DISSERTATION

Integrated Semiconductor Technology Analysis

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik
von

Christoph PICHLER

[redacted]

[redacted]

Wien, im März 1997

Kurzfassung

Die numerische Simulation der Herstellung und des elektrischen Verhaltens elektronischer Halbleiterbauelemente spielt eine wesentliche Rolle in der Entwicklung und Fabrikation integrierter Schaltkreise für die Verkürzung der Entwicklungszyklen und -kosten moderner *Very Large Scale Integration* (VLSI) Produkte. *Technology Computer Aided Design* (TCAD) umfaßt alle Computerprogramme und computerunterstützte Verfahren, die der Entwickung und Analyse von Halbleiterbauelementen und -prozessen dienen.

Diese Dissertation stellt eine TCAD Umgebung für die Simulation vollständiger VLSI-Herstellungsprozesse vor. Besonderes Gewicht wird auf die Integration unterschiedlicher Simulatoren, die Darstellung von Herstellungsprozessen und die Automatisierung komplexer Analyseabläufe gelegt. VISTA/SFC stellt eine Erweiterung des *Vienna Integrated System for TCAD Applications* (VISTA), einem Integrations- und Entwicklungsframeworks, im Hinblick auf die praktische Anwendung im industriellen Einsatz dar.

Die Integration unterschiedlicher Prozeß- und Bauelementsimulatoren erfolgt über standardisierte Schnittstellen zwischen dem steuernden Framework und externen Programmen. Externe Gittergeneratoren werden herangezogen, um die Konsistenz des Wafermodels nach den einzelnen Simulationsschritten zu gewährleisten. Externe Programme werden über LISP-Funktionen and das Framework angebunden. Als primäres Datenaustauschformat findet das *Profile Interchange Format* (PIF) Verwendung.

Die Darstellung von Herstellungsprozessen erfolgt mittels simulatorspefizischer Kommandos, wobei in einem Transformationsschritt prozeßspezifische und herstellungsspezifische Daten entsprechend umgewandelt werden. Prozeßsimulationsaufgaben werden parallel abgearbeitet und auf alle zur Verfügung stehenden Rechner verteilt, wobei Mehrfachberechnungen identer Teilaufgaben automatisch vermieden werden. Die wesentlichsten Gesichtspunkte bei der Realisierung der graphischen Benutzeroberfläche für die Prozeßflußerstellung, die Ablaufsteuerung und die Datenverwaltung bildeten Zuverlässigkeit, Fehlertoleranz und sichere Bedienbarkeit.

Die Erstellung und Bearbeitung komplexerer Simulations- und Analyseaufgaben wird durch eine Klasse von Objekten unterstützt, die verschiedene Operationen wie *Design of Experiments* (Doe), *Response Surface Modeling* (RSM) und Optimierungsaufgaben in kompakter Weise beschreiben; darüber hinaus stellen sie einfache Datenverwaltungs- und Nachbearbeitungsfunktionen zur Verfügung.

Zwei Beispiele beleuchten die Anwendbarkeit und Gebrauchstüchtigkeit der vorgestellten Lösungen: Ein kompletter CMOS-Prozeß wird unter Zusammenspiel unterschiedlicher Simulatoren durchgerechnet, die Durchführung einer komplexen Optimierungsaufgabe wird anhand des Beispiels der Minderung des Kurzkanaleffekts auf die Schwellspannung eines NMOS Transistors mittels zusätzlicher *Pocket Implants* gezeigt.

Abstract

The design and fabrication of semiconductor devices in modern integrated circuit technology rely crucially on the numerical simulation of fabrication processes and device behavior for achieving competitive cycle times and reducing development costs for Very Large Scale Integration (VLSI) products. Technology Computer Aided Design (TCAD) comprises all software tools, strategies, and methodologies that support the development and verification of semiconductor processes and devices.

This dissertation presents a TCAD environment for the simulation of complete VLSI fabrication processes that emphasizes tool integration, process-flow representation, and task-level automation. VISTA/SFC is based on the Vienna Integrated System for TCAD Applications (VISTA), a TCAD integration and development framework, and extends concepts and mechanism found in VISTA towards the production use of the framework.

The integration of heterogeneous process and device simulation tools is realized by providing standardized interfaces between the a task control layer and external executables. External gridding tools are used to automatically ensure the consistency of the wafer model after each process simulation step. LISP functions are used to bind external executables to the environment. The Profile Interchange Format (PIF) is used as a primary wafer-data exchange format.

The representation of process flows is based on simulator-specific statements, with a mapping step being used to transform factory-specific and process-specific statements to appropriate tool calls. Execution of process flow experiments is done in parallel across the computing network, dynamic split-point generation minimizes the computation load. Robustness and easy of use have been major concerns in the design and implementation of the graphical user interface (GUI) for process flow definition, experiment control, and data management.

For the convenient definition and representation of more complex tasks, a class of objects has been added to the environment that encapsulate task-level applications like Design of Experiments (DoE), Response Surface Modeling (RSM), and optimization, and provide built-in data management and post-processing capabilities.

The applicability and usefulness of the implemented prototype is demonstrated by two examples. A complete CMOS fabrication process is simulated using a variety of heterogeneous simulation tools, and the encapsulation of a complex optimization task for the reduction of the short-channel effect on the threshold voltage of an NMOS transistor by means of an additional pocket implantation step is shown.

Acknowledgment

This thesis owes its existence to the help, support, and inspiration of many people. In the first place, I would like to express my sincere appreciation and gratitude to Prof. Siegfried Selberherr for his support and encouragement during the more than four years of this thesis's work. He has provided for an optimum working environment at the Institute for Microelectronics, where a lack of resources is something unimaginable due to his managerial skills and foresight. His uncompromising quest for excellence significantly shapes everyone at the institute. I am also indebted to Prof. Fritz Paschke, who has not only been a source of enthusiasm and encouragement over many years, but has also agreed to serve on my examining committee on very short notice.

The discussions and cooperations with all of my colleagues have contributed substantially to this work: Stefan Halama has bravely upheld the vision of VISTA as a truly open and generic TCAD framework, laying a sound basis for all future development; Richard Plasun has fearlessly attacked the most intricate of problems, drawing from a rich expertise in software engineering and numerics; Gerhard Schrom has been an inexhaustible source of process and device technology wisdom and a relentless warner of real life's dangers, traps, and pitfalls; Ernst and Karl Strasser have been long-time companions during the undergraduate and graduate years, shaping my mind during many technical and non-technical discussions along the way. Rudolf Strasser has put a lot of enthusiasm and energy into eliminating many shortcomings in various areas, closing the gap between academic research and practical applications; Walter Tuppa has set the pace of implementation with mind-boggling ingenuity and speed. To all members of the group I am very grateful for the cooperative spirit and the excellent working atmosphere, creating a unique setting for intellectual explorations as well as for culinary and Bacchic adventures. I also extend my appreciation to all staff members of the Institute for Microelectronics for their assistance and support.

I am very grateful to Masuda-san and his group at Hitachi's Device Development Center for giving me the opportunity to work and live in Japan for two months as a regular employee, for classes on Japanese society and culture, and for introducing me to the art of the tea ceremony; to Prof. Andrew Neureuther and his group at the Electrical Engineering and Computer Science Department of the University of California at Berkeley for accepting me as a visiting researcher for five months, and for many hours of mind-opening discussions on all kinds of TCAD aspects; to Prof. Jan Slotboom, Dr. Peter Stolk, and everyone at the Semiconductor Device Architecture Group at Philips Natlab at Eindhoven for offering me a trainee position, for a fruitful exchange of ideas on practical applications of TCAD, and for the tours to Zeeland and Delft.

Finally, I owe special gratitude to my family for continuous and unconditional support of all my undertakings, scholastic and otherwise.

Funding and Sponsors

This work has been supported by the following companies and institutions:

Austria Mikro Systeme, Unterpremstätten, Austria;
Christian Doppler Forschungsgesellschaft, Vienna, Austria;
Digital Equipment, Hudson, Massachusetts, USA;
Forschungsförderungsfond für die gewerbliche Wirtschaft, projects 2/285 and 2/299, as part of ADEQUAT (JESSI project BT1B), ESPRIT projects 7236 and 8002.
Hitachi Device Development Center, Tokyo, Japan;
Motorola APRDL, Austin, Texas, USA;
National Semiconductors, Santa Clara, California, USA;
Philips Research, Eindhoven, The Netherlands;
Siemens EZM, Villach, Austria;
Siemens, München, Germany;
Sony, Atsugi, Japan.

Anyone who leaves behind him anything in writing and likewise anyone who takes it over from him supposing that such writing will provide something reliable and permanent would be a fool.

Socrates, *Phaedrus*

Contents

List of Figures

List of Tables

Terminology

1-D simulation: One-dimensional simulation. In TCAD, refers to the numerical analysis of semiconductor fabrication processes and devices with all physical quantities assumed to be constant in all directions except for the vertical direction perpendicular to the wafer surface.

2-D simulation: Two-dimensional simulation. Inquiry into the nature and causes of the behavior of a semiconductor wafer (during process simulation) or a semiconductor device (during device simulation) with all quantities assumed to be constant in a direction parallel to the wafer surface. The cross-section under scrutiny is a plane perpendicular to the wafer surface, usually cutting through the active area and the electrodes of the device.

1-D/2-D simulation: Combines a set of one-dimensional (and two-dimensional) simulations to build a two-dimensional model.

Chip: A minute square of thin semiconductor material cut off from a processed wafer. Most valuable when carrying an integrated circuit.

Device: Semiconductor device that is the basic component of an integrated circuit. Most of the times, a transistor.

Equipment: Piece of equipment found in semiconductor fabs used to apply treatments onto wafers.

Fab: Fabrication unit for manufacturing semiconductor devices. Comprises a clean room and numerous pieces of equipment; very expensive (new ones start at one billion dollar (US)).

Flow: Sequence of processing or simulation statements to be carried out one after the other to fabricate or simulate a device.

Integrated Circuit: Electronic circuitry realized on a single semiconductor chip. Usually comprises a lot of transistors (up to several million), but rather few inductances. Most prominent examples are microprocessors and memory components.

LISP: Interpreting programming language primarily used in artificial intelligence and CAD. Its basic data types are lists, hence the name, and atoms, which are not lists.

Threshold Voltage: Characteristic voltage applied to the gate electrode of a MOS transistor that marks the transition from an *off* state to an *on* state. Definitions for real devices vary from company to company. Internally, it indicates the onset of strong inversion in the device [Wol95a].

Tool: Simulator or other computer programm that serves a similar purpose for the ends of the user as does, e.g., a hammer for a carpenter. More specifically, a process, device, capacitance, or circuit simulator.

Wafer: Piece of semiconductor material that is gradually transformed into an electronic device during fabrication.

Widget: Building block of graphical user interfaces.

Yield: Ratio of chips accepted over chips started. The higher, the better, usually around 90% (for memory products) to 60% (for microprocessors).

Acronyms

ASCII	American Standard Code for Information Interchange
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BPFL	Berkeley Process Flow Language
CAD	Computer–Aided Design
CAD	Computer–Aided Manufacturing
CFI	CAD Framework Initiative
CIM	Computer–Integrated Manufacturing
CMOS	Complementary MOS
CMP	Chemical Mechanical Polishing
CPU	Central Processing Unit
CVD	Chemical Vapor Deposition
DoE	Design of Experiments
ECAD	Electronic CAD
EVE	Evaluable Entity
EVI	Evaluation Instance
GUI	Graphical User Interface
HDX	Hierarchical Data Exchange
IC	Integrated Circuit
LAT	Large-Angle Tilt
LDD	Lightly Doped Drain
LISP	List Processing
LM	Levenberg-Marquardt
LOCOS	Local Oxidation of Silicon
MOS	Metal–Oxide–Semiconductor
NFS	Network File System
PAI	PIF Application Interface
PAL	PIF Application Layer
PBFM	PIF Binary File Manager
PED	PIF Editor
PFR	Process Flow Representation
PIF	Profile Interchange Format
PIL	PIF Interface Layer
PLB	PIF Logical Binary File
PVD	Physical Vapor Deposition
RIE	Reactive Ion Etching

RPC	Remote Procedure Call
RSM	Response Surface Model
SFC	Simulation Flow Control
SIMS	Secondary Ion Mass Spectrometry
SPR	Semiconductor Process Representation
SRP	Spreading Resistance Profiling
STI	System Tool Interface
SVG	Simple Vector Graphics
SWR	Semiconductor Wafer Representation
TAC	Tool Abstraction Concept
TCAD	Technology CAD
TIF	Technology Interchange Format
VISTA	Vienna Integrated System for TCAD Applications
VLISP	Vienna LISP
VLSI	Very Large Scale Integration
VOS	Vienna Operating System
VUI	Vienna User Interface
WWW	World Wide Web

Chapter 1

Introduction

Semiconductor manufacturing relies to a large extend on computer simulation of fabrication processes and devices to reduce costs associated with experimental runs and to keep development times short.

1.1 Semiconductor Technology

The design and fabrication of an integrated circuit is a complicated endeavor [Bon91]. Modern Very Large Scale Integration (VLSI) technology uses fabrication processes that consist of several hundred process steps being applied one after the other to the raw semiconductor wafer to manufacture an integrated circuit. The development of new device generations continuously pushes the limits of knowledge of the physics involved, at the same time making equipment and fabrication installations more expensive and increasing the costs of experiments.

To cope with these challenges, semiconductor technology makes extensive use of computer-aided design (CAD) methodologies to reduce technology development costs and to improve product quality. Figure 1.1 gives an overview of the realms of Electronic CAD (ECAD), Technology CAD (TCAD), and Computer-Integrated Manufacturing (CIM) in VLSI manufacturing.

1.1.1 ECAD – Electronic Computer Aided Design

ECAD is concerned with the design of an integrated circuit in terms of schematics, netlists, and layout. An integrated VLSI circuit consists of up to several millions of active devices that are interconnected on a chip to implement the desired functions, i.e., to build a memory module or a microprocessor. The layout contains a description of the entire integrated circuit, including all active devices and interconnect structures, in terms of material layers[1]. Circuit simulation uses a netlist of the circuit, together with compact models of the electrical components, to compute the behavior of the integrated circuit. Model parameters are

[1] *Material layer*, in this context, also includes local material modifications such as implantation regions.

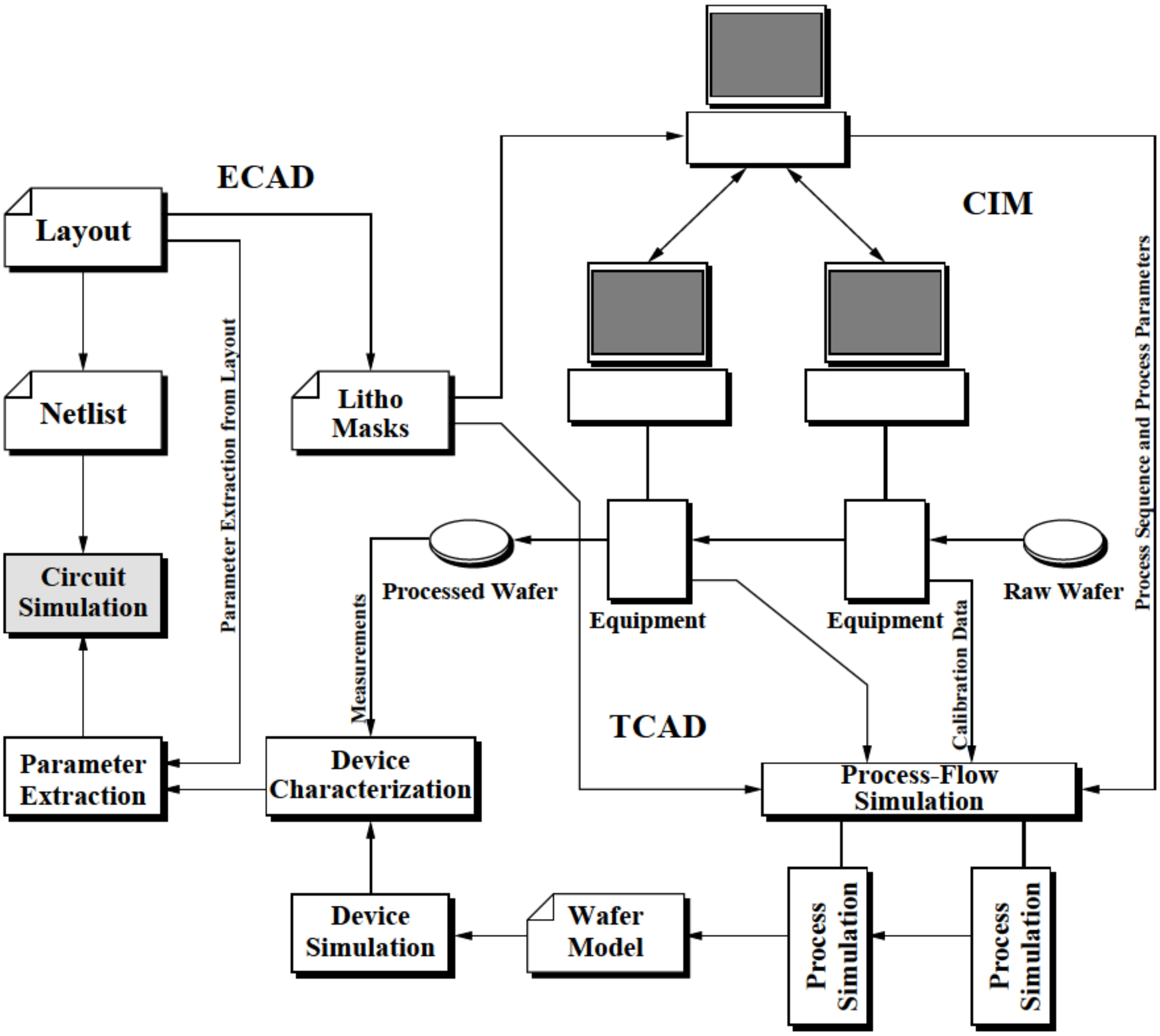


Figure 1.1: CIM, ECAD, and TCAD in VLSI manufacturing. Feedback loops are not shown.

extracted from device data obtained from measurements or from simulation. Some parameters – mostly concerned with interconnects – can be directly inferred from the layout[2]. The output of circuit simulation is used to verify the entire design. From the layout, lithography masks are generated to be used in fabrication and process simulation.

1.1.2 CIM – Computer Integrated Manufacturing

CIM takes care of the smooth, automated operation of the fabrication unit and all equipment. Raw wafers are processed by applying treatments – thermal treatment, ion implantation, gas atmospheres, etc. – to induce physical and chemical processes to alter the material surface structure and the distribution of dopants in the material bulk. Up to several hundred treatments are applied sequentially to each wafer before an integrated circuit is completed. Most fabrication equipment is controlled by workstations that manage recipes, monitor equipment state and operation, and detect malfunctions. The location of each wafer lot in the fab at any given moment is tracked automatically together with the processing history and the remaining process flow.

1.1.3 TCAD – Technology Computer Aided Design

TCAD uses process flow information and layout data to simulate the fabrication process as well as the device behavior prior to actual fabrication to determine viable technologies and to improve existing ones. For each fabrication step, numerical simulation tools are used to compute the changes of the wafer model in accordance with the real-world effect. The final wafer model contains the complete device structure including material layers and dopant distributions. On a two-dimensional or three-dimensional sub-domain of wafer model, the semiconductor equations are solved to compute the electrical behavior of the device at various operating points and under different conditions. The extracted device characteristics are the basis for optimizations of the fabrication process as well as of the circuit design.

As a single wafer takes many weeks to be fabricated, semiconductor process and device design can substantially benefit from the use of simulation and modeling to reduce the cost and time required to develop new or extend existing technology [Mar93] [Mar96]. A concise introduction into the use of simulation in semiconductor technology development is given in Cole et al. [CBF$^+$90].

As the complexity of the processes is steadily increasing – with respect to the number of fabrication steps as well as to the physics involved when going to smaller device dimensions –, the demands on simulation are constantly growing. At the same time, competitive markets call for shorter product development cycles. To help TCAD keep the pace with these trends, TCAD frameworks are used to share common tasks among simulation tools and applications, organize the management of simulation data, and assist the process and device engineers with tasks that are beyond the context of a single TCAD tool. For a brief evaluation of design frameworks in general, see Kleinfeldt et al. [KGMB94]; for a summary of TCAD frameworks, refer to Halama [Hal94]

[2] If additional technology parameters – sheet resistances, layer thicknesses, etc. – are supplied.

1.2 The VISTA/SFC TCAD Environment

In this work, an integrated environment for TCAD analysis is presented that intends to support a wide variety of TCAD tasks from tool development and calibration to process flow simulation and design optimization. Special emphasis has been put on using the generic concepts and mechanism found in VISTA (see Section 1.2.1) to support process and device engineering tasks in actual industrial settings. Although a thorough treatment of all aspects of the concepts, design, and implementation is not feasible, the points of main concern are discussed.

1.2.1 The VISTA Project

This work has been carried out as part of the VISTA project [HFF$^+$93] [FTS94] [HPR$^+$95] and continues a series of doctoral theses on the VISTA framework's architecture [Hal94], on its data level [Fas94] [Rie96], and on its application to CMOS process optimization tasks [Pim93]. CASE tools, system libraries, and the LISP interpreter used in VISTA are treated in [Tup96]. Examples of process simulators developed in the VISTA project can be found in [Str94] and [Puc96]. A graphical editor for defining semiconductor devices in two and three dimensions is described in [Rie96].

