TECHNOLOGY COMPUTER-AIDED DESIGN FRAMEWORKS AND THE PROSE IMPLEMENTATION

by

Alexander Shih-Wei Wong

Memorandum No. UCB/ERL M92/27

13 March 1992

TECHNOLOGY COMPUTER-AIDED DESIGN FRAMEWORKS AND THE PROSE IMPLEMENTATION

by

Alexander Shih-Wei Wong

Memorandum No. UCB/ERL M92/27

13 March 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

TECHNOLOGY COMPUTER-AIDED DESIGN FRAMEWORKS AND THE PROSE IMPLEMENTATION

by

Alexander Shih-Wei Wong

Memorandum No. UCB/ERL M92/27

13 March 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Dedicated To Mom and Dad

Contents

Acknowledgments

I would like to thank Professor Andrew Neureuther, my research advisor, not only for his superb research guidance, but also for his encouragement and suggestions on all facets of life during my stay at Berkeley. I am indebted to Professors William Oldham and Charles Stone for serving on my Ph.D. committee and for reading my dissertation. I am grateful to Dean David Hodges for serving on my committee.

My long-time cubicle-mates Rich Ferguson, Ed Scheckler, and Nelson Tam, and cubicle-neighbor Kenny Toh, made life fun and interesting with their computer games, cubicle sports, and the occasional technical discussion. I feel fortunate to think of them not only as my colleagues, but also my friends. The same is true of my more recent cubicle-mates John Helmsen and David Newmark. John Hutchinson and Bill Partlo are great friends, not to mention competitive Mac-game opponents. Ken Nishimura assisted me countless times with small computer requests that often turned out to be big ones. Rick Spickelmier answered all of my Octtools questions.

I thank Duane Boning and Stefan Halama for their collaboration on the Profile Interchange Format. Sandy Wisch transformed BPIF++ concepts into C++ reality. Chris Hegarty and Chris Williams provided much appreciated assistance with BPFL and general process flow language questions. Goodwin Chin offered many great suggestions on general TCAD issues, as well as specific PROSE and PIF ones. May he be forgiven for attending Stanford rather than Berkeley for graduate school. Andrej Gabara and Robert Wang openly shared SIMPL ideas and the latest software releases. It was a joy to work with Dick Gurtler, Norm Shelly, and Bob Tremain during my time at Motorola.

The members of the Semiconductor Wafer Representation Working Group were invaluable in promoting, describing, and implementing a wafer standards. I could not have asked for a more motivated, dedicated, and talented group of researchers to work with. Wally Dietrich deserves special mention for helping me with those tough committee issues that found their way into the SWR realm.

I thank my parents for their endless love, understanding, and support.

Finally, I would like to acknowledge the Semiconductor Research Corporation for their most generous financial assistance.

Biography

Alexander Wong received the B.S. degree in Computer Engineering from the University of Illinois at Urbana-Champaign in 1987, and the M.S. degree in Electrical Engineering from the University of California at Berkeley in 1989. His technical interests include computer-aided design tools and frameworks for process technology and integrated-circuit applications.

He has worked in summer positions at General Electric Semiconductor, AT\&T Bell Laboratories, and Lawrence Livermore National Laboratory in 1985, 1986, and 1987, respectively. In 1989, he was a consultant for the Advanced Technology Center at Motorola. He is currently the Chairman of the CAD Framework Initiative Semiconductor Wafer Representation Working Group, and is a member of Eta Kappa Nu, IEEE, Tau Beta Pi, and Phi Kappa Phi.

1
Introduction to Technology CAD Frameworks

Technology computer-aided design (TCAD) tools can help technologists analyze, understand, and predict the results of integrated-circuit processing technology in a timely and economical manner with computer simulations. TCAD tools can also calculate device behavior and offer insight into process latitude and design manufacturability. As process technologies become more complex, these capabilities make TCAD not only a desirable but an increasingly necessary part of the integrated-circuit design and manufacturing process.

The effective use of TCAD is highly dependent on the unification of specialized TCAD tools and seamless interfaces between TCAD and related CAD domains. The job of a TCAD framework is to provide facilities that enable TCAD tools, and tools in related domains, to work together. Current TCAD systems are still in their infancy - a broad and important set of framework topics must still be addressed. These include common wafer representations, graphical user interfaces, common process flow specification and execution, and ways to couple design, technology, and manufacturing information.

This dissertation examines the diverse and large number of framework issues facing process technology CAD. Solutions in domain interfaces, semiconductor wafer representation, process management, and graphical user interfaces are proposed and implemented in an experimental framework named the Process Simulation Environment (PROSE). PROSE explores these issues through a new open-architecture that allows tools to be easily integrated through common interfaces. PROSE also brings together integrated-circuit design, technology, and computer-integrated manufacturing CAD domains through the OCT/VEM/RPC framework. A graphical user interface based on the VEM layout editor is developed to allow for simultaneous wafer and mask editing, and interactive process simulation capabilities. A Binary Profile Interchange Format (BPIF) toolkit provides common tool interfaces for accessing, storing, and manipulating wafer data such as device topography and impurity fields. A manager for process simulators has been implemented. An interpreted command language, named Pcl, allows users to specify process parameters, query wafer data, and invoke simulators. A Pcl shell supports graphical and textual interfaces for entering and executing process flows.

The development of PROSE has been guided by a valuable early prototype described in the author's masters report [1]. The implementation has been substantiated through the integration of disparate process simulators and the linkage of these simulators to IC CAD through the OCT/VEM/RPC CAD framework. The development of a global phase-shifting mask application takes advantage of PROSE's merged design and technology CAD facilities. PROSE has also influenced, and has been influenced by, other framework projects and representation standards work within the research community. An updated version of the BPIF toolkit, BPIF++, uses many object and function definitions proposed by the Semiconductor Wafer Representation standards group, of which the author is Chairperson. Driving TCAD tools with Pcl and coupling with CIM has been demonstrated with an exploratory implementation using SIMPL.

This chapter introduces the role of IC processing technology simulation tools in the design cycle, evaluates the requirements they place on integration frameworks, and surveys existing systems and standards efforts.

1.1 CAD for Integrated Circuit Technology

Mainstream technology CAD tools can be grouped by tools simulating integrated-circuit fabrication processes (process simulation), tools simulating device behavior (device simulation), and support tools such as input parsers and output visualizers. When used together, these tools can explore device processing technology and device behavior through computer simulation. For example, the device in Figure 1-1 is fabricated on "virtual silicon" using process simulation. With device simulation, the current-voltage characteristics of the resulting device are computed. In this section, the types of tools used in process and device simulation are described. For a more detailed discussion of process and device simulators, several texts are helpful [2] [3][4] [5].

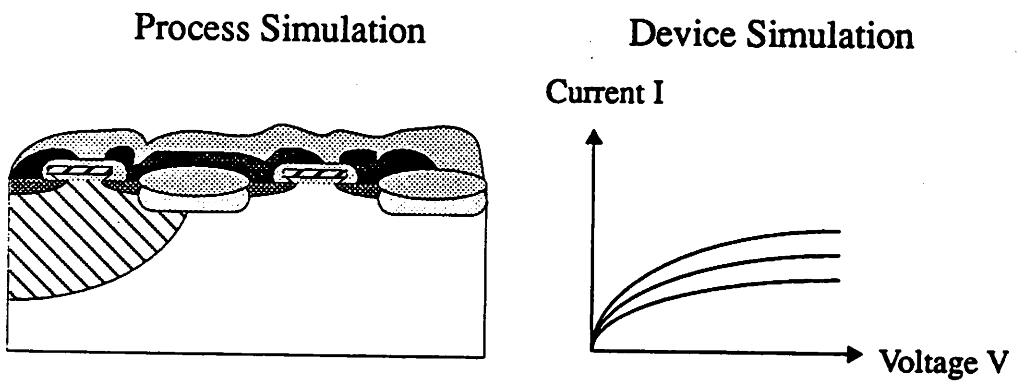


Figure 1-1. Technology CAD can be roughly grouped into process simulators, device simulators, and support tools such as those that display cross-sectional profiles and and current-voltage characteristics.

1.1.1 Process Simulation

Although cross-sectional profiles can be estimated with geometric simulators such as SIMPL-2 [6] and OYSTER [7], combining several specialized, "expert" tools results in a more accurate profile. These experts can be loosely divided into *above-silicon* and *below-silicon* simulators. Figure 1-2 shows how a variety of tools working together are needed to produce the example CMOS cross-sectional profile.

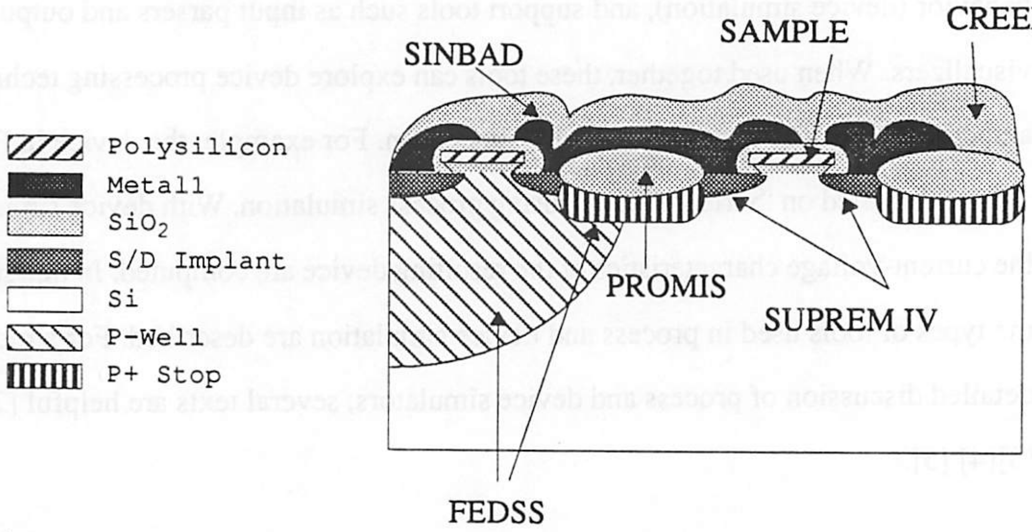


Figure 1-2. Conceptual CMOS wafer cross-section. To obtain the most accurate profile, several different TCAD tools, such as the ones shown above, must be used.

Above-silicon process simulators handle the processing technology that occurs, not surprisingly, above the silicon wafer surface. Pattern transfer, chemical vapor deposition, and device interconnect methods are all above-silicon processes. Pattern transfer is accomplished by a lithographic process involving photoresist application, exposure, and development. Commonly available programs for lithography include DEPICT-2 [8], PROLITH [9] and SAMPLE [10] . For depositing a material using chemical vapor deposition or selectively etching a material using wet or dry processes, ESPRIT [11] and SPEEDIE [12], as well DEPICT, PROLITH and SAMPLE [13], can be used. Metallization for device interconnect is also a deposition process. In addition to the programs mentioned above that treat the advancing surface as a string, atomistic approaches have also been taken, e.g., the SINBAD simulator [14].

Below-silicon processes cover thermal process steps such as dopant diffusion, ion implantation and thermal oxidation.[1] Because of the complex interactions between oxidation and diffusion, these steps are usually handled by a single simulator such as CREEP[15], FEDSS [16], PREDICT [17], PROMIS [18], or SUPREM IV [19].

1.1.2 Device Simulation

Device cross-sections generated by process simulation can be fed into device simulators for device characterization and calculation. Three equations are important in device simulation: the Poisson equation, the electron current-continuity equation, and the hole current-continuity equation. Many two-dimensional simulators that solve these equations have been developed, such as FIELDAY [20], PADRE [21], PISCES [22], and SIFCOD [23]. They can handle general structures for both bipolar and MOS devices, but are computationally intensive. A faster simulation can be achieved if the structure is limited to MOS devices, because then only the Poisson and majority-carrier equations need to be solved. This is the approach of CADDET [24] and MINIMOS [25].

1. Oxidation, which grows at the silicon interface, is generally grouped into the below-silicon category because it involves the physics of high temperature processes similar to diffusion.

Parasitic resistance and capacitance values may also be computed by solving the Poisson equation. This is particularly important due to the interesting and irregular topographic structures present in most advanced devices. Programs that can calculate parasitic values with arbitrary geometries include FCAP2 [2] and RACPLE [26].

1.2 TCAD Framework Requirements

An environment that supports the development and integration of TCAD tools, such as process and device simulators, is called a TCAD environment or *framework*. The ideal framework provides facilities for the TCAD "ABCs", that is, the ability to apply tools easily to new problems, build new tools, and chain tools together.

A common complaint from users is the large initial effort required in learning how to use TCAD tools in comparison with other types of CAD tools. This time sink is compounded by the observation that technologists typically use TCAD for sporadic but intense periods of time. They must therefore re-learn the tool each time it is used (since they have undoubtedly forgotten how to use them from their previous encounter). These problems hinder the ability to easily apply TCAD tools to new technology problems. Providing user-friendly interfaces for TCAD has been demonstrated to be an effective solution. Programs such as SIMPL-DIX [27] contain menu-driven interfaces but lack the ability to use command macros. For example, an analytical oxidation simulation requires the entry of six cryptic oxidation parameters each time it is invoked. Some forms of inherently graphical data, such as mask layout, must also be entered textually. To fully satisfy the ease-of-application requirement, the user-interface must provide menu and keyboard driven input mechanisms for single and multiple macro commands, and graphical means for entering graphic data such as mask layout and initial wafer profiles.

A TCAD framework should also aid the TCAD model developer, who is typically not a computer scientist but is often asked to be one when implementing a new model and tool. By providing generic routines that are common to all programs, such as input parsers, visualization routines, and math libraries, this task can be reduced significantly. For example, a breakdown of the SUPREM IV program shows that less than half of the source code is used to implement the actual models; the supporting routines make up the larger half [28]. Figures for other process simulators are similar. The time saved in building a new tool is valuable because it allows the developer to spend more time on model development.

Tool chaining is the third and final requirement of a TCAD framework. Researchers who model IC processing usually focus on a particular area of the technology, such as lithography, thermal processing, deposition, or etching. Thus, the resulting simulators are naturally developed along process technology boundaries. Although these individual simulators are valuable in understanding and advancing specific parts of IC technology, a much greater gain may be realized by combining these pieces into an integrated package that can handle a complete process flow. This integration can be accomplished through tool chaining.

1.3 The Role of Technology Simulation in CAD

Customer expectations of smaller, faster, and less expensive chips in a highly competitive marketplace places a tremendous burden on all aspects of CAD. Shortened product design cycles require speedy technology development and characterization that results in circuit models early in the life of a new technology. Aggressive design constraints require that these models are timely and accurate. The difficulty of these tasks is compounded by the rising complexity of both the technology and the circuit design. The results of integrated-circuit design CAD (IC

CAD) and technology CAD ultimately wind up in manufacturing, which also pushes CAD within the context of an automated factory. Computer-integrated manufacturing (CIM) can assist in wafer tracking, equipment allocation, process monitoring, process flow, and equipment modeling, and can also make use of design and wafer data from design and technology CAD.

A vertically-integrated CAD system that supports a full range of tasks from synthesis, circuit simulation, and mask layout in IC CAD, to process and device simulation in TCAD, to equipment modeling and process flow specification in CIM, is important to realize the needs of future designers and technologists. As shown in Figure 1-3, this system should provide pipelines for exchanging several key pieces of data among all three environments. In this organization, TCAD plays the central role of filling the gap between design and manufacturing.

1.3.1 TCAD and IC CAD

The distinction between IC CAD and technology CAD is blurring due to the increasing amount of process technology knowledge required to design a chip and the increasing amount of design information needed to drive the technology. In the future, one could imagine how designers can use IC CAD not only to design circuits, but also to see how their designs affect circuit characteristics and technology concerns such as manfacturability, yield, and reliability. Technologists, with the help of TCAD, can potentially produce an optimized technology for each design type, thereby giving designers more freedom in their layouts, or reverse engineer from product to design in order to solve manufacturing problems. The future is not as far away as one may think. There exist tools that make use of both IC CAD and TCAD data, e.g., mixed device and circuit simulators that calculate device characteristics based on layout [29],

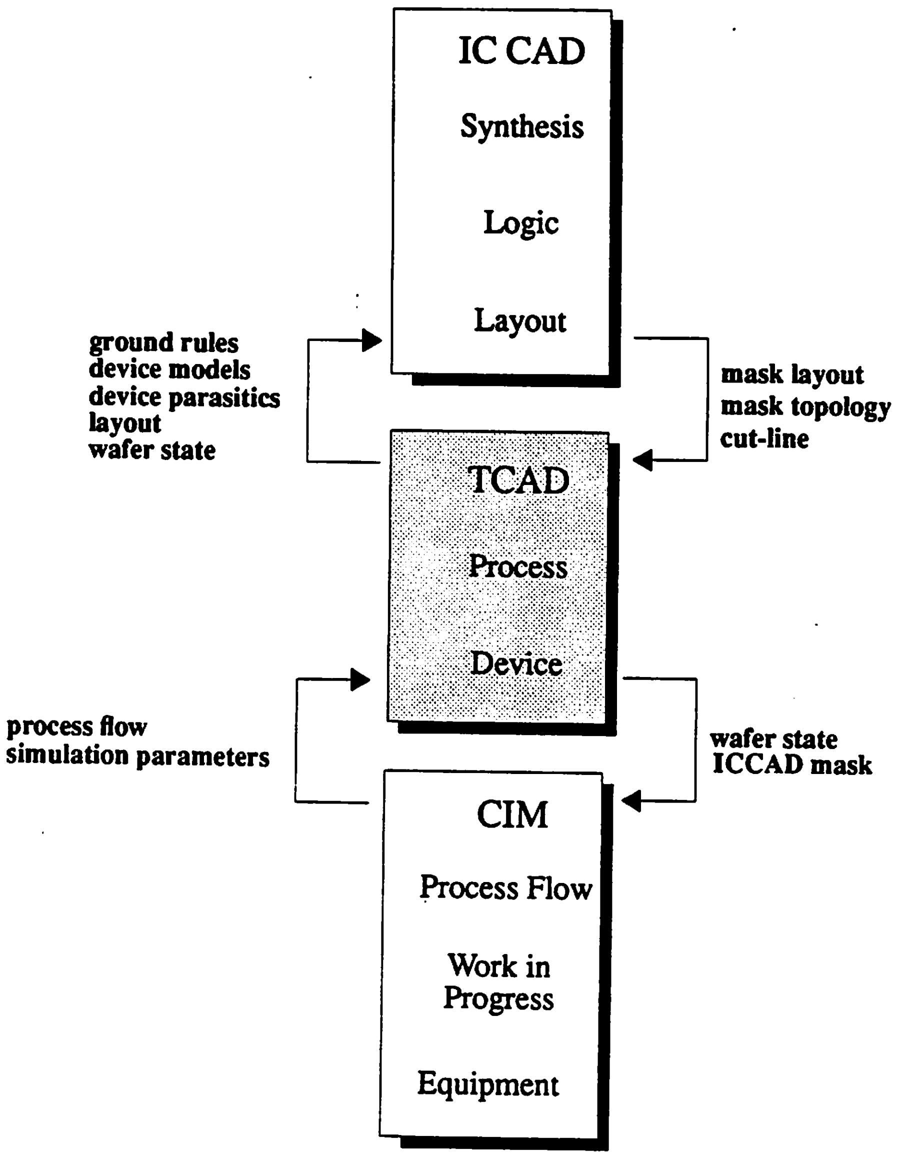


Figure 1-3. A vertically integrated CAD system with data connections between environments. Arrow labels suggest the types of information exchanged.

resistance and capacitance solvers that generate parasitic values due to mask topography [30], statistical simulators that predict process latitude and manufacturing reliability [31], and topography simulators that rely on mask data [6].

IC CAD provides three major types of information to TCAD: physical mask layout information for topography process and device simulation; topological mask information that describes the mask shapes and their relationships with each other (useful for optical imaging and other tools that depend heavily on the mask layout); and cut information that specifies the location on the layout associated with a cross-section view. TCAD, on the other hand, provides IC CAD with the necessary rules for layout and circuit design, including: design rules (also known as ground rules) for mask design; circuit models from device simulation; parasitic resistance and capacitance values for a particular topography; and mask layout information with processing effects.

1.3.2 TCAD and CIM

Interaction between TCAD and CIM is also necessary to transfer technology from the design phase into manufacturing. Because TCAD aims to model the fabrication processes that occur in the factory, it needs process flow, equipment parameters, and other data from the factory. These can be provided by CIM, which, incidentally, needs TCAD wafer information for verifying process flow accuracy, calibrating equipment parameters, etc. This information loop can feed-back upon itself, leading to a situation where TCAD wafer data and CIM rate data are exchanged multiple times. For example, pattern transfer of high numerical-aperture features may require increased etch time or benefit from changes in etching conditions, and interactions between oxidation and diffusion may reduce diffusion time.

Process flow representation is another area where TCAD and CIM intersect. It is very desirable to have the same process flow recipe for both the actual and simulated manufacturing process. The Berkeley Process Flow Language (BPFL) is a merged description that can drive both simulators and equipment [32]. If BPFL is used, the process flow requires both TCAD data, e.g., mask data for pattern transfer steps, as well as CIM data, e.g., equipment models and wafer tracking.

1.3.3 Programming Point of View

These CAD domains are also intertwined from a programming viewpoint. For example, Figure 1-4 shows how the most basic CAD components, such as operating systems, storage managers, and user interfaces, can be shared between IC CAD and TCAD. It makes sense not only to link different types of CAD, but also to take advantage of code re-use through common underlying platforms.

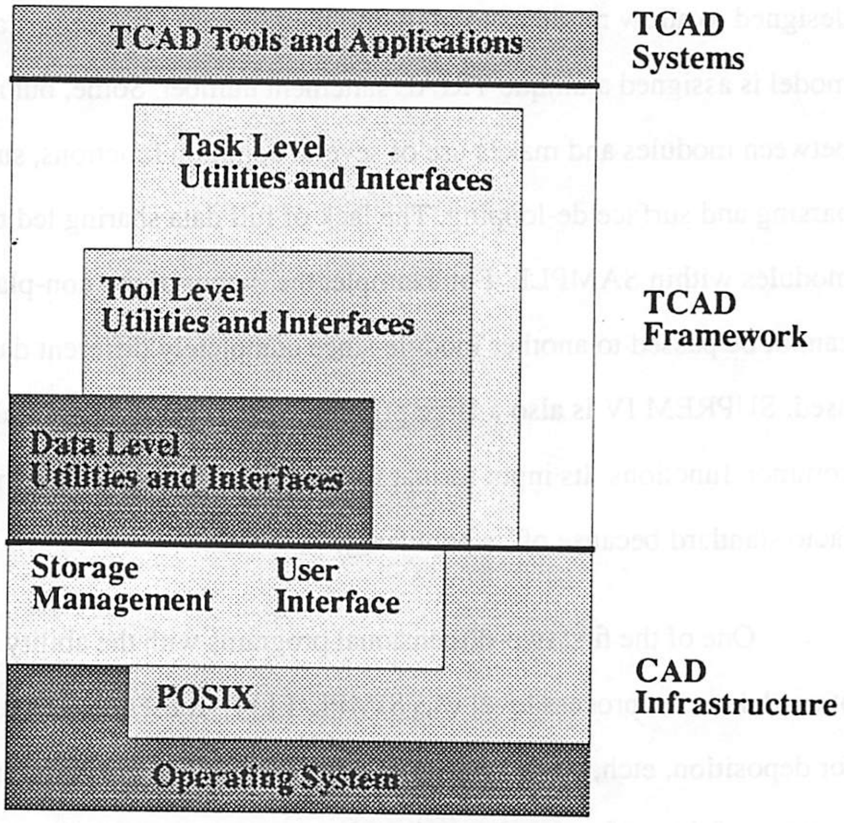

TCAD Tools and Applications
TCAD Systems
Task Level Utilities and Interfaces
Tool Level Utilities and Interfaces
TCAD Framework
Data Level Utilities and Interfaces
Storage Management
User Interface
POSIX
CAD Infrastructure
Operating System

Figure 1-4. A proposed TCAD framework architecture based on a CAD infrastructure [55].

1.4 Survey of Existing Implementations

True frameworks require a set of commonly agreed to infrastructural components and "plug and play" compatibility between tools. The latter is the so-called holy-grail of the TCAD framework community. On the evolutionary path to this long-term goal, many implementations that are framework-like. These include programs that pool models, systems that combine multiple tools, and environments that provide favorable surroundings for tool integration and development.

Early integration efforts resulted in large, monolithic programs that gathered multiple TCAD models. Examples include SAMPLE for photolithography and above-silicon processes and SUPREM-IV for thermal processing. SAMPLE was originally designed to allow modules to plug into the program [33]. In this architecture, each model is assigned a unique TRIAL statement number. Some, but not all, data is shared between modules and makes use of several common functions, such as those for input parsing and surface de-looping. The lack of full data sharing led to incompatible modules within SAMPLE. For example, the output of the non-planar etch module cannot be passed to another module since completely different data structures are used. SUPREM IV is also a large program with several, but not all modules, sharing common functions. Its input format for data and control has become somewhat of a de-facto standard because of its popularity.

One of the first two-dimensional programs with the ability to simulate a variety of models at the process level was SIMPL-1 [34]. It uses elementary analytical models for deposition, etch, development, and implantation to create rectangular cross-sections of the wafer. A polygon-based program named SIMPL-2, soon followed. Advancements in operating system technology also allowed SIMPL-2 to connect to external programs. For example, SIMPL-2 can execute the SAMPLE deposition module transparently, from the user's point of view, through UNIX operating system file transfers and system calls. The OYSTER program, a three-dimensional program that uses analytical models using a solid modeling approach, was also developed during this period. Although these programs used analytical models and computed cross-sectional profiles very quickly, their main weakness was the lack of physically accurate numerical models for precise simulation.

Since the mid 1980s, many groups have published on integrated systems that use time-consuming but accurate simulators to simulate a complete process flow. These include almost all large companies who use TCAD in the U.S., Europe, and Japan [35][36][37][38][39][40][41][42][43][44], universities and other research groups [45] [46], and TCAD vendors [47][48][49]. To a certain degree, all support two-dimensional, numerical-based simulation of the complete process flow and provide data and utility sharing, but within this group, some are more framework-like than others. Generally, those with more framework features, e.g., common wafer databases, shared utilities, etc., are still under active development. The VISTA system [50] and the PROSE environment described here are examples of such TCAD environments.

TCAD environments that target the development of new models are still a research topic and, at the current level of research effort, should be realized within the next five years. Systems such as PROSE and VISTA, although currently integration systems, are making strides towards supporting model development by providing common library routines for basic simulator functions. One system that has targeted model development since its inception is PROPHET [51]. It has two differentiating features. First, models in PROPHET do not use private data structures, but rely on C-function calls to access a common data area. Second, a shared set of math library routines is available to the models.

Two systems that look at the TCAD world from a different perspective are CAFE [52] and PREDITOR [31]. CAFE addresses the TCAD integration issue from the computer-integrated manufacturing viewpoint. PREDITOR is a statistical simulator that calls on fast analytical models many times rather than slow, numerical models only once, to capture the statistical variance that occurs in the factory.

A summary of TCAD past, present, and future "frameworks" is collected in Table 1-1. They are loosely divided into five classes: single monolithic programs, integrated systems, statistical and extention language approaches, systems based on a Profile Interchange Format (PIF)-like representation [53], and development or other types of framework-based systems.

1.5 TCAD Framework Standards

TCAD researchers and users worldwide are working on standards to reduce the number of different tool formats. The current situation is similar to that of the IC design CAD community ten years ago, with each site using its own custom suite of tools. Within the past five years, design CAD has been moving steadily towards framework standards. Rather than re-invent the wheel, TCAD can take advantage of many existing standards, such as those for layout and visualization, but it must also set new standards that are specific to TCAD, particularly for wafer and process flow representation.

