Fill: stippled  Border: empty
       Colors: (0,48323,0)
       Fill pattern: 4x4  "1000001000101000"
>>>fill pattern can't be represented with strokesLayer "OXID" looks:
  For a "XDEV_QDSS" device:
       Priority: 0     nColors: 1    Fill: stippled  Border: empty
       Colors: (51633,51633,52957)
       Fill pattern: 4x4  "1001011001101001"
              strokes:  (45.0 - 4) (135.0 - 4)
       Border pattern: 1x1  "0"

Layer "PSG" looks:
  For a "XDEV_QDSS" device:
       Priority: 0     nColors: 1    Fill: stippled  Border: empty
       Colors: (56267,56267,45013)
       Fill pattern: 4x4  "1010010101001011"
              strokes:  (53.1 - 4) (63.4 - 4) (116.6 - 4) (126.9 - 4)
       Border pattern: 1x1  "0"

Layer "NTRD" looks:
  For a "XDEV_QDSS" device:
       Priority: 0     nColors: 1    Fill: stippled  Border: empty
       Colors: (54281,39056,0)
       Fill pattern: 4x4  "0000011001100001"
>>>fill pattern can't be represented with strokesLayer "RST" looks:
  For a "XDEV_QDSS" device:
      Priority: 1    nColors: 1    Fill: stippled  Border: empty
      Colors: (54281,17211,17873)
      Fill pattern: 4x4  "0100000110000010"
>>>fill pattern can't be represented with strokesLayer "ERST" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (64211,31774,51633)
      Fill pattern: 4x4  "0000101000001010"
>>>fill pattern can't be represented with strokesLayer "IMPL" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (0,56929,0)
      Fill pattern: 4x4  "0000111100001111"
            strokes:  (0.0 - 2)
      Border pattern: 1x1  "0"

Layer "PLY1" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (55605,24492,20521)
      Fill pattern: 4x4  "0000111100001111"
            strokes:  (0.0 - 2)
      Border pattern: 1x1  "0"

Layer "PLY2" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (56267,20521,20521)
      Fill pattern: 4x4  "1010101010101010"
            strokes:  (90.0 - 2)
      Border pattern: 1x1  "0"

Layer "PSG1" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (61563,50309,0)
      Fill pattern: 4x4  "0010100101001010"
>>>fill pattern can't be represented with strokesLayer "PSG2" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (60901,48323,0)
      Fill pattern: 4x4  "1010010110100101"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

Layer "CUT" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (46337,51633,46999)
      Fill pattern: 4x4  "1010010110100101"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

Layer "MTL" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,58915)
      Fill pattern: 4x4  "0101101001011010"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

Layer "PSD" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (7943,48985,48323)
      Fill pattern: 4x4  "1010111111111010"
            strokes:  (0.0 - 2) (36.9 - 4) (63.4 - 2) (90.0 - 2) (116.6 - 2) (143.1 - 4)
      Border pattern: 1x1  "0"

Layer "PO1" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (39056,27140,27140)
      Fill pattern: 4x4  "1111100110011111"
            strokes:  (0.0 - 2) (45.0 - 4) (90.0 - 2) (135.0 - 4)
      Border pattern: 1x1  "0"

Layer "PO2" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (46999,22506,24492)
      Fill pattern: 4x4  "0110100110010110"
            strokes:  (45.0 - 4) (135.0 - 4)
      Border pattern: 1x1  "0"

Layer "CONT" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (31112,29126,28464)
      Fill pattern: 4x4  "1011010110101101"
            strokes:  (36.9 - 4) (45.0 - 2) (53.1 - 4) (104.0 - 4) (135.0 - 2) (166.0 - 4)
      Border pattern: 1x1  "0"

Layer "MTL1" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,58253)
      Fill pattern: 4x4  "1010010110100101"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

Layer "MTL2" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (39056,0,56929)
      Fill pattern: 4x4  "0101101001011010"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