1.2.2 Outline

The following chapter gives a more detailed account of the application of TCAD in industry and presents typical TCAD tasks and tools. Chapter 3 outlines the architecture of the VISTA/SFC simulation environment and discusses basic concepts and mechanisms. In Chapter 4, the tool integration methodology adopted is presented. Chapter 5 focuses on the representation of semiconductor process flows for simulation purposes, Chapter 7 deals with the management of simulation data for TCAD experiments. Chapter 8 presents the task level management and encapsulation mechanisms available in VISTA/SFC. A uniform representation of various TCAD tasks is described.

Chapter 9 sketches two typical TCAD analysis and synthesis tasks. Section 9.1 demonstrates the simulation of a complete CMOS fabrication process with heterogeneous simulation tools, Section 9.2 shows the automated interplay of task-level components to optimize a given process for a reduced short-channel effect. Finally, this thesis concludes with a brief discussion and gives some suggestions for future work.

Chapter 2

TCAD Scenario

Industrial TCAD applications cover a broad range of analysis and synthesis tasks aimed at characterizing fabrication processes, tuning device performance, improving yield numbers, and evaluating and selecting technology alternatives. The role of TCAD and TCAD applications differ from company to company, ranging from the characterization of critical process sequences in a stand-alone fashion using manually written simulator scripts to fully integrated modeling and simulation of actual factory-floor processes and procedures. However, the fact that the number of experimental wafers that can be processed is being steadily reduced by escalating equipment costs and more complex and therefore increasingly time-consuming processes makes TCAD an indispensable methodology for continuing progress in semiconductor development [Mar96].

Figure 2.1 sketches VLSI design loops from circuit specification to process design. The innermost design loop[1] is based on device fabrication and represents the most expensive tasks. The target process has to be tuned to yield devices matching the device specifications derived from circuit design. Circuit modeling reads device parameters extracted from device data to simulate the behavior of the target circuit prototype. Deviations from the circuit specifications – themselves being imposed by system and product design – lead to an adjustment of the circuit prototype.

This chapter gives an overview of typical TCAD tasks found in the semiconductor industry and in research laboratories together with a brief summary of applicable TCAD tools and methodologies.

2.1 TCAD Tasks

TCAD tasks range from unit-process analysis and device characterization to yield optimization and technology selection. With tools and application strategies gradually maturing, emphasis is being shifted from point-tool applications to fully integrated and automated VLSI design modeling. Two dimensional simulation is the regular case, although shrinking

[1] Not shown in the picture are loops at equipment level used to adjust equipment settings according to process step specifications. See [SGHH91] for an example of an equipment-level process control system.

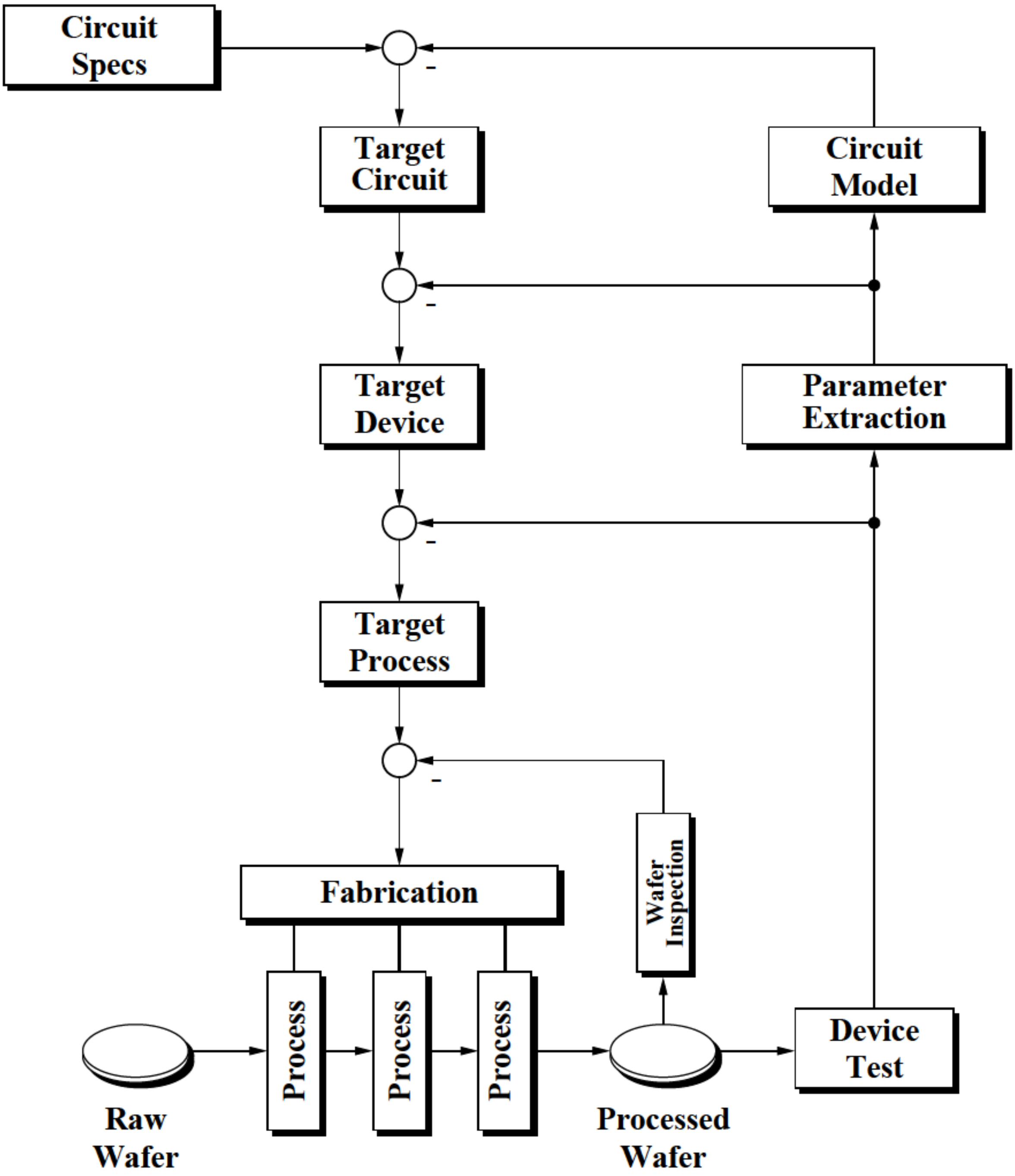


Figure 2.1: Nested VLSI design loops. Circuit specifications establish constraints for circuit design, calling for specific device characteristics to be realized by process design.

device dimensions[2] and topologically more complex designs render three-dimensional simulation more and more important. Mixed one-dimensional and two-dimensional simulation is still being used in cases where trends have to be analyzed quickly or extremely expensive models are being used.

2.1.1 Unit Process Analysis

The traditional application of TCAD focuses on the analysis of unit fabrication processes used in VLSI manufacturing. All of the basic physical processes, including ion implantation, diffusion and oxidation, epitaxy, lithography, and etching and deposition, have been successfully modeled and implemented as numerical simulation modules, and form an indispensable basis for all further TCAD applications [Sel84] [Eng86] [Sze88] [DY93] [Mey95]. The objectives for unit process modeling are twofold:

1. Numerical simulation provides insight into the physical nature of fabrication processes that are not observable otherwise. A prominent example is given by the simulation of ion implantation, where the spatial distribution of dopants in the semiconductor material can only be measured with a spatial resolution far coarser than the typical feature size in modern devices. Therefore, a picture of the two-dimensional and three-dimensional dopant distribution can only be found by simulation.

2. The effects of variations of process settings for new or unfamiliar device geometries, for new materials or etchants can be most conveniently studied by process simulation. Unit processes can be tuned to a nominal target behavior before being grouped together with other process steps to form process modules for fabrication.

2.1.2 Device Analysis and Characterization

The numerical simulation of semiconductor devices represents the most important means of characterizing and evaluating both fabrication processes and device designs. Based on a numerical model of the device structure, including wafer geometry and dopant distribution, the distribution of the electric potential, the current density, carrier densities, and the temperature are computed as a function of externally applied voltages. Most types of devices can be simulated; available simulation tools range from simulators specialized on certain device types [SSP80] to general semiconductor structure simulators [Fis94] [SKRS95] [Sim96]. The fact that the distribution of physical quantities in the semiconductor device is not observable in nature due to the limited resolution of measurement techniques holds even more for device operation. Simulation provides the only means for looking into the active device. For example, [RSS95] treats the analysis of the transient behavior of charged-coupled devices.

2.1.3 Interconnect Simulation

The increasing complexity of integrated circuits causes metalization layers to become more and more important, as more complex circuitry needs more interconnections and therefore

[2] According to [Dav96], channel lengths are expected to reach $0.05\mu m$ by 2004, gate oxide thicknesses $2.5nm$. See [Ass94] for a summary of expected trends and developments in semiconductor technology.

a larger number of more densely packed interconnect layers. Together with the increase of chip size, the electrical behavior of the interconnecting layers gets more and more important for the operation of the whole circuit. Interconnect simulation analyzes the capacitances between different parts of the interconnect system to predict dynamic load and propagation delay. Simulations are performed in two or three spatial dimensions, sometimes including the non-linear behavior of the semiconducting material. Capacitances and resistances obtained from interconnect simulation are used in circuit simulation – together with data for the active devices – to compute the electrical behavior of the whole system.

2.1.4 Circuit Simulation

Although not part of the traditional TCAD domain, circuit simulation forms an important part of the VLSI design loop as it establishes an interface between the "vertical" and the "lateral" analysis of semiconductor devices. From the electrical behavior obtained by device simulation, device model parameters are derived and used in models of electrical circuits to verify the circuit design.

2.1.5 Process Tuning

Combining process and device simulation to compute the fabrication and operation of semiconductor devices allows to study the influence of process parameters on the device characteristics. In a typical industrial environment, circuit designers ask for a certain device behavior that has to be met – within a certain range – by the devices leaving the fabrication unit in order for the integrated circuits to work properly and meet their respective specifications[3]. By adjusting the process flow sequence and process parameters[4], process design aims at ensuring that a large fraction of all devices fabricated fall into the acceptable range. Due to the extremely high costs of real-silicon experiments and the notorious lack of time and man-power in a highly competitive industry, the necessary parameter adjustments are frequently done in the virtual world of simulation, with only a very small number of actual experimental wafer runs granted to the process designers.

2.1.6 Calibration

For simulators to accurately predict the results of fabrication process steps as well as the device behavior, calibration is of utmost importance. By adjusting certain *calibration parameters*[5], the computed results can be made to agree with measured data over a certain range of variation in the process parameters and operation conditions. In general, the more accurate the result has to be, the smaller is the valid range of a calibration. Only calibrations and uncalibrated model together form a valid model for a given application. Depending on the

[3] Traditionally, circuit design has been at the demanding side, while process design strives for fulfilling these demands.

[4] A modification of the process sequence is opted for only very reluctantly, as it introduces additional degrees of freedom; existing process flows are used as long as possible.

[5] Calibration parameters are either explicitly provided by the simulator, or they may be introduced artificially in a pre-processing fashion.

level of detail that is of interest as well as on the data available, calibration can either be performed locally on the tool level or globally for the complete process.

Tool Calibration

Provided measured data and a calibration mechanism exists for a tool, single simulation tools can be tuned to model the actual behavior of fabrication processes and devices in a given fabrication setting. Figure 2.2 shows this approach for a sequence of three tools, each being calibrated separately. In principle, the individual calibration of each simulator is most desirable, because in this fashion, tools can be defined[6] that can be used at any position of the process flow as well as in different contexts. Unfortunately, process interdependencies as well as lack of sufficient experimental data make this strategy rather cumbersome.

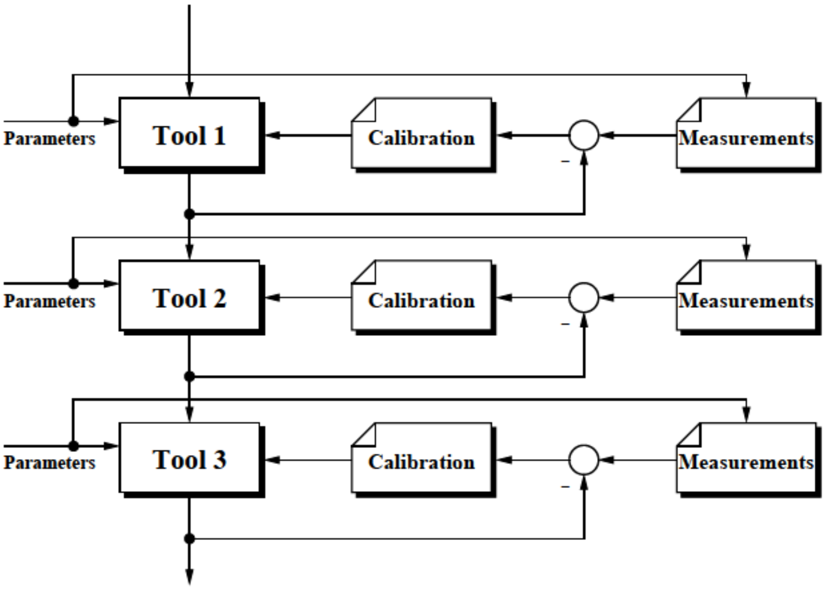


Figure 2.2: Local tool calibration. The parameter settings are used to select corresponding measurement data during calibration.

Global Calibration

In most cases, the exact numerical solutions of intermediate steps during process simulation are not of pronounced interest to the process designer. Rather, overall trends of the system behavior should be modeled accurately to guide design decisions. To make the simulation results comparable to target specifications, only the very last stages of the simulation task – usually device simulation – need calibration. Figure 2.3 shows the principal mechanism for globally calibrating a sequence of tools. All calibration data is subsumed in a single

[6] In the sense of a combination of generic tool plus calibration.

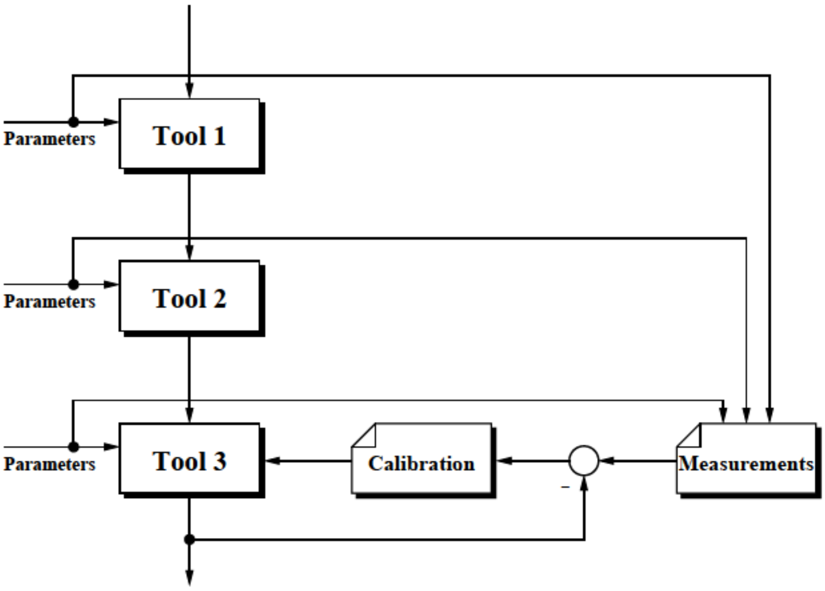


Figure 2.3: Global tool calibration. Combined parameter settings of all tools together are used for fitting with measured data during calibration.

set of calibration data that encapsulates all previous – potentially unknown – calibration information.

In a more general approach, the complete sequence of tools can be replaced by a single model – a response surface model, table lookup, or other – that is calibrated for some sets of input parameter values. Figure 2.4 show the principle structure of global modeling and calibration.

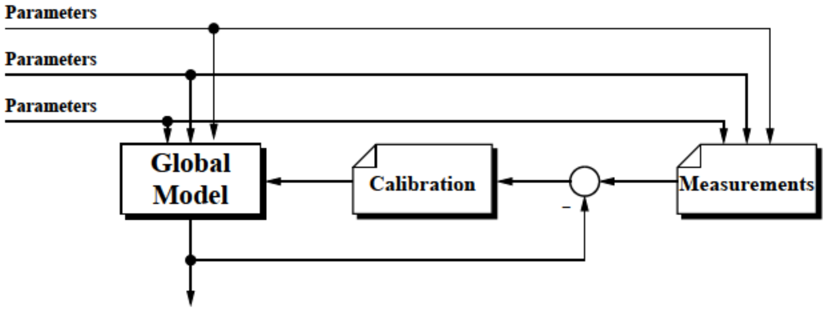


Figure 2.4: Global calibration of global model. The complete system is represented by a single model.

Experimental Data

One of the main complications in calibrating models and simulation tools is caused by the fact that in many cases, experimental data are not easily obtained. The numerical results are usually much more accurate than available measurements, with respect to both the spatial solution as well as the significance of the readings.

2.1.7 Sensitivity Analysis

A typical process has a large number of control parameters that can have a significant impact on the performance of the circuit as well as on the yield of the product. To establish a basic knowledge of the behavior of a design under variation of input parameter values and to estimate the relative importance of the input parameters, sensitivity analysis applies small changes to the nominal values of input parameters according to an axial design scheme. For measurements and numerical simulation, variations of the input parameter values cannot be made infinitely small. The sensitivity of an output variable x_i with respect to an input parameter x_j is therefore defined as $\Delta x_i/\Delta x_j$, with Δx_j sufficiently small. In order to compensate for highly nonlinear system responses, it is useful to transform the input parameters appropriately before computing sensitivity numbers. For example, the dose parameter of most ion implantation steps exhibit a logarithmic influence on the device behavior; to obtain comparable sensitivity data, Δx_j is computed from $\log x$ instead of x, x being the dose setting in cm^{-2}.

2.1.8 Design Centering and Yield Optimization

As opposed to the “ideal” world of computer simulation, where numbers are exact, the real fabrication process has to deal with fluctuations and variations in all parameters. Therefore, for a given process and given process parameters, the measured electrical characteristics of the fabricated devices vary within certain ranges[7]. These variations are modeled as appearing at the input of the fabrication units, e.g., a statistical variation is assigned to a process parameter, with the system itself assumed to behave completely deterministic. The parameters of these statistical variations can be obtained from long-term measurements in the fabrication unit. Given a model of the fabrication process – either a sequence of process simulators of a global model –, a large number of experiments can be generated from the nominal values and the statistical parameters[8]. In order to optimize the yield of the product, the fraction of experiments that leads to results that fall into the manufacture acceptance window has to be maximized. By adjusting the nominal process parameter values, the design is centered with respect to the output distribution. In practical applications, the system is represented by a response surface model, and experiments are generated by using the Monte Carlo method[9]. Figure 2.5 shows the experimental loop used for design centering.

[7] So do, of course, the processing results after each fabrication process steps. As these effects are, in the general case, statistically uncorrelated, only the overall impact is of interest.

[8] See [AAY$^+$88] for the application of statistical methods to VLSI design.

[9] See [HH83] for an introduction to Monte Carlo methods.

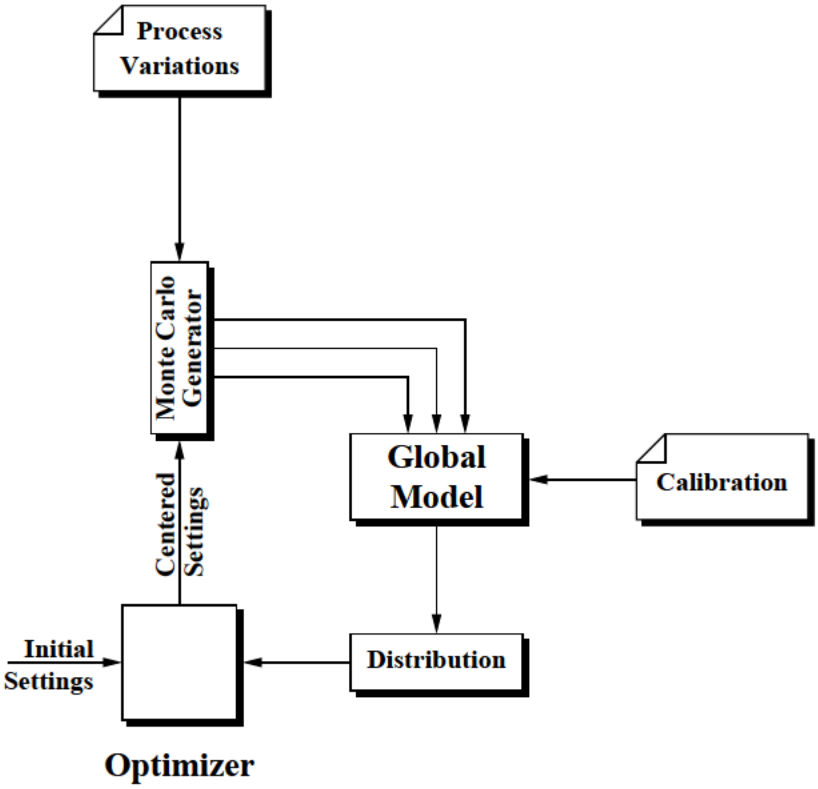


Figure 2.5: Design centering using Monte Carlo method. An optimizer centers nominal process parameter settings to maximize the number of experiments generating acceptable results.