SURVEY OF TCAD SYSTEMS		
Name	**Affiliation**	**Notes**
Monolithic General Process Simulators		
MECCA	AT&T Bell Laboratories	AT&T Internal
FEDDS	IBM	IBM Internal
SAMPLE	UC-Berkeley	Above-Silicon Processes
SUPREM IV	Stanford University	Below-Silicon Processes
Integrated TCAD Systems		
COMPOSITE	Fraunhofer	Part of STORM
HP's System	Hewlett Packard	HP Internal
IDDE	Phillips	Object-based Graphical Interface
MASTERPIECE	Silvaco Data Systems	Vendor Integrated System
P&D Workbench	NEC	Networked Computers
SATURN	Siemans AG	Process and Device Simulation
SIMPL-IPX	UC-Berkeley	X Window User Interface with Mask Layout
STUDIO	Tech. Modeling Assoc.	Vendor Integrated System
Supervised Simulation	Toshiba	Geometric Data Interface
TITAN	CNET/CNS	Part of the STORM Project
Statistical and Extension Language Approaches		
Hitachi's System	Hitachi	Includes Numerical Simulation
Preditor	Carnegie Mellon Univ.	Statistical simulation/Chip Database
SIM-C	Dawn Technologies	Vendor, CIM Extension language
Profile Interchange Format (PIF)-like Systems		
CAFE	MIT	Emphasis on CIM
EASE	Intel	Uses a PIF Predecessor
PROSE	UC-Berkeley	This Work
VISTA	TU-Vienna	LISP Tool And Data Control
Development and Other TCAD Framework-Based Approaches		
Framework Prototype	CFI TCAD Framework Group	Early Standards Prototype.
PROPHET	AT&T Bell Laboratories	Development Environment

Table 1-1. Summary of TCAD programs, systems, and environments with framework properties.

The profile interchange format (PIF) effort was the first TCAD standards group. It concentrated on a specification for cross-sectional profile information. This effort resulted in a document that described a representation philosophy and ASCII file format [53]. PROSE and several of the environments mentioned above use a PIF approach.

The CAD Framework Initiative (CFI) and the Electronic Design Interchange Format (EDIF) organizations are current undertaking the definition of standards for IC CAD. The TCAD community has piggy-backed onto these efforts through three sub-groups. The CFI TCAD framework group contains three working groups addressing programming interfaces for semiconductor wafer representation (SWR), semiconductor process flow representation (SPR), and information modeling (DM&V). The information modeling effort is also being pursued by two technical sub-committees within EDIF, the device modeling and verification (DM&V) group and the process and device (P&D) group (which originally handled the PIF work). In August 1991, the TCAD framework group successfully demonstrated a prototype of their standards [54], and efforts are being made to commercialize the standard.

These committees are well represented by most TCAD researchers in the US and a few in Europe. It is the hope that future work in the area of TCAD integration and development frameworks will endorse these emerging standards.

1.6 Summary

Technology CAD plays an important role in understanding and advancing semiconductor technology development. There are many tools today that can simulate process and device behavior in integrated-circuit technology. TCAD frameworks can increase the value of these tools making them easier to apply, reduce the time needed to develop new tools, and allow tools to be chained together. Most groups have recognized the need for a TCAD framework and have developed their own systems, and standards efforts are underway to unify semiconductor wafer and process representations among tools.

1.7 Dissertation Outline

The goals of the Process Simulation Environment (PROSE) are to develop a TCAD system that enables the efficient application of process simulation to technology problems by users, effortless integration of TCAD tools by developers and system integrators, and the interaction between different types of CAD. In addition, a conscious effort is made to assist in forging framework standards that benefit the community at large.

The role of PROSE as a TCAD tool integration environment is covered within the first five chapters of this disseration. Chapter 2 describes the overall framework architecture and summarizes each of the components. Component interaction is explained from a user point of view through a practical example that simulates a CMOS EEPROM bit-cell. Chapters 3 through 5 tackle the difficult problem of exchanging semiconductor wafer information between tools in a uniform and robust manner, and agreeing on a common representation between different groups. Wafer representation requirements, standards efforts, and an object-oriented functional interface to wafer data are presented.

Interactions with design and manufacturing environments are then covered. Chapter 6 describes how the joint use of TCAD and IC CAD, especially strong in PROSE because of its access to a general CAD framework, can be used to advantage. Chapter 7 describes a phase-shifting mask CAD toolkit application that makes use of both TCAD and design CAD within PROSE. This application is able to analyze large, non-regular cells for phase-shifting mask applicability using global and local techniques. Chapter 8 covers the management and description of process flow information within PROSE using a tool command language, and the link to computer-integrated manufacturing systems through a common process flow language. Conclusions and perspectives on future trends in TCAD frameworks are discussed in the final chapter.

2
The Process Simulation Environment

In exploring TCAD issues it is important to balance unconstrained "what if" thinking with the reality of what can be efficiently implemented. PROSE has the grounding in reality through the implementation of a TCAD system for tool integration, yet also explores future sources of stimulation by working with available resources in IC CAD. Starting with an overview of PROSE will give the general reader an example of a TCAD system, and provide the experienced TCAD developer with a survey of the strengths of PROSE.

The quest for a system that satisfies the ABCs of TCAD (tool application, tool building, and tool chaining) has led to advances in framework architecture, wafer representation, and process flow specification. PROSE has the following features and capabilities. A friendly user-interface contains graphical, menu, and textual editing capabilities. This allows even casual users to use TCAD tools for their application. Common utilities for input, output, and wafer manipulation help TCAD developers implement new models more easily. A common TCAD database significantly reduces the time required to integrate tools by providing a central repository for cross-sectional device profile information and a functional interface to access that data. A new process command language allows users to specify and execute process flows in a programming language style, with procedure calls to access process simulators and wafer data.

An equally important emphasis in PROSE is on the coupling of TCAD to different environments and the development of industry standards for TCAD data representation. PROSE was designed with an "open architecture", meaning all components are modular and communicate by way of standard interfaces accessible from technology, design, and manufacturing CAD domains. Several ideas generated by the TCAD community's standards efforts, such as the architecture described by the SRC group [55] and the CFI TCAD Framework Group [56], have been incorporated into PROSE and vice-versa.

Section 2.1 chronicles the history of the concept by a motivating example taken from the SIMPL system. Section 2.2 describes the overall architecture of the environment and summarizes each of the components. The interaction of PROSE with design and manufacturing CAD environments is discussed in Section 2.3. Section 2.4 describes the flow of data between components from the programmer's perspective. Section 2.5 explains PROSE from a user's perspective, using an EEPROM device as an example. A chapter summary is given in Section 2.6.

2.1 History

The idea for a process simulation environment began in 1987 after the evolutionary nature of the SIMPL-DIX [27] architecture caused it to become unwieldy. As shown in Figure 2-1, this fourth generation system has an architecture where tools are directly connected using specific translators. Each arrow represents a link between two programs, typically requiring a few graduate student-months to develop. Therefore, the fifteen arrows linking SIMPL-DIX to other tools represents almost three graduate student years of work for only a partially connected system.

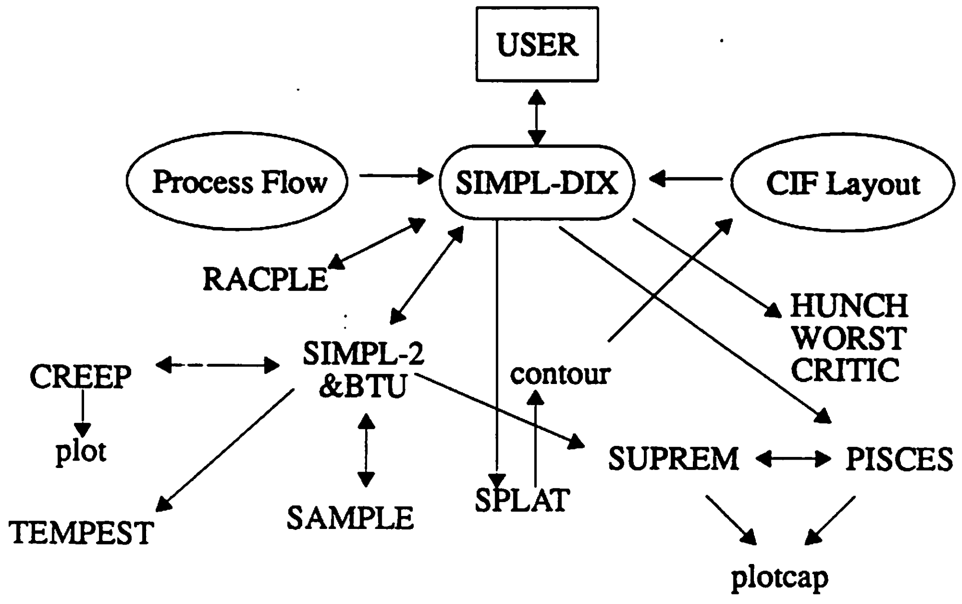


Figure 2-1. The SIMPL-DIX system implementation.

The intent of PROSE is to define an architecture and implement a system aimed at modular TCAD tool integration to ease the burden of adding new tools. An initial version of the architecture and an exploratory implementation were developed in the Spring of 1989 [1]. The encouraging results of this work led to a continued effort in the area and resulted in the version of PROSE described in this dissertation.

2.2 Architecture

The architecture contains an interesting mix of new TCAD specific components for wafer and process flow representation, and existing generic CAD components for user interaction, data storage, and tool communication. Figure 2-2 summarizes this architecture in its component form. Tool and task level applications are designed to plug directly into the system through interfaces provided by the TCAD database and process flow manager.

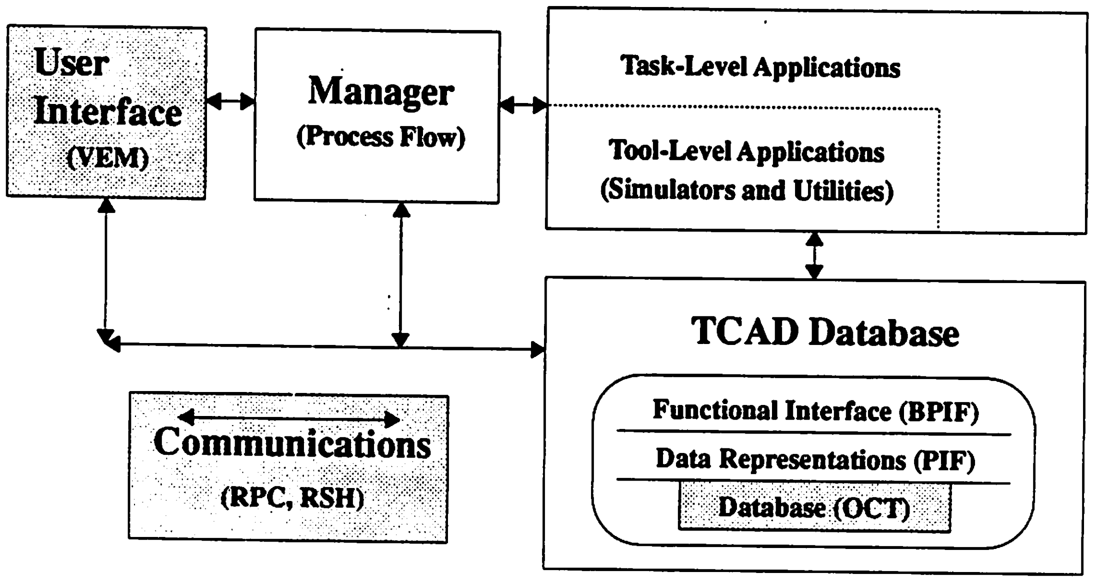

User Interface (VEM)
Manager (Process Flow)
Task-Level Applications
Tool-Level Applications (Simulators and Utilities)
TCAD Database
Functional Interface (BPIF)
Data Representations (PIF)
Database (OCT)
Communications (RPC, RSH)

Figure 2-2. The PROSE architecture. Shaded components make use of the OCT/ VEM/RPC CAD Framework.

The TCAD database component uses the Binary PIF (BPIF) interface which defines a set of geometry and mesh objects, and implements C and C++ function libraries that can access and manipulate them. The process flow manager combines simulation, control, and BPIF commands in a command language named Pcl. A layout editor allows users to edit mask and wafer data, and invoke process commands, while a separate graphical user-interface controls process flow execution.

2.2.1 OCT/VEM/RPC

Early in the definition process, it was recognized that the requirements for TCAD environments were similar to those of IC design CAD environments, and therefore IC CAD could provide hints and/or implementations for parts of the architecture. PROSE takes advantage of previous CAD implementations in the user interface, database, and communications areas by using the OCT/VEM/RPC CAD

Framework [57]. OCT (Octopus[1]) is an object-based data manager, VEM (View Editor Monolith) is a combination layout editor and user interface, and RPC (OCT Remote Procedure Call) is a means for tool communications across different machines and operating systems. In addition, the open architecture of OCT/VEM/RPC allows it to be modified with relative ease. PROSE makes use of these components where possible, and provides extensions to the components for specific TCAD needs to from an infrastructural base for TCAD applications.

2.2.2 Graphical User Interface

The graphical user interface is an interactive shell and geometric editor. The shell is menu and/or keyboard driven, with dialog-boxes that "pop-up" when additional information from the user is required. The geometric editor supports the display and editing of both mask layout and cross-section data. This editor is much more flexible than the SIMPL-DIX user-interface [27], which only allows layout and cross-section viewing. Multiple windows and menus are supported in this X-windows graphical environment.

2.2.3 The TCAD Database

The most important, and the most difficult to implement, component within PROSE is the TCAD database. This database acts as the central repository for all cross-sectional wafer data. Applications access this data through a standard functional interface named the Binary PIF that allows programs to store, retrieve, and manipulate wafer objects such as material boundaries and the fields within them [58]. Object types are based on the Profile Interchange Format (PIF) representation [53]. This philosophy is better.than previous wafer storage schemes, e.g., SIMPL-IPX, because a central

1. The first Octtools database was named SQUID. OCT is its successor. According to folklore, the term OCT stands for octopus, to suggest the database's many arms.

representation reduces the number of translators needed between tools and the time required to add new tools. The underlying persistent database implementation uses the OCT database.

A second responsibility of the TCAD database is to provide a pipeline to and from other CAD environments. For IC CAD, this primarily means mask layout and wafer data. Because the OCT database is used for both wafer and mask data storage, this link naturally exists within PROSE. Programs can access both types of data through similar functional interfaces - the Binary PIF for wafer data, and the standard OCT interface for mask data. For CIM, wafer and process flow data can also be exchanged through functional interfaces.

2.2.4 Manager

The manager coordinates all components needed to accomplish a given task. From the user's viewpoint, input to the manager can originate either from interaction with the user, or from a pre-determined process flow specification. In the first case, process simulation commands can be invoked from a menu within the VEM layout editor. In the latter, a set of process flow language commands can be interpretively executed using the PROSE command language (Pcl), which is a modified version of the Tool Command Language (Tcl) [59]. Pcl contains many programming language features such as variables for holding simulation parameters and conditional statements for complex process flows.

An X-window user-interface using the Tk toolkit [60] allows Pcl commands to be run from a more friendly graphical environment. Process flow management by the Berkeley Process Flow Language (BPFL) [32], which unifies simulation and manufacturing process flow information, has also been investigated for use in PROSE.

In either interactive or interpretive mode, the manager processes commands one at a time. Upon receipt of a command, the manager uses either the RPC or UNIX shell communication mechanisms to initialize and execute the necessary component(s). For example, a lithography simulator might be invoked to perform a lithography step. The manager would handle the communications necessary to retrieve the initial profile from the TCAD database, perform the simulation, and write back the resulting profile.

2.2.5 Applications

Both tool and task level applications are supported. Tool level applications are generally a single program such as a process or device simulator. Task level applications, on the other hand, can invoke one or more tools multiple times, e.g., a process parameter optimization program.

Three process simulators have been incorporated into PROSE: the SAMPLE topography simulator [13], the SIMPL-2 general process simulator [6], and the SUPREM-IV thermal simulator [19]. From SIMPL-2, translators are also available to variety of other process and device simulators [45]. PROSE utilities have been developed for display of profile boundary and impurity information. Additional applications can be added with relative ease by replacing their specific interfaces with the interfaces provided for process flow and semiconductor wafer data.

2.3 Coupling with Other Types of CAD

In addition to providing an environment for TCAD tools, an equally important goal within PROSE is to couple TCAD to related CAD environments.

At best, current TCAD systems have a very weak link to mask information, which is the intersection point between TCAD and IC CAD. PROSE tightly couples these two environments through the shared use of the OCT/VEM/RPC CAD Framework. Programs may access both wafer data and mask or higher level design information through similar OCT interfaces, thereby removing the distinction between the two environments.

PROSE also attempts to provide links between TCAD and CIM at the process flow and equipment modeling levels. Pcl serves as a common input language for TCAD tools and contains constructs for tool invocation, error handling, event looping, and conditional branching. Calls to CIM can potentially be made from this language to invoke the BPFL interpreter or the Berkeley Equipment Modeling system [61]. Wafer information is transferred to CIM through the Binary PIF.

2.4 Data Flow

To explain the operation of PROSE in more concrete terms, this section describes the dataflow between each of the components for a typical unit step. This operation is summarized graphically in Figure 2-3. Typically, each unit step causes three sub-steps to occur. These are signified in Figure 2-3 by the annotated arrows.

Two types of information, control and physical, flow between the components. Control information controls the component, i.e., input commands to a process simulator, and is generated by the manager. Physical data, on the other hand, is the physical quantity, such as the process flow, wafer, or mask layout data that the tool needs to perform its task.

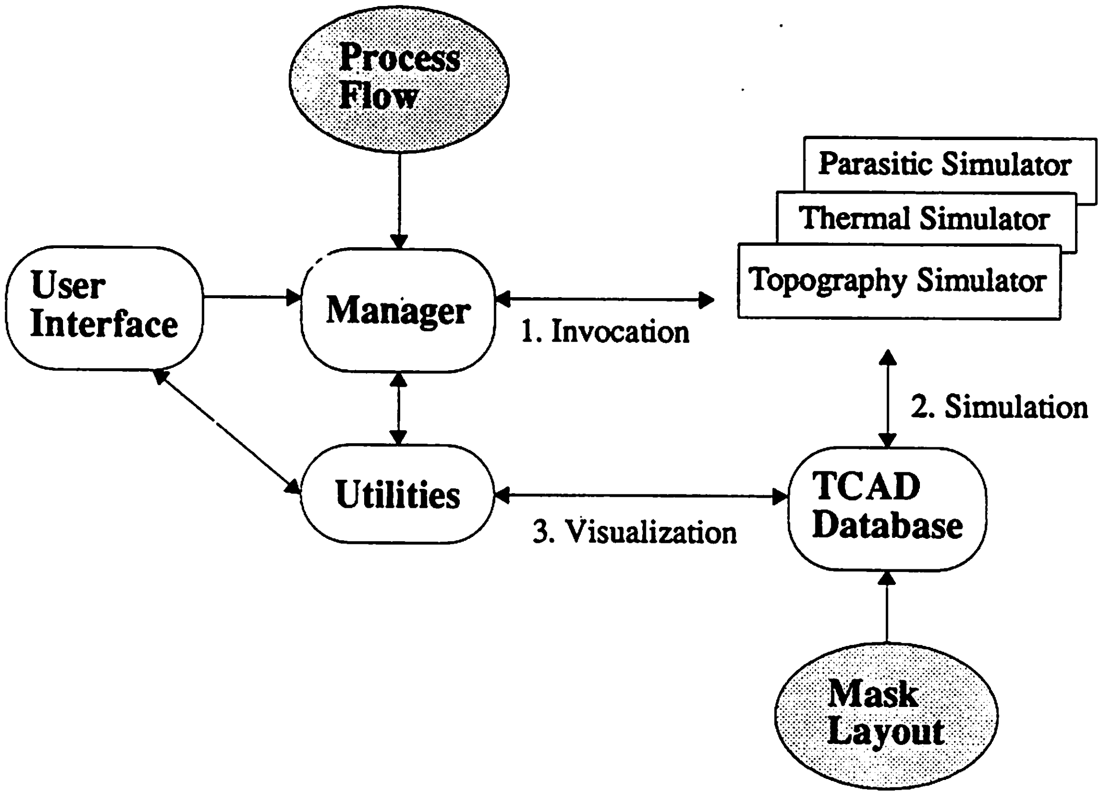


Figure 2-3. Data flow within the environment. Process flow and mask layout information is provided by the computer integrated manufacturing and IC design CAD environments.

2.4.1 Invocation

The process begins when the manager receives a command either from the user through the menu interface in interactive mode, or from the next Pcl command in interpretive mode. The manager then determines which simulator should be invoked based on the command type and initializes that program through the remote shell mechanism. After the simulator is awakened, the manager either translates the process command into a simulator input deck, or makes a direct function call to the simulator. This sub-step consists entirely of control information - no physical data is exchanged.

2.4.2 Simulation

The simulator reads in the input deck that contains the commands to be executed, and a pointer to where the mask and wafer data is stored. After retrieving the physical data from the TCAD and/or IC CAD database, the simulator performs the simulation and stores the resulting data back into the database.

2.4.3 Visualization

Since the communication is synchronous, the manager waits until the simulator completes its task before the visualization utility is invoked. The visualizer, like the simulator, reads the most recent versions of the profile and mask data, and transforms it into a form compatible with VEM. The graphic is then displayed in VEM, and the process step is marked "done".

2.5 EEPROM Bit Cell Example

This section shows how PROSE can be used to generate a CMOS electrically erasable read-only memory (EEPROM) bit-cell similar to the one described by Chacherelis et al. [62]. The process flow and mask layout for the cell are shown in Figures 2-4 and 2-5.

2.5.1 Interactive Mode

To use PROSE interactively, the user starts the session by invoking the VEM layout editor, which also acts as the user interface shell (see Figure 2-6). The initial mask layout is then drawn into the *physical mask facet* of the cell using VEM mask layout editing functions, a tutorial of which can be found in [63]. The complete mask layout as seen in VEM is shown in Figure 2-5. The OCT facet representing the cross-sectional view of the device, named the *cross facet*, must also be opened from the VEM shell. The interactive manager is then invoked by selecting the PROSE

EEPROM PROCESS FLOW

```
Twin Well 1um CMOS Front End
Modified Locos Isolation
Buried N+ Mask and Implant
Gate and Coupling Oxidation
Enhancement Vt Adjustment Implant
Poly Deposition/Definition
Interlayer Oxide
Metal1 Deposition/Definition
Interlayer Oxide
Metal2 Deposition/Definition
```

Figure 2-4. CMOS process flow for the EEPROM example [62].

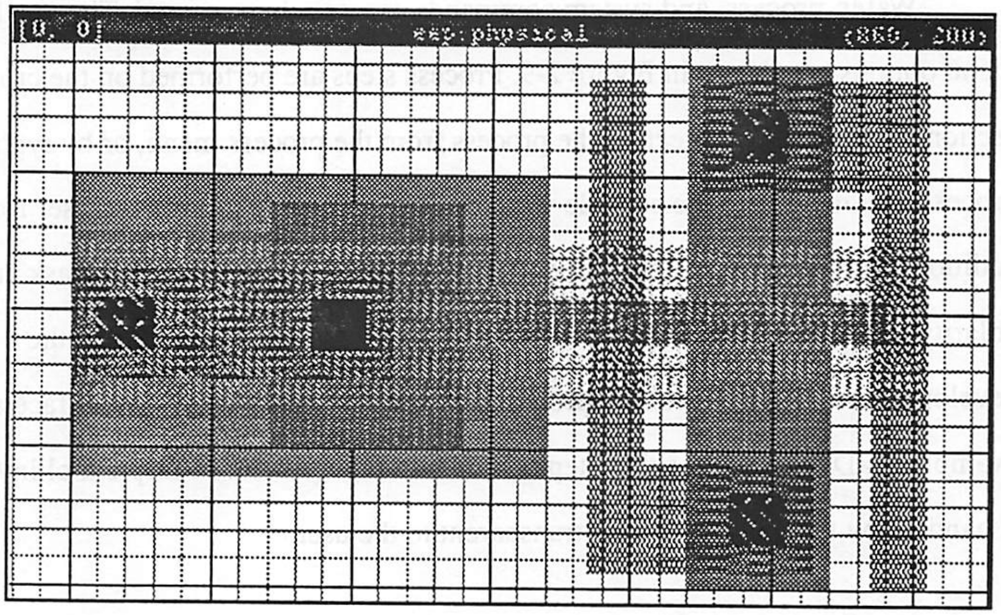
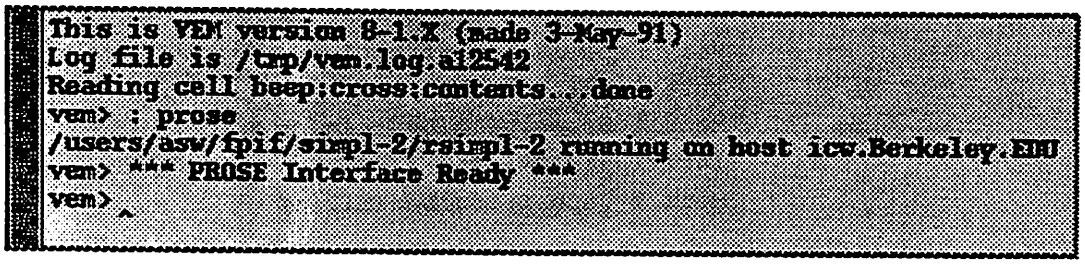


Figure 2-5. EEPROM mask set [62].

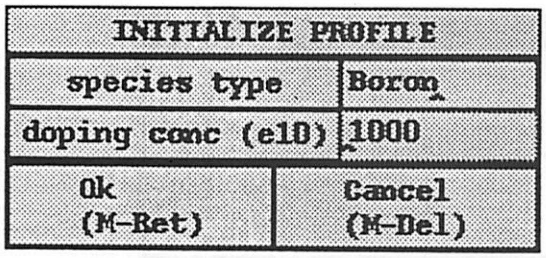
application in the VEM menu from the cross facet. After PROSE prompts the user for the background substrate doping and the desired cut-line is entered into the *cutline facet*, the starting silicon substrate is displayed in the cross facet, and the cut-lines associated with the mask are shown in the *cuts facet* (see Figures 2-7 and 2-8).



```
This is VEM version 8-1.X (made 3-May-91)
Log file is /tmp/vem.log.a12542
Reading cell beep:cross:contents...done
vem> : prose
/users/asw/fpif/simpl-2/rsimpl-2 running on host icw.Berkeley.EDU
vem> *** PROSE Interface Ready ***
vem>^
```

Figure 2-6. Starting PROSE by invoking the VEM layout editor and selecting the PROSE application. This creates a VEM console window and enables the PROSE menu.

Wafer, process, and system commands can now be executed. The available menu options are shown in Figure 2-9. Process steps are performed on the cross-section facet either by selecting the process from the process menu, or by typing the process step name into the console window. Pop-up menus prompt the user for additional information, such as resist thickness in the spin-on step, the mask layer and polarity to be used in the exposure step, and the layer to be developed in the development step. The resulting profiles are then displayed as new cross facets. An example CVD oxide deposition step is shown in Figure 2-10. Multiple tool invocation is handled by the manager and is transparent to the user.



INITIALIZE PROFILE

species type	Boron
doping conc (e10)	1000

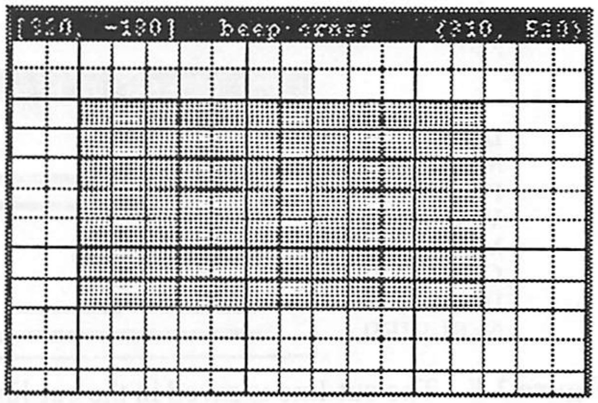


Figure 2-7. The cross facet. A pop-up menu obtains the substrate background doping from the user. The initial wafer profile, a piece of bare silicon, is then displayed.

The final profile for the EEPROM cell is shown in Figure 2-11. Because it is boundary-based, the VEM layout shows only the topography of the device, even though both boundaries and fields are stored in the OCT database. A separate viewer, based on the SIMPL-DIX program [27], can be used to display the fields within each layer (see Figure 2-12).