Layer "VIA" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,5957)
      Fill pattern: 4x4  "1001010000101001"
            strokes:  (126.9 - 4) (135.0 - 4) (143.1 - 4)
      Border pattern: 1x1  "0"

Layer "M1" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (0,0,60239)
      Fill pattern: 4x4  "1010010110100101"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"
```

```
Layer "M2" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (53619,0,48323)
      Fill pattern: 4x4  "0101101001011010"
            strokes:  (45.0 - 2) (135.0 - 2)
      Border pattern: 1x1  "0"

Layer "OXPO" looks:
  For a "XDEV_QDSS" device:
      Priority: 0    nColors: 1    Fill: stippled  Border: empty
      Colors: (34422,45675,52295)
      Fill pattern: 4x4  "0110111111110110"
            strokes:  (0.0 - 2) (45.0 - 4) (90.0 - 2) (135.0 - 4)
      Border pattern: 1x1  "0"

Layer "dop10" looks:
  For a "XDEV_QDSS" device:
      Priority: 80   nColors: 1    Fill: solid      Border: empty
      Colors: (0,0,0)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

Layer "dop11" looks:
  For a "XDEV_QDSS" device:
      Priority: 79   nColors: 1    Fill: solid      Border: empty
      Colors: (32436,23168,15225)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

Layer "dop12" looks:
  For a "XDEV_QDSS" device:
      Priority: 78   nColors: 1    Fill: solid      Border: empty
      Colors: (45675,32436,27140)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

Layer "dop13" looks:
  For a "XDEV_QDSS" device:
      Priority: 77   nColors: 1    Fill: solid      Border: empty
      Colors: (50309,35746,39718)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

Layer "dop14" looks:
  For a "XDEV_QDSS" device:
      Priority: 76   nColors: 1    Fill: solid      Border: empty
      Colors: (52295,44351,31112)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"

Layer "dop15" looks:
  For a "XDEV_QDSS" device:
      Priority: 75   nColors: 1    Fill: solid      Border: empty
      Colors: (56929,43028,51633)
      Fill pattern: 1x1  "1"
            strokes:  (0.0 - 1)
      Border pattern: 1x1  "0"
```

```
Layer "dop16" looks:
  For a "XDEV_QDSS" device:
       Priority: 74   nColors: 1    Fill: solid      Border: empty
       Colors: (59577,58915,55605)
       Fill pattern: 1x1  "1"
              strokes:  (0.0 - 1)
       Border pattern: 1x1  "0"

Layer "dop17" looks:
  For a "XDEV_QDSS" device:
       Priority: 73   nColors: 1    Fill: solid      Border: empty
       Colors: (58915,53619,0)
       Fill pattern: 1x1  "1"
              strokes:  (0.0 - 1)
       Border pattern: 1x1  "0"

Layer "dop18" looks:
  For a "XDEV_QDSS" device:
       Priority: 72   nColors: 1    Fill: solid      Border: empty
       Colors: (56929,61563,0)
       Fill pattern: 1x1  "1"
              strokes:  (0.0 - 1)
       Border pattern: 1x1  "0"

Layer "dop19" looks:
  For a "XDEV_QDSS" device:
       Priority: 71   nColors: 1    Fill: solid      Border: empty
       Colors: (61563,61563,4633)
       Fill pattern: 1x1  "1"
              strokes:  (0.0 - 1)
       Border pattern: 1x1  "0"

Layer "dop20" looks:
  For a "XDEV_QDSS" device:
       Priority: 70   nColors: 1    Fill: solid      Border: empty
       Colors: (63549,64211,62887)
       Fill pattern: 1x1  "1"
              strokes:  (0.0 - 1)
       Border pattern: 1x1  "0"

Layer Palette: "~asw/simpl-rpc/lib/technology/simpl/layers:physical:contents:"
       52 Layers
```

Appendix B

SIMPL-RPC Generated PIF Example



PIF Structure: 10 points, 3 polygons, 232 lines.

(Note: not to scale)

SIMPL-RPC Generated PIF Output: May 1989 ASW