2.1.9 Device Matching

Circuit design requires devices of different types – NMOS and PMOS transistors – to be closely matched in order to be used in complementary circuits. Matching devices exhibit an identical electrical behavior when discounting for the sign of the electrical currents and voltages. More precisely, the respective threshold voltages are identical. Due to process variations and different process sequences[10] for the N and P type devices, the distributions of these electrical parameters do not coincide automatically. In a fashion similar to the one described in the previous section, statistical simulation can be used to establish a set of nominal process parameters that minimizes statistically the differences between the two groups.

2.1.10 Technology Development and Evaluation

At the early stages of the development of a new process technology, unexplored territory has to be probed to limit the number of potential technology alternatives and select the most promising one. Although no accurate calibrations exist at this stage of process development, the existing body of information can be used to extrapolate process characteristics to a new technology or generation [Mas95] [Mar96].

2.1.11 Process Synthesis

Given the layout of a device, the sequence of process steps required to build the material layers needed can be determined automatically [YGM94] [HM96]. A number of possible solutions can be obtained, that are fed into process and device simulation to select the best result.

2.1.12 Inverse Modeling

As mentioned above, process and device simulation often provide the only means of looking into the semiconductor material to determine the distribution of physical quantities like dopant concentration or carrier density. Given a "real" device, one can combine these techniques to indirectly measure two-dimensional dopant profiles[11] by matching electrical measurements obtained from experiments and simulation [Kha95] [KFBS95]. Figure 2.6 shows the optimization loop identifying a doping profile by means of electrical device characterization and inverse modeling. Given a parameterized model of the doping profile, an optimizer tries to minimize the deviation of the simulated device parameters from measured data by adjusting the parameters of the doping model. The model may be an implantation simulator, a sequence of process simulation tools, or an analytical model of the doping profile. In the latter case, care has to be taken not to introduce false or unjustified assumptions on the nature of the solution sought.

[10] In the sense that some implants are only applied to one type of device.

[11] As of now, no measurement techniques exist for determining the two-dimensional distribution of dopants in the wafer material.

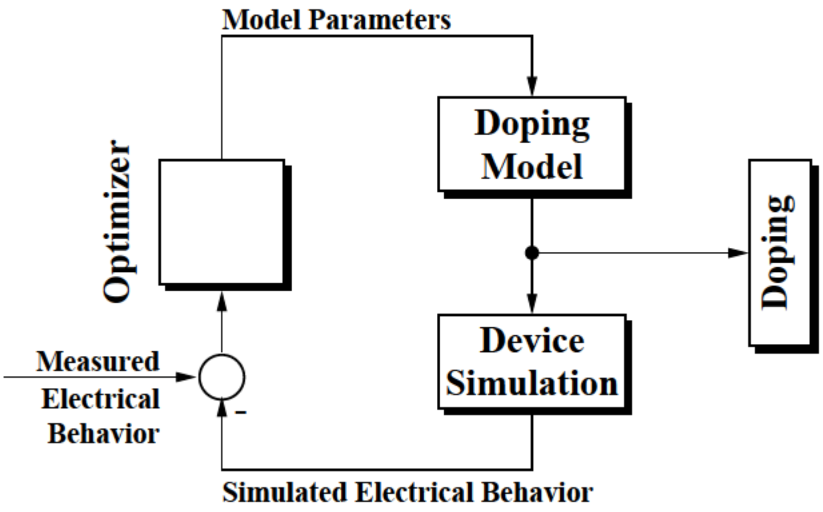


Figure 2.6: Inverse modeling of doping profile.

2.2 TCAD Tools

A large variety of process and device simulation tools exist in the TCAD realm, each having been designed for special purposes and applications. Simulators model processes and devices in one dimension, two dimensions, and three dimensions. Process simulators are either geared towards the detailed analysis of single process steps or towards modeling complete process flows. The application of tools to a TCAD task is supported by frameworks, that cover data management and automation aspects.

2.2.1 Point Simulators

Point simulators focus on the accurate and sound modeling of single fabrication processing steps and are used primarily to analyze advanced effects. They do not intend to support process integration tasks, but to gain insight into the underlying physical mechanisms of treatments and processes.

2.2.2 Multi-Step Simulators

Multi-step simulators or process flow simulators emphasize complete process flow simulation capabilities and usually neglect to a certain degree the physically sound treatment of every single one of the available individual process steps. This approach is in good agreement with the fact that even the most advanced tools need to be calibrated in order to deliver results matching the “real life” situation; trends are often enough to characterize a technology.

2.2.3 Response-Surface Models

Response-surface models (RSM) provide analytical models for arbitrary systems, with the dependency of each output variable on the set of input parameters being formulated as an analytical function. The most common RSM use polynomial functions of second order, although in principle, any function can be used[12].

2.2.4 TCAD Frameworks and Virtual Fabs

While TCAD frameworks try to support different aspects of TCAD applications, with emphasis on the tool control, experiment generation, and data handling tasks, virtual fabs aim at modeling the complete VLSI design cycle, from single process simulation to statistical analysis and optimization. Widely used applications include PDF Solution's pdFab [WKKS93], Silvaco's MASTER framework [HB93], and TMA's CAESAR [AGJ93]. NORMAN/DEBORA [CBK$^+$93] offers powerful experiment generation capabilities for response-surface modeling.

[12] See, for instance, [SC96] for a completely unstructured alternative to polynomial modeling.

Chapter 3

The VISTA/SFC TCAD Environment

3.1 Introduction

The VISTA/SFC TCAD environment has been designed to support the development of semiconductor processes and devices at different stages of the design process. It has evolved from early attempts based on the VISTA TCAD framework [SFF+91] [Fas94] [Hal94] to create a tool for controlling and automating the sequential execution of semiconductor process simulation tools [PS93a] [PS93b]. At the present stage of development, it comprises simulation tool integration, process flow representation, process flow simulation, and task-level automation for optimization, calibration, and response-surface modeling tasks. The remainder of this chapter gives an account of the objectives and the overall architecture of the simulation environment.

3.2 Objectives

3.2.1 Tool Integration

At the early stages of development, the primary goal of the Simulation Flow Control module (SFC) [PS93b] was to use a set of single-process simulation tools like PROMIS[1] and SAMPLE [Ele91] to automatically simulate complete VLSI fabrication processes, the main concern being the integration of process simulation tools by providing uniform interfaces for the exchange of simulation data and for controlling the execution of the tools. Chapter 4 presents in detail the strategies adopted in VISTA/SFC for coupling heterogeneous process and device simulation tools.

[1] Initially, the PROMIS process simulator [HPW91] [HHW+90] was capable of simulating fabrication process sequences including ion implantation and diffusion processes. In order to make different numerical and process-modeling related components independent from each other and to foster the modularization of the simulator, it has been divided into a set of tools each modeling only a single fabrication process [Sti93] [Puc96].

3.2.2 Experiment Management

In addition to simulating sequences of fabrication processes, the support of large-scale simulation experiments plays another important role in the TCAD domain. Simulation experiments are an indispensable aid in the development of VLSI processes and devices for studying the influence of parameter changes on the behavior of the fabricated device. Typical tasks involve about five to ten independent process parameters that are varied independently of each other, leading to up to several hundred simulation experiments for every system aspect under consideration. Due to the high costs[2] of two-dimensional and three-dimensional simulation, the number of simulator calls – the actual invocation of an executable to numerically simulate a given fabrication process or device characteristic – submitted to a workstation or other type of computer needs to be made as small as possible, with automatically enforcing the reuse of existing results where possible. Tool control and data management necessary for such experiments in a fashion suitable for application in an industrial setting constitutes another core part of this work (cf. Chapter 6).

3.2.3 Task Automation

The generation of a set of design-space sample points by process and device simulation experiments forms the basis for further analysis. As described in Section 2.1, various TCAD tasks aim at characterizing a VLSI process in terms of statistical behavior and sensitivity to uncertain control parameter values as well as at tuning its performance to specifications derived from chip design and management. Furthermore, TCAD creates a number of tasks on its own dealing with the calibration of simulation tools and the generation of response surface models, to name a few. To liberate the TCAD engineer from defining and performing these tasks manually[3], automation and programming support on the task level are a major concern for a TCAD environment. VISTA/SFC offers *evaluable entity* (EVE) objects that establish an object-oriented approach to many task-level tasks; they shall be addressed in more depth in Chapter 8.

3.3 Architecture and Components

The VISTA/SFC simulation environment comprises several internal modules as well as external applications[4]. Figure 3.1 gives an overview of the TCAD environment's principal functional components.

3.3.1 Task Control Layer

As the central coordinating instance, the *task control* layer takes care of controlling all activities initiated via the GUI, the ASCII interface, or a batch file. It is implemented as a set of dedicated VLISP functions and establishes object-oriented interfaces for all task-level

[2] In terms of CPU time, disk space, network traffic, and user patience.

[3] I.e, manual setup of scripts for optimization, design of experiments, etc.

[4] *External applications* run as separate processes on the operating system.

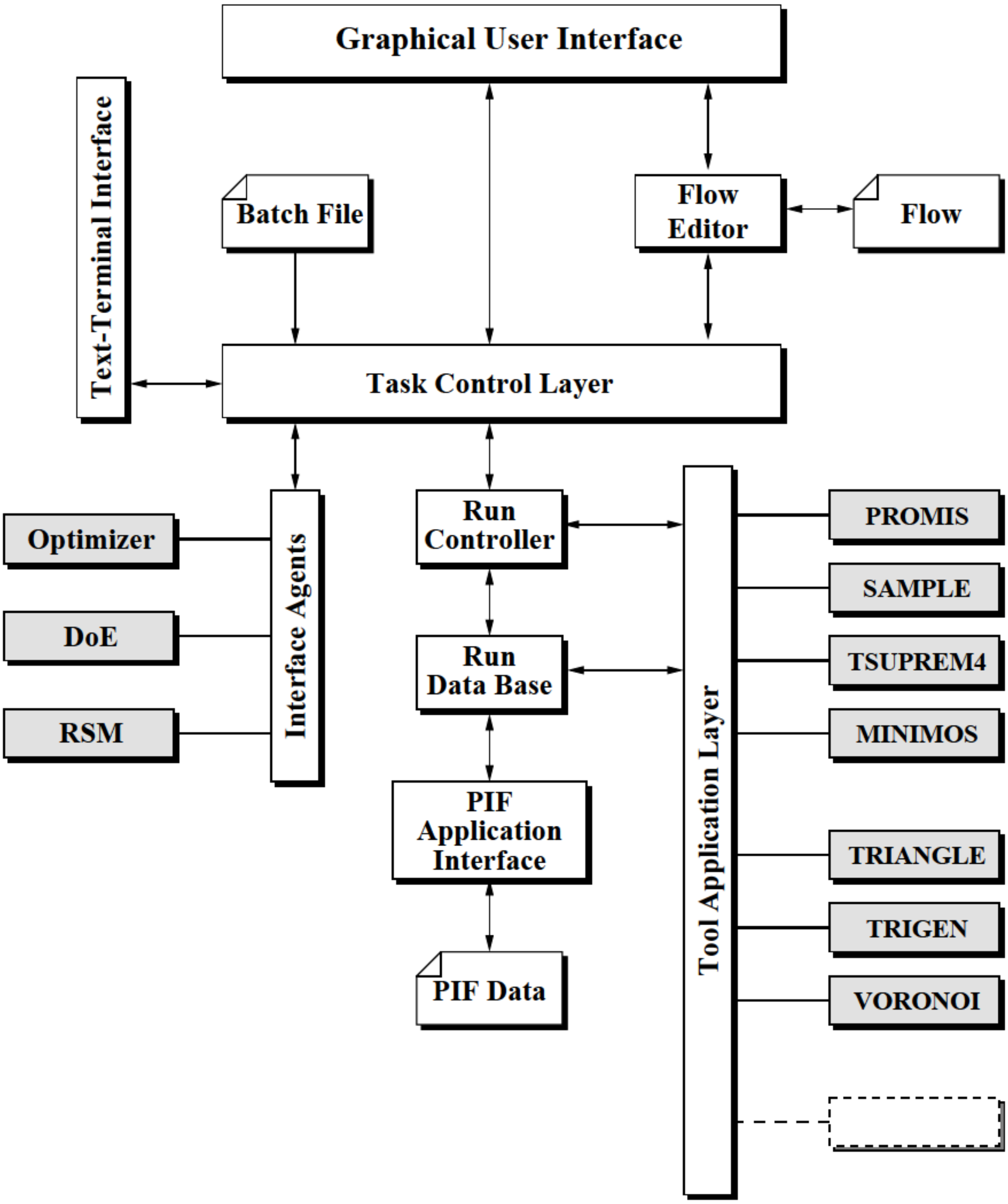


Figure 3.1: Main components of the VISTA/SFC TCAD environment. Dark rectangles indicate external executables.

services. The VLISP *shell interpreter* – not shown in the figure – provides the basis for the implementation of all other internal modules. It provides interfaces to the operating system, the graphical user interface (GUI), and the PIF Application Interface (PAI) [FFS$^+$91] to conveniently access simulation data stored in the Profile Interchange Format (PIF) [Duv88]. All operating-system dependent services are encapsulated by the VLISP interpreter, which ensures portability over a wide variety of operating systems and platforms; see below in Section 3.6 for more details.

3.3.2 Simulation Flow Control

The *flow editor* offers an intuitive and convenient graphical interface for writing process flows. It supports the definition of process flows in a hierarchical and modular manner in terms of tool-independent process statements as well as explicit tool statements. Chapter 5 discusses the capabilities of the flow editor as well as the highlights of the process flow representation used. Process flow information is interpreted by the *run controller*, which together with the *run data base* forms the core components for the management of iterative and parallel split-lot experiments. The run controller takes care of the detection of splits, of scheduling multiple runs in parallel operation on workstation clusters, and offers a number of operation modes to facilitate development and debugging of both processes and simulation tools. The run data base stores and retrieves simulation output data and extracted data of any format, with the PIF format being used as primary exchange format for wafer data.

3.3.3 Tool Binding

All simulation tools and auxiliary data-manipulation tools and gridders[5] are accessed through a layer of *binding functions* that encapsulate tool specifica, establishing a set of VLISP functions to allow the invocation of all simulation tools in a uniform manner as simple function calls. All tool parameters as well as output and error redirections are passed as arguments to the binding function, ensuring a maximum of independence of the tool binding layer and the task control layer. All binding functions together are organized as the *tool application layer* that makes all tools available to the task control layer.

At the other end of the functional hierarchy, task-level tools are interfaced with the task control layer by interface agents that establish communication channels with concurrent executables like optimizers, design-of-experiment (DoE) tools, and response-surface-modeling (RSM) modules. While being run as clients of the task control layer, they also operate as servers for more complex applications, and are linked together by callback[6] connections. Interface agents are fully integrated with the EVE class of objects; see below in Section 3.5 for more details.

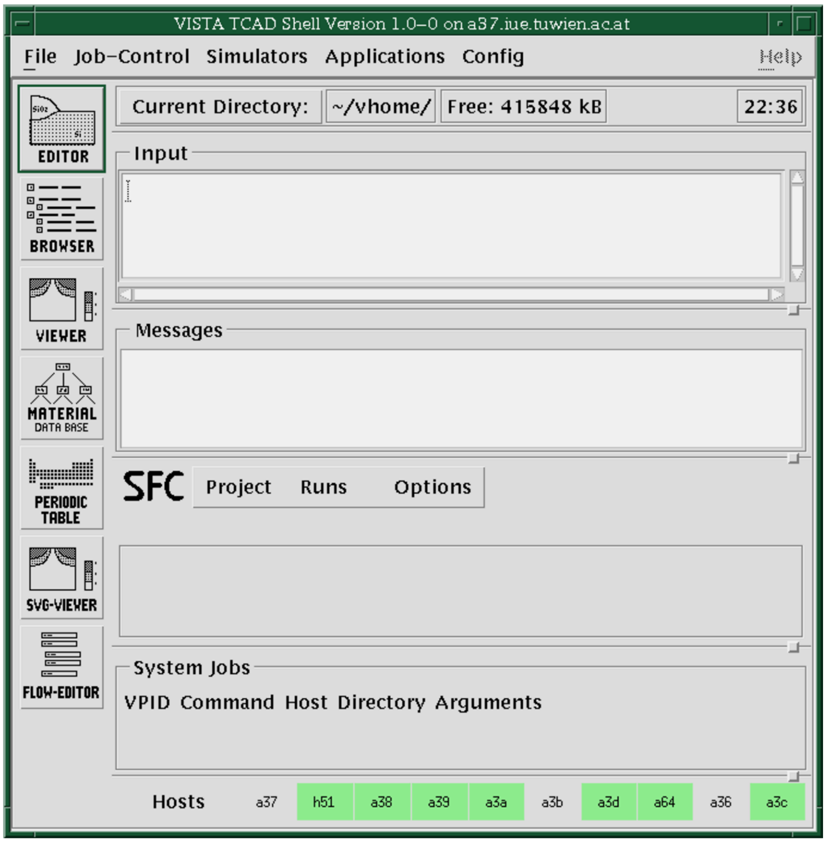
[5] VISTA/SFC uses external gridding tools to ensure data consistency after simulation steps; see Section 4.3.
[6] See Appendix B.

3.4 VISTA/SFC and the VISTA Framework

While the VISTA framework as described in [HPR+95] provides a host of functionalities and services for a variety of simulation tasks in the TCAD domain and elsewhere, implementing, e.g., abstractions of the operating system and the X Window-based graphical user interface to ensure portability across platforms, as well as providing a PIF data base interface and higher-level libraries for PIF data manipulation, and additional support for tool development, VISTA/SFC bridges the gap between generic versatility and practical application.

3.4.1 Graphical User Interface

The VISTA/SFC graphical user interface as shown in Figure 3.2 intends to make all tools and services available directly at the user's fingertips[7]. It is subdivided into several windows that



Current Directory: ~/vhome/ Free: 415848 kB 22:36

VPID Command Host Directory Arguments

Figure 3.2: Main GUI shell of VISTA/SFC.

can be made visible and invisible by the user according to the current task. The *Input* window

[7] The level of task abstraction presently supported by the VISTA/SFC GUI does not completely reflect all available functionality. Future work is required to establish concepts for the representation of complex TCAD tasks.

lets the user enter commands for the VLISP interpreter in the same way as at the normal command line, with the added benefit of providing a command history and a recall mechanism as well as full screen-edit support. For the programmer's convenience, information can be inquired for all symbols known to the VLISP interpreter. The *Message* window essentially prints out all messages generated by the VLISP interpreter; the destination of these messages can be dynamically configured to the message window, or to the terminal where the session was started, or to both.

The *SFC* window gives access to the run controller and the run data base, which will be discussed in more detail. The top organizational entity for all work to be carried out in the environment is the *project*. A project contains a *design*, which contains a process flow and mask data, and additional information necessary to build the design. These additional data are stored as project *resources*, containing settings for global parameters and user preferences defined for a project. Table 3.1 gives examples of available project resources. Due to the object-oriented design of the environment, any number of projects can be active at the same time. However, whereas more than one project may be in progress at a time, only one can be simultaneously accessed through the GUI.

Resource	Description
`directory`	Top directory for all project data.
`direct-tool-output`	Writes all tool output directly to terminal to avoid buffering when debugging.
`discard-tool-output`	Discards all tool output to save space.
`echo-tool-output`	Echoes all tool output to the terminal for monitoring progress of operation.
`single-step-mode`	Executes flows in single step mode for debugging; see Section 6.5.2.
`file-check-mode`	Selects policy to check for existing output files; see Section 6.2.1.
`purge-mode`	Allows purging of tool output data that is not needed in the current split experiment.
`float-epsilon`	Maximum relative difference of two floating point numbers for split checking.
`gridding-tool`	Gridding tool for wafer-state update; see Section 4.3.4.

Table 3.1: Examples of project resources.

The *System Jobs* window gives an overview of all active and queued system jobs, including executing host, working directory, and command line arguments, with direct access for terminating jobs as well as for sending data to a process in case its input is connected to the environment. The *Hosts* window shows all computers enabled for execution with color codes indicating availability and idle state. An automatic load balancing mechanism [KPR$^+$96] distributes the computation load evenly across the computing environment. Specific hosts may be marked as *restricted* to indicate that they should only be used if a job explicitly asks for it; see Section 4.6 for more details on the interface between the computing environment and the simulation environment.

3.5 Concurrent Clients

A central concept that the VISTA/SFC simulation environment concentrates on *concurrent clients*. Concurrent clients are external executables or processes[8] that run concurrently with the controlling process – the simulation environment – and communicate with it. Most numerically extensive or critical applications, especially optimization tasks, are not implemented as internal services but are delegated to concurrent clients. To establish access to these external modules, an encapsulation strategy has been developed and implemented.