2.5.2 Interpretive Mode

In interpretive mode, The process flow can also be entered textually. This is useful when the process flow is overly time consuming to accomplish interactively, when a process flow has been previously defined and is run on a new cut-line or mask

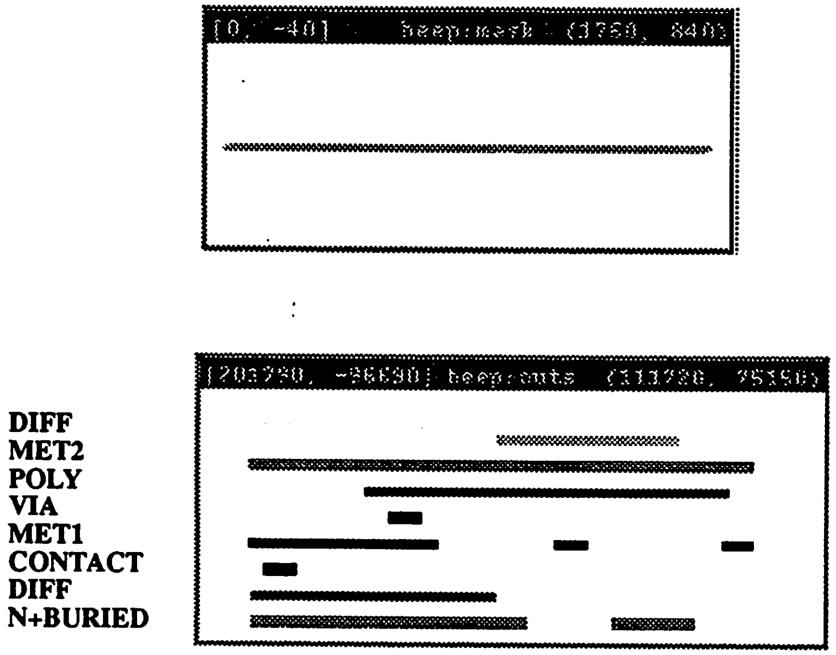

[0, -40] beep:mask (3760, 840)
[203720, -86690] beep:cuts (313720, 75150)
DIFF
MET2
POLY
VIA
MET1
CONTACT
DIFF
N+BURIED

Figure 2-8. The cut-line is saved in the cut-line view (top). The cut-lines for each mask layer are automatically extracted from the general cut-line and shown in the cuts view (bottom).

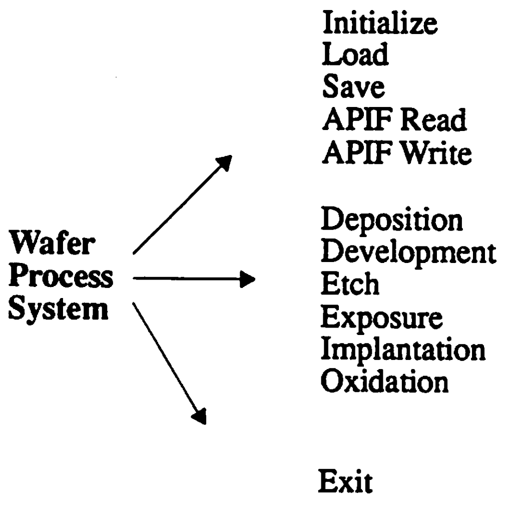
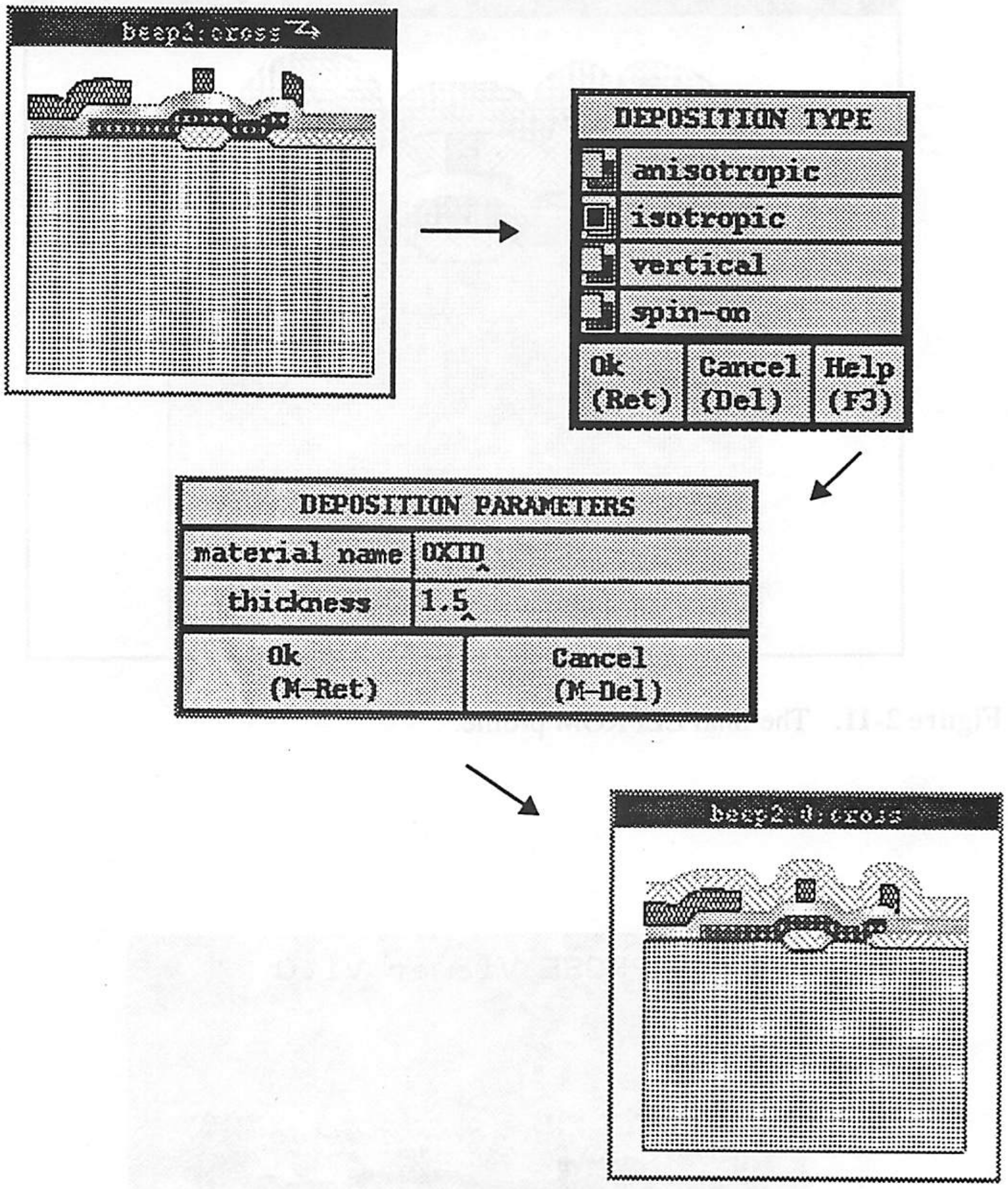


Figure 2-9. The menu of PROSE commands.



material name OXID
thickness 1.5

Figure 2-10. An example deposition step. (Top) the original wafer, (Middle) the menu invocation and dialog boxes, (Bottom) the resulting wafer.

set, or when only slight modifications to an existing process flow are made. The LOCOS oxidation and N+ implant steps for the EEPROM process flow, expressed in Pcl, are shown in Figure 2-13. Analytical SIMPL-2 models are used in this example. The interpretive manager interprets the process flow one step at a time, much like a

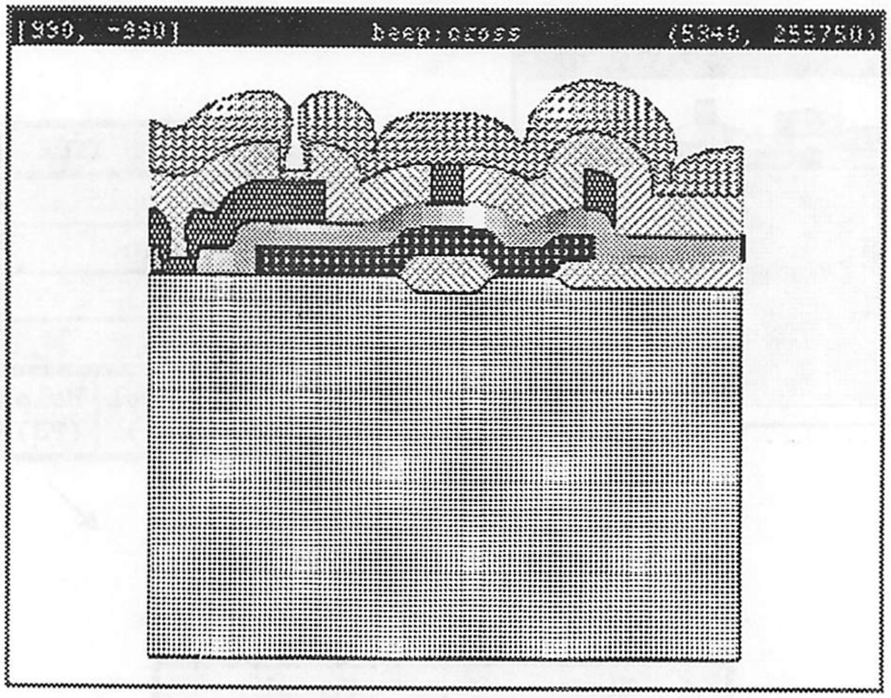


Figure 2-11. The final EEPROM profile.

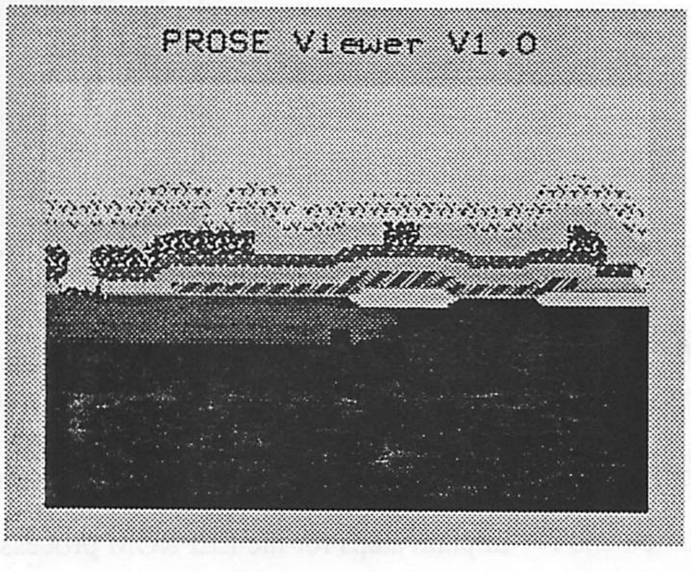
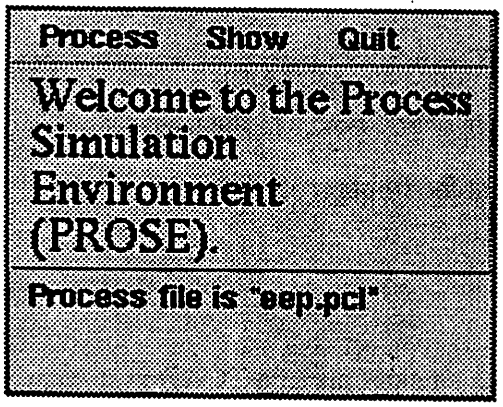


Figure 2-12. EEPROM substrate doping is displayed in the PROSE viewer.

code debugger, e.g., dbx [64], interprets a section of source code in a C-language program. The flow is shown in the simple X-windows based user interface that assists users in executing process flows. The filled in check-boxes in the figure signify that the LOCOS steps have been completed, but that the implant steps have not.

2.6 Summary

This overview chapter provides both an example of a TCAD system and highlights some of the strengths of PROSE. The architecture uses the OCT/VEM/RPC CAD framework for its generic graphical-user interface, database, and communications needs. Extensions have been made to the framework so that TCAD specific requirements such as wafer data representation can be handled. Using a CAD framework allows a close coupling between TCAD and IC CAD, since exchange of wafer and mask layout information uses the same underlying interface. PROSE operates on unit steps. Each unit step causes a series of interactions between each of the components, which reads in a starting cross-section, performs a simulation step or steps, and writes back the resulting cross-section. An EEPROM example shows how PROSE can be used both in interactive and interpretive modes. The following chapters will examine the key issues in TCAD systems and the PROSE implementation in greater detail.

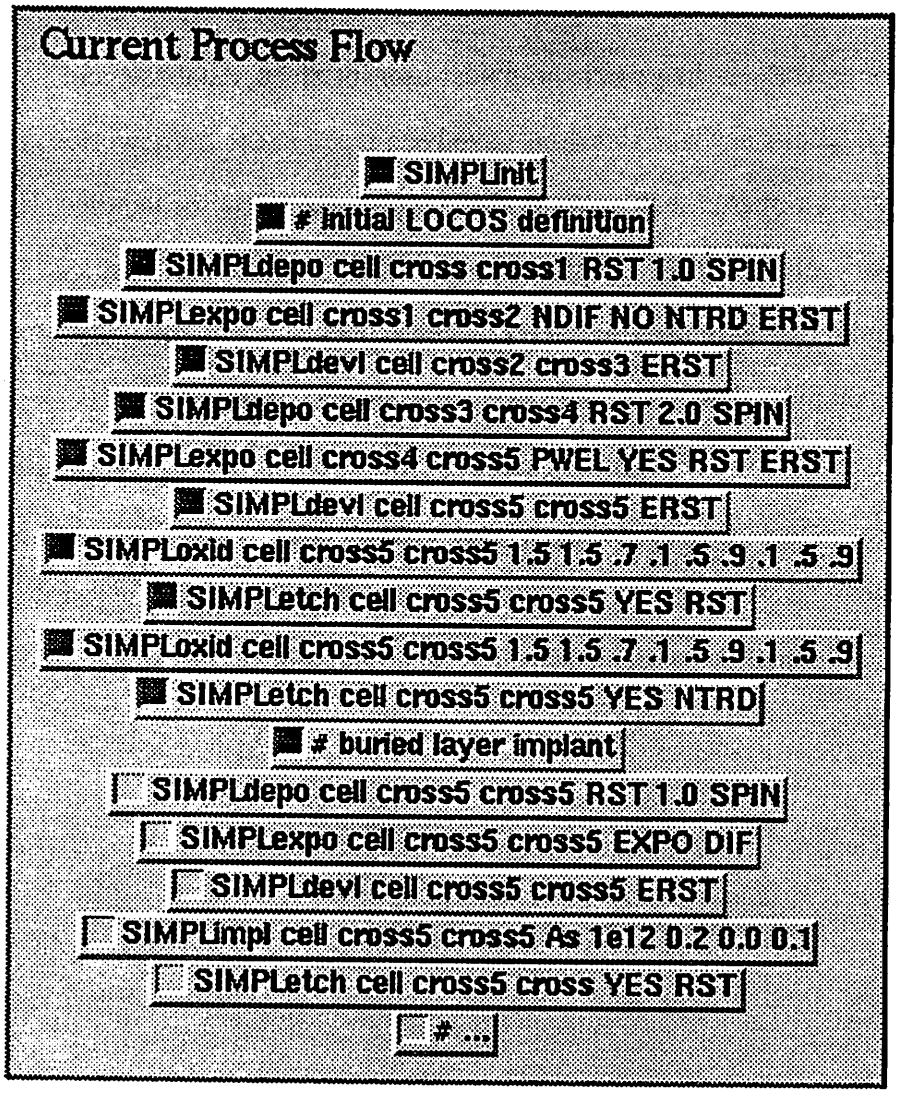


Welcome to the Process Simulation Environment (PROSE).

Process file is "eep.pcl"

Show Menu Options

show $\longrightarrow$ open file
show flow
run step
run all


Current Process Flow

```
■ SIMPLinit
■ # initial LOCOS definition
■ SIMPLdepo cell cross cross1 RST 1.0 SPIN
■ SIMPLexpo cell cross1 cross2 NDIF NO NTRD ERST
■ SIMPLdevl cell cross2 cross3 ERST
■ SIMPLdepo cell cross3 cross4 RST 2.0 SPIN
■ SIMPLexpo cell cross4 cross5 PWEL YES RST ERST
■ SIMPLdevl cell cross5 cross5 ERST
■ SIMPLoxid cell cross5 cross5 1.5 1.5 .7 .1 .5 .9 .1 .5 .9
■ SIMPLetch cell cross5 cross5 YES RST
■ SIMPLoxid cell cross5 cross5 1.5 1.5 .7 .1 .5 .9 .1 .5 .9
■ SIMPLetch cell cross5 cross5 YES NTRD
■ # buried layer implant
□ SIMPLdepo cell cross5 cross5 RST 1.0 SPIN
□ SIMPLexpo cell cross5 cross5 EXPO DIF
□ SIMPLdevl cell cross5 cross5 ERST
□ SIMPLimpl cell cross5 cross5 As 1e12 0.2 0.0 0.1
□ SIMPLetch cell cross5 cross YES RST
□ # ...
```

Figure 2-13. The first two steps of the EEPROM process flow are specified using SIMPL-2 models in Pcl and displayed through the user interface.

3
Representing Wafer Information

Technology CAD (TCAD) places specific demands on the representation, access, and manipulation of semiconductor wafer information. TCAD tools, such as process and device simulators, need to store and exchange different types of information, such as multi-dimensional geometric, field and attribute data. Furthermore, tools in other areas, e.g., integrated-circuit CAD and computer-integrated manufacturing, also benefit from access to wafer information.

This chapter begins in Section 3.1 by describing what semiconductor wafer information is and how it relates to other types of CAD data. Section 3.2 takes a pragmatic, bottom-up look at how existing process and device simulators model wafer information. Section 3.3 motivates the need for a common wafer representation, and Section 3.4 surveys previous approaches. The chapter is summarized in Section 3.5.

3.1 What is Wafer Information?

Semiconductor wafer information models the wafer state at a particular time instance. Pictorially, wafer data may be represented by device cross-sections and impurity profiles, such as the MOSFET shown in Figure 3-1. These cross-sections contain information such as device topography, impurity concentrations, and electrode connections to the outside world. Two related types of data are mask and process flow information. Although they are used to generate the wafer cross-section, they are not considered part of the wafer representation.

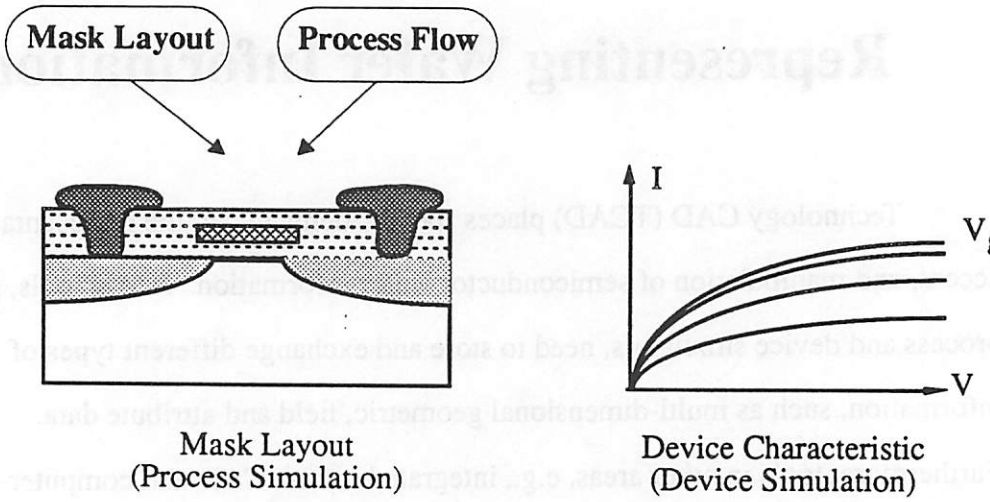


Figure 3-1. TCAD Wafer Data

Depending on the requirements of a particular TCAD tool, wafer data can be represented in several ways. To better understand these requirements, different TCAD tools are examined in terms of how they represent and make use of this data.

3.2 Who Uses Wafer Information?

Process simulators are arguably the most demanding users of semiconductor wafer data. Topography, oxidation, and other simulators that model the fabrication process have a varied set of requirements for manipulating cross-sections.

3.2.1 Boundary Representation Tools

Topography simulators that model deposition and etching processes generally rely on boundary representations [12][13]. This is because topography simulators are interested in profile shape and material adjacency information. A popular representation is the string-model and its variations [65][66], and an example implementation is the SAMPLE non-planar etch machine [67]. In this model, each material layer is represented by a string of vertices that are connected by line segments (see Figure 3-2); the shape of the profile evolves by moving the vertices based on their etch rates. One drawback of this particular string model implementation is the requirement that all strings span the cross-section. This is a problem when materials reach only halfway. A more general approach is the polygonal data structure in SIMPL-2 [6]. In this representation, connectivity information is stored on vertices at material boundaries (see Figure 3-3). This representation is similar to winged-edge data structures used in graphics applications [68], but still does not properly handle cross-sections with completely enclosed islands of material.

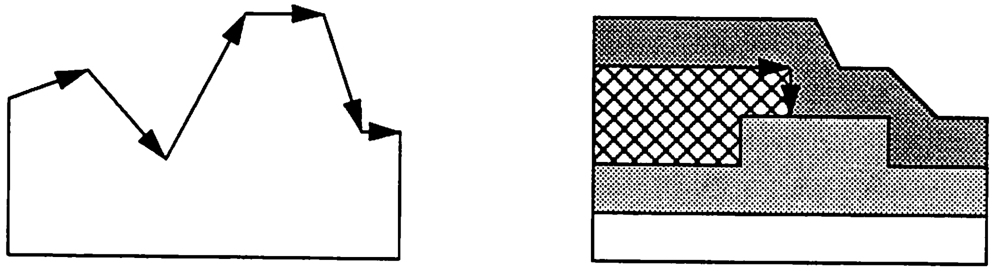


Figure 3-2. (Left) A string represented by a set of connected line segments. (Right) A problem can occur when the string does not span the entire cross-section.

A different approach to topography simulation is the use of cells that represent the atomistic nature of materials. In the ballistic deposition model proposed by Brett [14], circular shaped cells (or disks) with sticking and other coefficients are shot at a profile (see Figure 3-4). An approximate boundary based on cell density can then be computed. For etching, the cell-model can be used to remove "atoms" of the material based on their exposure to the etching mechanism [69].

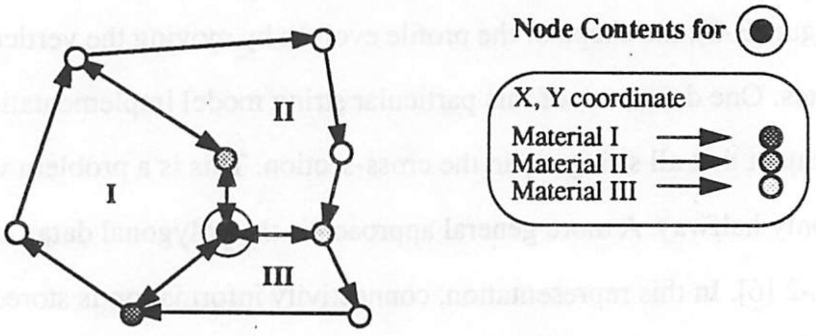


Figure 3-3. The SIMPL-2 polygonal representation. Vertex nodes store information about touching materials. If material name pointers are followed, a clockwise outline of a material can be obtained.

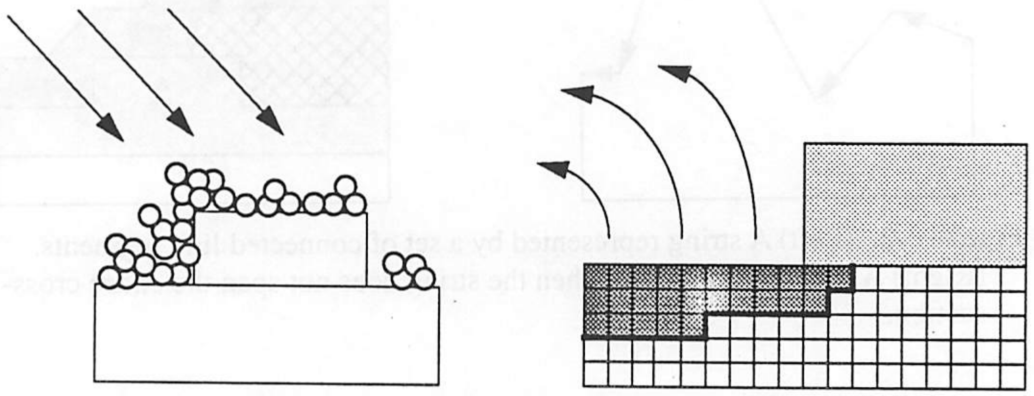


Figure 3-4. Atomic-based approaches, such as ballistic deposition (left) and cell-based etching (right).

3.2.2 Mesh Based Tools

In photolithography, meshes are used to track the amount of photoactive compound in a photoresist during exposure and development. The SAMPLE program, for example, implements this model with a tensor product mesh.

Thermal simulators for oxidation, diffusion, and implantation are also generally mesh-based due to the field nature of the data they manipulate. A well known thermal simulator that uses a triangular mesh is SUPREM IV [19]. Whether finite-element, finite-difference, or other discretization methods are used, local element neighbor information must be part of the structure for computational efficiency. Global material boundary information, on the other hand, is lower priority in these simulators, and is specified only implicitly through the material type for each element (see Figure 3-5).

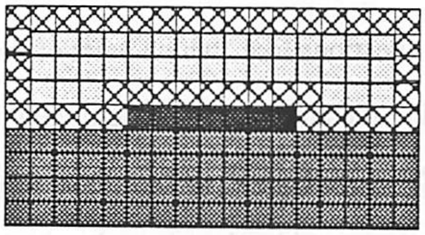


Figure 3-5. Material boundaries are obtained through the slow process of searching for boundary elements. For example, the boundary elements of the lightly stippled material are cross-hatched.

The majority of TCAD simulation tools are mesh based. This is also true for resistance and capacitance solvers and device simulators, which use meshes to solve the Poisson and electron-hole current continuity equations. There are a variety of mesh types in use. Some of the most common are triangular, rectangular, and tensor-product meshes, but others, such as Octrees [70], have also been successfully applied to process and device simulation. Auxiliary representations are also necessary for specifying boundary conditions in both mesh and boundary-based representations.

Tools may also rely on analytical or *unevaluated* functions for wafer representation. This is the case of the FABRICS statistical simulator [71]. Because each complete simulation requires thousands of simulations (to account for the statistical randomness of the process), full numerical simulation is too slow. Instead, fast analytical solutions are used instead. Unevaluated fields also occur in device simulator output such as current-voltage and capacitance curves.

In summary, wafer data is represented most commonly by its geometry and field components (see Figure 3-6). The geometry view, also called the boundary or topography view, provides information about the material boundaries of the cross-section, and can answer questions like "which parts of the wafer are polysilicon" or "where is metal exposed to the ambient". The field view best describes fields within a cross-section, such as a MOSFET's source and drain regions or the gate threshold voltage adjustment, as represented by the triangular mesh.

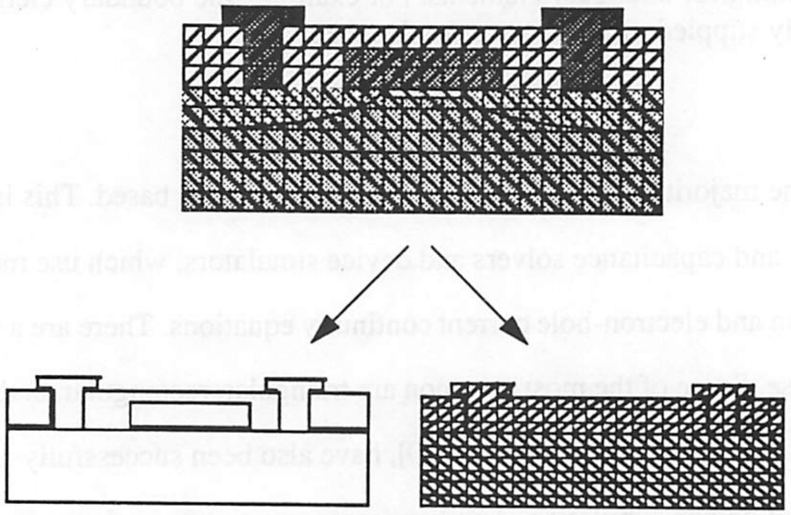


Figure 3-6. Separation of a MOSFET into geometry and field views.