```
-------------------------------------------

(PIF
 (pointList
  (pointListName "P")
  (dimension 2)
  (units "microns")
  (coordinates
    0.000000  5.000000
    4.870000  5.000000
    4.870000  0.000000
    0.000000  0.000000
    0.000000  6.000000
    1.240000  6.000000
    1.740000  6.000000
    4.870000  6.000000
    1.240000  7.000000
    1.740000  7.000000
  )
 )
 (lineList
  (lineListName "L")
  (pointNameList P1 P2)
  (pointNameList P2 P3)
  (pointNameList P3 P4)
  (pointNameList P4 P1)
  (pointNameList P1 P5)
  (pointNameList P5 P6)
  (pointNameList P6 P7)
  (pointNameList P7 P8)
  (pointNameList P8 P2)
  (pointNameList P9 P10)
  (pointNameList P10 P7)
  (pointNameList P6 P9)
 )
 (faceList
  (faceListName "F")
  (lineNameList  L1 L2 L3 L4)
  (lineNameList  L5 L6 L7 L8 L9 -L1)
  (lineNameList  L10 L11 -L7 L12)
 )
 (segmentList
  (segmentListName "Segment")
  (faceNameList F1)
  (faceNameList F2)
  (faceNameList F3)
 )
 (geometry
  (geometryName "SIMPLGeometry")
  (segmentNameList
   "SI"
   "OXID"
   "POLY"
  )
 )
 (grid
  (objectType "Segment")
  (objectName "SI")
  (gridName "SIMPLGrid")
  (gridType "Rectangular")
  (dimension 2)
  (nodeList
   (coordinateList
    (direction x)
    (coordinates
      0.000000
      1.240000
      1.740000
      4.870000
    )
    (direction y)
    (coordinates
      0.000000
      0.100000
      0.200000
      0.300000
      0.400000
      0.500000
      0.600000
      0.700000
      0.800000
      0.900000
      1.000000
      1.100000
      1.200000
      1.300000
      1.400000
      1.500000
      1.600000
      1.700000
      1.800000
      1.900000
      2.000000
      2.100000
      2.200000
      2.300000
      2.400000
      2.500000
      2.600000
      2.700000
      2.800000
      2.900000
      3.000000
      3.100000
```

```
          3.200000
          3.300000
          3.400000
          3.500000
          3.600000
          3.700000
          3.800000
          3.900000
          4.000000
          4.100000
          4.200000
          4.300000
          4.400000
          4.500000
          4.600000
          4.700000
          4.800000
          4.900000
          5.000000
          6.000000
       )
     )
   )
 )
 (attribute
  (attributeName "substrateDoping")
  (table
  (objectType "grid")
  (objectName "SIMPLGrid")
  (objectPart "nodes")
  (data
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
    4  1.000000e+13
```

```
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
      4  1.000000e+13
     )
   )
   (attribute
    (attributeName "maskInfo")
    (objectType "stranger")
    (objectName "SIMPLGeometry")
    (data
     (POLY
       ( 1.240000    1.740000)
     )
     (MTL
       ( 3.290000    3.790000)
     )
     (ACTV
       ( 0.000000    4.870000)
     )
    )
   )
   (attribute
    (attributeName "cutlineInfo")
    (objectType "point")
    (objectName "SIMPLGeometry")
    (coordinates
    -149 5
    338 5
   )
  )
  (attribute
   (attributeName "windowInfo")
   (objectType "point")
   (objectName "SIMPLGeometry")
   (coordinates
     0.000000   0.000000
     4.870000   10.000000
   )
  )
  (snapshot
   (snapshotName "SIMPLCross")
   (units "Micron")
   (usesGeometry "SIMPLGeometry")
   (usesGrid "SIMPLGrid")
   (usesAttribute "substrateDoping" "maskInfo" "cutlineInfo" "windowInfo")
  )
 )
```