3.5.1 Encapsulation of External Clients

For several TCAD tasks, external clients[9] are used by the VISTA/SFC environment to generate the desired results. For the internal representations of external clients *agents* are provided that take care of all data and message handling required for smooth operation. For example, when an optimization task is initiated by the framework, an agent is assigned to the optimizer tool, which establishes a connection between the task-level EVE interface[10] and the optimizer. The agent is represented by a VLISP object and can be subclassed to implement encapsulations of other external clients. It takes care of passing messages between the optimizer and the environment by means of a callback-based, asynchronous connection, allowing for the execution of multiple optimization tasks at the same time. Figure 3.3 shows the interaction between the optimizer agent and the EVE interface on the one hand, and between the agent and the external optimizer on the other hand. The environment passes a description of the model to the optimizer, defining the model's type and its control and response variables. During the course of the optimization, the optimizer requests the evaluation of the model for a certain set of control values by sending a message to the environment. Messages between the optimizer and the agent are transmitted using VISTA's operating-system independent standard-input/standard-output redirection capabilities. The details of this connection, however, are encapsulated by the agent, and other means of communication like TCP/IP sockets can be supported as well. Depending on the internal operation of the optimizer, evaluation requests may be sent synchronously, or a number of requests may be sent at a time. Upon termination of the optimization, the result found and diagnostic information are passed back to the environment.

3.6 Implementation

All modules of the VISTA/SFC environment are coded in VLISP and are run by the VLISP interpreter of the VISTA framework. The entire VLISP code amounts to approximately 1.8 megabyte of ASCII source text, 1 MB being attributed to the implementation of the core functional modules shown in Figure 3.1 and 0.4 MB each being alloted to the interfaces for

[8] Processes also include daemons and servers that run completely independent of the simulation environment, whereas executables are started on behalf of it.

[9] The distinction between *client* and *server* sometimes cannot be clearly made. In the example of an optimizer, for instance, it can be regarded as a server that provides the optimization service and delivers the optimum found, or as a client that requests evaluations from the simulation environment.

[10] See Section 8.3 for a description of the class of Evaluable Entity (EVE) objects.

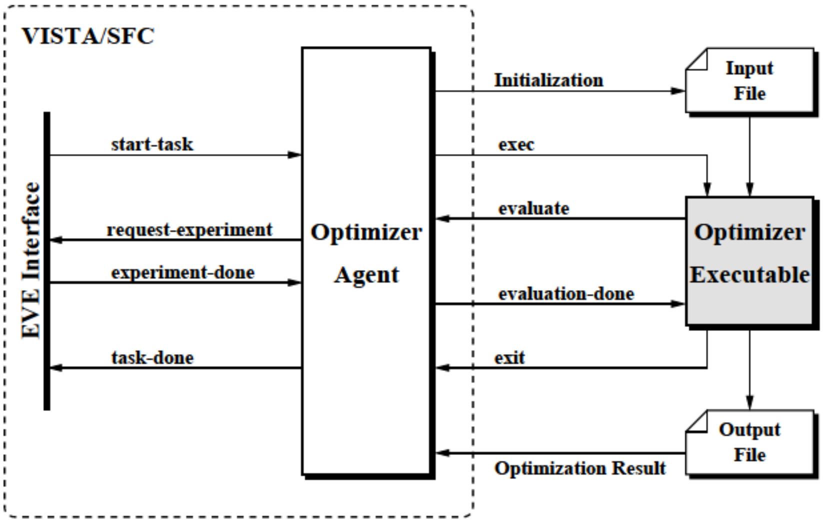


Figure 3.3: Communication between EVE interface, optimizer agent, and external optimizer.

currently integrated external tools and to basic functionality for GUI generation, system interface, message handling, and resource handling. Figure 3.2 gives an overview of the various VLISP modules constituting the simulation environment.

Module	Description
PIL	PIF Interface Layer for reading and writing PIF data.
SVG	Graphics library for plotting.
VBS	Base-system libraries providing operation-system independent access to system processes, file system, regular expressions, resource data base, system timers.
VUI	High-level interface to X window system.

Table 3.2: VISTA/SFC VLISP modules.

VLISP was chosen as the implementation language for a variety of reasons that are related to the LISP language itself [WH89], to the XLISP implementation [Bet89], and to various extensions available in VLISP[11]. The following list highlights those aspects that were relevant in the decision process in the context of the VISTA/SFCenvironment.

- Interpreter allows fast prototyping and code-update and code-reload during sessions.
- Automatic memory management supports problem-oriented programming.
- Run-time code generation capability

[11] See [Tup96] for more information on the Vienna Base System and the Vienna LISP Interpreter. Appendix A gives a brief overview of the LISP programming language and the XLISP and VLISP implementations.

- Built-in debugging interface facilitates code development and debugging.
- Runs on various operating systems and platforms.
- Vienna Operating System (VOS) provides powerful support for file handling and process management tasks in an operating-system independent way.
- Vienna User Interface library (VUI) provides high-level functionality for defining graphical user interfaces quickly and comfortably.
- Built-in error handling and error message system standardizes error handling.
- Built in object classes support object-oriented design.
- Multiple inheritance facilitates the reuse of object classes spread across different modules.
- PIF Application Interface (PAI) [FFS$^+$91] [Fas94] provides direct access to simulation data stored in PIF format.
- Easily extensible by binding external functions and libraries coded in C or C++ to the interpreter automatically using the Tool Abstraction Concept (TAC).
- Built-in regular expression library simplifies text-based data exchange.
- Byte-code compiler increases speed of execution and enforces syntax and type checking at compile-time.
- Variety of additional LISP data types improve code quality and form basis for compile-time and run-time type checking. Type-specific print and read routines liberate the programmer from handling different types explicitly. Some examples of valuable data type extensions are given in Table 3.3.

Node Type	**Description**
`PROCESS`	Running or terminated system process
`QUANTITY`	Real number with units[a]
`REGEXP`	Compiled regular expression
`TIMEOUT`	Timer entry
`TIMESTAMP`	File time stamp, system time stamp
`VOSPEC`	Operating-system independent file specification

Table 3.3: Selected VLISP data types.

[a] All arithmetic operations available for floating point numbers are also supported for quantities. The VUI library provides widgets for the direct GUI representation of quantities.

Chapter 4

Integration of Simulation Tools

The simulation of complete VLSI fabrication processes and devices involves the cooperation of a variety of different simulation tools to generate a model of the desired semiconductor structure, to compute the behavior of the resulting device, and to extract characteristic electrical parameters for further use in circuit simulation[1]. Figure 4.1 gives an overview of all simulation stages involved in the computer-aided analysis of an integrated circuit; feedback loops of the design process are omitted for clarity.

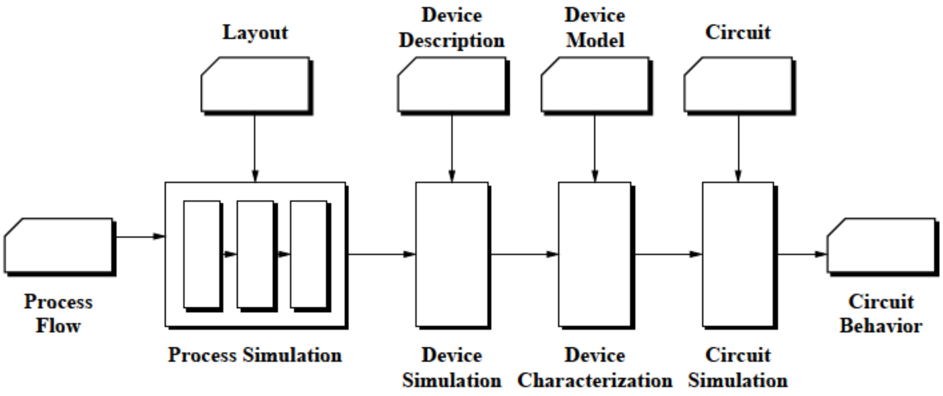


Figure 4.1: Simulation stages of TCAD analysis of integrated circuits.

Each stage shown in the picture calls for running one or several simulators, with process simulation almost always requiring the complex interplay of many dissimilar programs, reflecting the fact that the fabrication process itself consists of a sequence of numerous process steps of a broad variety of types applied one after the other onto the semiconductor wafer. Interfacing these tools with one another and controlling them in a uniform way for the purpose of completely automating the entire procedure outlined in Figure 4.1 are the main concerns of this chapter.

[1] Note that the scope of this chapter does not include device characterization or circuit simulation, although most of the concepts and solutions presented can be extended to these domains.

4.1 Heterogeneous Process Simulation Tools

Among the components shown in Figure 4.1, process simulation plays an outstanding role with respect to tool integration for the large variety of available tools on the one hand and for the need of arbitrarily changing the order in which tools are invoked on the other hand. Moreover, whereas device simulation, device characterization[2], and circuit simulation transform their respective inputs to outputs of entirely different domains, process simulation is domain-invariant inasmuch as process simulation tools produce output that again serves as input for other process simulation tools[3]. More precisely, a device simulator reads a model of the wafer structure and writes numerical values for electrical quantities; a diffusion simulator reads a wafer model and writes a wafer model.

4.1.1 Tool Specialization

The large – and constantly growing – number of available process simulation tools can be explained with the rapid evolution of semiconductor technology as well as with the lack of a unified and established basis for the development and implementation of TCAD tools. Fast technology changes call for the development of specialized simulation tools, each holding an expert's position for a particular process simulation task. For example, the definition of arbitrary diffusion models on each material segment of a wafer is an advanced feature of the PROMIS NT process simulator [PS95] [Puc96], which other diffusion simulators do not offer. Of all available tools, to optimally model a process technology, it is of paramount importance to use the tool best suited for each process step instead of confining oneself to a particular product's or vendor's solutions. (Cf. Table 4.1 for some examples for existing process simulators; this list shows only a negligible fraction of all process simulation tools in use at universities and in industry).

4.1.2 Tool Categories

Apart from being specialized in particular physical processes, process simulation tools fall in three categories with respect to the usability for process flow simulation:

1. Tools that model the effects of a single fabrication process.
2. Tools that model the effects of a short sequence of closely related fabrication process steps.
3. Tools capable of simulating a complete semiconductor process flow in its entirety.

Moreover, tools differ with respect to the completeness of the solution produced: Tools either generate a complete model of the wafer as it appears at the end of a process step considered, or they produce only a partial model, limiting the output, for instance, to dopant distribution

[2] In lieu of *device characterization*, the term *parameter extraction* is often used.

[3] Trivial as it sounds, this fact is not always present when process simulation tools are being devised. Rumor has it that even at the most prestigious institutions tool developers are sometimes baffled when learning of the requirement for a tool to be able to understand its own output.

data or geometry data. Table 4.1 gives some examples of process simulation tools belonging to either of these two categories.

Tool	Supported Processes	Output
ATHENA [Sil94]	All processes	Complete output
CREEP [SOK87]	Oxidation, Reflow	Complete output
FLOOPS [Law95]	All processes	Complete output
IMPACT-4 [BCF+95]	Diffusion, Oxidation	Complete output
PROMIS Analytical Implant [Sti93]	Ion Implantation	Geometry, new doping
PROMIS Monte Carlo Implant [Boh96]	Ion Implantation	Geometry, new doping
PROMIS Diffuse [Wim93]	Diffusion	Geometry, all doping on silicon segment
PROMIS NT [Puc96]	Diffusion	Complete output
PROMIS ETCH [SS95]	Deposition/Etching	Geometry
SAMPLE [Ele91]	Deposition/Etching/Lithography	Geometry
SKETCH	Deposition/Lithography	Geometry
SPEEDIE [MRC+90]	Deposition/Etching	Geometry
SPLAT [Toh88]	Imaging	Light intensity distribution
SUPREM4 [Tec95]	All processes	Complete output

Table 4.1: Process simulation tools, processes, and output. *Complete output* indicates a complete description of the wafer at the end of the process step.

The ability to integrate a broad variety of heterogeneous simulators and to present them in a uniform manner is one of the main concerns of this work. The following sections present in more detail the tool integration methodology used in VISTA/SFC.

4.1.3 Tools and Tool Applications

As shown in Table 4.1, some tools are capable of providing a number of different services to the simulation environment. For example, the SKETCH program serves as a simple deposition tool as well as a lithography tool, depending on a command line option; the TSUPREM4[Tec95] simulator is capable of simulating a number of different fabrication processes, which are specified in an input deck. From the framework's point of view, only the service rendered is of interest, not the executable invoked. To reflect this fact, a service provided by a tool is called a *tool application*. Figure 4.2 shows the relation between tool applications and tools in the light of the SKETCH simulator. On the other hand, a particular service may need more than one tool to be called in a strict order. On the right side of Figure 4.2, a tool application is shown that provides an encapsulation of all simulator calls and intermediate data manipulations required for an oxidation simulation[4].

[4] See Section 6.2.1 on how to avoid the potential computational overhead arising from this abstraction.

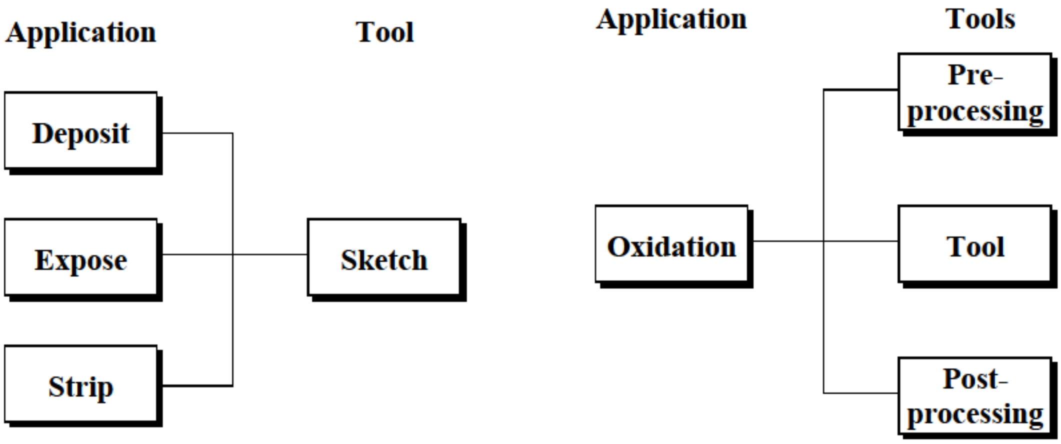


Figure 4.2: Applications and Tools. A single tool may used for several distinct applications, a single application may call for several tools to be invoked.

4.2 Tool Integration

The integration of external tools into the broader context of a simulation environment requires coupling on the *data level*, i.e., the exchange and conversion of tool input and output data between syntactically and possibly semantically different data representations, and on the *control level*, i.e., parameter passing, input deck generation, tool invocation, and handling of diagnostic messages. Furthermore, on the *presentation level* a user interface has to be provided to allow editing of tool settings in an error-proof and intuitive way, supporting both graphical as well as text-terminal operations.

Tool integration in this context deals with executable simulators that themselves remain unchanged by the integration process. In contrast to this approach, other methodologies have been devised to integrate tools by linking numerical modules that perform the solution of the physical equations with generic server modules that manage the data produced. For an example of this strategy, see Sahul et al. [SWH+96].

4.3 Data Level

Integration at the data level is concerned with the exchange of simulation results between simulation tools. The two fundamental problems in this domain arise from different data formats and from different information models employed for representing simulation data. In general, the first problem is easily solved by a data format converter that maps a given data format onto another one. The second problem poses by far the greater challenge. Care has to be taken to correctly convey all information represented in one model to another one and vice versa. In general, this problem cannot be solved completely, and a certain loss of information needs to be reckoned with. However, in some situations, data *bypassing* can help to limit the loss of information to the position of a particular tool in a sequence of tool steps; after that step, information can be regained by appropriate merging operations (Figure 4.3). For

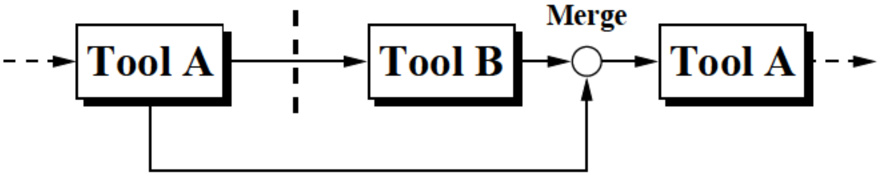


Figure 4.3: Different information models of tools A and B lead to loss of information for tool B. Bypassing data regains that loss for tools subsequent to B.

the general case, it is more instructive to focus on the subsequent modifications of a wafer state model instead of thinking of chaining tools together. In Figure 4.4, the wafer model is modified from step to step by the operation of tools whose respective inputs and outputs are derived from, and merged with, the current contents of the model. According to this scheme, a tool's input may be completely unrelated to state of the model.

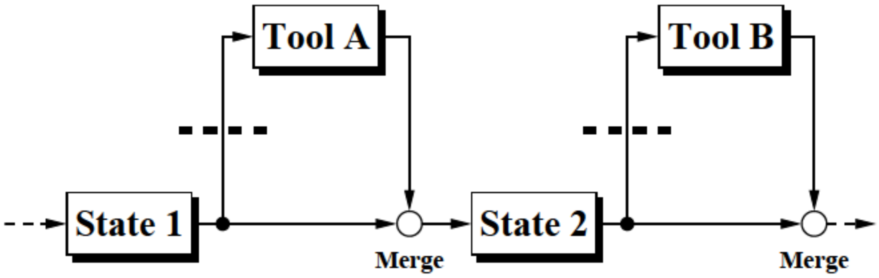


Figure 4.4: Wafer model and tool input and output: The state of the model is iteratively modified by the operation of tools.

4.3.1 PIF – **Profile Interchange Format**

The Profile Interchange Format (PIF) [Duv88] is used as the primary data format for representing wafer data and for exchanging simulation data between tools. It allows a hierarchical boundary representation of geometries and defines a set of distributed or concentrated attributes on geometry segments and on grids to store material and dopant distribution data. Grids are built from elements of a variety types, including triangular and rectangular elements [Fas94]. Dopant concentration data, e.g., are subdivided into *interstitial* and *active* fractions, with the total (chemical) concentration of a dopant species being the sum of the two. Figure 4.5 gives a summary of the objects found in a typical PIF wafer file. Arrows represent references from an object to another one. If a PIF object is a collection of smaller entities, an index specifies the entity addressed[5]. Figure 4.6 shows the navigation tool provided in

[5] Geometry objects like points, lines, and faces are not represented as separate objects, but are organized in lists. E.g., a `pointList` object contains all points of a geometry, a `segmentList` contains all segments. To access a single point from a point list, an indexed reference is used.

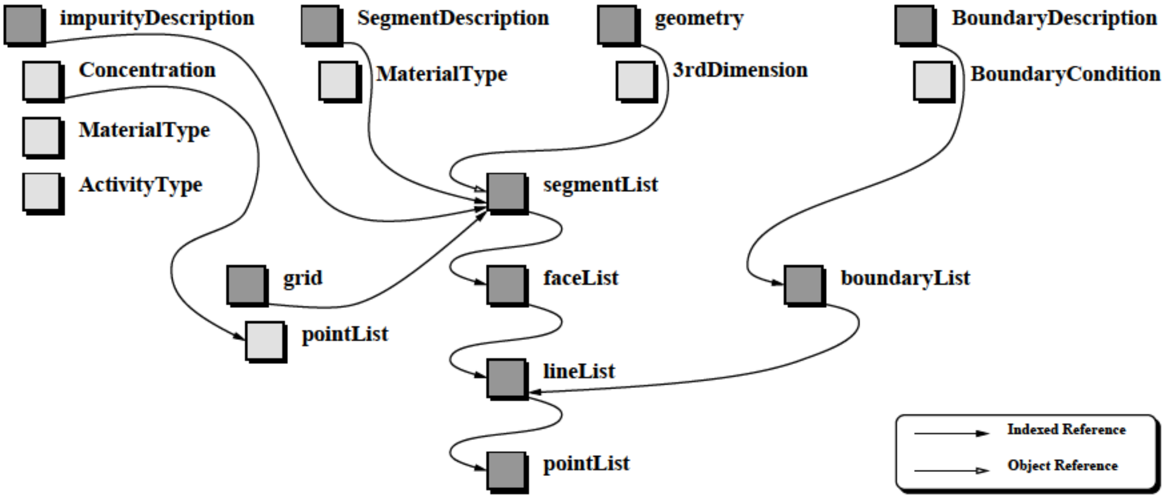


Figure 4.5: PIF objects in PIF wafer state file.

VISTA/SFC to view and manipulate PIF data.

PIF data are accessed by applications using the PIF Application Interface (PAI) [FFS+91], one of VISTA's core components. The original PIF specification [Duv88] leaves ample room for defining different data models. The PAI implementation enforces some conventions on the structure of PIF objects while still allowing ambiguities in the definition of geometrical objects. To avoid any ambiguities, a *wafer state* convention has been devised that imposes a number of restrictions on acceptable data representations for a wafer state model and greatly enhances the practical usability of the PIF format with respect to inter-tool data exchange. The most important restrictions are given in Table 4.2; see also [Rie96].