3.3 Issues in Common Wafer Representation

A common wafer representation goes a long way to helping solve the tool connection or chaining TCAD framework requirement. The most obvious reason for using a central representation is to reduce the number of translators and therefore the amount of translator development time necessary between tools from O(n^2) to O(n), as shown in Figure 3-7. Another important reason is the loss of data accuracy due to incomplete representations and/or poor translation between them.

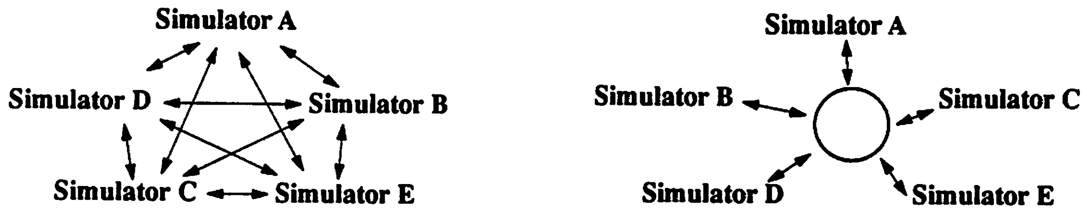


Figure 3-7. The number of translators grows O(n^2) with the number of simulators to be integrated. This can be reduced with a central representation.

Unfortunately, wafer representations optimized for specific process steps, such as those described in Section 3.2, do not necessarily produce an optimized common representation when combined. For instance, pioneering integration systems used evolutionary approaches to arrive at a common representation. Often, the data structure or the associated savefile format of the most used tool in the system was taken as the initial "common" representation. As additional requirements were discovered, the database evolved to meet those new needs. In the SIMPL system, the initial design goal was to integrate topography tools using the SIMPL-2 data structure. When tools that made use of node and line segment representations were added to the system, such as the creeping flow simulator CREEP, the SIMPL-2 data structure was extended with additional slots for marking tracing boundaries. The implementation of the SIMPL/SUPREM link has also led to some extensions of the SIMPL-2 mesh structure but not enough to completely represent the information needed by SUPREM IV. Since element data remains with the SUPREM program, data consistency becomes a problem. After four generations of SIMPL, the latest version, SIMPL-IPX, has turned into an "integration jungle," with many specific translators and no central location for wafer data.

The practical experiences gained from the SIMPL implementation have identified several major issues that must be addressed when developing a common wafer representation. These issues are now discussed.

3.3.1 Multiple Wafer Views

The scope of the representation must adequately cover all wafer views that are needed by the TCAD tools using the framework. For instance, the MOSFET profile in Figure 3-6 may need to supply its boundary nodes for an etching simulator, or its mesh nodes for an oxidation module. Maintaining consistency between the two major views, i.e., field and geometry, is difficult, since each time one view changes, the other one must be updated, or at least be marked invalid. As mentioned before, there are a variety of different mesh types used in TCAD, each of which can be considered a different view.

The situation is complicated even more if the tools within the framework also use unevaluated functions, have special representations for boundary conditions, or use custom views. Multiple views also place demands on storage requirements, since each profile may need redundant information to meet performance constraints for a particular view.

3.3.2 Data Mapping

Associated with the multiple view problem is the problem of translating the data between views, or *data mapping*. The inaccuracies of translating between boundaries and meshes in between SAMPLE and SUPREM is one example of a data mapping problem that has been previously investigated [72]. Consider performing an etch step with both view types. Many problems arise when the resulting boundary is "stitched-back" into the mesh. For instance, imbalances in the boundary and mesh densities must be equalized, or the boundary may have to "snap" to the grid (see Figure 3-8). The quality of the translation is highly dependent on the way the data mapping is performed.

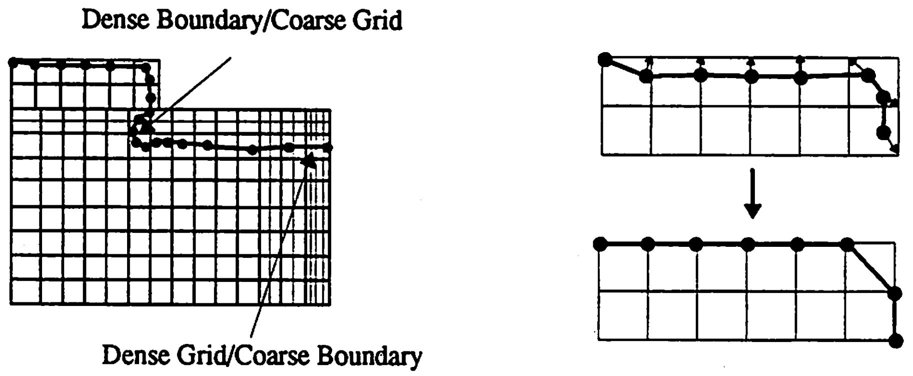


Figure 3-8. The "stitch-back" of a boundary etch-front causes a data mapping problem. (Left) Mismatches in the boundaries and/or field densities must be handled. (Right) Additional constraints might require the boundary to be snapped to the mesh elements.

Translation between mesh-types is another example of data mapping. Translators are needed for converting between different forms of uniform and non-uniform meshes. Even the straightforward conversion of a rectangular mesh to a triangular one by dividing each rectangle into two triangles, for example, may lead to problems of a mesh that is too dense for the TCAD tool to handle. Density adjustment or other adaptive grid techniques must usually be employed.

3.3.3 Application Constraints

Wafer representation is dependent on the type of application using it, i.e., whether the tool exists both in source code and binary forms, or only the latter, makes a difference. When both are available, the internals of the tool can be changed to conform to the common wafer representation. In the binary only case, a tool *wrapper* must be used. The wrapper translates the incoming common representation into the tool's format, performs the function, and translates the tool's output back into the common representation, as shown in Figure 3-9. Wrappers limit wafer representation performance because the entire profile must usually be passed into and out of the wrapped program, as opposed to only passing the necessary data. Nevertheless, wrappers can be used whenever it is desirable to leave the TCAD tool in its original form, as is true of most existing tools. New tools, however, should avoid the use of wrappers.

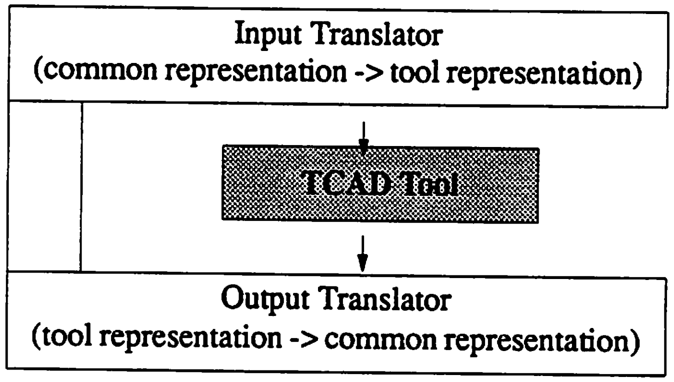

Input Translator
(common representation -> tool representation)
TCAD Tool
Output Translator
(tool representation -> common representation)

Figure 3-9. Wrappers allow existing tools to use a common representation without modifying the internals of the tool.

The types of data access provided are also dependent on whether the framework is to be used for tool integration or development purposes. For example, set operations are important when integrating tools to perform operations the stitch-back function shown in Figure 3-8, which can be easily implemented with a geometric "inset" of the etch front into the original profile. Their use in supporting TCAD model development may be limited though, since algorithms usually operate on individual elements and segments rather than entire geometries. "Move-point" and "move-element" would be more helpful in this case.

3.3.4 Performance

Performance requirements differ depending on whether TCAD tools plan to use the data representation for tool integration or model development. In the first case, performance is much less of an issue because the time spent within the tool is much greater than the time taken in exchanging the data. Performance is obviously very important when the representation is used within the tool.

3.4 Common Wafer Representation Approaches

Common wafer representation approaches currently in use are uniform file formats, functional interfaces, and shared-data structures. The choice of implementation is highly dependent on the level of functionality desired and the targeted application.

3.4.1 File Interchange Formats

An approach that allows for a loose coupling between tools is the uniform file interchange format. This concept is similar to the textual formats used in IC CAD for layout information, e.g., Caltech Intermediate Form (CIF) [73]. Tools exchange information by reading and writing data files using the format. If the format is complicated, the parsing function can be difficult to implement though. Generally, file formats are used for integration. The Intersite Profile Interchange Format [53], discussed in Chapter 4, is an example of a common file format.

3.4.2 Functional Interfaces and Client-Server Models

Functional interfaces allow tools to access data directly without the need of a file format, and have been used in the most recent wave of PIF-like environments being developed [50][74]. These interfaces range in complexity from low-level interfaces that provide file parsing functions, medium-level interfaces that access TCAD data as objects, to high-level interfaces that can manipulate multiple or composite objects. Low- to medium-level interfaces usually address the integration problem, while higher-level interfaces can be used for integration as well as development purposes.

A related approach is the client-server model. In this model, a *wafer server* maintains the state of the wafer. Client TCAD tools, such as simulators and visualizers, request wafer data from the server through a functional programming interface. In effect, the server acts as an oracle that answers questions about the wafer state posed by the clients. This approach is geared towards new tool development. The functionality of the client-server model depends on the amount of functionality placed on the server side relative to the client side. The Semiconductor Wafer Representation Architecture [56], which is presented in Chapter 4, as well as the Chip Database (CDB) [75] make use of the client-server model.

Both functional interfaces and the client-server model have the advantage of being database independent, since the storage mechanism is hidden from the programmer through the function call.

3.4.3 Common Data Structures

The tightest coupling of tools is achieved through a shared data structure. This can be quite effective if the same language is used for all tools, there are a limited number of tools, and the agreed-upon data structure does not change. Shared data-structures can support both integration and development, depending again on the level of functionality that the common function library provides. A prime example of this type of environment is DAMSEL [76]. Common data structures have the disadvantage of having a high maintenance cost, because each time the data-structure is changed, all related modules must also be updated.

3.4.4 Performance Considerations

When considering data access performance, the data structuring approach is the most efficient, followed by functional interfaces, the client-server model, and file interchange formats. The difference in performance between data-structures and functional interfaces is minimal. Initial results show that the client-server model can also be competitive to data structures to within ten percent [77]. File formats are a factor of ten slower.

3.5 Summary

Understanding what wafer information is and how it can be represented is an important part of building a successful TCAD framework. Two widely representations are boundaries and meshes. Multiple data views, data mapping, application type, and data access performance make development of a common representation very difficult. Several different approaches that address this problem are currently in use, including file formats, functional interfaces, and shared-data structures.

4
Wafer Representation Standards

The use of standards in semiconductor manufacturing, such as wafer size, type, and crystal orientation, is analogous to the use of standards in TCAD. A standard wafer representation is particularly valuable to the TCAD community because of the complex physics involved, requiring multiple areas of expertise.

Several de-facto wafer standards based on popular simulator programs such as SIMPL-2 [6], and SUPREM IV [19] are already in use, but their formats are over-burdened with the demands placed on them by new tools. Since 1986, two committees addressing the Profile Interchange Format (PIF), and the Semiconductor Wafer Representation (SWR), have been actively developing standards based on input from multiple groups in the U.S. and Europe. Finding the right path to standardization is difficult. Technical considerations such as the choice of operating system, programming language, and programming paradigms, e.g., the client-server model, an object-oriented approach, etc., must be sorted out. Development costs, membership from universities and companies on different continents, and other political issues also arise.

This chapter gives a perspective on the progress and key issues in developing standards for wafer representation. It is based on the author's experiences from co-developing the SIMPL-IPX system, designing PROSE, participating in the PIF standards, and chairing the SWR working group. Section 4.1 discusses the motivation for a standard and the parties involved. The progress and potential of the PIF and SWR efforts are then charted in Sections 4.2 and 4.3, respectively. Section 4.4 describes standards work on information modeling. A quantitative cost analysis of standards for integration and development are given in Section 4.5. Major areas of concern in the SWR are discussed in Section 4.6. Section 4.7 suggests smoother ways to run the standards process. A possible scenario for future standardization is presented in Section 4.8, and conclusions are made in Section 4.9.

4.1 Motivation

4.1.1 Benefits

There are three main arguments for developing and using a wafer standard. First, a standard reduces the amount of effort each group needs to devote to the common representation problem. With the vast amount of work remaining in wafer representation (especially in three-dimensions), the pooling of expert minds makes the best use of limited resources. Second, a standard wafer representation allows tools to share data, allowing them to be easily connected to synergistically address new technology issues. This is the main benefit of an *integration standard.* Third, supporting routines written for the standard can be used by all TCAD tools, reducing the development time for new tools and encouraging model development. This is the motivation for a *development standard.*

4.1.2 Everyone Wins

Three types of TCAD developers make up the wafer standardization interest - universities, industry, and vendors - each of who have their motivations for promoting a standard.

Professors and their students are interested first in development standards, and second in integration standards. A development standard reduces the time necessary for graduate students to develop and test out new models. A side benefit is the research content of development standards. Since it is still an active research area, publications and degrees can result from standards work. Integration standards are also desirable, mainly because it allows researchers to accelerate technology transfer to industry and other users, as well as address new technology issues by combining simulators. Tool sharing through integration standards also allows for testing a specific tool with other tools. For instance, an etch tool may need to be linked to deposition and lithography tools to prove its usefulness. Finally, there should be flexibility in the standard to allow universities to explore and expand on them in their research.

Research and development groups in industry require a standard for comparing models and integrating the large number of tools provided by universities and industrial partners. A wafer standard for tool integration is top priority for companies who do not have a way to link their tools. Companies with integrated systems are more inclined to support the development standard, which is a long-term project. Either way, the standard must be "industrial strength" in specification and implementation to satisfy the requirements of these practical users, but still remain flexible enough to support future extensions.

Vendors can also use an integration standard to help them connect their tools more easily. In the future, the ones who do will be at an advantage over those with closed systems. This advantage is important because of the small number (certainly less than five) of vendors that the market can support. The general opinion is that vendors will be driven to use standards only if demanded by the user, but there is also a potentially lucrative business in standards documentation and implementation. To date, the development standard has less of an impact on vendors because it is still in its research and definition rather than implementation phase.

Led by the universities, supported by the industry, and closely watched by the vendor community, wafer representation standardization has been pursued since 1985. The wafer standards effort was first organized by Neureuther in an open letter mailed to TCAD groups worldwide [78]. This letter proposed the formation of a committee to standardize on a cross-sectional profile representation. Since then, two major efforts have taken on the challenge. The Profile Interchange Format addresses mainly the integration standard, while the Semiconductor Wafer Representation addresses both integration and development.

4.2 The Profile Interchange Format (PIF)

The Process and Device Technical Subcommittee of the Electronic Design Interchange Format (EDIF) organization was formed in 1986 to formalize a wafer standard by a working group of university and industry researchers. The result of this first effort was the publication of the textual Profile Interchange Format (PIF) in 1988 [53]. Since its initial definition, the PIF has evolved into a storage file-format as well as an object definition for a functional interface. These two versions are known as the Intersite PIF and the Intertool PIF, respectively. When combined, the PIF provides the features of a file format for exchanging data across sites, as well as an efficient functional interface to wafer data (see Figure 4-1).

4.2.1 File Format

The Intersite PIF allows for data exchange between multiple tools through files that have a EDIF-like syntax. An example of an ASCII PIF file describing a MOSFET can be found in [53]. This file format has gained only limited acceptance as a standard file format for describing wafer information due to its parsing complexity. Several tools have been modified to write to variations of the format [1][50][79][80][81], but none are able to read in and understand all files generated by others. Another

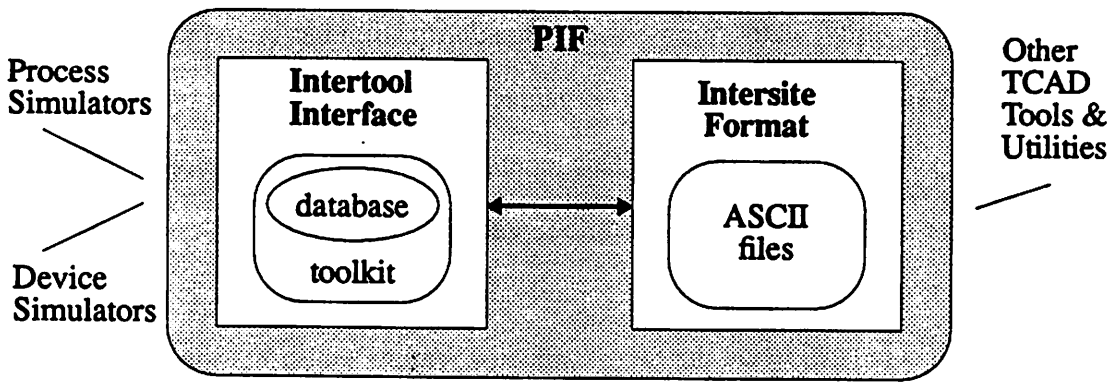


Figure 4-1. The Profile Interchange Format (PIF).

disadvantage is the inefficiency of writing and accessing data serially through a file , since tools are often only interested in selected portions of the representation. To address these problems, efforts have been made to provide low-level parser functions for retrieving data from a PIF file [82][83], but only a handful of tools have made use of them due to their limited functionality.

4.2.2 Functional Interface

The next step in the evolution was the broadening the PIF concept in ways more suitable for rapid interaction between simulators. The Intertool PIF uses the same conceptual model as the Intersite PIF, but represents wafer data as object instances in a graph, and provides a functional programming interface that can access and manipulate these instances. Besides allowing TCAD tools to access PIF data commonly, the functional interface has other advantages. It promotes code reuse. For example, each tool no longer needs a PIF file-parser, since object data can be readily retrieved through function calls. Programmers may also work with these objects at different levels of abstraction, e.g., as single objects such as a point, or as higher-level composite objects such as a geometry containing multiple points. Finally, storage of data as a graph of objects allows for direct rather than serial access of PIF information. These advantages lead to higher programmer productivity and more efficient access to the data.

Several TCAD database implementations use the Intertool PIF concept, including an early PIF-like database developed at Intel [36], the Binary PIF described in Chapter 5, PIF/Gestalt [79], and the Binary PIF in VISTA [84]. Although these implementations do not currently conform to a single standard PIF model, they are all members of the PIF family.

4.2.3 Discussion

Technically, the PIF has the potential to become a standard for tool integration. The Intertool PIF shows promise as a functional interface to wafer data. The PIF can also help in development needs by providing shared utilities, but the level of the sharing that it can achieve will depend on future research. Unfortunately, the PIF is remains a philosophy and not a standard. Various groups working on the PIF have and continue to make new and necessary extensions to the initial file format version, resulting in multiple PIF versions that are incompatible with one another. The PIF puzzle now has all the pieces, but needs a champion to put it all together.

Politically, the PIF is a good first effort at an integration standard and has played an important role in drawing together TCAD framework researchers from CMU, UC-Berkeley, MIT, Stanford University, and TU-Vienna. The research era of tool integration is almost over though as universities turn their attention to a development standard. Commercialization by companies and vendors is the next logical step. Unfortunately, the PIF has not succeeded in obtaining the necessary backing from these groups. Other than the attendance of an occasional meeting and the support university research efforts, the PIF has unfortunately fallen on the back-burner of the commercial TCAD stove.

4.3 Semiconductor Wafer Representation

In June 1988, the Semiconductor Research Corporation (SRC) organized a workshop on TCAD Frameworks at Stanford University. Discussion at the conference ignited a ten-man effort on this topic. An initial architecture document outlining the requirements for such a framework was completed in March 1989 [55]. This work generated a shopping list of capabilities that future systems should have. Out of this list, wafer representation was identified as one of the two most important items for standardization.[1]

In June 1989, the TCAD Framework Group was formed as a committee of the CAD Framework Initiative (CFI). Within this group, the Semiconductor Wafer Representation (SWR) working group was assigned the task of standardizing on wafer representation. The members spent one year developing an architecture and programming interface that would support both integration and development needs. The SWR contains the experiences of many groups who have looked at the wafer representation problem in university and industry. At the kick-off meeting of the SWR, ideas from many different implementations (FOXI, PREDITOR, PROPHET, PROSE, SUPREM/PISCES) and previous standards (EDIF DM&V and EDIF P&D) were collected, and their union was used as a basis for initial standards discussions [85].

1. Process representation was the other item identified as being very important.

4.3.1 Architecture

The architecture, which was approved by the committee for prototyping purposes in April 1991, is shown in Figure 4-2 [56]. It uses an object-oriented approach to describe wafer data and the client-server model to access it. The SWR representation is divided into field and geometry components, which provide boundary and mesh based views of the wafer cross-section, respectively. Clients access wafer data by way of a standard programming interface defined in C++ [86]. The types of calls that the client may make to the server are shown in Figure 4-3. The SWR architecture and programming interface documents are publicly available and can be obtained through the CAD Framework Initiative. A general discussion of the SWR may be found in [87].

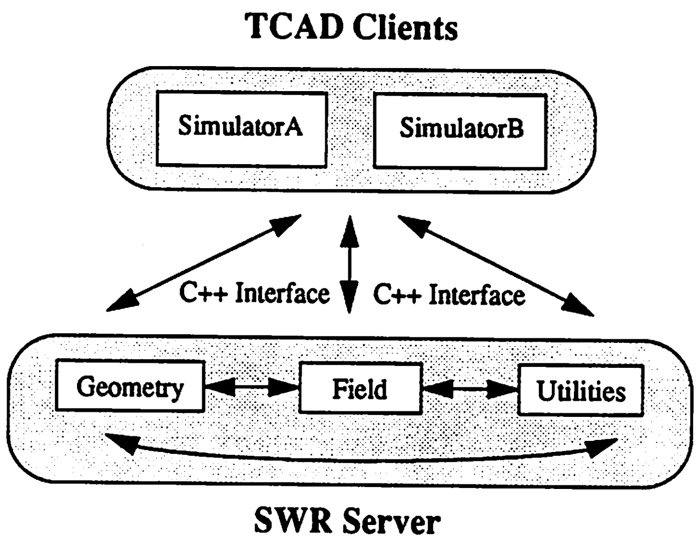


Figure 4-2. The SWR architecture [56].

Geometry Server

Objects:
point, edge, face, cell, cell complex

Operations:
create, delete, access (general);
section, coalesce, inset (set)
create geometry from mesh (other)

Example: make a square
```
SwrgPoint<2> p(0,0);
SwrgPoint<2> q(1,1);
SwrgWafer2D cellComplex(p,q);
```

Field Server

Objects:
field, mesh, element

Operations:
conform, extract, mesh geometry, validate (interactive); evaluate; update

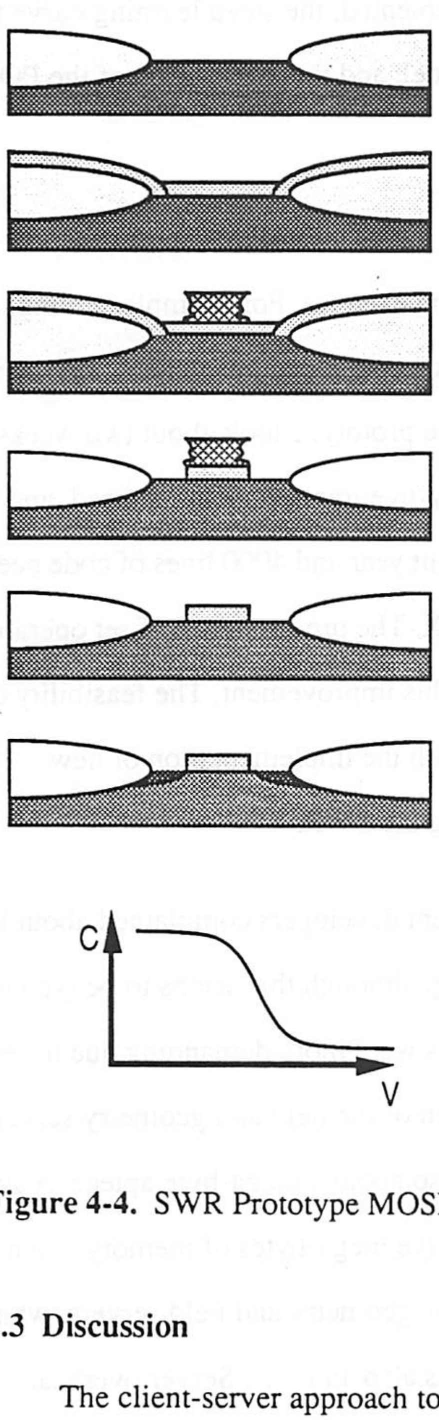
Example: mesh an existing geometry
```
f = Field.mesh_geometry(cell1, flag);
```

Example: evaluate function
```
f = Field<T>.evaluate(point)
```

Figure 4-3. Example SWR objects and operations [86].

4.3.2 Prototype

A prototype SWR standard was first demonstrated in August 1991. The demonstration simulated the construction of a MOSFET from the initial field oxidation through the source and drain formation sketched in Figure 4-4. The main components of the SWR server are the geometry server from IBM [88] and the field server written by Giles, Chin, and Law [77]. Universities, companies, and vendors contributed the nine clients listed in Figure 4-4. Simple new clients used the SWR to test it out as a emerging development standard, while larger existing clients used the SWR as an integration standard.



Process Step	Client Simulator
Initial Profile	SUPREM IV
Deposit Polysilicon Tool	TI Deposition
Gate Lithography	SAMPLE
Etch Gate	Intel Etching Tool SPEEDIE
Strip Photoresist	TI Stripping Tool
Create S/D Junction	SUPREM IV
Simulate Device	PISCES MINIMOS

Figure 4-4. SWR Prototype MOSFET process flow summary.

4.3.3 Discussion

The client-server approach to wafer representation is new to TCAD, as is the use of the C++ language and an object-oriented programming style. The emphasis on software engineering techniques from the computer-science field is a radical departure for physicists and electrical engineers who are still used to the "old programming ways." Some typical concerns expressed are the large existing base of FORTRAN and C programs that must be wrapped or re-implemented, the steep learning curve for C++, the applicability of the client-server model, and the assumption of the POSIX operating environment.

Both the technical advantages and the difficulties of this approach have been better understood through the SWR prototyping exercise. For example, adding tools to the framework was a much easier task than expected, once one understood the server. Wang reported that wrapping SAMPLE for the prototype took about two weeks and 800 lines of code [90]. This is factor of twenty-five improvement in speed, and factor of five improvement in size over the one student year and 4000 lines of code needed to develop the stitch-back routines in SIMPL-IPX. The pre-existence of set operations in the geometry server was the main reason for this improvement. The feasibility of a development standard has also been shown with the implementation of new elementary deposition and etching codes using the SWR.

Several concerns were also raised. Client developers complained about lack of detailed documentation during the prototyping, although this seems to be typical of any prototyping project. Machine requirements were more demanding due to server and wrapper size. The binary code size for each of the field and geometry servers was about one mega-byte. Client wrappers were also about a mega-byte apiece. A system with ten clients therefore requires an extra twelve mega-bytes of memory when integrated. The complexity of implementing the geometry and field servers, which are 68,000 and 10,000 lines of code respectively, is also an issue. Server overhead does not seem to be a problem though. Field server calls for point examination is about a factor of two slower than direct C++ function calls, but this is still only ten percent of the time used in function evaluations such as sqrt(x^2+y^2) [77]. In general, these issues are not enough to offset the advantages of the SWR approach.

From a political point of view, the SWR has been somewhat successful at working towards a standard. The group has more active industrial participation than the PIF, which was mainly a university effort. The twenty active SWR members are split evenly between universities and companies. A second difference is the strong participation of computer scientists and mathematicians who have assisted with software engineering and geometry issues, and who have pushed for approaches such as the client-server model.

The SWR prototype was the committee's key accomplishment not only because of the technical understanding gained, but also for political reasons. First, it proved that the different groups within the TCAD community could work together on a joint project. Thirteen organizations participated in the prototype. Second, it generated interest, confidence, and credibility. The initial demonstration was seen by over fifty people, and subsequent showings at member companies, Sematech, and the International Electron Devices Meeting (IEDM) have been well attended by managers, funding agency representatives, and the casual observer. Third, the prototype was a concrete project with a set goal and deadline for the group. Standard specifics had to be agreed upon in the documents, and had to be implementable within the allotted period. On the downside, the time required to coordinate the effort was considerable. Code integration and coupling tools between competitive companies required lots of legal paperwork. Even with these barriers, the prototype was by far the most productive three months spent by the SWR working group.

During the tenth meeting of the SWR in February 1992, the architecture and programming interface documents were updated to Version 0.9C.[1] It is the hope that Version 1.0 of both documents will represent the state of the SWR world after the prototype experience, will be geared as an integration framework, and can be used to support simple new tools. All concepts described in this version have either been exercised in the prototype or are those that the committee feels are well understood, useful, and easily implementable extensions to the current framework. Ironically, the success of the prototype has raised everyone's expectations, thus making it more difficult to formulate future plans, e.g., how to commercialize the software, whether to work on integration or development issues, and most importantly, who will do the work.

4.4 Information Modeling

An underlying information model for wafer data is being developed in parallel with the PIF and SWR efforts. The EDIF Device Modeling and Verification subcommittee, which became a CFI working group in 1991, has proposed an information model for the PIF in 1989 and more recently for the SWR [91]. Figure 4-5 shows the proposed device and geometry views in EXPRESS-G graphical notation [92]. Device information is separated into segment and boundary parts. Segments make up the layer materials and geometric information such as points, lines, and faces, as well as mesh and field attribute information. Boundaries specify boundary conditions for the device.

1. After voting approval by the TCAD Framework Group, this version will be designated Version 1.0.

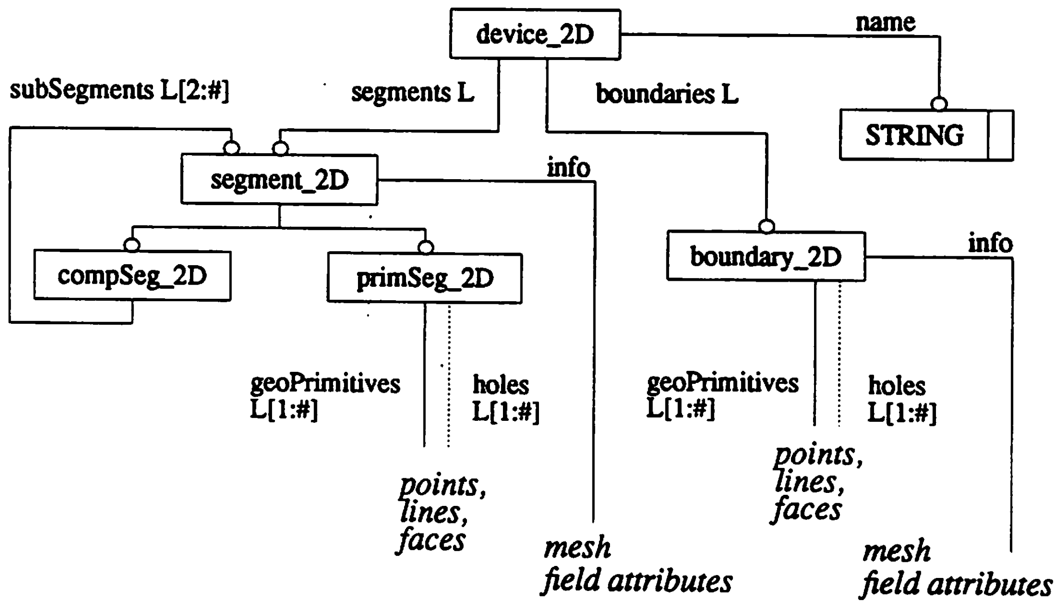


Figure 4-5. Top level DM&V information model for a two-dimensional device [91]. Segments and boundaries contain geometries, e.g., points, lines, and faces, and fields, e.g., mesh and field attributes.

The information model is the most basic way of expressing wafer data information, and may be the starting point of agreement between all groups. The DM&V group was chosen to develop the information model because of their modeling expertise and interest. An initial review of the model by members from the SWR and VISTA groups shows no major inconsistencies with the SWR or VISTA architectures [93]. A more complete review is currently underway by the DM&V and SWR groups.

4.5 Cost Analysis

In this section, the cost of developing and using a standard is compared to the cost of not using a standard. Wafer standardization can progress along two paths: agreement on a standard for integration systems; and continued research in the area of wafer representation for development systems. These two approaches to standards are addressed separately.

4.5.1 Integration Standard

The main advantages of an integration standard are: (1) a reduction of the time required to integrate new tools, (2) the ability to compare different tools that perform similar functions, e.g., judging the results of two lithography simulators; (3) the ability to share general utilities such as visualization tools; and (4) increased opportunities for smaller research groups to contribute equally in the TCAD arena. Assuming ten significantly different new models are developed by research groups each year worldwide, two-man months of effort are required per module for understanding and integration, and one month is needed to make the tool user-friendly using current techniques. Therefore, each integration organization (company) would spend approximately three person-years on (1) and (2).[1] The cost for (3) is also included in the three-months per tool estimate, since most sites already have a set of common utilities for their system. The cost for (4) is unknown but TCAD groups without integration projects, such as those at the University of Alberta and the University of Wisconsin, would benefit. If we assume there are ten integration sites worldwide then the total yearly effort spent on TCAD integration without standards is thirty person-years. This estimate does not include the unknown cost of (4).

1. Three person years is often more than a company can afford. Because of this, often only a subset of these models are integrated. The cost of the technology that is not transferred is unknown.

The cost to produce a document specifying the integration standard based on the PIF and SWR work is a two person-year effort (four people, each working half time). The price of developing an implementation depends on the amount of previous work that is both applicable and available, but can be conservatively placed at ten person years. Thus, the total cost is approximately twelve person-years for documentation and implementation of an integration standard. Taking figures from the SWR prototype, integration of a new tool into the framework should take no more than a week if the standard is used, but the time necessary to understand how the tool works remains constant. Thus, the cost to add a new tool is roughly one month and one week. The effort for ten tools at ten sites would be about ten person-years or less, depending on whether there is integration sharing among sites. The tool integration burden can also be shifted from the development group to the users using this approach.

An integration standard is ripe for commercialization or cooperative development because there is a savings of at least a full-time person per organization per year, and the system could be developed with less effort that the savings from the first year.

4.5.2 Development Standard

In SAMPLE and SUPREM, less than one-third of the code implements the process models; the other two-thirds is for model support, such as input parsing and simulator output. This is thought to be typical for most simulators. A development standard can conservatively reduce the coding time by one-half, substantially increasing the productivity of simulator developers and reducting the duplication of work. For instance, half of the graduate student coding effort would be saved. A development standard might also address how the scope of TCAD tools can be used through enhanced controllers such as task level managers.

There are more unknowns in costing out a development standard because it is further in the future than an integration standard. A paper definition is estimated to take between two and four person-years. Since this is still an active research area, it is hard to estimate the cost of an implementation, but it is on the order of tens of person-years. A TCAD project of this scale is probably not manageable by any individual organization, and will therefore require a joint effort between universities, companies, and vendors.

Some overlap between the integration and development standard exists. An integration standard would allow some degree of utility sharing, making it somewhat easier for developers of new tools. The development standard, on the other hand, is a superset of the integration standard, since tools in the integration framework can also act as models in the development framework.

4.6 Outstanding Issues

Although we have come a long way in defining wafer standards, many outstanding issues, both technical and political, remain. The way that these issues are resolved will greatly affect the future of the standards work in general, and the SWR working group in particular.

4.6.1 Technical Issues

Remaining technical issues can be grouped into issues created by the decisions made, issues that have not yet been addressed, and future issues that are still unknowns.

For each choice made in a definition, many other choices are sacrificed. For example, the major architectural decisions made in the SWR are the use of the client-server model, the definition of the representation through the C++ language using an object-oriented paradigm, the separation of the representation into geometry and field components, and the implementation of a geometry interface based on set operations. These decisions were made by the SWR group after careful consideration, but other choices are also valid. The group could have selected a layered model instead of the client-server model, a FORTRAN instead of C++ specification, unified instead of separated geometry and field data, and moving boundaries rather than set operations. There are reasonable arguments for each of these approaches, and the best choice is still being debated.

Many issues related to the wafer representations must also be addressed before a true standard exists, for example, error handling, data persistency, definition for multiple programming language implementations, and intertool communications mechanisms. Although there has been some work within the CFI group to study these issues, they have not been specifically addressed in the SWR.

Finally, there are still many remaining unknowns. Development frameworks are in their infancy. As we continue to work towards them, they will no doubt bring up more issues. Three-dimensional process and device simulation needs will place additional demands on framework representation and performance requirements. The magnitude of the legacy system problem, i.e., the headache of maintaining an old monolithic system or transferring all the tools to a new framework, must also be considered.

4.6.2 Political Issues

The main political barriers are the cost of standards and who will pay for them, the issue of committee membership by those in Europe and Asia (when funding is mainly through US sources), and the complications of implementing standards by committee.

The cost of standards development to date has been quite high and almost completely supported through volunteer time. Future costs must not only be reduced, but also charged to a customer. The SWR group has so far held ten meetings, with each meeting averaging of two days in duration.Writing the architecture and programming interface documents have consumed one person-year, the prototype integration has taken two person-years of effort, and chairing the group has been a half time job for over a year. It is unlikely that these volunteers will agree to work for free for much longer. New developers and new sources for dollars must be found. The large number of SWR meetings is also an issue, as travel is always a sore spot with companies and funding agencies.

Participation by the Europeans and the vendors is also key. It is imperative to coordinate efforts with TCAD groups and standards projects within Europe, e.g., TU-Vienna, ETH, DASSI, and the STORM project, and vendors such as Dawn Technologies, Silvaco, and TMA. There seems to be a general lack of interest among the Europeans, who seem to view the SWR effort as a waste of time. A concrete proposal that describes a coordinated effort would be a good first step. This would also tie in the vendor community, which has expressed an interest in seeing standards documents produced from the SWR work.

Finally, the extent of the standard needs to be clarified on whether it includes only documentation, or documentation with an implementation. The argument for the latter is that a standard must be widely available and shown to be useful in order for it to be accepted. But a standards implementation limits commercial possibilities for vendors, brings up many code ownership issues, and makes the coordination effort between members more complex and more costly.

4.7 Moving Forward with the Standards Process

The number of issues raised in Section 4.6 is large but not overwhelming. From the cost analysis of Section 4.5, standards for wafer integration and development are definitely worth pursuing. Given the current state of the SWR, this subsection provides two suggestions for moving forward in the standards process: the division of standards work into integration and development thrusts, and technically-oriented meetings and project driven standards. A commercialization plan for an integration standard is presented to identify some more specific suggestions for a successful cooperative arrangement. A research plan for SWR Version 2.0 (the development framework) should also be crafted to meet the needs of the TCAD framework research community, but is left to future work.

Suggestion #1: Let the researchers do research, and let the implementors implement.

People tend to do better at things that they like to do, rather than are forced to do. Given that the SWR group is mostly university and industrial researchers, and the fact that Version 1.0 of the SWR is almost complete, it is unlikely that they will be willing to devote any more time to an integration standard. It is better to unleash them on the specification and prototyping of a development framework, and to put them in an advisory role for the integration standard. Companies who show an interest in the representation for integration should join forces with universities who have related implementations, e.g., the PIF people, to formally develop a standard using new blood.[1] The effort spent on pursuing an integration standard is worthwhile for the reasons cited in Section 4.1, even if only supported by a handful of companies.

Suggestion #2: Strive for technically oriented meetings and project driven standards.

The most productive SWR meetings have been the ones where political issues are either not discussed, or discussed only briefly. Unfortunately, roughly a third of SWR meeting time has been spent on politics, bureaucracy, or rehashing previous decisions.[2] In the future, political issues at technical meetings should be delayed until the last hour of the last day. In addition, many discussions require only a small group of people. Meeting in task groups of three or four appears to be effective means of making specific decisions that can be recommended to the committee at large. Projects are also extremely useful for motivating people and achieving goals. The SWR prototype project, and its deadline, encouraged the group to make decisions much more readily than multiple discussions without a practical application. The CFI organization has also recognized the importance projects and is using them to drive the standards process for design CAD.

1. The keyword here is *develop*. The integration standard is a development, and not a research project. Thus, the burden of development should mostly rest on industry, with universities providing help to the transfer of technology in their respective implementations by students and the undergraduate or master's level.

2. The committee chair assumes only partial responsibility. SWR members like to talk a lot too!

4.8 An Implementation Plan for SWR Version 1.0

Rather than specify specific recommendations for solving standards issues such as implementation personnel, code ownership, and funding, these elements are presented within an overall proposal for implementing Version 1.0 of the SWR. This framework would be useful for tool integration and simple tool development. Such a proposal has long been needed in guiding the future role of the SWR. Only a summary and key recommendations of this project-centered proposal are presented here. A more detailed proposal has been distributed to the SWR group at large for comment and consideration [94].

4.8.1 Project Goals

The key goals of this project are the commercialization of the SWR server and its demonstrated use by four key TCAD client types: above-silicon (topography) process simulators, below-silicon (thermal) process simulators, device simulators, and resistance/capacitance parasitic extractors (see Figure 4-6). A scenario is proposed where the SWR servers, based on their prototypes, are commercialized by vendors and SWR client wrappers for proprietary tools are implemented by companies. These wrappers would then be shared with the universities to assist them in wrapping their tools. This should not be large burden on universities because most tools of the same type are similar in nature. Figure 4-6 identifies some possible industry and university tools that could be wrapped in this project.

4.8.2 Benefits

Vendors would benefit because it would be easier to integrate their tools, and there would potentially be a new market based on SWR compliant applications. Companies would be able to buy and use a supported server. Each company would

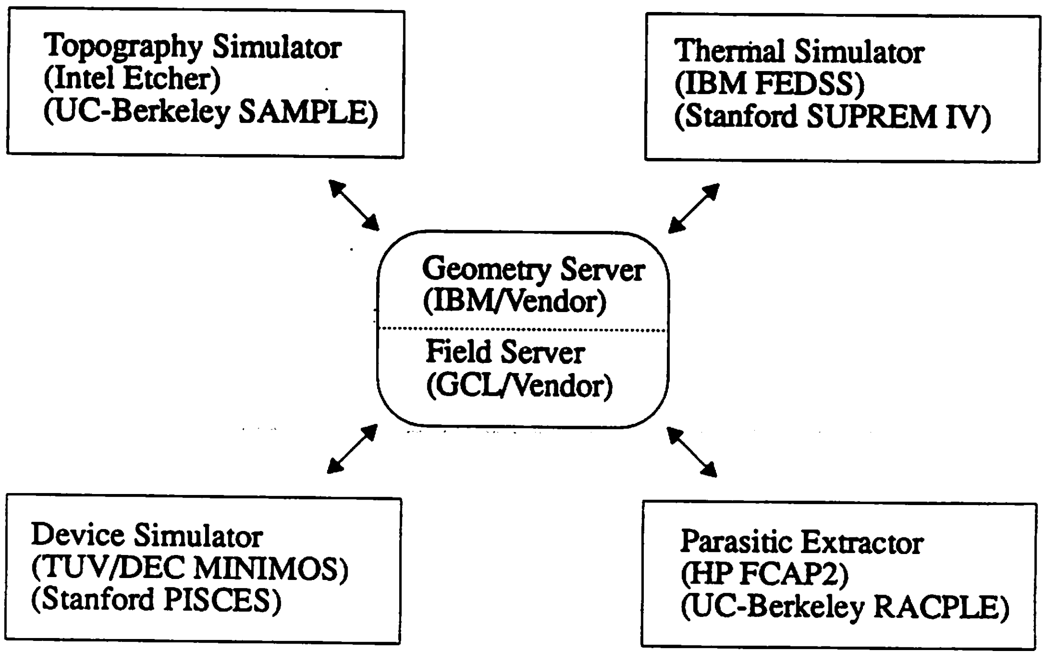


Figure 4-6. The SWR V1.0 would demonstrate robust servers working with the four key client types. GCL stands for the Giles/Chin/Law prototype implementation.

have their tool connected to the server at the end of the project, and would be able to add more tools easily by modifying wrappers written by other participating companies. Universities would have a skeleton server to use, and would no longer have to worry about technology transfer issues to companies.

4.8.3 Costs and Funding

A very rough guesstimate summarizing the work involved for this project is listed in Table 4-1. The basic idea is as follows: companies would be expected to contribute mostly person-power; consortiums such as Sematech (or possibly a group of companies) would be asked to provide integration facilities and seed money to vendors for server development; and universities would receive some travel and student assistance for their contributions.

Component	Cost (in Person Years)	Who	Financial Support
Field Server	3	Vendors	Consortium/Sales
Geometry Server	3	Vendors	Consortium/Sales
Client Wrappers (Industry)	0.5 for each client	Companies	Companies
Client Wrappers (University)	0.2 for each client	Universities	Consortium
Project Management	0.5	Consortium	Consortium
Facilities (Integration Site)	--	Consortium	Consortium
Totals	9.3	All	Companies/Consortium

Table 4-1. Guesstimate of component costs for the SWR V1.0 implementation project.

4.8.4 Ownership

The paper definition of the standards should remain in the public-domain to encourage worldwide participation. On the other hand, it is only fair to allow parties who implement the code to retain ownership. Servers are owned and supported by the vendors, with special discounts available to the participants. Client wrappers are to be shared among participants (so that they can more easily wrap their own proprietary code). Binary versions of the clients may also be made available at the integration site. Universities should be given versions of the SWR server at minimal or no cost. Arrangements could be made release a reduced version of the server to research partners, e.g., universities in Europe.

4.9 Conclusions

Wafer standardization is exciting yet frustrating. The whole committee smiles when agreements are made. The committee frowns when bureaucracy or differences get in the way. The PIF work addresses mainly the integration standard through file format and functional interfaces. More recently, the SWR working group has been working on a development and integration standard using new concepts such as the client-server model and the object-oriented programming style. This is a multi-disciplinary group with talents in physics, electrical engineering, and computer science. A prototype of the SWR has provided many useful comments, and more prototype projects should be pursued in the research of development frameworks, or SWR Version 2.0. Integration frameworks, on the other hand, are ripe for commercialization. In spite of the technical and political issues that remain, standards can reduce TCAD integration and development time by a factor of three and should be pursued. A plan for implementing Version 1.0 of the SWR that targets specific goals and benefits all participants has been proposed as a starting point for discussion of future efforts.

5
The Binary PIF

The "Binary PIF" (BPIF) defines a set of objects and a library of functions that describe an intertool version of the profile interchange format (PIF), a representation for semiconductor wafer information. The "BPIF Toolkit" function library enables technology CAD (TCAD) tools, such as process and device simulators, to access PIF data through a common interface, thus facilitating tool integration and encouraging utility sharing. This approach defines a toolkit that is easy to use, and which meets the tool integration and performance needs within a TCAD environment. The chosen environment, PROSE, uses the OCT/VEM/RPC CAD framework, which gives BPIF a convenient location for storing persistent data through the OCT database, and access to mask information.

The definition and implementation of BPIF began in 1988, a year before the Semiconductor Wafer Representation (SWR) working group, described in Section 4.3, was founded. Therefore, it played a role in helping formulate the initial standard. During its mid-life, BPIF development was carried out concurrently with the SWR work. One of its main tasks then was to uncover issues in wafer representation by carrying out experiments on next generation communications of cross-sections. The BPIF has since been improved on to address these issues through a new BPIF++ toolkit that uses an object-oriented C++ interface.

Sections 5.1 - 5.7 describe an experimental BPIF Toolkit, based on C and the OCT data manager. These sections were published as an article in IEEE Trans. on Computer-Aided Design.[1] Examples are shown of how TCAD tools may use the toolkit to access wafer data, such as grids and geometries, within a graph of PIF objects. A BPIF package supporting the use of the intertool PIF within the PROSE environment, which includes a link to the PIF intersite file format, is then presented. Performance measurements for BPIF are found to be fast enough for general use. Comparisons are made with the PIF/Gestalt implementation to assist in the standardization of the intertool PIF.

Sections 5.8 - 5.10 contain recent work on BPIF++. These include the advantages and implementation of BPIF++, the link to the Berkeley Topography Utilities (BTU), and the interoperability potential of BPIF++. Chapter conclusions are drawn in Section 5.11.

5.1 BPIF Object Definition

Conceptually, BPIF objects are designed to be the same as those defined in the intersite or ASCII file version of the PIF [53]. There are three object types - geometries, grids, and attributes. A hierarchical approach is used to represent the *geometries* (boundaries) of the cross-section [83]. In this scheme, the most basic geometric building block is the point. Lines are constructed from a set of points, and faces are constructed from sets of lines, as shown in Figure 5-1. Fields are represented on *grids* (or meshes). Grids are constructed similar to geometries by specifying a set of points and then connecting the points to form elements, or by using pre-defined grid types. Doping impurities and other *attributes* can be associated with the grid. Certain attributes may also be attached to geometries; for example, the association of an electrode name to a material boundary.

1. A. Wong, et. al, "The Intertool Profile Interchange Format: A Technology CAD Environment Approach," *IEEE Trans. on Computer-Aided Design*, September 1991 [58].

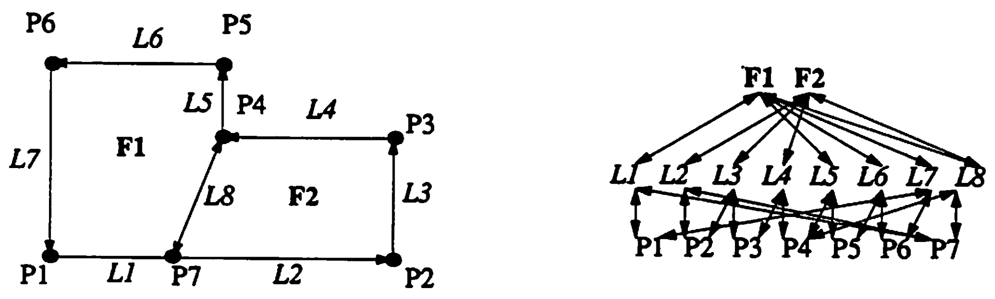


Figure 5-1. Constructing a geometry hierarchically.

Several changes and additions were made to the initial BPIF definition, based on findings from exercising BPIF. One change involves representing grid node indices and elements as large arrays, as opposed to individual objects. Access time for large numbers of individual node and element objects was found to be too slow in applications involving grids with many points. Storing nodes and elements as large, array objects increases access efficiency. Specific objects for rectangular, triangular, and tensor-product grids are also supported due to efficiency concerns. Additional objects have been added to handle mask and cut-line information. The "cut" object uses a PIF geometry to represent a cut through a mask or mask set; the cut may be a point, line, or face. The "mask" object provides a reference, in the form of a character string, to external layout information. Finally, attribute handling has been simplified. Two pre-defined attributes, "name" and "impurity", are tested in the current prototype. A "stranger" object has been included to allow for general attributes and other user-definable data. A summary of BPIF objects may be found in Table 5-1.

BPIF objects are described by C-language structures. The generic object contains a union of specific PIF object types, and the field `pifObjectType` determines the type of the object. Each specific type is in turn defined by its own C-structure. For example, Figure 5-2 shows the structures of both the generic object and

Type	Objects
Organizational	File, Snapshot
Geometry	Geometry, Point, Line, Face Solid, Segment, Boundary
Grid	Grid, GridNodeArray, GridPointArray
Layout Interface	Mask, Cut
Pre-Defined Attributes	NameAttribute, ImpurityAttribute
Other	Stranger

Table 5-1. BPIF objects used by the toolkit.

the specific "point" object type. Values of the object are accessed through the structure's data members. Since objects must be explicitly stored and retrieved in the database, there are generally two copies of each object: one in memory, and another in persistent storage. To create a new object in the database, the programmer first creates and initializes the in-memory copy of the structure, and then calls the `pifCreate` function with pointers to the object and its parent as arguments, e.g., `pifCreate(*parent, *point)`. To update an in-memory copy of the object from the database, a call is made with `pifUpdate`, e.g., `pifUpdate(*point)`. The behavior of these two functions is shown pictorially in Figure 5-3.

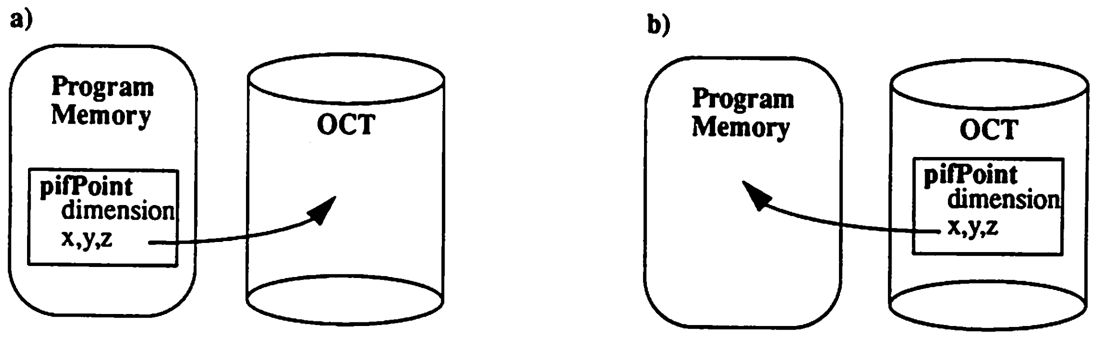


Figure 5-3. Use of (a) pifCreate and (b) pifUpdate to create and retrieve objects in the OCT database.

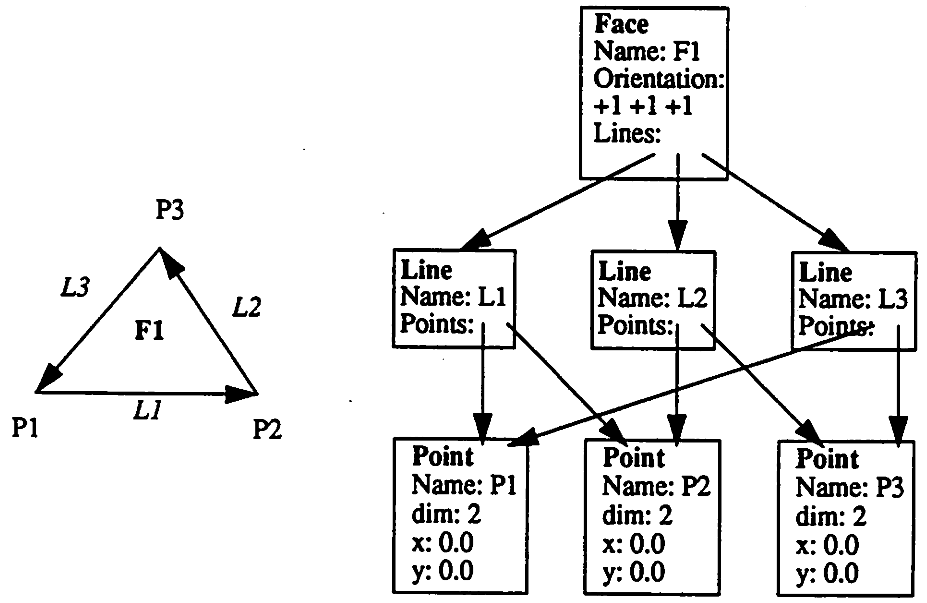
```
typedef int pifObjectType;
struct pifObject {          /* PIF_OBJECT */
pifObjectType type;
     union {
              pifFile file;
              pifSnapshot snapshot;
              pifGrid grid;
              pifGeometry geometry;
              pifPoint point;
              ...
              pifStranger stranger;
     } pif;
};

struct pifPoint {
     pifDimension dimension;
     pifCoord x,y,z;
};
```

Figure 5-2. C-Structures for PIF_POINT and PIF_OBJECT.

Relationships between BPIF object instances are governed by a set of attachment guidelines, or *PIF Policy*. An attachment is a bi-directional connection between two objects that is maintained by BPIF; all of the required attachments between instances are established upon object creation. Optional attachments, such as that associating attribute information with part of the geometry, may be established with the `pifAttach` function. Currently, no automatic checks are made on the legality of an attachment. This has the advantage of allowing new relationship rules to be explored with relative ease, but places the responsibility of adhering to PIF policy on the programmer.