Item	PIF	**Wafer State**
Geometry orientation[a]	arbitrary	counter-clockwise
Geometry connectivity[b]	undefined	single
Number of grids per segment	any	1[c]
Numbers of points per line	$\geq$2	2

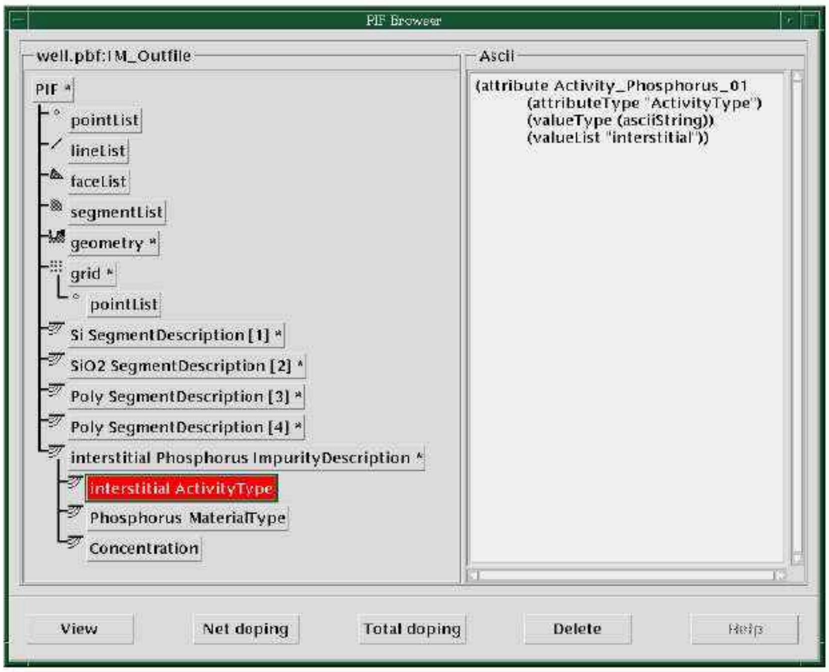
[a] *Geometry orientation* refers to the orientation of lines in faces and faces in solids.

[b] *Geometry connectivity* refers to the topological dimension of faces and solids.

[c] In the VISTA/SFC wafer state convention, grids have to be boundary-conform to their respective segments.

Table 4.2: Original PIF specification vs. VISTA/SFC PIF wafer state convention.

For accessing PIF data, tools are either linked with the PAI or use external wrappers to convert data to the PIF format (Figure 4.7). The second approach is often preferable as no changes of tool code are necessary and tool upgrading is simplified.



```
PIF *
├ pointList
├ lineList
├ faceList
├ segmentList
├ geometry *
├ grid *
│ └ pointList
├ Si SegmentDescription [1] *
├ SiO2 SegmentDescription [2] *
├ Poly SegmentDescription [3] *
├ Poly SegmentDescription [4] *
└ interstitial Phosphorus ImpurityDescription *
  ├ interstitial ActivityType
  ├ Phosphorus MaterialType
  └ Concentration
```

```
(attribute Activity_Phosphorus_01
        (attributeType "ActivityType")
        (valueType (asciiString))
        (valueList "interstitial"))
```

Figure 4.6: GUI for viewing PIF data. The left part of the window provides navigation through the hierarchical data structure, the right part shows the data in ASCII representation.

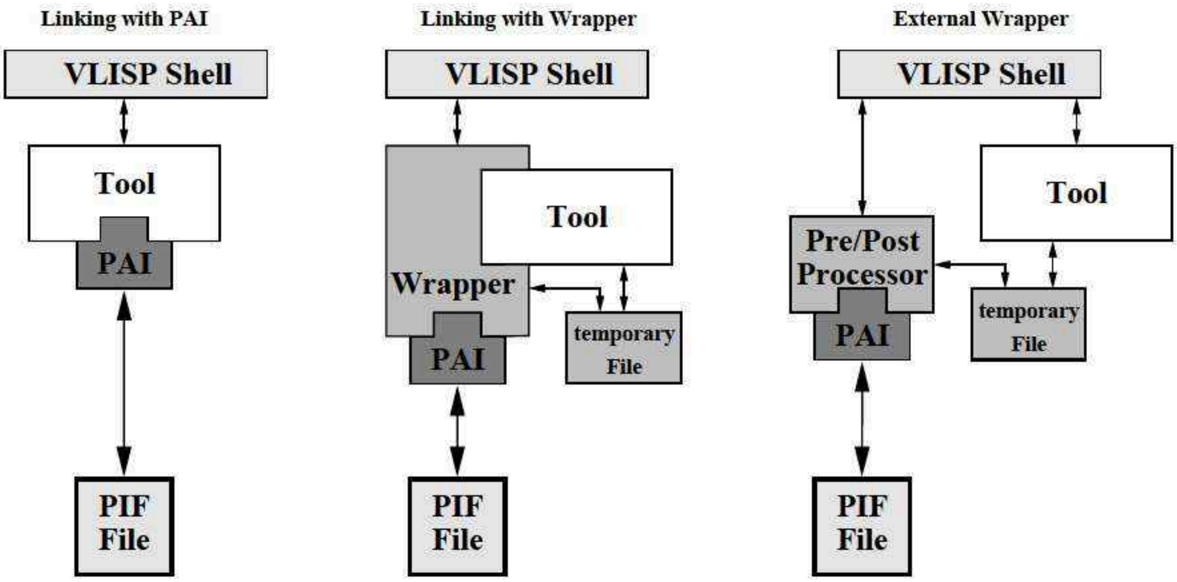


Figure 4.7: Data level integration schemes.

4.3.2 Native Tool Formats

Although a single common data format is highly desirable for the integration of heterogeneous simulation tools, enforcing the conversion to a common data format after each simulation step causes a significant overhead in computing time and a loss in numerical stability of the overall data and can be avoided if two subsequently executed simulators share a common data format, or if, as it is often the case, only tools of a single family are used in one project. To take advantage of this potential increase in speed and numerical accuracy, native tool formats are also fully supported, with conversions taking place as necessary. A prominent example for a native tool format that is supported by the VISTA/SFC environment is the TIF format used by TSUPREM4 [Tec95]. In addition to predefined data types, new ones may be defined to specify appropriate conversion routines and tools (cf. Section 7.1.3).

4.3.3 Consistent Wafer Representation

Tool specialization has led to expert tools dealing with isolated aspects of a wafer only. For example, the SAMPLE [ONS$^+$80] [Ele91] and PROMIS [SS95] etch and deposition modules operate only on the wafer *geometry*, causing inconsistencies between the grid-based dopant distribution data and the geometry boundary information, as they do not take care of reconciling the two after updating the geometry[6]. To ensure a correct, consistent, and concise wafer representation after each process simulation step, data merge operations have to be performed to reflect geometry alterations in the grid structure, to purge superfluous grid elements, and to merge dopant information from before and after a simulator call (cf. Figure 4.4). Figure 4.8 sketches the data flow in the case of a PROMIS etch operation. Starting from the geometry information present in the PIF wafer state model, the tool call generates

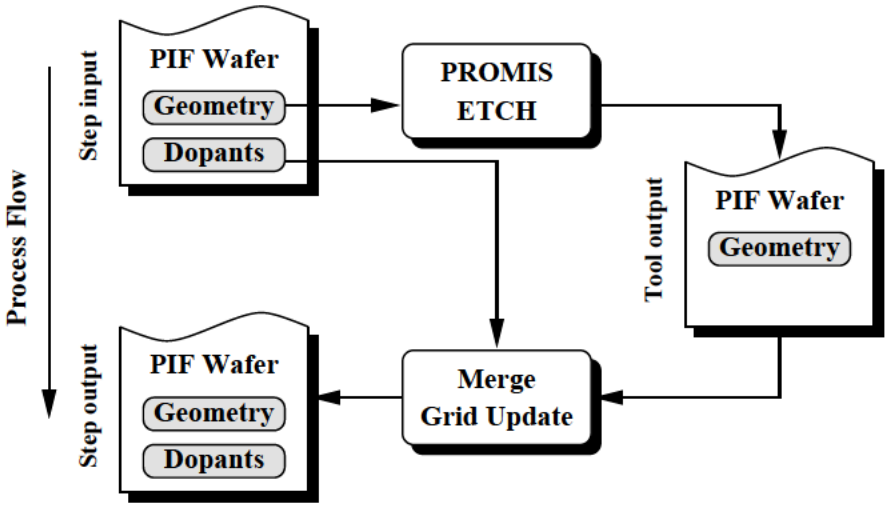


Figure 4.8: Data flow for creating a consistent wafer model after a PROMIS etch operation.

[6] These inconsistencies are not a consequence of negligent behavior on part of the tools in question, but necessary simplifications to let tool developers focus on the core issues of the simulation task.

the new wafer geometry reflecting the effects of the simulated etch or deposition process. In the subsequent merge and update operation, segment grids on all segments of the geometry are checked and adjusted to conform with their respective segments[7]. After the merge and grid-update operation, a complete model of the wafer is available for further processing. For the general case, Figure 4.9 shows the basic flow graph for updating the PIF wafer state representation after termination of a tool.

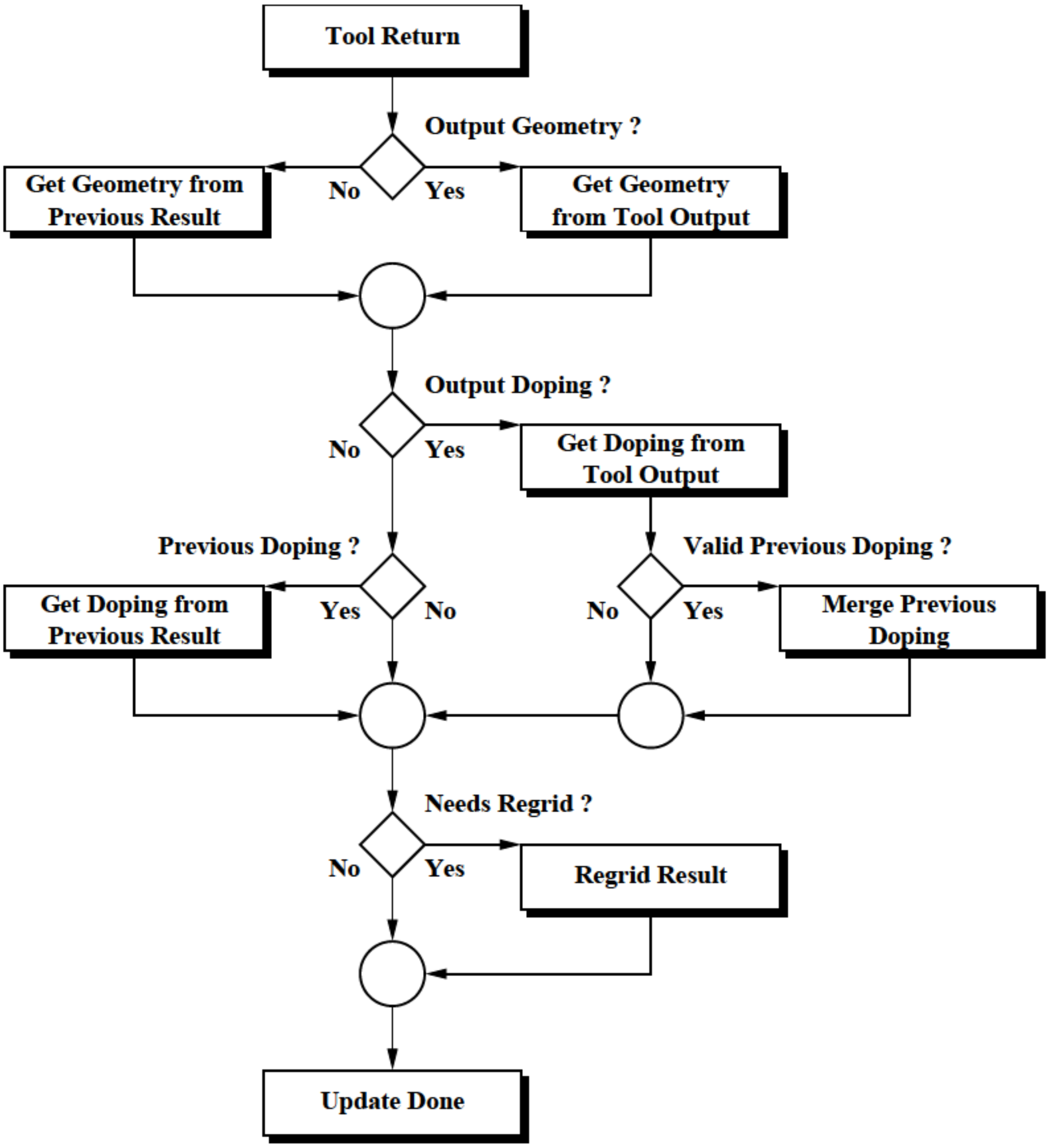


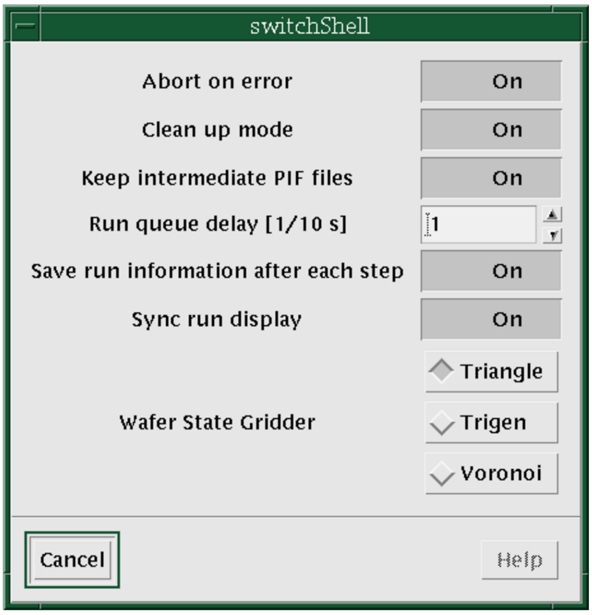
Figure 4.9: Flow graph for updating the PIF wafer state representation after a simulation tool call.

[7] For a detailed analysis of grid-related tasks during merging see [Hal94].

4.3.4 Gridding

During process flow simulation, the PIF wafer model is subsequently undergoing a number of modifications of the boundaries of the material segments defining the wafer geometry and of the dopant distribution data in the bulk and along material interfaces of the wafer. To fulfill the requirements of the PIF wafer state convention adopted in VISTA/SFC for the PIF representation of a wafer model (cf. Section 4.3.1), grids have to conform to the boundaries of their respective segments to form a valid wafer state.

The reconciliation of existing grid and dopant data, modified grid and dopant data, and existing and modified geometry data is taken care of by external gridding tools that are called automatically by the framework. The selection of an appropriate gridding tool potentially poses problems similar to the choice of the right simulator. Various gridding tools have been reported, with new developments constantly challenging established solutions. Therefore, it is advisable to provide a mechanism for using the gridding tool of one's choice in a fashion similar to selecting the simulator best suited for a process step. At present, VISTA/SFC offers a choice of three grid generators, Triangle [She96], Trigen [Ban90], and Voronoi [Hal94], for re-gridding purposes, with a clear interface provided for the integration of additional gridders. In the GUI, gridders are selectable by mouse click similar to other configuration parameters (Figure 4.10).



Abort on error On

Clean up mode On

Keep intermediate PIF files On

Run queue delay [1/10 s] 1

Save run information after each step On

Sync run display On

Wafer State Gridder ◆ Triangle ◇ Trigen ◇ Voronoi

Figure 4.10: Selection of wafer state gridders: GUI module.

At present, Triangle has proven the most reliable and robust gridder of the three alternatives. It is based on a highly flexible delaunay gridder and takes care of enforcing the wafer state convention with respect to grid and dopant data, re-gridding of the entire structure from scratch, using local grid refinement with respect to dopant concentrations, merging existing dopant data and additional data generated by an implantation or diffusion step, and for merging a geometry arising from a topography simulation step with the dopant data of the original wafer. As it makes use of existing grids, numerical inaccuracy is reduced to a minimum and the creation of additional grid points is restricted to areas where it is necessary[8]. TRIANGLE provides a mechanism for the generation of quality meshes by means of various grid constraints including minimum angles and maximum areas of triangles and dopant dose conservation criteria. An enhanced refinement technique delivers rapid variations of grid density and thus minimizes the global number of grid points.

In general, the choice of gridding parameters and constraints exerts a strong influence on the quality of the simulation result. Therefore, all gridding-related resources are specific for a project and do not need to be specified globally. For example, the **`max-triangle-area`** parameter specifies the maximum size in μm^2 of all triangles generated by TRIANGLE, the **`min-angle`** parameter defines the minimum angle in degrees. Typical values for these resources are 0.1 and 15, respectively.

Wafer state gridding is mainly concerned with the storage of distributed quantities, not with requirements arising from numerical and computational considerations. Most simulation tools employ their own strategy to construct an internal grid representation that is determined by numerical aspects of the equations to solve. In general, it is not possible to find a grid that is optimally designed for all possible computational strategies. Therefore, data representation on grids is supported by the environment, whereas the generation of computational grids is declared responsibility of the simulation and solver modules[9].

4.4 Control Level

The control level takes care of all aspects of tool invocation and tool run-time management:

1. Command line assembly.
2. Input deck generation.
3. Input and output redirection for establishing communication links and for message and error logging.
4. Tool invocation.
5. Return code interpretation
6. Extraction of results from output files.
7. Interpretation of diagnostic messages.

One fundamental problem in TCAD tool integration is caused by the fact that many simulators understand complex input deck languages that are rather difficult to represent on the

[8] In a typical multi-tool process flow simulation scenario, numerous calls to the gridding utility are made to reconcile tool output with pre-existing data. We have found it to be extremely helpful to differentiate between "new" data generated by a simulator, and "old" data that already is the result of a re-gridding operation.

[9] PIF-tool developers can rely on powerful libraries that support the internal handling of grids [Fas94].

framework level in a uniform manner. Therefore, we treat each functional subset of a simulator as a separate task-level *application*(cf. Section 4.1.3). For each fabrication process we define an application that appears to the framework as an independent tool. The application is accessed by calling a binding function that reads resources derived from local and global settings.

4.4.1 Binding Function

Applications are either based on external executables or on functions internal to the environment. All operations necessary to invoke an application and to return its output data are encapsulated by a *binding function*, which establishes a uniform interface between the environment and the application[10]. Applications can be defined and added to the framework during sessions without need for interrupting any active tasks. Figure 4.11 shows a piece of code used to define an application based on an external simulator. A GUI is generated completely automatically on demand from this declaration without any further coding required. Nevertheless, to reflect complex interdependencies of an applications parameters, more sophisticated interfaces can be defined by the user via an object-oriented GUI generator [Str96].

4.4.2 Tool Resources

Each application is represented as a *resource client* that understands a set of resources. Similar to the resource mechanism employed in the X Window system [Jon89], a *resource* is identified by its name, which is unique in the context of a given client, and a resource type. Interdependencies and default values may be specified for all resources of a client. All command-line arguments, required and optional parameters, operation modes, etc. of a tool are accessible as resources. This mechanism ensures that:

1. All tool parameters are explicitly accessed, and
2. Parameters of all tools are accessed in a uniform way.

Resources and resource clients are implemented as object classes in VLISP. Resource types comprise all basic LISP data types[11] plus additional ones defined by the application.

4.4.3 Input Deck Templates

Specialized simulators often understand a rich input deck syntax, enabling the experienced user to formulate complex analysis tasks. From the process engineer's point of view, however, only a few variables in the input deck may be of interest. Therefore, a given simulator together

[10] Using a standardized VLISP function pointer as interface between the run controller and external tools keeps the interface as flexible as possible, while still allowing for the automatic generation of the function code from a more abstract *tool description* to minimize manual coding efforts. As VLISP does not differentiate between code and data, it is ideally suited for code generation applications.

[11] For the sake of clarity, the LISP representations of boolean data, `T` and `NIL`, are mapped to a pair of strings, e.g., `"on"` – `"off"` or `"yes"` – `"no"`.

```
;; binding code for SKETCH simulator
(defun sfc::sketch (&key

        ;; ----- standard keys -----
        cob         ;; context object containing input data, working
                    ;; directory, input/output streams

        ;; ----- input file key -----
        pif-file    ;; physical PIF file
        pif-logical ;; logical PIF file

        ;; ----- tool-specific resources -----
        (process "subdomain") ; type of process
        (material "Si")       ;
        (length 3.0)
        (depth 2.0)

        ;; ----- local variables -----
        &aux
        output-plb
       )

  ;; ----- define local callback function for system call -----
  (flet ((sketch-done (process client-data return-code)
           ;; ----- pass output file to context -----
           (send cob :check-output output-plb :type 'pif-file)
           ;; ----- pass return code back to context and return -----
           (send cob :call-done return-code)))

    ;; ----- create output file name -----
    (setq output-plb (list (send cob :base-name :suffix "sketch" :ext "pbf")
                           "Sketch_Outfile"))

    ;; ----- queue system command for execution on network -----
    (sfc::queue "sketch"                ;; system command
        ;; ----- pass command line arguments -----
        (append (list pif-file          ;; physical PIF file
                      pif-logical       ;; logical PIF file
                )
                (list 'opbf (first output-plb)
                      'oplb (second output-plb))
                ;; ----- read key arguments from local environment -----
                (get-keys :begin 'process :end 'depth)
        )
        ;; ----- pass working directory and output/error streams -----
        :directory (send cob :inquire 'directory)
        :output (send cob :inquire 'output)
        :error (send cob :inquire 'error)
        ;; ----- set termination callback of system  -----
        ;;       call to local function
        :term-cbl #'sketch-done
    )
  )
)
```

Figure 4.11: Example code for tool binding function for SKETCH simulator.

with a specific input deck *template* that implements an analysis task may be defined as an application by itself. The template is derived from an existing input deck by simply marking relevant fields with enclosing them in `<(` and `)>` as shown in Figure 4.12. The simplest substitution statement just assigns a name and a default value to a field in the input deck, while additional statements are used to define units, value ranges, and numerical expressions to derive a field's value; Figure 4.13 gives some examples of template substitution statements. Table 4.3 gives a summary of currently implemented additional statements.

```
TITLE MOS CHARACTERISTIC
COMMENT Y.MAX=0.8 MODEL,METHOD SOLVE ***
mesh infile=<(mesh-file input.pi)> tsuprem4 elec.bot poly.elec y.max=0.8
models conmob fldmob srfmob
contact number=Gate n.poly
symbolic newton carriers=1 hole
solve v(Substrate)=0.0 electrod=Substrate vstep=0.1 nstep=6
solve elec=Substrate V(Substrate)=<((bulk-bias :units "V") 0.0)>
solve elec=1 continu c.vmin=-3.3 c.vstep=-0.05
log outfile=<(log-file md.log)>
solve elec=Gate continu c.vmin=-0.1 c.vstep=-0.02
solve v(Gate)=-0.1 electrod=Gate vstep=-0.1 nstep=15

PLOT.1D   infile=<(log-file md.log)>
+ device=1/post
+ Y.AXIS=I1  X.AXIS=V(Gate) Y.LOG   POINTS
+  left=0 right=-3.2 bottom=1E-22 top=1E-2
+  title="DATA" SYMBOL=1 ^ORDER color=1
+ outf=<(data-file md.dat)>
```

Figure 4.12: Input deck template example. Expressions enclosed in matching angle-bracket – parenthesis pairs `<(...)>` are replaced by values supplied by the simulation environment to generate the simulator input deck.

```
<(alpha 15.0)>

<((alpha :units "deg") 15.0)>

<((alpha :units "%" :calc "0.15 * alpha") 100)>
```

Figure 4.13: Template substitution statements examples.