Large collections of PIF objects are not organized in a static file as in the ASCII PIF, but rather in a dynamic graph of instances of PIF objects. A fragment of a PIF *instance graph* detailing the structure of a triangular face is shown in Figure 5-4. A more general instance graph is shown in 5-5. Application programs generate, manipulate, or access these instance graphs to: capture a wafer structure, to change that structure by simulation or other means, and to examine the result. The instance graph is an informal depiction of the objects and the relationships between them; programmers generally have these types of graphs in mind when using a PIF Toolkit. The purpose of the PIF toolkit is to enable application programs to create and directly manipulate these instance graphs. Consistency of the attachments in the object graph is left to the programmer (which is defined by the PIF Policy).



P3
L3
L2
F1
P1
L1
P2
Face Name: F1 Orientation: +1 +1 +1 Lines:
Line Name: L1 Points:
Line Name: L2 Points:
Line Name: L3 Points:
Point Name: P1 dim: 2 x: 0.0 y: 0.0
Point Name: P2 dim: 2 x: 0.0 y: 0.0
Point Name: P3 dim: 2 x: 0.0 y: 0.0

Figure 5-4. A graph of PIF object instances for a triangular face.

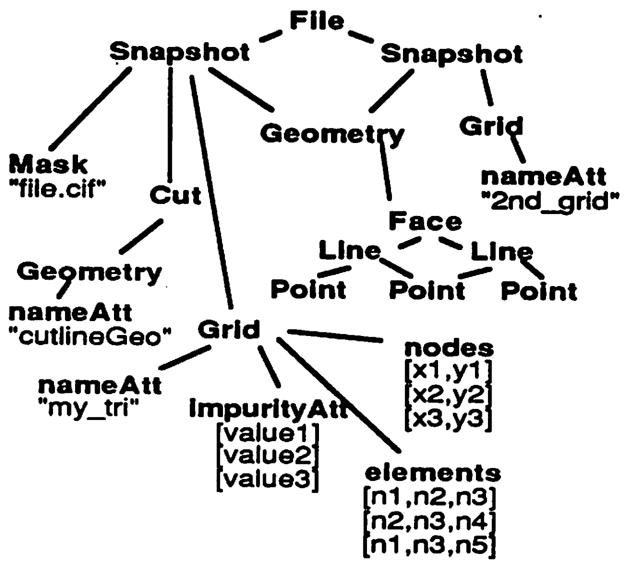


Figure 5-5. An example PIF object instance graph.

5.2 BPIF Toolkit Functions

The BPIF toolkit provides a database independent set of functions that allows programmers to access, modify, and store objects. BPIF functions are separated into basic and extended toolkits. The basic toolkit is a minimal set of low-level functions needed to manipulate the instance graph, as is used mainly by programmers. The extended toolkit, on the other hand, provides higher-level functions that generally operate on many objects and is meant for both developers and programmers. In Figure 5-6, for instance, the function `pifCreatePoint` is a low-level function that creates a point, whereas the `pifDeleteGrid` operates on multiple objects related to a grid, and is therefore a high-level function.

5.2.1 Basic Toolkit

The set of basic PIF toolkit functions is separated into five groups. These functions, summarized in Table 5-2, are described below.

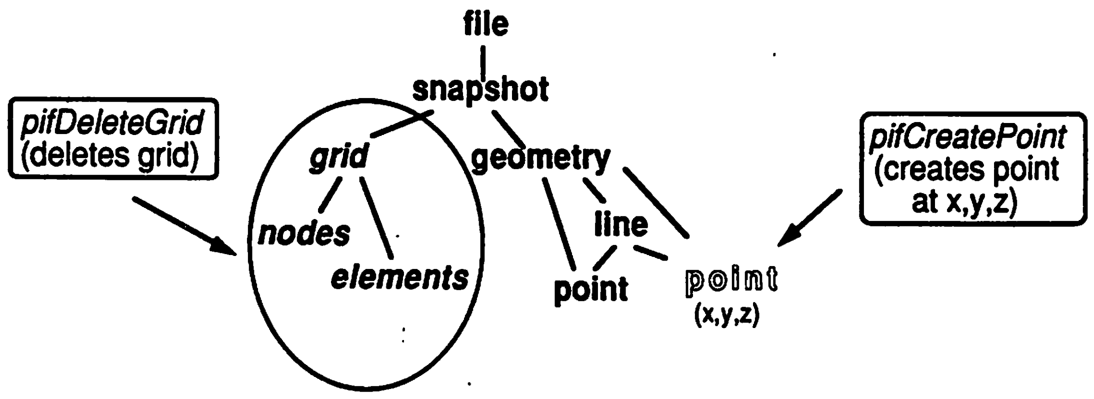


Figure 5-6. Basic and extended toolkit function examples.

The "file" functions open and close a PIF file. This area delimits the working name and address space for a set of PIF objects, and corresponds to a file in the ASCII PIF.

The "basic" functions create, retrieve, and modify objects. Most of these functions have an equivalent OCT operation; details may be found in the OCT User's Guide [95].

For walking around and retrieving instances in a graph, the BPIF toolkit offers a set of instance generator functions in the "traversal" group. The function `pifInitGenerator(*object, pifObjectMask)` creates a generator object that contains all directly related instances specified by a PIF *object mask.* The object mask allows the programmer to indicate the types of objects that are to be included in the resulting generator. For instance, one can generate all combinations of points, lines, and faces from a geometry with the function call `pifInitGenerator(*geometry, PIF_POINT_MASK | PIF_LINE_MASK | PIF_FACE_MASK)`. This generator can then be used by `pifGenerate` to retrieve each of these objects. In addition to single

Group	Name	Function
file	pifOpenFile pifCloseFile	opens an instance file saves an instance file
basic	pifCopy pifCreate pifAttach pifDetach pifUpdate	copies an object creates an object attaches two objects detaches an object reads object from database
traversal	pifInitGenerator pifGenerate pifGenNumber pifReverseGenerator	finds objects based on criteria gets next object gets number of objects in generator reverses the generator order
attribute	pifGetByAttributeName pifGetByAttributeImpurity	gets an object by name gets an object by its impurity type
other	pifError pifPrintObj pifVersion	error handler "pretty prints" object returns BPIF version number
Other Types	**Name**	**Function**
deep	pifInitGeneratorDeep	multi-level generator
recursive	pifCopyRecursive pifDeleteRecursive	recursively copy object tree recursively delete object tree
extended	pifCreate2DSubstrate pif2DAddTopLayer	create starting substrate add top layer to profile

Table 5-2. Summary of functions in the BPIF toolkit.

level "shallow" generator functions, multiple level generators are also available, and are characterized by the "deep" suffix, e.g., `pifInitGeneratorDeep`. For example, one could use a deep generator to find all the points from the face in Figure 5-4.

The "other" group contains functions for debugging, error handling, and other miscellaneous tasks. Examples of these are `pifPrintObj` to "pretty print" the contents of an object and `pifVersion`, which returns the current version of the toolkit. Exception handling is through a common error routine `pifError`. All functions return `PIF_STATUS`, which evaluates to either `PIF_OK or PIF_NOT_OK`.

Functions in the "attribute" group have been implemented for frequently used attributes such as "name" and "impurity". These are pre-defined as special PIF objects in the BPIF object model. For example, there is often a need to get an object by its name or its impurity type. The functions `pifGetByNameAttribute` and `pifGetByImpurity` perform this task.

5.2.2 Extended Toolkit

Commonly used, higher-level functions, which operate on multiple objects, are contained in an extended PIF toolkit. These functions are subroutines built on the basic toolkit functions. For example, two extended toolkit functions that have been implemented are `pifCreate2DSubstrate` and `pifAddTopLayer`. `PifCreate2DSubstrate` creates a starting snapshot instance graph of a two-dimensional substrate with a thickness and doping concentration. This snapshot contains the necessary sub-graphs of the geometry, grid, and attribute instances. `PifAddTopLayer` modifies the snapshot by depositing a new material with a finite, positive thickness onto an existing profile.

5.3 Layering on the OCT Database

The PIF toolkit is intended to be layered on an object-oriented database in a manner transparent to the toolkit user. A standard PIF toolkit should allow implementation across a wide range of programming languages and database types, although this is limited by practical considerations, e.g., one rarely finds a TCAD tool that uses COBOL. The implementation of the BPIF toolkit, which is similar to other PIF toolkit definitions such as [79], was found to be only a moderately difficult task.

The BPIF toolkit implementation is built on OCT, a data manager that uses object-oriented concepts. At the lowest level, all PIF objects are cast into OCT "stranger" objects, with an automatic translation between the PIF object and the OCT stranger object occurring each time an instance is stored into or retrieved from the database. For instance, in the `pifCreate` function, the data in the PIF object union is first copied into an OCT stranger object, and this resulting object is stored into the database. Objects are retrieved by `pifUpdate` through the reverse procedure. Although these implementation details are transparent to the programmer, they do affect toolkit performance, as discussed in Section 5.6. This translation is eliminated in the BPIF++ implementation described in Section 5.8

5.4 Illustrative Examples

The BPIF extended toolkit provides a mechanism that makes it easy for applications to access PIF data. Programming time is saved, since, unlike the ASCII PIF, there is no need to parse the input file. Instead, programs link in the BPIF library and make function calls with high-level objects. Suppose a programmer has a deposition routine that only needs to work with the top-string of a profile. Figure 5-7 shows how the extended toolkit can be used to extract this string from the current snapshot, allow the programmer to manipulate the string, and then write back the resulting profile into the database.

Programmers may also use lower-level functions in the basic toolkit to access PIF data. For example, assume a programmer would like to construct a face from a list of points, producing an instance graph such as the one shown in Figure 5-4. The function `makeFaceFromPoints`, shown in Figure 5-8, implements this task. Line 9 uses the function `makeLinesFromPoints` to create an array of lines that make up the face. Line 10 allocates memory for a PIF_FACE object and sets the `face` pointer to it.

```
modifyTopString(file, snapshot, newString)

pifObject *file, *snapshot;  /* PIF file and snapshot */
stringStructure newString;   /* structure for new string */
{
 pifObject *topString;       /* holder for profile top string */

/* pifGetTopString is a function which retrieves the top string of
        a profile within a given snapshot */
pifGetTopString(file, snapshot, topString);

/*******************************************************
programmer's routine to the modify top string goes here
*******************************************************/

pifPutTopString(file, snapshot, topString); /* replace string */
};
```

Figure 5-7. Simple example showing how the BPIF extended toolkit may be used by an application.

The fields of the PIF_FACE object are then set: `face->pif.face.array` is a pointer to the array of lines, and `face->pif.face.directory` points to the orientation of each line, which is assumed to be 1. Each field must contain a valid value. For example, `array` may only contain a pointer to an array of points, and `direction` must be set to -1 or 1. A list of object types and their fields can be found in [96]. Recall from the discussion in Figure 5-3 that the PIF object in memory is not automatically stored in OCT, but must be explicitly saved. This is done in line 16 with the `pifCreate` command, which takes pointers to the object's parent (the geometry) and the object (the face) as arguments. Finally, the result, a pointer to the new face, is returned to the calling procedure.

```
1 pifObject *makeFaceFromPoints(geo, pts, n)

2 /* This function creates a face from an array of points and
       stores them in the database. Lines which make up the face
       are also created in the process. The starting geometry and
       point array are passed in as "geo", "pts", and "n", the
       number of points. These points are assumed to be in
       counter-clockwise order around the face. This function
       returns a pointer to the new face. */

3 pifObject *geo, *pts;      /* pointers to geometry & points */
4 int n;                     /* # of points */
5 {
6 pifObject *lines[], *face;
7 int i;
8 /* makeLinesFromPoint creates a line from an array of points and
       stores them in the database */
9 lines = makeLinesFromPoints(geo, pts, n); /* create lines */
10 face = AllocPIFObject(1); /* allocate pifFace object */
11 face->type = PIF_FACE; /* face points to pifFace object */
12 face->pif.face.array = lines; /* create face from lines */
13 face->pif.face.direction = AllocPIFInt(n);
14 for (i=0; i<n; i++)       /* set line orientation */
15   face->pif.face.direction[i] = 1;
16 pifCreate(geo, face);
17 return(face);
18 };
```

Figure 5-8. Example function using the basic toolkit which creates a face from an array of points.

Figure 5-8 demonstrates how this function is used to create a triangle. Line 3 reserves program memory for an array of three points. Next, each of the required fields in points `p[0]` through `p[2]` are set. The `makeFaceFromPoints` function is called to create the face in line 9. For comparison, this example implemented with the PIF/

Gestalt toolkit using CommonLisp may be found in [79]. This function may be added to the BPIF toolkit if it is frequently used, since the extended toolkit is intended to be a growing library of "helper" routines contributed from BPIF programmers.

```
/* Example use of makeFaceFromPoints */
/* Assumes the geometry "geo" to which the face is attached
      already exists */
pifObject p[3];
p[0].type = PIF_POINT;     /* create first point in memory */
p[0].pif.point.dimension = TWO;
p[0].pif.point.x=0; p[0].pif.point.y=0;
pifCreate(file, &p[0]);    /* create point in OCT database */
p[1].type = PIF_POINT;
p[1].pif.point.dimension = TWO;
p[1].pif.point.x=10; p[1].pif.point.y=0;
pifCreate(file, &p[1]);
p[2].type = PIF_POINT;
p[2].pif.point.dimension = TWO;
p[2].pif.point.x=5; p[2].pif.point.y=5;
pifCreate(file, &p[2]);

makeFaceFromPoints(geo, p, 3); /* create face */
```

Figure 5-9. Creating a triangle using the `makeFaceFromPoints` function.

5.5 Using BPIF in PROSE

BPIF is used by PROSE to examine the practicality of the object model and toolkit functions within the setting of an environment for operating process and device simulators. A BPIF package, consisting of the toolkit and PIF utilities, is used to store, access, manipulate and view this data from a common database, as well as provide a link to the ASCII PIF. Figure 5-8 shows the scope of the BPIF package implementation. The shaded components are part of the OCT/VEM/RPC CAD framework [57]. Three process simulators - SIMPL, SAMPLE, and SUPREM, have been modified to use the toolkit. These tools were chosen because they represent different types of simulators (analytical, topography, and thermal), and therefore can be used as a proving ground for experiments on BPIF. Circuit designers would also be interested in the results since they the cross-sections are linked to the layout through SIMPL.

Several PIF utilities are also included with the package. A BPIF parser and writer links the ASCII and Intertool versions of the PIF. The writer is able to generate ASCII PIF, and the parser can read in PIF files written by the writer. Although the capabilities of this parser are limited, it does enable comparisons between the intertool and ASCII PIF. These results are given in Section 5.6. A second utility is the BPIF viewer translator. Cross-sections manipulated using BPIF are stored using BPIF Policy within OCT and are not directly displayable by the VEM layout editor. The viewer utility translates BPIF facets a form that VEM can display, i.e., OCT physical facets.

SIMPL-2 [6] has been modified to translate between its internal data structure and the intertool PIF through the BPIF toolkit. Using this interface, any SIMPL-2 process function can be applied to a BPIF cross-section. Two routines, `simpl2bpif`, and `bpif2simpl`, have been added to SIMPL to accomplish this. The former builds an internal data structure compatible with the PIF object model, and then makes BPIF calls to store them in OCT. The latter performs the inverse process. In addition, the commands within SIMPL have converted into OCT-RPC format and can be invoked using RPC menus within VEM. This version of SIMPL-2, named SIMPL-RPC, contains functionality similar to SIMPL-DIX with the additional ability to edit mask layout and cross-section information. SIMPL-RPC is the interactive component of the general PROSE manager described in Chapter 8. For each step, the current state of the profile is read from the OCT database into SIMPL-RPC using the toolkit. After completion of the process step, the resulting cross-section is written back into OCT, translated by the viewer routine, and automatically displayed by VEM. Since a common database is used, both layout and cross-section geometries can be displayed simultaneously in the VEM editor.

The BPIF interface to SAMPLE uses the wrapper technique. Instead of modifying the SAMPLE code, a translator operates between BPIF and the SAMPLE output file. This has the advantage of leaving the SAMPLE binary intact, but the disadvantage of requiring that all necessary data is stored in the savefile format. This is the case for SAMPLE's string-based algorithms, e.g., deposition, etching, and lithography. The `sample2bpif` and `bpif2sample` routines allow these algorithms to be accessible from within PROSE. For example, Figure 5-10 shows a photoresist profile generated by SAMPLE, stored in BPIF and viewed in VEM.

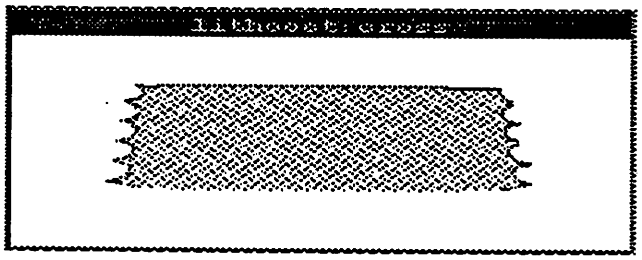


Figure 5-10. BPIF photoresist profile generated by SAMPLE.

The BPIF interface to SUPREM IV also uses a wrapper. In this case, it is the grids rather than strings that need to be translated between the two formats. The BPIF reads and stores these grids using the `PIF_GRID` attribute. Since VEM is string-based, it can not be used to display raw BPIF grid information, but there are two other ways that this can be viewed. String contours can be generated and displayed in VEM, or a separate simple grid-viewer based on the SIMPL-DIX graphics routines can be used.

5.6 Performance

Measurements are taken on a DECstation 3100 workstation to determine the relative data access and storage performance of the ASCII versus the Binary versions of the PIF. The read and write access times, and storage sizes of three different types of profile objects were tested: 10,000 point objects; a single object containing an array of 10,000 impurity values; and a DRAM bit-cell whose cross-section is shown in Figure 5-11. Results are summarized in Table 5-5.

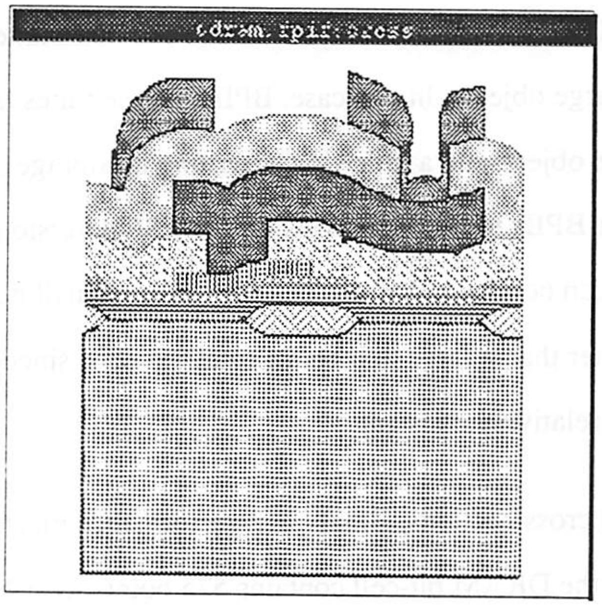


Figure 5-11. DRAM bit cell used in the performance measurements.

	10,000 Points			10,000 Value Array			DRAM Bit Cell		
Access Type	Read	Write	Size	Read	Write	Size	Read	Write	Size
ASCII PIF	3	1	400	1.5	0.5	140	<1	<1	43
Binary PIF	6	7	469	0.6	0.7	80	<1	<1	51

Table 5-3. ASCII and Binary PIF performance numbers. All times are in seconds, and sizes are in kilobytes (Kb).

The 10,000 PIF point test is used to determine raw object storage and retrieval performance. The ASCII PIF is two times faster than BPIF for reading in an object, and seven times faster in writing and object. This difference in performance is due to the extra overhead imposed by BPIF on any object that is stored or access by the program. The storage requirements of both are comparable at forty and forty-seven bytes per point.

The 10,000 value impurity array test is used to determine speed and size performance for large objects. In this case, BPIF is three times faster than the ASCII PIF for reading the object, and a little slower writing it. Storage sizes range from eight bytes per value for BPIF, to fourteen bytes for the ASCII version. When compared to the point test, we can conclude that it is better to store a small number of objects that are very large, rather than a large number of small objects, since object access overhead remains relatively constant.

For typical cross-sections, Binary PIF performance meets or beats the ASCII PIF. For example, the DRAM bit-cell contains 575 points, two 450 point impurity arrays, and additional information such as the definition of the tensor product mesh, simulation window, and cut-line information. In this case, ASCII and Binary access times are both under a second, and storage requirements are comparable. This test is more reflective of real performance because it takes into account object organization and access style.

In the ASCII PIF, all data must be accessed serially, while BPIF uses a tree structure. Thus, BPIF can easily make up for the difference in raw access speed if only a portion of the cross-section is used, which is usually the case. For example, a SAMPLE deposition module needs only the top profile-string. With a simple analysis, a rough performance comparison can be made. Assuming the "top-string" data contains fifty turning points and is (on average) in the center of the PIF file, the total number of objects that an ASCII PIF parser must read and search is fifty plus half the number of objects in the file. For a single snapshot in BPIF, the average search through a graph for the top-string is seldom more than ten objects; thus sixty total objects are accessed. Assuming BPIF is three times slower than the ASCII PIF in access time, an equation for relative practical performance can be stated as follows:

$$\text{ASCII PIF} \longrightarrow \; ObjectsToRead + \frac{TotalObjects}{2} \geq (10 + ObjectsToRead) \times 3 \; \longleftarrow \text{Binary PIF}$$

$$50 + \frac{TotalObjects}{2} \geq (10 + 50) \times 3$$

$$TotalObjects \geq 260$$

Therefore, BPIF will be faster when there are more than 260 objects in the snapshot, i.e., when the file size is larger than 10Kb. This performance gap increases with the size and complexity of the cross-section.

Finally, a comparison was made to determine the relative times taken by BPIF versus the OCT data manager. In this measurement, it was found that the creation time for 10,000 OCT points was three seconds out of a total of seven seconds. This indicates that access time is split about evenly between BPIF and OCT. These figures are reasonable for this prototype and compare favorably to the performance of other intertool implementations such as [79], which uses a relational database and is 100 times slower. An optimized BPIF implementation on a commercial object-oriented database would increase performance by at least a factor of two.

5.7 Comparison with Other PIF Implementations

Toolkit performance and ease of programming are most important in BPIF. The toolkit offers adequate performance in most one-dimensional and some two-dimensional applications; some confidence of this fact is given by the examples. The use of C-structures to access data within object instances, and a compact set of functions to manipulate this data, has also made the toolkit relatively easy to learn and use. However, BPIF lacks a formal, object definition that guarantees consistency between PIF objects, and between the intertool and ASCII versions of the PIF. A formal object definition is better addressed by a related Intertool PIF approach, PIF/Gestalt [79], developed at MIT.

Besides PIF/Gestalt, two other PIF relatives deserve mention. The first is the representation used in Intel's EASE system [36], which is a predecessor of the intertool PIF. The rigid FORTRAN programming language and lack of a flexible hierarchical file system such as UNIX made it difficult to extend this database, but many of its concepts have been incorporated into the current intertool PIF. The second is the wafer representation used in the VISTA system under development at TU-Vienna [84]. This system makes extensive use of a modified PIF, both in intertool and intersite forms, and has been successfully integrated the PROMIS process simulator and MINIMOS device simulator. Through discussions among the groups working on intertool PIF, it is the hope that a single conceptual model and a standard interface can be achieved in the future.

5.8 BPIF++

A logical step for BPIF is a toolkit implementation using an object-oriented language and data manager. This extension has been made in a C++ version of the BPIF toolkit, named BPIF++ [97].[1] BPIF++ improves on the original in three ways. First, an object-oriented interface rather than C-structures are used to access the data members, resulting in greater programming language independence through better hiding of the data representation. This is important for future compatibility with the SWR specification, which also uses C++, and other languages that do not use C-like structures, such as LISP and FORTRAN. Second, performance and simplicity are increased by the eliminating the extra translation step previously necessary when displaying the cross-section in VEM.Object policies are compatible between the layout-editor and BPIF++. Third, BPIF++ maintains consistency between the in-memory object and the persistent object stored in OCT. Except for opening and closing the BPIF file, automatic OCT creation and manipulation makes the underlying database virtually transparent.

The BPIF object and function definitions have been changed in BPIF++ to track the evolving SWR standards. BPIF++ objects are very similar to BPIF objects, with minor changes to follow SWR naming convention. BPIF++ geometry objects are stored as OCT geometry objects ,and grid objects are stored as OCT properties. A list of BPIF objects and their OCT counterparts are shown in Figure 5-12. Each time a BPIF object is created, a corresponding OCT object is also automatically created, e.g., creating a BPIF++ vertex creates an OCT point. Object instance graphs for a BPIF++ file, and its corresponding OCT representation, are shown in Figure 5-13. The major difference between a BPIF++ and BPIF graph is the redundant information stored in

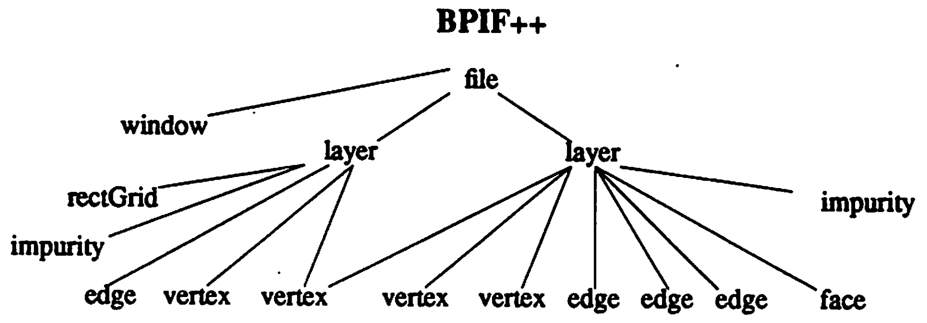
1. Another name for BPIF++ is FPIF, which stands for "finalized PIF".

the edge and face instances corresponding to the OCT edge and polygon instances. Similarly in OCT, point information is maintained both in edges and polygons as well as the point objects representing this data. The reason for this implementation is so that VEM can draw the facet directly without additional translation.

BPIF++ Object	**OCT Object**
file	facet
layer	layer
vertex	point
edge	edge
face	polygon
gen	generator
impurity	property: real-array
window	property: real-array
rectgrid	property: real-array

Figure 5-12. BPIF++ objects and their mapping into objects in OCT physical policy.

The BPIF++ implementation provides a variety of functions equivalent to the BPIF low-level function library for accessing and manipulating instance graphs. Specific functions rather than general ones are used. For example, a vertex `v1` is created using the command `v1 = new (container, x, y)`, instead of filling in the fields for v1 and issuing the pifCreate command. The values for `X` and `Y` can be retrieved using the `getX()` and `getY()` commands, e.g., `xlocation = v1.getX()`, or by accessing the member in the structure, e.g., `xlocation = v1.x`. A summary of BPIF++ functions is shown in Table 5-2.



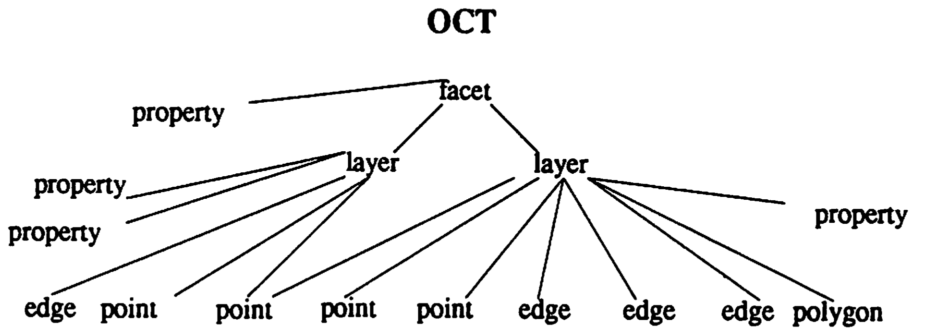


Figure 5-13. Example object instance graphs. The top graph is the BPIF++ representation, while the bottom graph is the corresponding OCT representation.

Performance measurements of BPIF++ show a significant improvement over BPIF, and are listed in Table 5-5. Point access is two to three times faster, while large array access performance remains the same. BPIF++ file sizes are thirty to sixty percent smaller due to the use of compact OCT structures rather than the larger general PIF_STRANGER object type. The DRAM cross-section took under a second to read and write, and was 25% smaller.

Translation routines between BPIF++ and SIMPL-2 were implemented to test the practical usefulness of BPIF++. `BPIF++2Simpl` reads in an OCT facet and converts it into the SIMPL-2 data structure. `Simpl2BPIF++` performs the reverse operation, the result of which can be graphically viewed in the PROSE environment.

Object	Function Name	Description
all	create attach print printOct ~ getOct	creates an object attaches two objects prints object information prints OCT version of object delete object gets OCT version of object
file	open close	opens an instance file saves an instance file
layer	setName getName	sets layer name gets layer name
vertex	getX, getY changeX, changeY	get X or Y coordinate value modify X or Y coordinate value
edge	getVertex getPntX, getPntY count	get array of vertices get array of points number of points
face	getEdge getPntX, getPntY	get array of edges get array of points
gen	gen	create object generator
impurity	getNumber getData setName setData	get number of impurities in array get impurity data set impurity name set impurity data
window	getXmin, getXmax getYmin, getYmax setVals	get X simulation window boundaries get Y simulation window boundaries set boundary values
rectGrid	setVals getX, getY getxArr, getyArr	set number of grid lines get number of X and Y grid lines get array of X and Y grid lines
attribute	createName, createRef, createDim setName, setRef, setDim, setImp getName, getRef, getDim, getImp	create attribute set attribute values get attribute values

Table 5-4. Summary of functions in the BPIF++ toolkit.

	10,000 Points			10,000 Value Array			DRAM Bit Cell		
Access Type	Write	Read	Size (kb)	Write	Read	Size (kb)	Write	Read	Size (kb)
ASCII PIF	1	3	400	0.5	1.5	140	<1	<1	43
Binary PIF	7	6	469	0.7	0.6	80	<1	<1	51
BPIF++	3	2	170	0.8	0.6	50	<1	<1	44

Table 5-5. ASCII, Binary PIF, and BPIF++ performance numbers. All times are in seconds.

The ability to use an object-oriented programming style made it much easier to implement these translators when compared to the translators for BPIF. The code size for these translators was about 210 lines for `Simpl2BPIF++`, and 450 lines for `BPIF++2Simpl`.

5.9 Berkeley Topography Utilities via the PROSE/SIMPL Interface

The Berkeley Topography Utilities (BTU) provide a set of functions for manipulating geometric boundary information [72]. Functions such as stitch-back for non-planar etching and string to grid translation are provided in this set of functions. The SAMPLE/SUPREM IV link makes use of these high-level functions in the stitch-back of a SUPREM IV profile into SIMPL-2 cross-section.

BPIF and BPIF++ currently do not have topography functions implemented internally, but can take advantage of BTU functionality through the PROSE/SIMPL interface. For example, out-diffusions can be performed using this link. The original profile is generated by the SIMPL program and stored in OCT using BPIF++. The diffusion step reads the BPIF++ profile into the SIMPL-2 structure, performs the SUPREM IV diffusion, and updates the mesh values using BTU functions. The diffused profile is then saved back using BPIF++.

5.10 Interoperability

When two programs are tightly coupled and have a high communication bandwidth between them, they are said to *interoperate*. For example, oxidation and diffusion algorithms interoperate since the two processes intimately depend on each other during each time step. In this section, the potential of BPIF for handling separate modules that need interoperability is evaluated through two examples - two-dimensional oxidation/diffusion and three-dimensional deposition.

A typical oxidation step typically uses 5000 node points in the grid due to equation solver limitations. Using BPIF++ performance figures, point write time is 0.4 seconds and read time is 0.3 seconds for this case. Assume that there are ten scalar attributes such as position, velocity, stress, and impurity doping at each point. If each attribute is stored as an array, it will take three seconds to read and four seconds to write. Therefore, the communication time for one tool is seven seconds per step. With oxidation and diffusion tools working together, the total transfer time is fourteen seconds per step (see Figure 5-14). If a one hour oxidation requires an exchange of information every fifteen seconds, the total time spent transferring data between the tools would be about an hour. If the total computation time required is also an hour, BPIF++ interoperability overhead is about 50%.

BPIF++ interoperability performance for three-dimensional topography is similar. For example, in the three-dimensional SAMPLE-3D, about 2,000 triangular elements (4,000 edges) and 1,000 points are used in the gaussian etching simulation of a surface corner [98]. The total number of BPIF++ objects is therefore 7,000 (2,000 + 4,000 + 1,000), which takes 1.4 seconds to read and 2.1 seconds to write. To exchange this data between a movement/visibility calculation module and a regularization module would require seven seconds (2 x (1.4+2.1)). If 50 regularization and 300

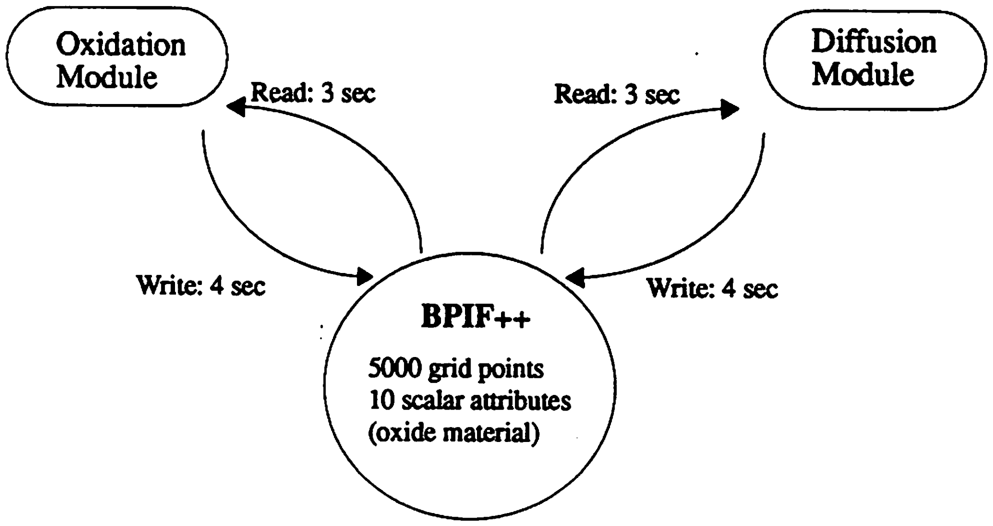


Figure 5-14. Oxidation/diffusion interoperability example. Each transfer between the two modules takes about one second.

movement steps are required, then the total time spent moving the data around is forty minutes (7 seconds x 350). A typical SAMPLE-3D run takes twenty minutes, so the forty minute interoperability (200%) overhead is not acceptable, except for possibly accessing a computationally intensive de-loop algorithm that requires a smaller number of data exchanges, such as [99]. For instance, if ten de-loop calls are required for the simulation described above, then the exchange time using BPIF++ would be 140 seconds (2 x (7 seconds * 10)). This is only twelve percent of the total simulation time.

5.11 Conclusions

Three years of experimenting with the PIF wafer representation philosophy has uncovered many TCAD requirements and issues in the areas of direct program access, database support, and visualization. These items have been addressed in the BPIF implementation. In the Intertool PIF style, BPIF consists of a library of functions that

TCAD programs may use to create, access, manipulate, and store objects using objects defined similarly to the textual PIF. During this time, this work has evolved from a C-language interface in the original BPIF to an object-oriented, C++ interface named BPIF++.

BPIF is a prototype toolkit that explores the feasibility of intertool PIF usage within a TCAD environment, PROSE, and is driven by the needs of tools. Performance and ease-of-use requirements of the toolkit have been confirmed through the development of a BPIF package used within PROSE, which connects the SIMPL, SAMPLE, and SUPREM process simulators. BPIF is similar to other PIF-like implementations and efforts are being made to unify the different approaches.

An object-oriented version of BPIF, named BPIF++, has been defined and implemented. BPIF++ objects are stored in OCT, and a set of C++ functions is provided to manipulate these objects. BPIF++ improves on BPIF in performance and ease of use. Consistency is maintained between in-memory and database objects, and objects follow the SWR programming interface style. A targeted use of BPIF++ is to support data transfer between modules that interoperate. Interoperability performance for BPIF++ is plausible for solving oxidation/diffusion problems in two-dimensions, and reasonable for special cases in three-dimensions, such as the computationally expensive de-loop procedure. The BTU topography toolkit can also be linked into PROSE by BPIF++ and SIMPL. The advantages of BPIF++ make it the representation of choice for future PROSE development.

6
Coupling TCAD and IC Design CAD

A coupling between TCAD and IC CAD can be a potent combination in advancing the capabilities of CAD. It can foster increased understanding for both the circuit designer and technologist, and create powerful new applications that make use of both mask and wafer data. This chapter describes the philosophy behind the TCAD/IC CAD coupling, several potential applications, and PROSE's contribution of wafer and phase-shifting mask data and applications to the OCT/VEM/RPC CAD framework.

Section 6.1 describes the need for a TCAD/IC CAD coupling. Examples of several potential applications of this link are described in Section 6.2. Section 6.3 discusses how PROSE adds TCAD components to the OCT/VEM/RPC CAD framework to achieve a unified environment. Section 6.4 uses SIMPL as an example to explain how new applications can be easily added. Section 6.5 summarizes the chapter.

6.1 Introduction

In the past, CAD for technology (TCAD) and CAD for design (IC CAD) have been treated as separate disciplines in both industrial and academic research groups. Designers created circuit and mask designs based on "magic" technology parameters supplied by technologists in the form of circuit models and layout ground rules. Technologists had little need for IC CAD because the TCAD did not really need anything more than simple mask configurations such as lines and elbows. The situation today is different due to more aggressive designs and technologies. Circuit designers and technologists must work together to balance design and technology considerations that transcend traditional boundaries.

Coupling IC CAD and TCAD can significantly raise the level of communication between designers and technologists by improving understanding, knowledge transfer, and model accuracy. First, a combination IC CAD/TCAD system can answer designers' technology questions by providing device cross-sections and circuit models for a particular layout situation. Second, this system can speed up the transfer of design and technology data by automatically updating circuit and layout models within both systems, rather than relying on human transfers through files or written memos. Third, the system can take advantage of more sophisticated models. For instance, parasitic values can be extracted using TCAD tools and passed to the circuit simulator, rather than relying on an average resistance per unit length value. TCAD tools, on the other hand, can use typical mask sections of the real design rather than test pattern geometries. The main types of data that are exchanged between IC CAD and TCAD were presented in Figure 1-3 and are reproduced in Figure 6-1.

There are also synergistic effects in the development of TCAD and IC CAD tools. Infrastructure and utilities can be shared among tools to display and edit cross-section and layout information, store this data, and manipulate this data. For example, the VEM layout editor and OCT data manager are used for editing and storing both mask and wafer information in PROSE. CAD layout editors contain set operations for manipulating masks, that can, if represented similarly, be applied to cross-sections for merging geometries. Simulated annealing algorithms can be used for optimization problems in both areas, such as cell placement in CAD and image optimization in TCAD.

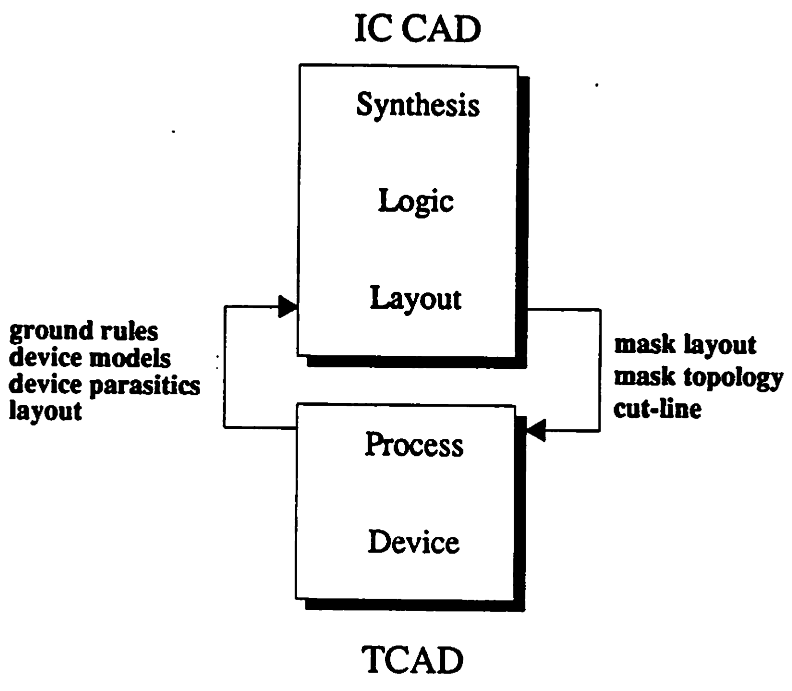


Figure 6-1. Information exchanged between technology CAD and IC Design CAD (or EDA) environments.

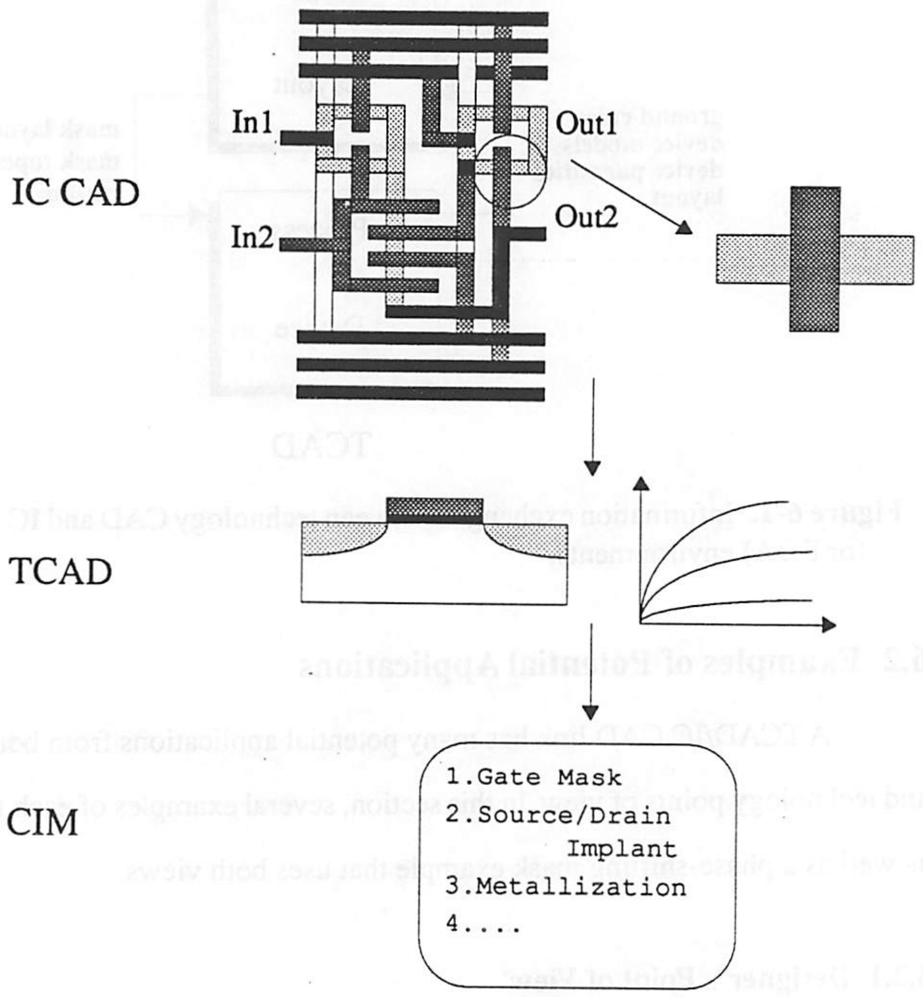
6.2 Examples of Potential Applications

A TCAD/IC CAD link has many potential applications from both the design and technology points of view. In this section, several examples of each type are given, as well as a phase-shifting mask example that uses both views.

6.2.1 Designer's Point of View

From the designer's point of view, having a tool that can reach from synthesis to layout to process and device simulation in TCAD opens the door to many new applications. Ideally, this "CAD Microscope" allows designers to "see" not only mask layout, but also cross-section, device, and manufacturing views of the wafer (see

Figure 6-2). In addition to educating designers on these new views, the microscope can also help in the improvement of circuit models and the understanding of design-rules.



```
1.Gate Mask
2.Source/Drain
      Implant
3.Metallization
4....
```

Figure 6-2. The conceptual CAD Microscope. By inspecting an area in the mask layout, the user would be able to generate the TCAD cross section, device simulation, and manufacturing process flow information.

Layout editors generally present a conceptual view of the mask patterns, which can be quite different from the images incident on the actual wafer. These differences, predictable by TCAD, result from the diffraction effects of light, the imperfections of the optical imaging system, and the non-planarity of the wafer. The ability to view patterns based on these parameters can play an important role in understanding mask printability and defect tolerances.

Viewing cross-section data graphically within the layout can also be a very powerful tool for understanding design rules. The number of design rules has increased dramatically from 30 rules for a 5 micron, 6 mask DRAM process to 100 rules for a 1.5 micron process with 13 masks [100]. Rules were meant to be broken, and the best designers are the ones who know when to "break the rules" to get the most out of a technology. A cross-section display tool built on the integrated IC CAD/ TCAD system can help both amateur and expert designers decide how to trade-off design rules for space and/or performance by quantifying the consequences. For example, a metal-to-metal spacing design rule can be important in avoiding sharp corners that cause reliability problems, but the difference between an acceptable or unacceptable angle is dependent on device topography. With a graphical interface, verification of a mask configuration can be achieved simply by viewing the cross-section and identifying the most critical angle [27].

Model accuracy can be improved by accounting for TCAD results in the circuit simulation loop. For instance, layout-driven device simulation can provide better models to the circuit simulator as well as provide device sizing information. The same ideas can also be used for parasitic resistance and capacitance elements, which can be calculated more accurately based on layout when compared to the simple unit length and width approach. This can be particularly important in cases with severe topography, where the actual resistance can be three times greater when compared to the planar case.

6.2.2 Technologist's Point of View

From the technologist's point of view, having a tool that can reach from process simulation, process integration to layout, terminal conditions and circuit performance opens the door to other classes of new applications. Looking through the microscope in reverse allows a broad view of the entire context of product design. Thus decisions in developing unit processes, integrating processes, predicting yields, and making the process manufacturing worthy can be made in the overall context of the products.

The lithographer and mask maker in establishing a base line process can, for example, examine the kinds of patterns which will be required to hold critical size. The process integrator will be able to examine area and speed penalties from various metallization step-coverage approachs to decide which technology to pursue. Equipment modelers in predicting tool yield impact will be able to realistically assess interaction of defect size distributions with the variation in circuit vulnerability as a function of defect location in the layout design. Manufacturing support will be able to anticipate problems caused by device context issues such as problems in alignment signal quality variations due to subtle film thickness changes, or resist notching due to reflections from device topography. The entire design team will benefit from being able to assess worst case scenarios from diverse factors such as changes in circuit loading, layout hedges, tools, materials and process conditions.

6.2.3 Phase Shifting Masks

Phase-shifting mask design is singled out to illustrate in detail the TCAD/IC CAD link because it has a strong dependence on both design and technology. Using alternating (0 and 180 degree) phases of light passing through transparent features allow much closer packing of mask patterns. The higher density for the same lithography tool results in potential feature-to-feature interactions for every pattern on a mask. These local effects are linked with global effects as the routing causes different phase assignments over the entire chip, since the computation time needed by the TCAD imaging tools is not practical for large chips. These tools depend on guidance from IC CAD for selecting the critical areas that require detailed simulation. Local simulations may also identify areas where the global view is affected. This co-dependence forms an iterative loop where trade-offs due to both IC CAD and TCAD constraints must be considered. A phase-shifting mask analysis and design tool, using both global IC CAD and local TCAD tools, is described in Chapter 7.

6.3 Coupling TCAD and IC CAD in PROSE through a Framework

One of the main goals of PROSE is to explore the coupling of TCAD with IC CAD. PROSE is well positioned to do this because is built to be an application under a general CAD framework. The chosen framework is OCT/VEM/RPC [57], which is designed to support a variety of IC CAD tools. It does this by providing uniform user interface, database, and communications facilities that CAD applications can make use of.

6.3.1 City Analogy

The infrastructure that a framework provides is analogous to the infrastructure provided by a city. Like the TV station, library, and highways of a city that attract skilled workers and businesses, the framework concept encourages tools to work and live together by supplying the basic CAD infrastructural components necessary for tool operation. In Figure 6-3, t he OCT/VEM/RPC CAD framework provides the infrastructural components such as the database (library), user interface (TV station), and intertool communications (highways) necessary to support a "city" of CAD tools. This infrastructure can also support new suburbs, such as a TCAD framework, which will in turn build new infrastructure. In PROSE, TCAD specific representations and design functions are added to enhance the overall framework for TCAD and IC CAD tools.

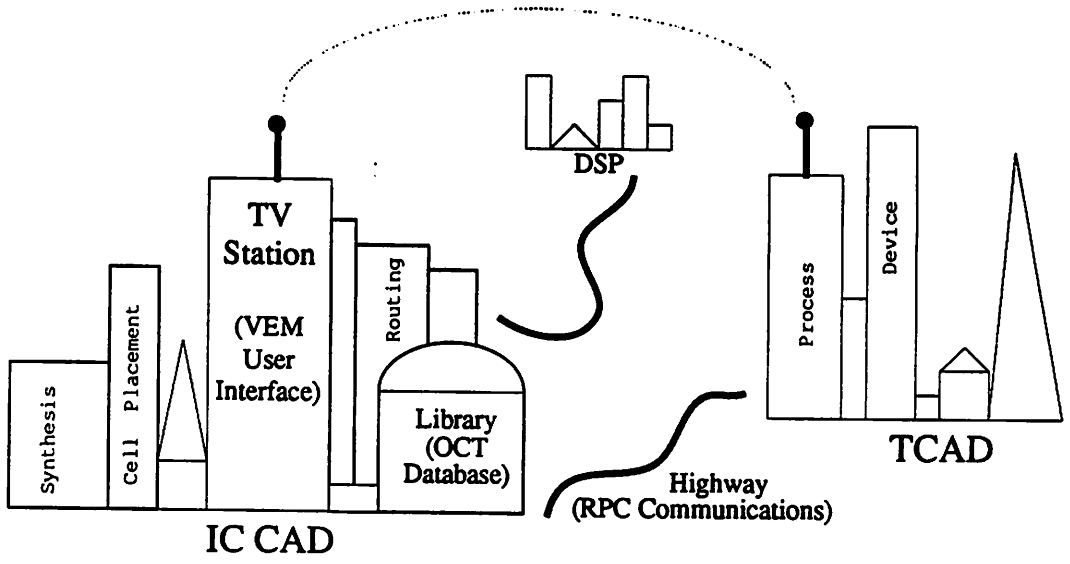


Figure 6-3. The OCT/VEM/RPC CAD framework provides the infrastructural components necessary to support a "city" of CAD tools and outlying suburbs. In this example, the downtown area is IC CAD, TCAD is a major suburb, and the metropolitan area is CAD, which includes other suburbs such as automatic digital signal processing synthesis tools.

6.3.2 OCT/VEM/RPC

The OCT/VEM/RPC CAD framework was developed in 1986 to support the IC CAD tools at UC-Berkeley. Since that time, there have been four releases of the Octtools, which contains the framework and the over fifty IC CAD tools that use it. Logic synthesis, placement, and routing tools are examples of applications that had traditionally used these framework facilities.

The framework is divided into three components: the OCT data manager, the VEM layout editor, and the RPC communications facility. OCT is a persistent object-like database for IC CAD information. Physical mask layout, symbolic "sticks" mask layout, and circuit schematics are the main views that OCT represents. OCT function calls enable applications to access these objects, which are stored in a graph structure. VEM is an X-Windows application that allows one to visualize and edit OCT data through a full-featured layout editor. Different functions are available in VEM, depending on the view being edited. For instance, one can add, delete, and change points, edges, and polygons when editing masks in the physical view, or add gates in a schematic view. The OCT Remote Procedure Call (RPC) mechanism allows OCT and VEM functions to be transparently called over different machines. This makes it possible to have a uniform interface between VEM and a designated remote application by enabling the application to take advantage of the menu, control, and query interfaces supported by VEM. Even if the application resides on a different machine, RPC still makes it appear to be a subroutine in VEM's address space.

Two of the framework's strongest features are its ability to represent, display and control multiple views of the design, and to allow user-supplied applications to operate on these views across different machines. An example of an integrated IC CAD application is shown in Figure 6-5. Three OCT facets representing different levels of design abstraction have been opened within VEM: the mask layout of a gate; a circuit diagram of a decoder; and an output driver cell schematic. The layout editing functions in VEM are used to coordinate the viewing and editing of these facets. An RPC application can also be dynamically associated with a view to add additional functionality to the editor. These applications can be user-defined and require no changes to the VEM source code. For instance, the layout of the gate might call on an interactive design rule checker, while the output driver may be linked to a timing simulator contributed by another user.

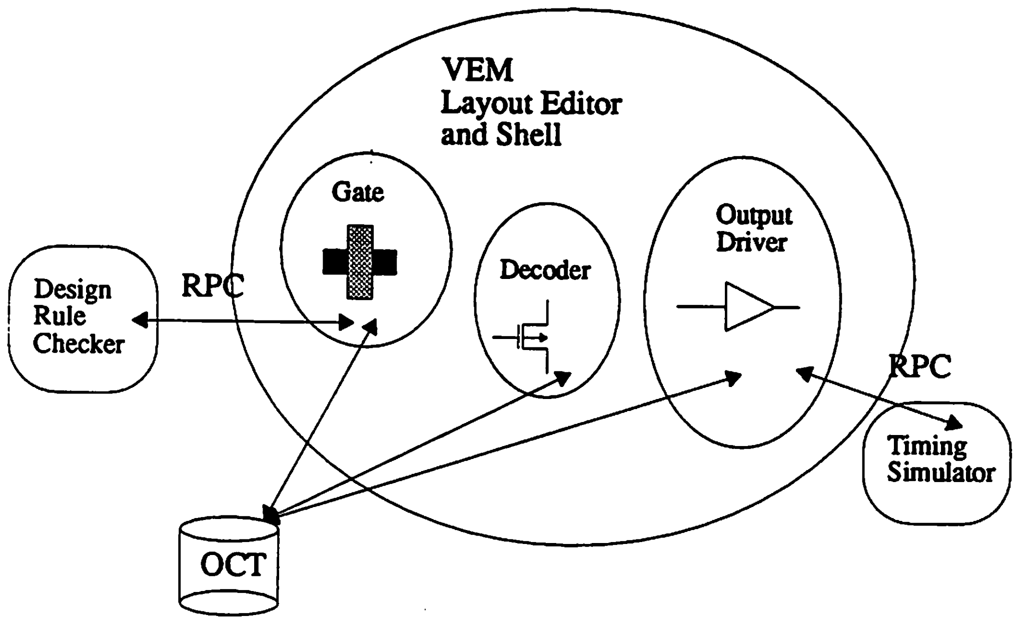


Figure 6-4. The VEM shell allows multiple views that are stored in the OCT database to be edited and controlled. User supplied applications, such as a design rule checker and timing simulator, can be added to the system through the RPC mechanism.

6.3.3 PROSE TCAD Additions to OCT/VEM/RPC

PROSE makes a new class of TCAD representations and applications available through the OCT/VEM/RPC framework. These additional capabilities are shown in Figure 6-5. The Binary PIF enables cross-section wafer data to be represented in OCT and accessed through the BPIF toolkit. Mask layers can also be either of single or multiple-phases, with the the latter being for phase-shifting mask applications. Both of these new views can be displayed as in VEM in addition to its standard views. Process simulation and phase-shifting mask applications have been linked to the framework by RPC.