If the value of a parameter is to be used more than once in a template, a simpler syntax can be used: Names preceded by the dollar sign `$` will be replaced by the current value of the parameter. Both the expression delimiters `<(` and `)>` and the substitution prefix `$` can be replaced by other characters to avoid conflicts with a simulator's native syntax. Using substitution expressions with the `:nooutput` flag along with substitution prefixes in a template allows all variable parameters to be declared in the header of the template, making for a more easily readable file.

For interactive editing, a graphical user interface is automatically created from the field definitions in the template file (Figure 4.14). Both the template and the input deck derived from the template – after substituting the default values from the parameter definitions in

Statement	Description
`:text`	Text string to be displayed in editor instead of the parameter name.
`:eval`	LISP Expression to be evaluated in the context of the substitution to derive the value of the parameter. All parameters read up to that point in the template can be used.
`:calc`	Same as `:eval`, but expression is assumed to be given in infix notation instead of prefix (LISP) notation.
`:use`	Value to be used if an error occurs during expression evaluation.
`:string`	Indicates that the value derived after substitution is to be printed enclosed in double quotes to the input deck generated.
`:log`	Parameter exhibits logarithmic behavior with respect to the system's responses.
`:min`	Hard-wired parameter minimum value.
`:max`	Hard-wired parameter maximum value.
`:units`	Units string to be displayed together parameter value.
`:type`	Parameter data type if not derivable from default value.
`:choices`	List of choices for parameter value.
`:noedit`	Disable interactive editing.
`:nooutput`	Skip entire statement when substituting.

Table 4.3: Template substitution expressions.

the template – can be viewed alternatively (Figure 4.15). In this fashion, pre-existing process and device simulation input deck libraries can be easily integrated without need to rewrite them in a new language.

4.4.4 Tool Description

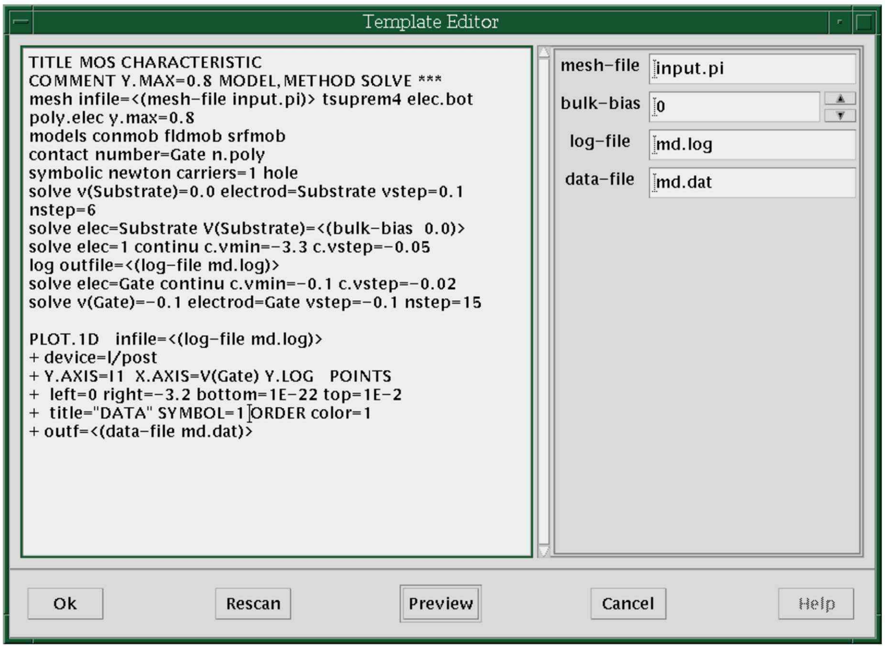
Ideally, all aspects of tool integration are based on a standardized and appropriately abstract tool description that contains all information for control level and presentation level integration. Unfortunately, many tools have proven rather elusive with respect to a complete description of their signature[12], behavior, and output. Therefore, a hybrid approach has been chosen that combines the flexibility and versatility of unrestrained LISP code with the power of abstraction of a formal description. The LISP code contains a function of standardized form (cf. Figure 4.11); a registration call assigns the function to an application. Parameters are automatically extracted from the function definition, additional information on ranges, defaults, types[13], etc. can be specified.

4.5 Calibration

A calibration refines a tool application for a specific purposes. The combination of an existing application and a calibration can be dynamically defined as a new application on its own,

[12] In analogy with the terminology of many programming language, the *signature* of a tool refers to the names, positions, and respective ranges of its input parameters.

[13] In most cases, the data type can be derived from the default value.



```
TITLE MOS CHARACTERISTIC
COMMENT Y.MAX=0.8 MODEL,METHOD SOLVE ***
mesh infile=<(mesh-file input.pi)> tsuprem4 elec.bot
poly.elec y.max=0.8
models conmob fldmob srfmob
contact number=Gate n.poly
symbolic newton carriers=1 hole
solve v(Substrate)=0.0 electrod=Substrate vstep=0.1
nstep=6
solve elec=Substrate V(Substrate)=<(bulk-bias  0.0)>
solve elec=1 continu c.vmin=-3.3 c.vstep=-0.05
log outfile=<(log-file md.log)>
solve elec=Gate continu c.vmin=-0.1 c.vstep=-0.02
solve v(Gate)=-0.1 electrod=Gate vstep=-0.1 nstep=15

PLOT.1D   infile=<(log-file md.log)>
+ device=l/post
+ Y.AXIS=I1  X.AXIS=V(Gate) Y.LOG   POINTS
+  left=0 right=-3.2 bottom=1E-22 top=1E-2
+  title="DATA" SYMBOL=1 ORDER color=1
+ outf=<(data-file md.dat)>
```

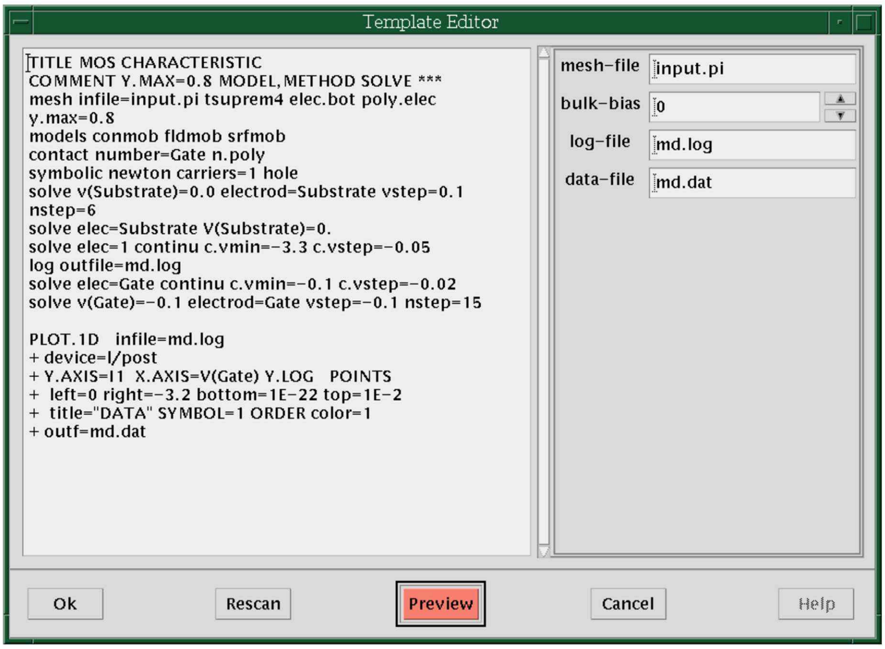
mesh-file input.pi

bulk-bias 0

log-file md.log

data-file md.dat

Figure 4.14: Graphical template editor generated automatically from template file.



```
TITLE MOS CHARACTERISTIC
COMMENT Y.MAX=0.8 MODEL,METHOD SOLVE ***
mesh infile=input.pi tsuprem4 elec.bot poly.elec
y.max=0.8
models conmob fldmob srfmob
contact number=Gate n.poly
symbolic newton carriers=1 hole
solve v(Substrate)=0.0 electrod=Substrate vstep=0.1
nstep=6
solve elec=Substrate V(Substrate)=0.
solve elec=1 continu c.vmin=-3.3 c.vstep=-0.05
log outfile=md.log
solve elec=Gate continu c.vmin=-0.1 c.vstep=-0.02
solve v(Gate)=-0.1 electrod=Gate vstep=-0.1 nstep=15

PLOT.1D   infile=md.log
+ device=l/post
+ Y.AXIS=I1  X.AXIS=V(Gate) Y.LOG   POINTS
+  left=0 right=-3.2 bottom=1E-22 top=1E-2
+  title="DATA" SYMBOL=1 ORDER color=1
+ outf=md.dat
```

mesh-file input.pi

bulk-bias 0

log-file md.log

data-file md.dat

Figure 4.15: Template editor in preview mode.

making a calibrated tool available under a unique name. A calibration is represented as a pre-processor for an application's input parameter vector that maps parameter settings on tool invocation to calibrated settings. In the simplest case, the calibration only supplies values for a set of calibration parameters accessible in the tool that do not depend on the input parameter values (*static calibration*). I.e., the calibration represents a specific set of default values for a given purpose (cf. Figure 4.16). In the general case, all input parameter values of the application depend on all parameter settings values; the calibration represents a multiplication of the invocation settings vector with a calibration matrix (*matrix calibration*). In both cases, the actual calibration parameters and factors can be derived using a procedure outlined in Section 8.8. Instead of a matrix, it is also possible to use a response surface model for mapping input parameters to calibrated parameters if a linear transformation is not sufficient.

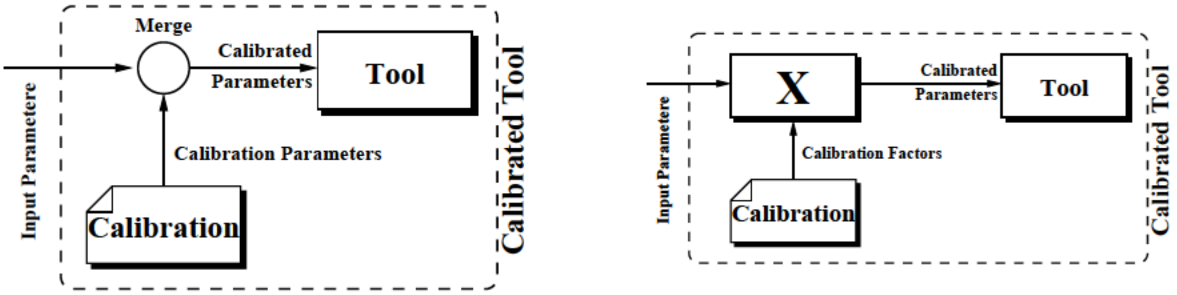


Figure 4.16: Static calibration vs. matrix calibration.

4.6 System Tool Interface

The lowest level of abstraction in the tool control layer of the VISTA/SFC environment is represented by the System Tool Interface (STI), that provides access to system executables in a uniform way. The purpose of this interface is to hide details of tool invocation that are related to queuing systems, file location of executables, authorized hosts for a given executable, and site-specific invocation procedures; basic logging and work-in-progress facilities are also offered. In this way, external tools can be accessed by symbolic names throughout all source code files and user-defined task-level applications. The actual mapping from the symbolic name to the operating system's command line is defined in a central configuration file[14].

4.6.1 Configuration

Figure 4.17 gives a simple example of the encapsulation of a non-standard call to a simulation tool. By the `deftool` statement, residing in a configuration file loaded on start-up of VISTA/SFC, the process simulator TSUPREM4 is defined to be called via the script

[14] It is not required to define an external tool before calling the STI. By default, an unknown name is used as-is on the command line.

`local-ts4`, which needs additional options. The VLISP command `vos::tool` returns an object `job` that represents the tool call. The last line of the example shows the UNIX command line that is generated by the tool call[15]. Figure 4.18 gives an example of a tool that is only authorized to run on certain hosts. In this case, only two hosts, `h50` and `h51` are allowed for the simulator to run on. On invocation time, the VISTA/SFC load balancing mechanism [KPR+96] selects the best available host and starts the simulator.

```
(deftool "tsuprem4"
  :command #"local-ts4"
  :options '(c z #\t 60)
)

(setq job (vos::tool "tsuprem4" #"deck.inp"))

myhost:~$ local-ts4 -c -z -t 60 deck.inp
```

Figure 4.17: System tool encapsulation. Non-standard calling procedure.

```
(deftool "tsuprem4"
  :hosts '("h50" "h51")
)

(setq job (vos::tool "tsuprem4" #"deck.inp"))

h51:~$ tsuprem4 deck.inp
```

Figure 4.18: System tool encapsulation. Authorized execution hosts.

4.6.2 UNIX Networking

A second STI task is the encapsulation of typical deficiencies occurring in a distributed computing environment. As VISTA/SFC is designed to distribute simulation tasks on a number of hosts connected by a network, with different tools executing on different machines and accessing data files on local and remote disks, the stable operation of the network layer is of greatest concern. As shown in Figure 4.19, on UNIX systems, remote executables are started using the remote procedure protocol (RPC), while data files are accessed across the network using the network file system (NFS)[16]. Due to the large size of simulation data files – typically several hundred kilobytes – reading and writing data from the remote host might take more or less time, depending on the bandwidth and the load of the network connection. If two tools are run one after the other, one reading the output of the previous one, situations occur where the RPC for the first tool has already terminated, but its output file has not been written completely by the NFS, or file locking is not synchronized. Starting the second tool immediately after the first one will result in a read error because of incomplete file data[17]. The STI takes care of appropriately scheduling the submission of system jobs to avoid network-induced malfunctions.

[15] VISTA/SFC is fully operational on both UNIX and VMS systems.

[16] As a matter of fact, NFS is based on RPC. See [Ste90b] for a detailed introduction to UNIX network programming.

[17] This might not cause trouble with sequential files; for random access files, it is disastrous.

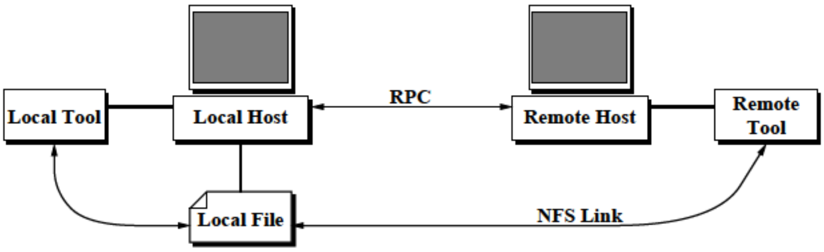


Figure 4.19: Remote tool call and local file: Propagation delay differences between remote job execution and network file data transfer causes loss of synchronization between job termination and file update.

Chapter 5

Semiconductor Process Flow Representation

The fabrication of modern semiconductor devices involves several hundred individual processing operations that are carried out sequentially on the initial semiconductor substrate (*raw wafer*) to manufacture the desired product [Wol90] [SS91] [WMF96]. VLSI process operations comprise a wide variety of different physical and chemical treatments. As they are applied one after the other on the – more or less – planar surface of semiconductor wafer, we speak of *planar technology*.

5.1 Fabrication Process Steps

Process operations include ion implantation for the introduction of dopant materials into the semiconductor substrate, thermal treatments to induce dopant diffusion and oxidation processes, etch and deposition processes to structure the semiconductor surface, spin-on processes to create resist coatings and protective films, optical and other varieties of lithography processes to transfer patterns from masks onto the semiconductor surface to determine the affected regions for subsequent etch and ion implantation processes, chemical-mechanical polishing (CMP) to create an even wafer surface, and others. See [Rug84] [WT86] [Sze88] for a more detailed discussion.

These processes can roughly be subdivided into four main groups that are used (1) to build material layers atop the wafer surface, (2) to transfer geometrical patterns onto the wafers surface, (3) to etch and remove material from the exposed part of the wafer, and (4) to incorporate doping in the wafer bulk. Table 5.1 gives a short list of common fabrication processes.

In addition to these primary types of process operations, other treatments are applied to the wafer to remove dust and oxides from the surface, to increase surface adhesion for subsequent coatings, to harden resist coatings by baking, etc. They do not serve a particular purpose in altering the wafer's surface geometry or in modifying the distribution of dopants in its interior, but are necessary to ensure the smooth and reliable operation of the primary treatments. As a consequence, these steps are usually not explicitly referred to when a process technology

Layer Growth	Epitaxy Thermal Oxidation Spin-On Chemical Vapor Deposition Physical Vapor Deposition
Pattern Transfer	Optical Lithography Electron-Beam Lithography
Etching	Wet (Chemical) Etching Chemical-Mechanical Polishing Physical Dry Etching Chemical-Physical Dry Etching
Doping	Ion Implantation Diffusion

Table 5.1: Examples of VLSI fabrication processes.

is discussed. Nevertheless, their importance for manufacturing must not be underestimated, and they play a crucial role in achieving a sufficiently high yield.

5.2 Process Simulation Tools

For almost all fabrication processes, numerical simulators are available that compute the effects of a given process step or treatment on the wafer. The wafer is represented as some kind of data model in one, two, or three dimensions, depending on the nature of the problem to solve. Some simulators are specialized on one kind of process step, e.g., lithography simulators [KS96a] [KS96b] concentrate on the light-propagation and resist-bleaching phenomena during exposure, while others are capable of computing a sequence of related process steps [Ele91] [STHN91] [WN95] or complete process flows including etching, deposition, ion implantation, and thermal treatments [Tec95].

The level of abstraction for specifying a fabrication step varies from tool to tool. E.g., etch and deposition processes in SUPREM [Tec95] are specified on an *effects*-basis, i.e., the user has to pass the expected effect in terms of the thickness of material removed to the program, whereas the etch and deposition modules PROMIS [SS93] [Str94] and SAMPLE [Ele91] expect etch and deposition rates to be given[1]. Therefore, a clear correspondence between process step parameters and simulator input in general cannot be defined easily.

5.3 Operations, Steps, and Flows

The sequence of process operations that is required to manufacture a given device is called a *process flow*. A process flow is only valid for a given technology (usually named after the minimum achievable feature size), and reads a fixed number of lithography masks which are

[1] Etch and deposition rates are again an abstraction from the treatment of the wafer, e.g., a CVD or RIE process taking place in a given piece of equipment.

derived from layout information provided by the device designer. Figure 5.1 gives a schematic overview of the fabrication of a semiconductor device.

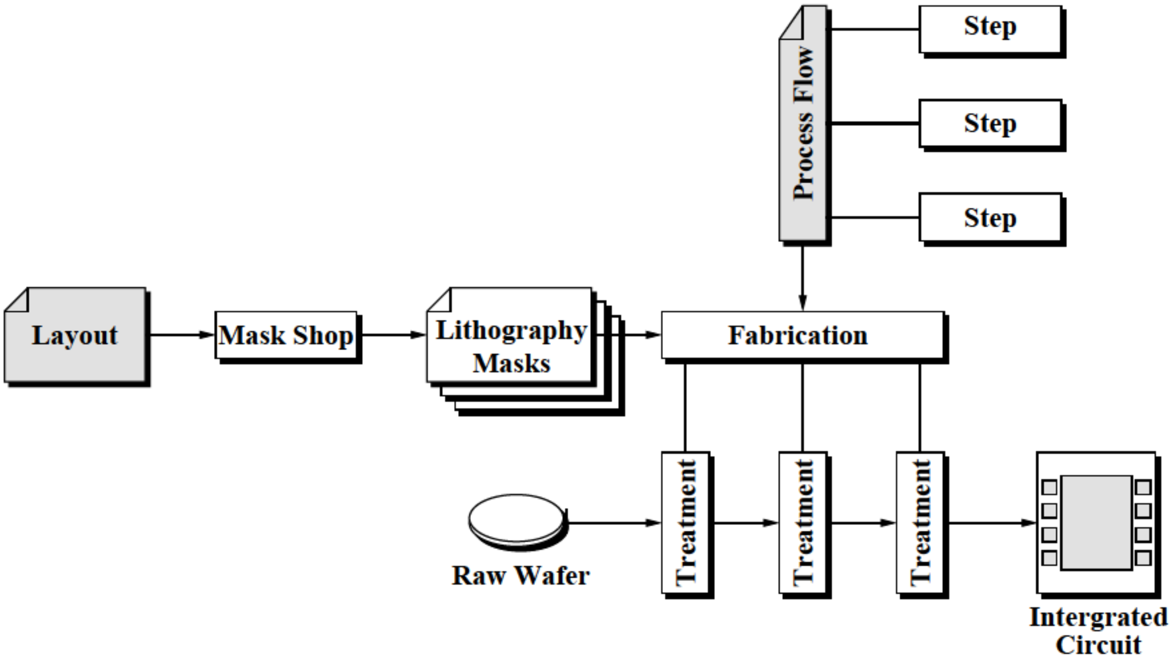


Figure 5.1: Fabrication of an integrated circuit from layout and process flow.

In this text, a *process step* refers to a primary process operation as described above and reflects the fact that an operation is applied to achieve a desired effect in the wafer or in the behavior of the semiconductor device. It is perceived as an atomic building block of a process flow and is usually carried out in a single piece of equipment. As a matter of fact, its granularity depends on what a given piece of equipment is expecting. Although a process step usually represents an indivisible unit of fabrication, it should rather be perceived as a milestone on the run traveler.

Most types of equipment are controlled by recipes that specify the internal operation of the machine during a process step. A recipe in itself contains a sequence of statements and can be viewed as a flow similar to a process flow. To clearly mark the difference from statements appearing in a process flow, we define an *equipment step* to be the basic building block of a recipe. Example of equipment steps include temperature ramp commands, gas flow switches, wafer pull operations, etc.

5.4 Process Flow Representations

Throughout the semiconductor industry, process flow representations (PFR) of various formats are being used to capture, communicate, and store technology data. Unfortunately, these protocols are not shared across different companies, as a lot of implicit information is required to actually translate a design given as process flow and mask data to silicon. These *hidden data* include silent agreements on proprietary procedures, recipes, and etchants, as well as detailed knowledge of the fabrication equipment available in a particular fab.

5.4.1 PFR for Manufacturing

A standardized representation of process flow information forms the basis for process development, process optimization, process centering, and yield improvement, and facilitates process data exchange between TCAD groups and fabrication sites. Attempts to define a standardized vocabulary for representation and interchange of process flow information have met several challenges from the complexity of the physical processes and the variety of equipments and recipes in a rapidly evolving technology.

Boning and co-workers [BMPS92] propose a general semiconductor process modeling framework based on state and state transformation information of wafers, machines, and facilities, and aims at establishing a rigorous mathematical model for the interactions between material and equipment during processing.

The Berkeley Process Flow Language (BPFL) [WR87] [HRW90] provides a procedural description of the fabrication of semiconductor devices; it is based on Common Lisp [Ste90a] and supports different views of a process sequence for different applications. Interpreters are used to translate a BPFL description into the appropriate format for a given tool or equipment.

Wenstrand [HRW90] presents an object-oriented approach to process flow representation, aiming at the integration of manufacturing, simulation, and design information.

The CAD Framework Initiative's Semiconductor Process Representation [GG91] [SG92] seeks to define an informational and procedural, generic and versatile model to capture all process-related information for semiconductor manufacturing.

Durbeck and co-workers [DCB93] focus on the capture and management of fabrication process information by means of objects organized in layers corresponding to the *equipment*, *step*, *process*, and *flow* views[2]

The process specification system SPEC [KN95] employs a similar conceptual approach as [DCB93], subdividing process information into *equipment*, *step*, *sequence*, and *wafer* levels. Moreover, as part of the Microelectronics Manufacturing Science and Technology (MMST) program [BCDH94] [MHM94], it stresses concurrent engineering and revision control of process data items, and is also used for actual wafer processing.

5.4.2 Simulator Input Languages

Some process simulators capable of simulating more than one process step provide process flow specification mechanisms in their input languages [Ele91] [HPW91] [Tec95]. Tailored to a particular tool's view of semiconductor technology, these process flow descriptions cannot

[2] *Equipment recipes, process steps, and process flows contain a great deal of information, and many different kinds of users generate or consume that information, including unit process developers, facility operators, technicians, and managers, as well as process flow and device developer* [DCB93].

be communicated between different tools. The existence of multiple formats for multiple tools has been identified as a significant barrier to the integration and use of these tools [GG92].

5.5 The VISTA/SFC Process Flow Representation

The VISTA/SFC process flow representation provides a convenient format for defining large process flows as sequences of *simulator-dependent* and *simulator-independent* statements. It has been designed primarily for the purpose of chaining together and controlling independently developed, heterogeneous process and device simulators, focusing on an open concept that allows additional statements to be defined and added to the vocabulary easily.

A set of simulator-dependent statements forms the core layer for all process and device simulation applications. They provide direct access to simulator parameters and are primarily used during tool development and tool calibration. Atop of the system-near, simulator-dependent layer, layers for process-specific and equipment-specific statements are provided. Figure 5.2 shows the the structure of these layers.

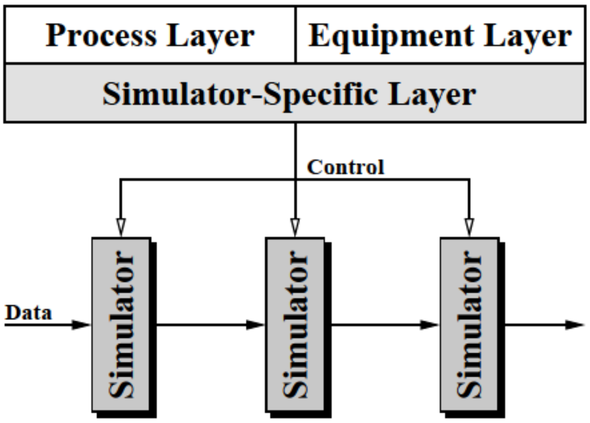


Figure 5.2: Simulator-specific, process-specific, and equipment-specific statements.

Only at the simulator-specific layer the numerical problem to be solved is completely defined in terms of tool calls and respective input parameters[3]. For both the process and the equipment layers, statements have to be mapped to simulator statements before actual computation.

[3] As a matter of fact, uncertainties and ambiguities still exist in terms of the executables invoked on the system level and their respective versions. Two identical system calls on the same system need not produce the same result, if, e.g., the search path (on UNIX) or the definition of a symbol or logical (on VMS) get changed between the calls. Furthermore, installing a new version of a tool may also lead to changes in the results computed. Some of these pitfalls may be circumvented by specifying complete path names, checking the file modification time of executables invoked, etc., but such procedures are rather cumbersome and therefore not widespread.

5.6 Equipment, Processes, and Simulators

A process representation for manufacturing purposes differs greatly from a description of simulation sequences required to simulate a process flow. Although process simulation tools have been devised to correspond – in more or less detail – to fabrication processes steps, this correspondence usually ends at sharing the same set of variables to describe basic processing conditions such as processing time and temperature.

Beyond this set of common parameters, fabrication equipments and simulation tools require quite different input data. While equipment reads instructions and parameter values for wafer movement, gas flow control, etc., which have no representation in the simulator, the simulator needs a host of additional information concerning numerical and computational aspects of its internal operation, which, in general, cannot be derived from process settings. Nevertheless, they might have some influence on the solutions calculated and have to be calibrated to accurately model a fabrication process step taking place in a given piece of equipment.

Process statements in the flow description specify fabrication process steps in a simulator-independent way. They form a technology-focused vocabulary that is used as common representation for TCAD and manufacturing. To capture all relevant technology information, each step can carry any number of attributes such as the precise type of equipment or additional parameters not available in the basic process step.

Equipment statements address specific pieces of equipment and describe equipment settings and procedures as used on the factory floor. Both process and equipment steps have to be mapped to simulator statements before starting simulation.

5.6.1 Mapping

In VISTA/SFC, mapping from process statements to simulator steps is accomplished by *tool agents*, which reconcile equipment and process views with simulator models and input requirements. Figure 5.3 shows the basic mechanism for converting a tool-independent process flow description into a simulator-specific flow.

A tool agent generates the required input for a simulation tool from the data in a process or equipment statement. Each step has attributes attached to specify the simulator and to supply values for simulator-specific parameters.

When determining the sequence of simulation steps corresponding to a process sequence, both the names of the tools to call as well as their respective inputs have to be established[4]. In the first place, the number of possible tool candidates has to be reduced by means of exclusion or explicit preference stated in the flow or in global resources. Once a tool has been decided on, parameter values get filled in from locally defined data, global defaults as well as calibration data by the tool agent, to yield a complete set of simulator input data.

[4] In many TCAD groups, only one dedicated simulation tool exists for a given process step, rendering the tool selection stage trivial. In general, though, a number of choices exist for each process step.

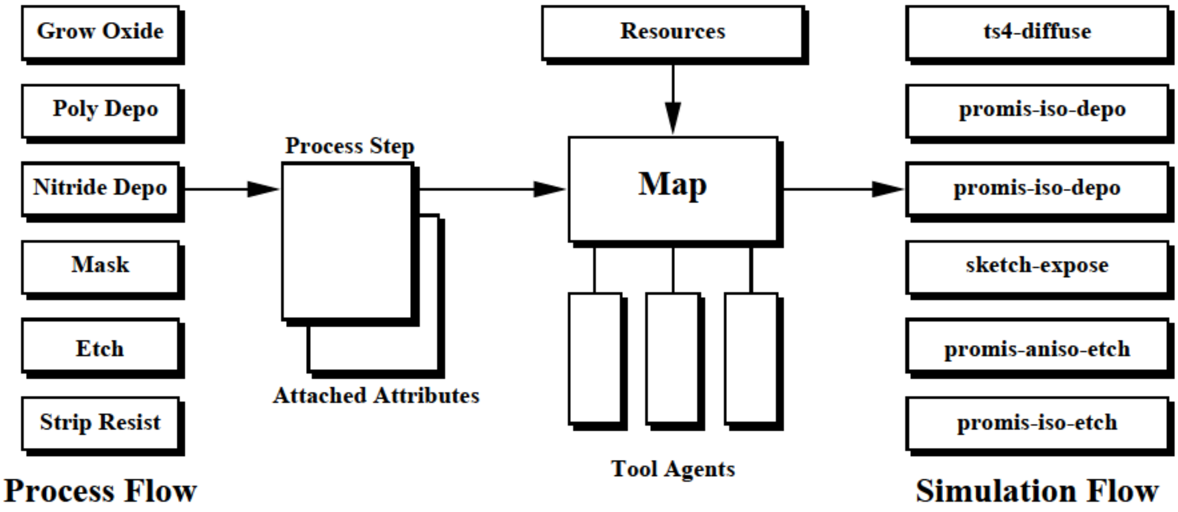


Figure 5.3: Mapping from process steps to tool steps. Tool agents generate simulator input statements for each step in the process flow.

5.7 Structure

The VISTA/SFC process flow representation consists of a sequence of steps to be carried out one after the other. Steps may be grouped in modules to define typical sequences.

5.7.1 Steps

Each step in a process flow is identified by a name. This name is assigned by the user on entering a new step into the flow and is used to reference split points and output data during simulation. Typical names can be registered for a given technology to reflect a company's standards. All input parameters as well as output data items generated during simulation are accessible by symbolic names for use in process analysis and optimization tasks.

5.7.2 Operations

While a step corresponds to a wafer fabrication step, operations are further subdivisions of process steps, describing in detail the sequence of manipulations or treatments applied to a wafer inside the equipment, forming a recipe for a given process step. Whereas process steps are considered building blocks of process flows, operations are interpreted by equipments and simulation tools. Operations are not interpreted by the controller, but represent additional information for tool agents and tools.

5.7.3 Modules

To foster modularity as well as design knowledge encapsulation and reuse, and to reflect the subdivision of industrial process flows in process modules, process submodules, process steps, and operations, the process flow representation supports hierarchical and parameterized process modules that allow the definition of large process flows in terms of predefined blocks (Figure 5.4).

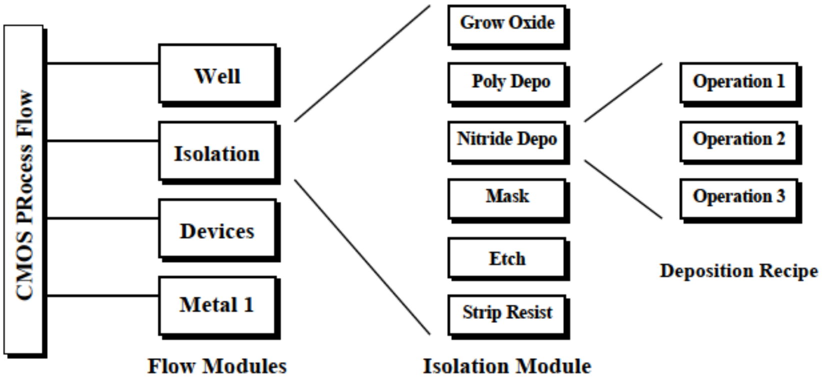


Figure 5.4: Hierarchical process flow representation. A recipe consists of a sequence of operations and corresponds to a step in a process module. Process modules are combined to form a process flow.

Modules contain any number of subordinate modules and process steps. They are either inserted into the process flow description or referenced by symbolic names, the former option ensuring a compact capture of the process flow for convenient data exchange with remote sites, the latter supports sharing of externally stored process module libraries by various users.

A module allows local symbols to be defined that can be used as parameters for all steps and sub-modules belonging to the module. Moreover, step output data can be assigned to local symbols and thus be passed to other steps or upstream in the flow hierarchy.

5.7.4 Libraries

To build up process flows from predefined modules, process libraries can be defined that hold collections of related, possibly parameterized, technology-specific process steps sequences. New statements are defined as specializations of existing steps by assigning a symbolic name. A library contains customized steps and modules, including resources for calibration and tool selection.

5.7.5 Data Flow

By default, it is assumed that each step reads the output of its predecessor. If necessary, a step's output data may be assigned to a symbol that is read by another step further down in the process flow.

5.7.6 Variational Flows

For production use, a semiconductor process description contains a sequence of process steps to be carried out on the wafer substrate, with additional constraints and target values for certain physical and electrical parameters measured during fabrication. During process development, on the other hand, process engineers wish to specify conditional statements, loops, and branch operation in a fashion similar to writing computer programs. However, in order to separate the nominal process description from experimental alterations, modifications to the process flow can be assigned externally, using step names and parameter names and leaving the process flow description file unchanged[5]. See Section 8.4.1 for details.

5.8 Statements

Table 5.2 gives a list of presently integrated simulation tools, respective fabrication processes, and symbolic names. Additional simulators and applications can be easily added by registering a binding function, a set of default values, and some additional information for editing and validity checks.

Table 5.3 gives a list of available basic process steps. To accommodate additional processes and to specify process steps in greater detail, the process flow statement vocabulary is extendible by defining new entries. For a given statement, any number of specializations can be added as new statements with certain parameters set to the desired values without need for additional coding. For example, a *CVD oxide* step is derived from a more general *CVD* step by permanently assigning the value SiO_2[6] to the *material* parameter.

5.9 Programming Interface

For programming purposes, an object-oriented interface has been implemented that simplifies creation and modification of flow descriptions by user-written applications (Figure 5.5), and also provides interactive editing functions for plain text terminal operation.

This interface forms the standard entry point for the generation and manipulation of process flows by other application and tools. For example, an automatic conversion tool uses this interface to produce a VISTA flow description from input deck specifications in a NORMAN [CBK+93] input file.

[5] This approach is also aimed at sharing process flow files among several users, with each of them running different experiments and performing different analyses without need to create private versions of the complete file.

[6] Possible ambiguities in material names are resolved by a material data base [Ins96b] that uses an inheritance-based strategy to identify material specifications.

Simulator	**Process**	**Symbolic Name**
PROMIS Analytical Implant	Ion Implantation	promis-implant
PROMIS Monte Carlo Implant	Ion Implantation	promis-mc-implant
PROMIS Diffuse	Diffusion	promis-diffuse
PROMIS-NT	Diffusion	promis-nt-diffuse
PROMIS Etch	Hemispherical Deposition Isotropic Deposition Unidirectional Deposition Anisotropic Etching Isotropic Etching Reactive Ion Etching	promis-hemi-depo promis-iso-depo promis-uni-depo promis-aniso-etch promis-iso-etch promis-ri-etch
SAMPLE	Conical Deposition Dual-Source Deposition Hemispherical Deposition Isotropic Deposition Planetary Deposition Unidirectional Deposition Anisotropic Etching Isotropic Etching	sample-conical-depo sample-dual-source-depo sample-hemi-depo sample-iso-depo sample-planetary-depo sample-uni-depo sample-aniso-etch sample-iso-etch
SKETCH[a]	Expose Spin-On Strip	sketch-expose sketch-spin-on sketch-strip
SUPREM4	Diffusion Ion Implantation	ts4-diffuse ts4-implant

[a] SKETCH is an auxiliary geometry manipulation tool used for "faking" lithography, deposition and etch processes where no actual numerical simulation of the physical process is required.

Table 5.2: Simulation tools and processes for process simulation.

Process	**Step**
Deposition	process-depo
Etch	process-etch
Expose	process-expose
Furnace	process-furnace
Ion Implantation	process-ion-implantation
Spin On	process-spin-on

Table 5.3: Basic fabrication processes.

```
> (SelectFlow #"~/FLOWS/schumicki.sfe")
#<Object(sfc::flow-edit-class): #140b8a230>
> (ShowFlow)
  1   Start: START
  2    CREATE SUBSTRATE: sketch-create-subdomain
  3    Oxidation: sketch-spin-on
  4    N-well Definition: BLOCK
 32*   Adjust Punch Through: BLOCK
 33     Mask N-well Lithography: BLOCK
 34      Deposit Resist: sketch-spin-on
 36      Expose: sketch-strip
 37      Develop: sketch-strip
 38     Etch Pad-Oxide: promis-ri-etch
 40     THA IMPLANT: promis-implant
 42     PT IMPLANT: promis-implant
 44    Strip Resist: sketch-strip
 45    Field Oxidation: ts4-diffuse
 46    Strip Nitride: sketch-strip
 47    Etchback: promis-ri-etch
 48    Gate Oxidation: ts4-diffuse
 49    Gate Formation: BLOCK
 62    LDD: BLOCK
 75    SOURCE DRAIN REGIONS: BLOCK
 95    Contacts: BLOCK
106    Metal1: BLOCK
117    CVD-Oxide: promis-iso-depo
118    PLAN: sketch-spin-on
119    BACK: promis-iso-etch
120    CONTACTS 2: BLOCK
123    METAL2: BLOCK
> (GotoStep "tha implant")
#<Object(sfc::flow-class): #140b30fd8>

> (ShowStep)
   THA IMPLANT: promis-implant
    anginc = 0.
    damage = off
    dose = 5e+13
    rmoden = 16.
#<Object(sfc::flow-class): #140b30fd8>

> (SetStep :dose 2.7e13)
#<Object(sfc::flow-class): #140b30fd8>

> (GetStep :dose)
2.7e13
>
```

Figure 5.5: Object-oriented flow description interface forinteractive and automatic editing.

5.9.1 File Format

Flow information is stored on the file system using an easily readable, LISP-like format with non-positional parameter passing[7]. Figure 5.6 shows a short sequence of simulator-specific steps defining a lithography module as a block of a larger process flow. The file format is completely encapsulated and need not be accessed by the user or by any application, but only through the programming interface and the GUI for the flow description. It is easy to read with any LISP interpreter or, for C or FORTRAN applications, through a library for hierarchical data exchange (HDX) [Pla96], which provides an object-oriented memory representation of nested data structures and supports reading and writing of nested lists of data.

5.10 Graphical Flow Editor

Defining and maintaining process flows is supported by an interactive GUI-based flow editor (Figure 5.7). In the left part of the window, a hierarchy browser provides direct access to all levels of the flow description, the right part shows details of the currently selected entry. Figure 5.8 shows the selection window for all available simulation tools.

The flow editor is implemented as a layer atop the flow programming interface (see Section 5.9). It operates independently of any file representation, using directly the interface's object-encapsulation.

5.11 Layout Data

The interface between device design and process simulation is based on lithography mask data that are either specified as numerical coordinates or derived directly from the device layout. For two-dimensional simulation, mask coordinates define a one-dimensional sequence of transparent and dark areas. Figure 5.9 shows a two-dimensional cross-section of the wafer with the one-dimensional lithography mask cross-section defining the poly-silicon gate atop the gate oxide layer during exposure and after removing the exposed photo resist and etching the poly-silicon not covered by photo resist.