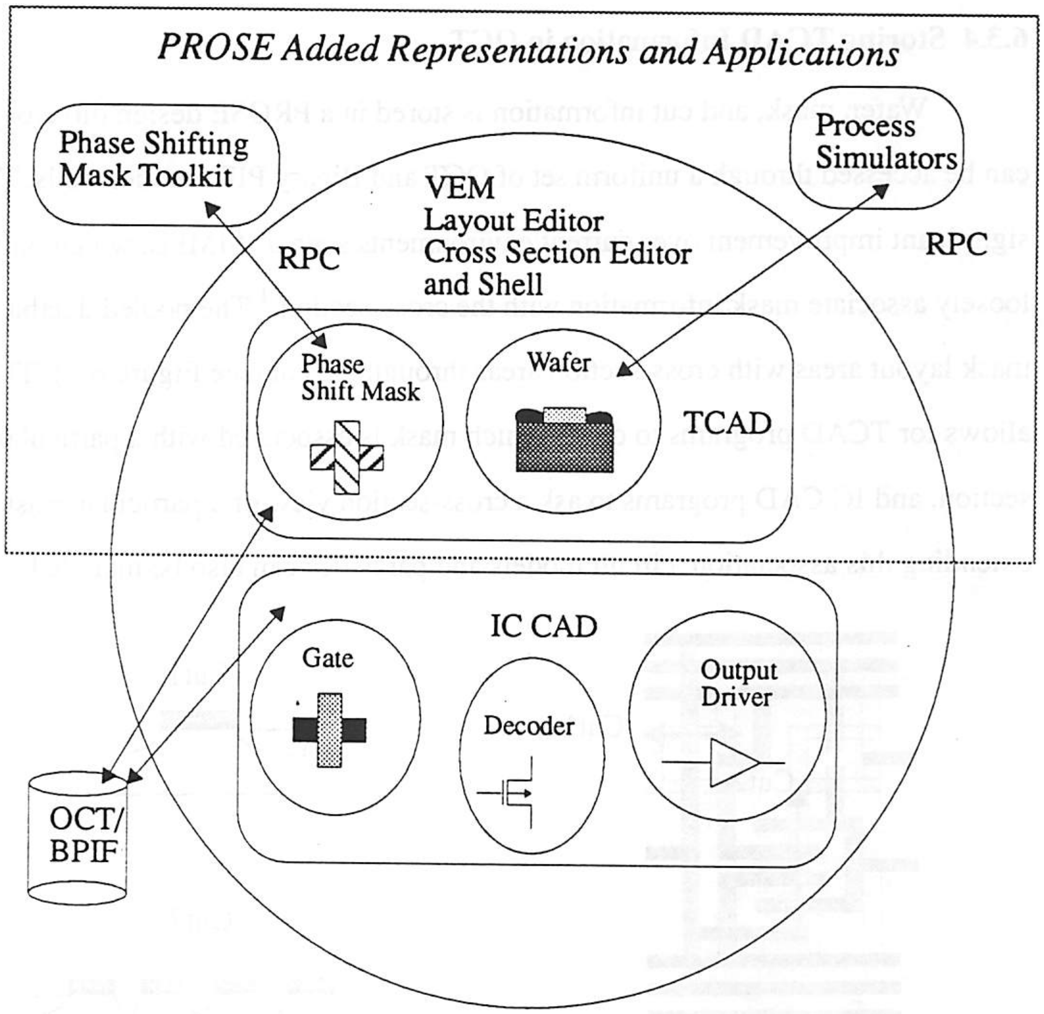


Figure 6-5. PROSE couples IC CAD and TCAD by adding TCAD cross-section and phase-shifting mask views, as well as process simulators and a phase-shifting mask toolkit, to the OCT/VEM/RPC framework.

Most of the CAD microscope concept discussed in Section 6.2 has been realized through the PROSE extension with OCT/VEM/RPC, which implements a dual TCAD/ICCAD system. Users have access to circuit design, mask layout (both regular and phase-shifting masks) and wafer profile information. Users can indiscriminately make use of all three types of data from single or multiple applications.

6.3.4 Storing TCAD Information in OCT

Wafer, mask, and cut information is stored in a PROSE design directory that can be accessed through a uniform set of OCT and Binary PIF function calls. This is a significant improvement over current environments such as SIMPL, which only loosely associate mask information with the cross-section.[1] The pooled database links mask layout areas with cross-section areas through the cut (see Figure 6-6). This allows for TCAD programs to query which mask is associated with a particular cross-section, and IC CAD programs to ask a cross-section view of a particular mask. By extending this association, circuit models and parasitics can also be included.

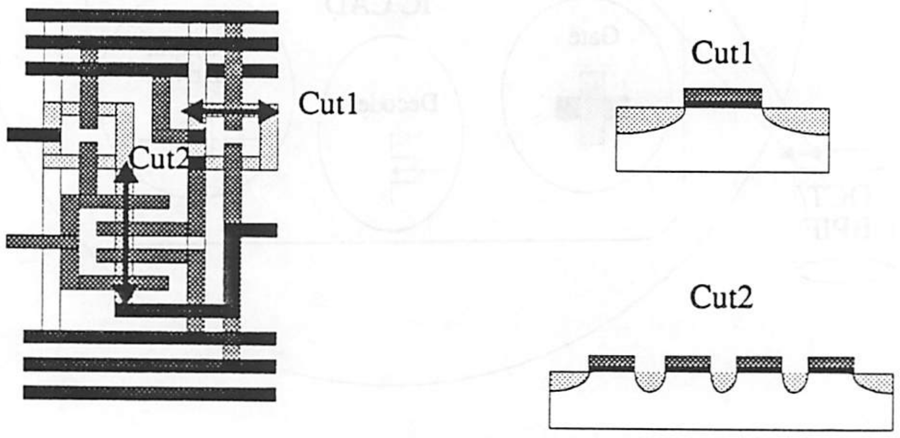


Figure 6-6. Each mask may have many cross-sections associated with it. These are determined by the different cut-lines.

1. In SIMPL-IPX, the cut, and the mask segments associated with the cut, are stored with the cross-section, but the mask itself is not linked. Inconsistencies can arise when the mask is modified because the cut and mask segments are not notified of this change.

PROSE implements each design as a UNIX directory consisting of four sub-directories called *views*, one each for the physical mask, cross, cut-line, and cuts on the mask (see Figure 6-7). Each sub-directory in turn contains a contents facet and an optional interface facet, which are UNIX files. The *contents* facet holds the actual data, while the *interface* facet provides an abstraction of the data for tools that may only need a subset of the view's information, e.g., the boundaries for a cell. The *mask* view (also called the *physical* view) contains the geometric shapes of the mask and optional parasitic values associated with each geometry. The *cross* view holds the device cross-section and is accessible by the Binary PIF. The *cut-line* view contains the location of the cut-line across the mask. The *cuts* view saves the one-dimensional openings associated with the cut-line for the particular mask set (see Figure 6-8).

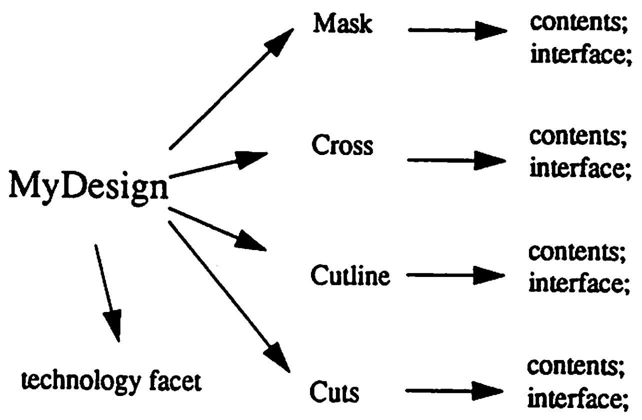


Figure 6-7. Organization of mask, wafer, and cut information for a PROSE design.

6.3.5 Editing TCAD Views in VEM

The VEM editor provides a uniform user interface for both IC CAD and TCAD applications, eliminating the need for the user to switch between two different interfaces. Mask editing is accomplished using standard VEM physical-mask

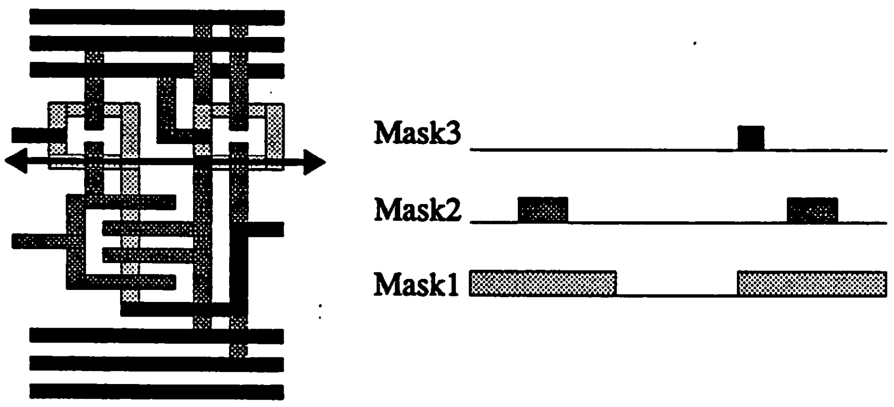


Figure 6-8. One-dimensional mask openings are saved with the cuts facet.

commands as described in [63]. Polygon cross-section editing is accomplished in exactly the same manner in VEM, but the results are stored in a BPIF wafer view. As previously described, masks and cross-sections are associated by the cut. Users can ask for a particular cross-section by drawing a two-dimensional cut-line across the mask and invoking a menu command to view the cross-section. If one already exists, it is displayed. Otherwise, the cross-section is generated by the TCAD Manager either interactively or through batch processing. The interactive interface is demonstrated in Section 2.5, and uses the same editing style. Users can also open a cross-section and query for the associated layout. In this case, two layout views are displayed: the mask and cut-line and the one-dimensional mask cuts.

6.3.6 Communicating with RPC

In PROSE, the TCAD Process Manager (or just "the manager") and Phase-Shifting Mask Toolkit described in Chapter 7 are implemented as RPC remote applications to VEM. These applications have access to all VEM user-interface facilities including: (1) deck-of-cards menus; (2) dialog boxes; (3) graphical "pointing"; and (4) console input commands. These facilities are explained in the context of the TCAD Interactive Manager.

The deck-of-cards main menu of the manager is created through a C-structure that specifies the names that will appear on the menu and the corresponding routines that will be invoked. When the manager needs more information from the user, it can use dialog boxes, such as those shown in Figure 2-10, to query for additional parameters. Graphical "pointing" capabilities can also be used in cases where menu or textual entry is cumbersome. An example of how the manager uses this feature is in identifying the mask required for an exposure step. In this case, The user can point to the mask in the layout window rather than typing in the mask name. Commands from the user can also be typed directly into the VEM console window, which passes the command to the manager.

6.4 Using and Creating a New Application

The procedure for using PROSE in the framework is as follows. After the VEM shell is started, the user can open an OCT facet, e.g., the cross-sectional view of the device, by typing its name into the VEM console window. The PROSE RPC application, e.g., an modified process simulator such as SIMPL modified to use the RPC functions (SIMPL-RPC), is then invoked. This opens an RPC connection between VEM and SIMPL that allows data to pass between VEM, SIMPL, and OCT. User input is entered into VEM, which then directs it to SIMPL if appropriate. Wafer and layout data are passed between SIMPL and OCT through VEM. Any changes to the facet are also updated in the VEM graphical view. When the application is finished, the RPC connection is broken and full control returns to VEM. Each wafer cross-section can run a different RPC application, but no more than one application can be running at any time in a facet. All internal VEM functions remain accessible even when there is an RPC application running, so capabilities such as geometry editing, cell hierarchy querying, opening new windows, etc., are can also be used.

New applications can be easily added to PROSE to take advantage of the features described above. Converting an existing program that uses BPIF and/or OCT calls into an RPC application involves two major modifications and one minor one. The major changes are the modification of the "main" routine of the program to initialize the RPC link, and the modification of routines to use a "callback" setup. The minor change involves using the RPC library rather than the OCT library during program compilation. A tutorial on adding new RPC applications can be found in the RPC Programmer's Guide [101].

The SIMPL program demonstrates how a simulation program can be modified. First, the main routine in SIMPL is changed to SIMPLMain and a UserMain routine is added. The UserMain routine is called when the RPC application is invoked by VEM. VEM passes this information about the facet, mouse location, and status that led to its invocation. SIMPL calls on its initialization routines, e.g., the routines originally in SIMPLMain. If there is no cross-section in the facet, SIMPL creates one by asking the user for the substrate background doping and making BPIF calls to create it. If a cross-section exists, then it is read into the SIMPL data structure. When the initialization is complete, SIMPL returns control to VEM with a pointer to the menu entries and the associated routines that are called when the user selects that entry. For example, if the deposition command is selected, the rpcSIMPLDepo routine is invoked. This routine is shown in Figure 6-9. RPC menus prompt the user for the deposition type and parameters, such as the material to be deposited, the deposition thickness, and the deposition model to be used. The SIMPL deposition function is then called, the cross-section is modified internally in SIMPL, and the data is written back into OCT.

If a tool uses BPIF, then it is extremely easy to convert it into an RPC application. One week's time was needed to switch SIMPL from the textual interface over to the RPC graphical interface. The most time consuming aspect of the conversion was to write the user dialog menus for each of the callback functions.

6.5 Summary

One of PROSE's greatest strengths is its ability to couple tools and data between both IC CAD and TCAD worlds. This coupling has many new potential applications from both the designer's and technologist's viewpoints. PROSE adds TCAD components the OCT/VEM/RPC framework, implementing the IC CAD/TCAD link. The result is a system where mask layout (regular and phase-shifting) and wafer information, and the tools that support them, are seamlessly presented using uniform user interface, database, and communications mechanisms. Implementing new applications in this integrated environment is easy, as demonstrated by the conversion of the SIMPL program. This component of PROSE has been shown to meet the needs of an integrated IC CAD/TCAD system.

```
/*** rpcDepo - SIMPL deposition example ***/
/* define deposition type menu shown in Figure 2-10 */
dmWhichItem depotype[] = {
               {"anisotropic",0,0},
               {"isotropic",0,0},
               {"vertical",0,0},
               {"spin-on",0,0}};
/* define deposition parameter menu shown in Figure 2-10 */
dmTextItem depoitemsvert[] = {
               {"material name",1,20,"POLY",0},
               {"thickness",1,10,"1",0}};

int rpcDepo(spot,cmdList,uow)
RPCSpot *spot;                   /* where the mouse is */
lsList cmdList;                  /* params on VEM command stack */
long uow;                        /* user option word (not used) */
{
     ...
/* create depotype menu and wait for response, returned in "num" */
     if (dmWhichOne("DEPOSITION TYPE",4,depotype,&num,0, help-
        String) == VEM_NOSELECT)
      return(RPC_OK);            /* return if no selection */
...
/* vert. deposition - ask for parameters, returned in depoitems */
     if (dmMultiText("DEPOSITION PARAMETERS",3,depoitems) == VEM_-
        NOSELECT)
      return(RPC_OK);
/* all answers are strings; convert thickness to floating point #
        */
     thickness = (float)atof(depoitems[1].value);
     ...
     VEMMSG("Running Deposition...");
     bpifpp2Sim(spot);           /* read BPIF++ wafer into SIMPL */
     /* perform SIMPL vertical deposition */
     SIMPL_Deposition(depoitems[0].value, thickness, spin);
     sim2bpifpp(spot,cmdList,uow); /* write back into BPIF++ */
     VEMMSG("done!\n");          /* report to user that we're done */
     return(RPC_OK);}            /* notify VEM that we're done */
```

Figure 6-9. A code fragment of the SIMPLdepo function showing how an elementary SIMPL vertical deposition step is performed on a BPIF++ wafer. The depotype menu structure creates the selection menu shown in Figure 2-10.

7
PROSE for Phase Shifting Masks

A particularly challenging example that highlights the need for a TCAD link to IC CAD is the phase-shifting mask application. Using this technology, features can be spaced more closely by using alternating phases of light passing through transparent areas, resulting in higher resolution patterns using current generation lithography equipment. This chapter describes a PROSE Phase-Shifting Layout Toolkit, nicknamed PLUTO, that implements a phase-shifting mask CAD system. PLUTO contains an automatic analysis tool for determining the phase-shifting viability for a particular design and a toolkit for designing phase-shifting masks within the layout editor. These tools are applied to the periphery circuits of a 16Mb dynamic RAM scaled for 64Mb rules using phase-shifting technology.

Section 7.1 introduces the phase-shifting mask problem and the architecture of the PROSE implementation. Section 7.2 introduces the goals the PROSE phase-shifting mask implementation. Major phase-shifting mask components are described in Section 7.3. The analysis tool is discussed in Section 7.4, and the interactive toolkit in Section 7.5. Section 7.6 investigates a 16Mb DRAM using the tools and is an adaptation of material published in the 1991 Intl. Electron Devices Meeting Technical Digest[1]. The advantages of a symbolic layout approach are described in Section 7.7. A summary of this chapter is presented in Section 7.8.

1. A.S. Wong, et. al, "Investigating phase-shifting mask issues using a CAD toolkit," Intl. Electron Devices Meeting, Washington D.C., Dec. 1991 [102].

7.1 Introduction

Phase-shifting mask technology has recently become more than just a novelty due to its potential for increasing resolution using current-generation lithography equipment as well as extending the ultimate limits of optical lithography. CAD tools for phase-shifting masks have not been developed in a large part because of the difficulties in linking global IC data and tools with local TCAD analysis mentioned in Chapter 6. This section gives some background of phase-shifting mask technology and proposes an architecture for a phase-shifting mask CAD system.

7.1.1 Levenson Phase Shifting

Emerging phase-shifting mask technologies have the potential of creating more compact designs at the small cost of a few additional passes in writing the masks. First proposed by Levenson for projection printing in 1982, this technique uses opposite (0 and 180 degree) phases of light so that features can be more densely packed [103]. This alternating approach is particularly attractive because the spill-over of light between transparent areas that would normally add can now be subtracted, allowing features to be placed closer together. Image contrast can also be increased and defocus effects can be reduced, leading to a smaller printable feature size. Figure 7-1 compares the phase-shifted and non-phase shifted case for an alternating line and space pattern. The final intensity incident on the phase-shifted wafer has higher intensity and therefore better resolution. Other phase-shifting techniques such as the use of rims [104], outriggers [105], and a chromeless technique [106][107], might also be included in the toolkit but are not addressed here.

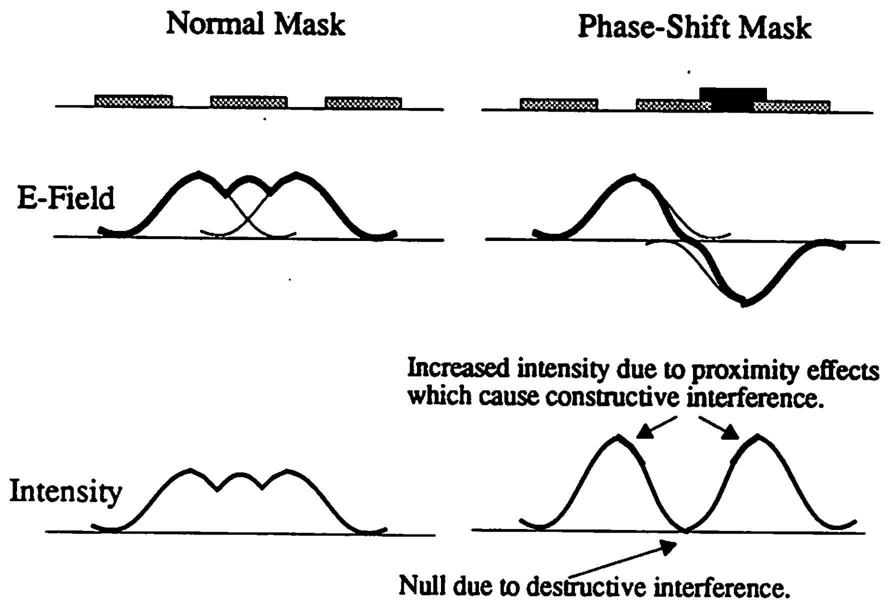

Normal Mask
Phase-Shift Mask
E-Field
Increased intensity due to proximity effects which cause constructive interference.
Intensity
Null due to destructive interference.

Figure 7-1. Phase-shifting masks can increase image intensity, contrast, and therefor feature density by using opposite phases of light. Intensity is proportional to the square of the electric field.

7.1.2 Architecture

Ultimately, one would like to have a phase-shifting mask CAD system that can automatically translate a traditional, single phase design into a phase-shifted one that uses less area, while maintaining the same printability as the original mask (see Figure 7-2). A possible flowchart for accomplishing this task is shown in Figure 7-15. The first step determines the amount of shrinkage that can reasonably be achieved using phase-shifting mask techniques. This factor can either be set a priori by the designer who requires a specific shrinkage amount, or can be based on constraints such as image contrast and intensity. Next, each mask region is phase-shifted by the chosen approach, e.g., alternating line or rim shifter. A design-rule checker (DRC) then checks the new design for violations. These violations are corrected, possibly resulting in a different global phase assignment or even a different shrink factor. The proposed flowchart is only one of several possibilities. For example, phase assignment can occur before shrinkage to allow for different shrink-factors to be used for different phases. This has the disadvantage of not providing the phase assignment mechanism with data about the shrink. A local feedback loop between these two steps may therefore be necessary.

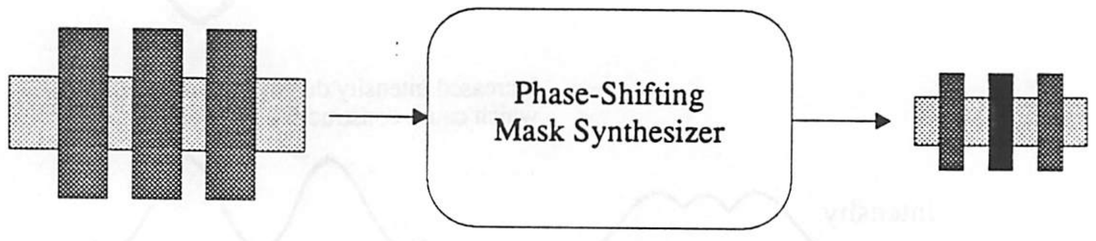


Figure 7-2. The ideal automatic phase-shifting mask system takes a regular mask as input, and magically produces a phase-shifting mask.

Theoretically, one would like to use detailed TCAD analysis to solve for or optimize the entire chip through image simulation. Realistically, several constraints make this level of analysis infeasible. Computation time increases as the fourth power of the area, and storage requirements are proportional to area. A practical example is the SPLAT program, which uses transmission cross coefficients to compute the areal image [108]. SPLAT requires ten minutes and forty seconds to compute the image of a 0.8 λ/NA sized contact hole in a 5 λ/NA by 5 λ/NA region on a Sun 4/280 [109]. The non-linear relation between the mask geometry and the image at the wafer also requires optimization algorithms to compensate the mask pattern using non-linear optimizers. Even using simulated-annealing techniques, thousands of SPLAT runs are necessary for optimizing a twenty-by-twenty array of pixels [110].

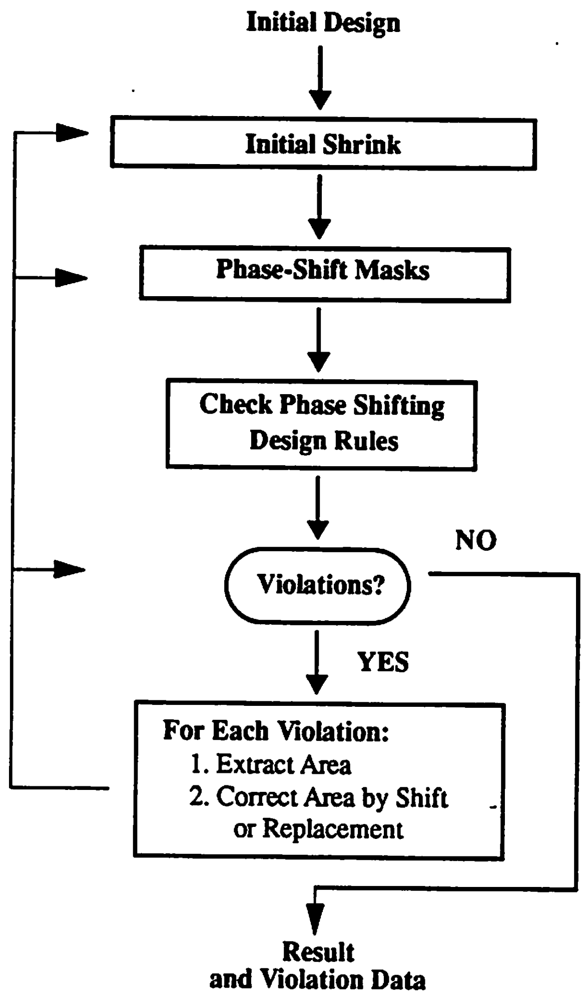

Initial Design
Initial Shrink
Phase-Shift Masks
Check Phase Shifting Design Rules
Violations?
NO
YES
For Each Violation:
1. Extract Area
2. Correct Area by Shift or Replacement
Result and Violation Data

Figure 7-3. Phase-shifting mask synthesizer flowchart.

Trade-offs between detail versus size must be made. The inclusion of IC CAD can relieve this problem by applying rule-based phase-shifting diagnostics at a global level, and calling on TCAD only in instances where design rules need to be established for new patterns. Using this global/local division, the IC CAD and TCAD components required of such a system are shown in Figure 7-4. In addition, several components, such as a user interface and manager, must span both environments.

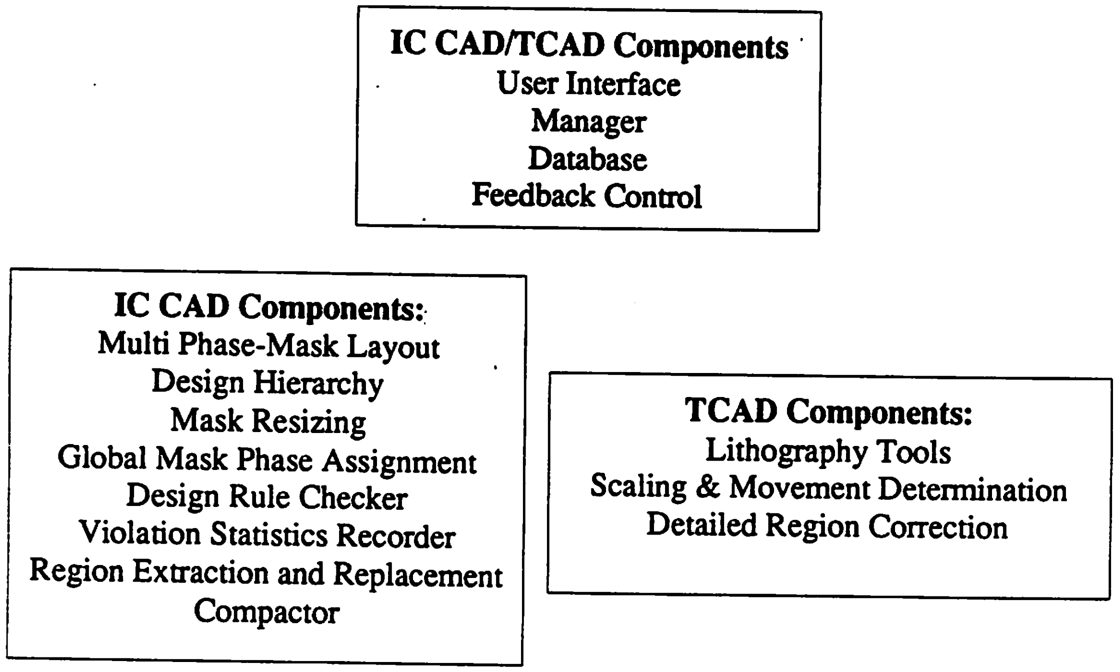


Figure 7-4. Requirements for a two-level phase-shifting mask system using both TCAD and IC CAD.

7.2 PROSE Implementation Goals

The goal of the PROSE implementation is to develop the major components necessary for a phase-shifting mask design system that uses both global ICCAD and local TCAD information. Four major components were built to perform automatic phase assignment, design-rule checking, mask area capture and extraction, and phase-shiftability analysis. Phase assignment uses a global recursive algorithm to assign opposite phases to neighboring mask shapes. The design rule checker can flag minimum space and width requirements for layers of different phases and shapes. Small sections of the global mask can be extracted from the layout for local analysis, redesigned, and replaced. An analysis tool uses design rule violation information to