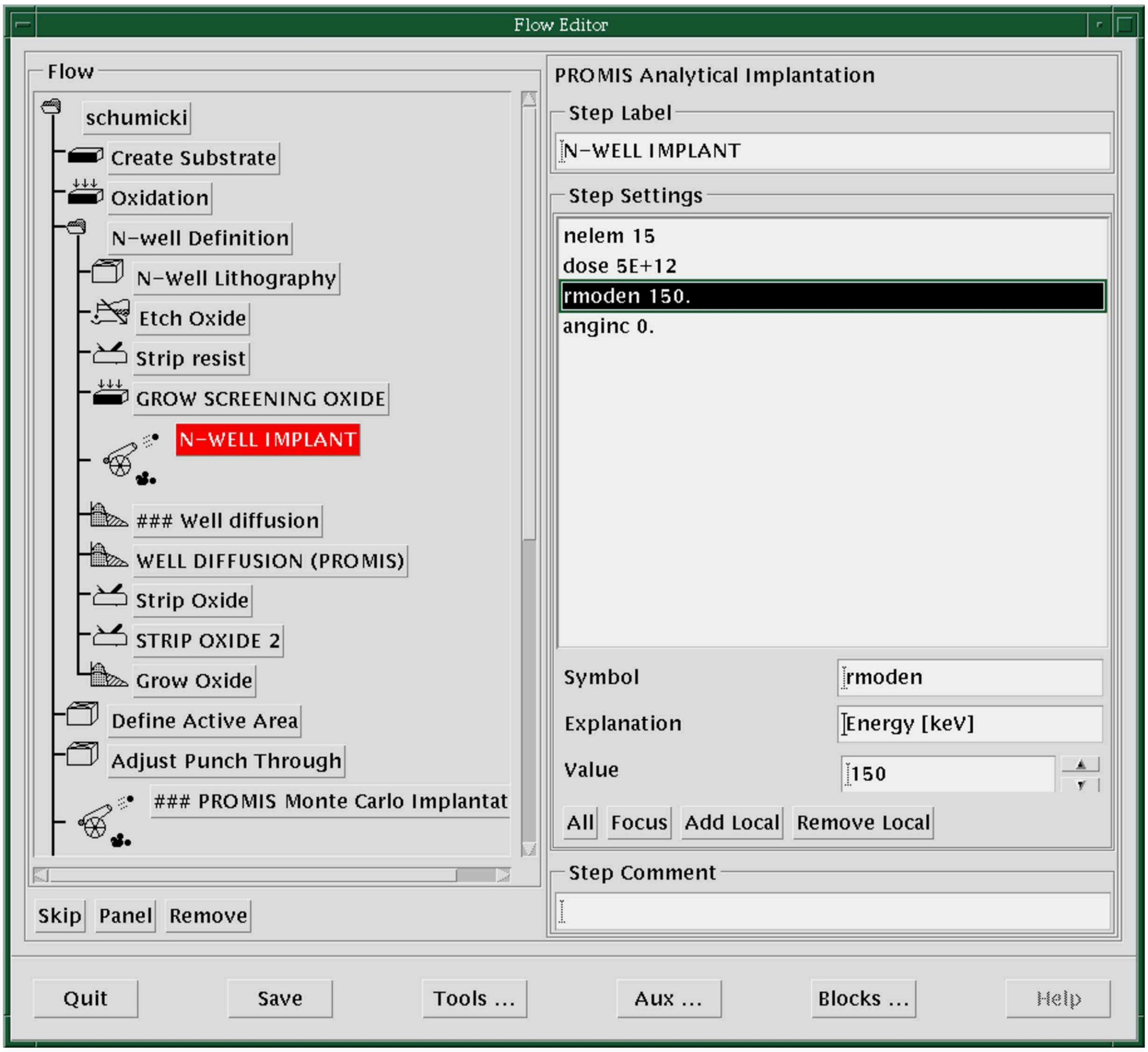
Cut-lines and cross-sections for defining simulation areas are specified in an integrated layout editor [MS96] based on the PIF editor [RHS95] [Rie96][8] Figure 5.10 gives an overview of the generation of one-dimensional mask data from layout data imported either from a ECAD system or entered manually by the user using the PIF editor. The one-dimensional mask file is used in process simulation to define mask coordinates at lithography steps in the process flow. Lithography masks are specified by name, boolean operations are used to to define a location on the wafer surface as a function of various mask geometries.

[7] *Non-positional parameter passing* does not enforce a strict order for passing parameter values, but allows each value to be specified together with the parameter's name. A default value may be specified for each parameter that is not required to be set explicitly.

[8] The editor supports different modes of operation, one providing editing capabilities for layout data as well as for cut-line and sub-domain definition.

```
(node ((assignments NIL)
       (block t)
       (comment NIL)
       (focus NIL)
       (label "SN-Lithography")
       (name "BLOCK")
       (settings NIL)
       (skip NIL)
       (body ((node ((assignments NIL)
                     (block NIL)
                     (comment NIL)
                     (focus NIL)
                     (label "Spin On Resist")
                     (name "spin-on")
                     (settings ((dope NIL)
                                (material "Resist")
                                (ortho-ticks 3)
                                (thickness 2.)
                                (unit "um")))
                     (skip NIL)))
              (node ((assignments NIL)
                     (block NIL)
                     (comment NIL)
                     (focus NIL)
                     (label "SN-Mask")
                     (local-symbols NIL)
                     (name "expose")
                     (settings ((invert NIL)
                                (mask-name "SN")))
                     (skip NIL)))
              (node ((assignments NIL)
                     (block NIL)
                     (class tool)
                     (comment NIL)
                     (focus NIL)
                     (label "Develop")
                     (local-symbols NIL)
                     (name "strip-material")
                     (settings ((material "Exposed")))
                     (skip NIL)))))))
```

Figure 5.6: ASCII representation using LISP syntax of a lithography process module. The module comprises a resist spin-on step, an exposure step (with mask), and a develop step. NIL is the LISP representation for the empty set or void.



Flow

- schumicki
  - Create Substrate
  - Oxidation
  - N-well Definition
    - N-Well Lithography
    - Etch Oxide
    - Strip resist
    - GROW SCREENING OXIDE
    - N-WELL IMPLANT
    - ### Well diffusion
    - WELL DIFFUSION (PROMIS)
    - Strip Oxide
    - STRIP OXIDE 2
    - Grow Oxide
  - Define Active Area
  - Adjust Punch Through
  - ### PROMIS Monte Carlo Implantat

PROMIS Analytical Implantation

Step Label: N-WELL IMPLANT

Step Settings

```
nelem 15
dose 5E+12
rmoden 150.
anginc 0.
```

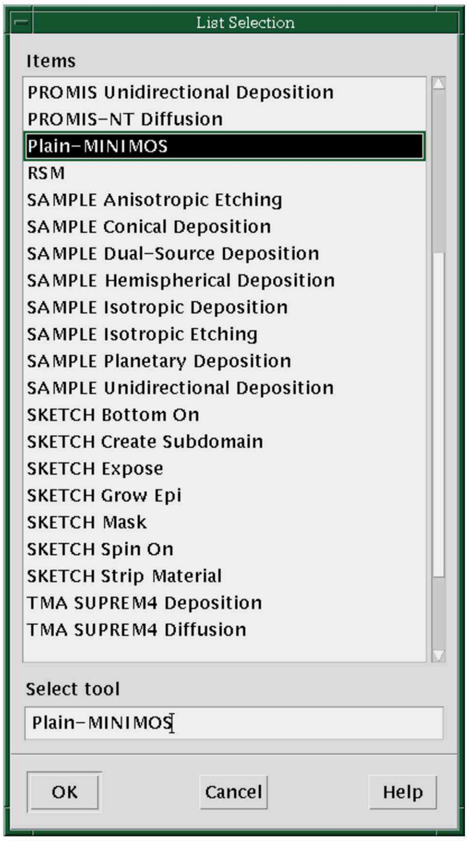
Symbol: rmoden

Explanation: Energy [keV]

Value: 150

Step Comment:

Figure 5.7: Graphical flow editor for hierarchical flow definition: hierarchy browser/editor and step editor.



PROMIS Unidirectional Deposition
PROMIS-NT Diffusion
Plain-MINIMOS
RSM
SAMPLE Anisotropic Etching
SAMPLE Conical Deposition
SAMPLE Dual-Source Deposition
SAMPLE Hemispherical Deposition
SAMPLE Isotropic Deposition
SAMPLE Isotropic Etching
SAMPLE Planetary Deposition
SAMPLE Unidirectional Deposition
SKETCH Bottom On
SKETCH Create Subdomain
SKETCH Expose
SKETCH Grow Epi
SKETCH Mask
SKETCH Spin On
SKETCH Strip Material
TMA SUPREM4 Deposition
TMA SUPREM4 Diffusion

Select tool: Plain-MINIMOS

Figure 5.8: Graphical tool selection interface for available simulators in VISTA/SFC.

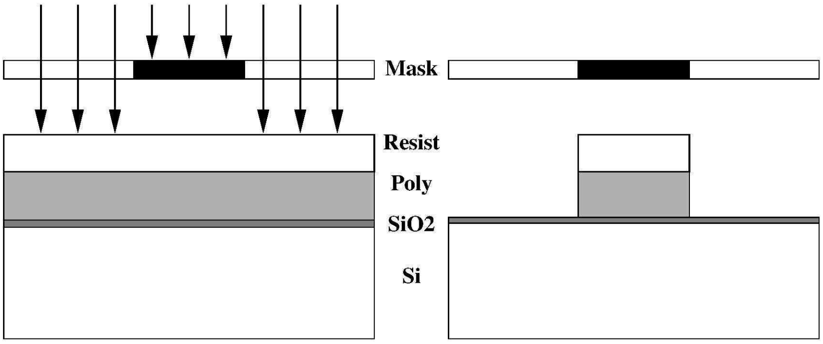


Figure 5.9: Two-dimensional wafer cross-section and one-dimensional lithography mask cross-section during exposure and after etching the poly-silicon layer.

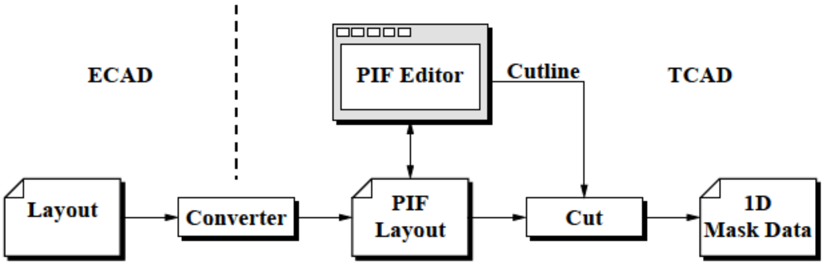


Figure 5.10: Generation of one-dimensional mask data from layout data. A cut-line is entered interactively by the user or supplied by the controlling environment. A converter is available to import layout data from other layout programs.

5.11.1 Variational Masks

When analyzing the behavior of a semiconductor device as a function of its geometrical dimensions, e.g., the dependence of the threshold voltage and other characteristics of a MOS transistor on the gate length (*short channel effects*[9]), all lithography steps have to be supplied with varying coordinate values. These variations are usually performed after the link between ECAD and TCAD has been severed, i.e., in a stand-alone fashion just generating appropriate values without using mask files. While this approach is acceptable when there exists only a loose link with the ECAD domain, is imposes severe limitations on the application of ECAD tools for checking the coordinates generated against constraints stemming from design rules or from the two-dimensional (mask) situation.

To couple ECAD and TCAD more tightly, and also to facilitate the feed-back process once optimum coordinates have been found for a geometrical optimization problem, the variations must already be applied at the layout stage. To this end, layout needs to be parameterized and subjected to modification for each design space point to analyze. Figure 5.11 shows the principal mechanism for deriving varying coordinate values from the layout. As the challenges that arise when parameterizing geometrical data are far from being trivial, active research has still to be done in this area.

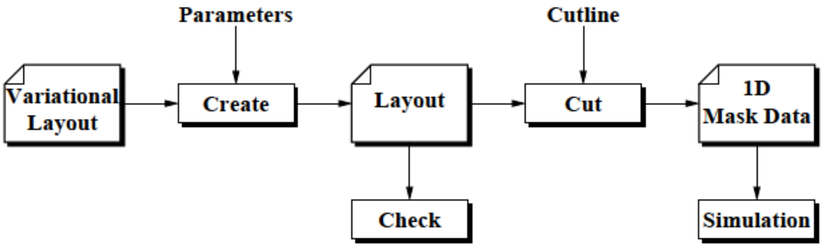


Figure 5.11: Generation of mask data from variational layout. Before data is used, geometrical checks can be run on the layout produced from the template.

[9] See Section 9.2 for an overview, [Wol95b] for a concise treatment.

Chapter 6

Run Control

The VISTA/SFC simulation environment has been designed to support different stages of TCAD process development, from tool development and process design to automatic scheduling and parallelization of split-lot experiments. The automatic simulation of process flows on a computer network using a variety of heterogeneous simulation tools represents the main objective of this work.

Process flows are submitted for simulation either from the graphical user interface, from a text terminal, or from a batch file. In visual mode, a comfortable interface is used to access the run control module and the run data base. Due to the asynchronous nature of the operation of the framework, all user interactions are fully available during active simulations. Moreover, process flow simulations can also be requested by other applications in a client-server manner. For instance a series of process simulation runs may be requested by a design-of-experiments module or an external optimizer, which are notified upon completion to read the generated wafer data and extracted parameters for further processing[1].

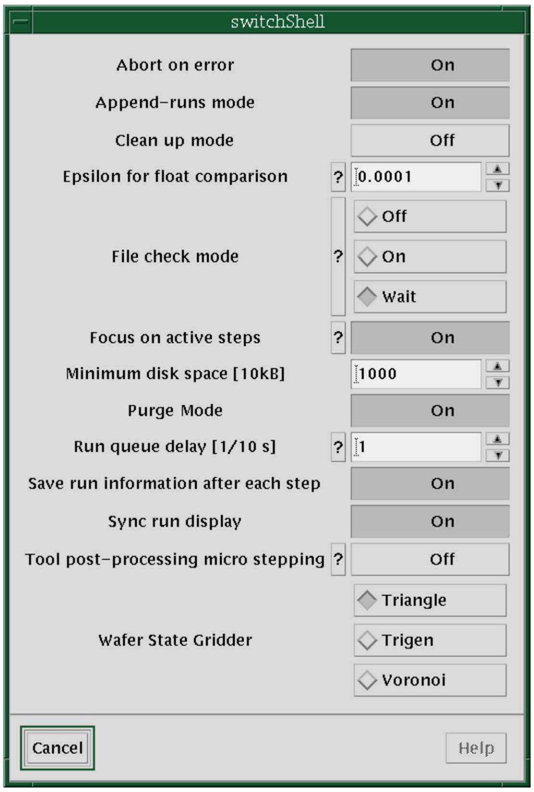
6.1 Projects

All tasks to be accomplished are organized in projects. A project contains a description of the design to work on, holds work-in-progress information, and carries additional resources and user settings that modify the behavior of the framework. Projects are represented as objects at run-time and stored to files between sessions. Figure 6.1 shows a selection of customizable project resources as they appear in the graphical user interface.

6.1.1 Multiple Projects

Due to the asynchronous nature of the operation of the framework and the internal representation of all functional components as objects, it is possible to have any number of projects active at the same time within a single framework session. For instance, a large-scale experiment may be defined and submitted. After that, the user may choose to start working on a different analysis of the problem in question, or on a completely different task. While

[1] See Chapter 8 for more details.



Abort on error: On

Append-runs mode: On

Clean up mode: Off

Epsilon for float comparison ? 0.0001

File check mode ? ○ Off ○ On ● Wait

Focus on active steps ? On

Minimum disk space [10kB]: 1000

Purge Mode: On

Run queue delay [1/10 s] ? 1

Save run information after each step: On

Sync run display: On

Tool post-processing micro stepping ? Off

Wafer State Gridder: ● Triangle ○ Trigen ○ Voronoi

Figure 6.1: GUI for customizable project resources.

all other projects remain active, the user interface focuses on the new project and provides state monitoring and direct access to all data and information. Switching between active projects gives complete control over all work in progress, while at the same time allowing a more detailed view of the selected project.

6.2 Experimental Splits

Process flow split points[2] define positions in the flow where variations in parameter values or process steps occur. Recomputing new experiments only from split points on is a powerful strategy to minimize the total computational effort and to yield results more quickly. Split points are generated by comparing runs to be submitted against the split tree of all existing runs of a given process and determining the most similar one that contains valid output data. Comparison is based on the complete tool parameter set actually used to invoke the tool rather than a potentially incomplete – with respect to the computation of output data – specification of process steps. Figure 6.2 shows three runs with parameter modification at the second and the third step. As the specifications - and thus the results - of the first steps of all runs and of the second steps of runs two and three are identical, they need not be computed. Split points are defined to reflect the flow of data taking place.

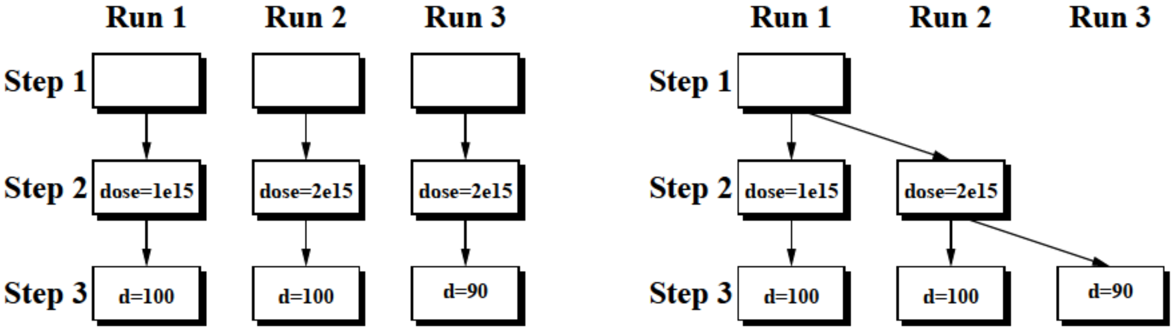


Figure 6.2: Reduction of total computation by splitting. Splitting avoids recomputation of results that remain unchanged across different runs.

When comparing step parameter sets, the numerical accuracy of the comparison has to be limited to an appropriate relative difference, i.e., two floating point numbers are treated as equal when their relative difference is smaller that a given number. In VISTA/SFC, this *epsilon* is adjustable by the user (see Figure 6.1), a value of $1e^{-4}$ has proven to be well-suited for most applications.

6.2.1 Data Sharing Across Split Branches

A single step in a flow, in general, leads to the invocation of a small number of system executables one after the other in fixed sequence. For instance, a pre-processing tool has to be called before a process simulator can be started[3]. Figure 6.3 depicts the case where a step

[2] While a *split point* denotes the location the first difference between two runs, a *split branch* represents the part of a run that differs from its *split parent*.

[3] See Chapter 4 for more details.

leads to the invocation of a sequence of two executables, the first one preparing the data for the second one in a uniform manner without being influenced by any parameter settings. As

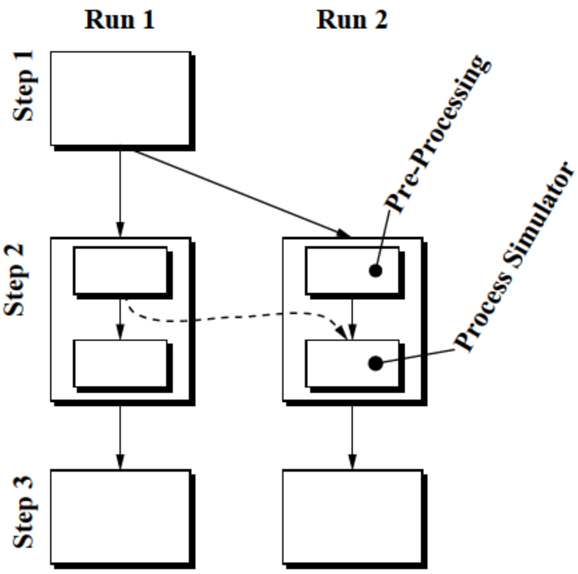


Figure 6.3: Data sharing across splits.

splitting occurs on the step level, and chaining of the two executables is defined at a lower level, the same splitting would only prevent step 1 to be recomputed, but it would not prevent running the pre-processing tool twice on the same input.

To prevent these excess computations, a mechanism has been implemented that takes care of registering operations and looking up existing results. Figure 6.4 gives an overview of the events occuring during simultaneous submission of two runs splitting at a step that requires a sequence of tools to be run. Before the pre-processing operation is started, existing results are looked up using a formal description of the operation. In general, this description is a string containing all information that specifies the operation completely, i.e., the name of the tool, the input files, and all parameters that affect the result. If no existing results are found, the operation is started and a work-in-progress entry is generated to indicate that the result is being computed. At the successful completion, the entry is marked as done. The next step that intends to run this operation receives the existing output file.

During parallel operation, the second inquiry may appear any time after the first one, especially during the time the first operation executes. In this case, no output is available yet, but there is no need to compute it, either, as it is in progress. Therefore, a callback can be registered that gets called when an operation completes to deliver the generated output.

In addition to checking the existence of an operation's result, it is also checked for validity in terms of file modification times. If an operation's result exists and is inquired, the existing output file is checked for being younger than all files that were used as input to the operation. Otherwise, recomputation is initiated. This strategy is similar to the one employed by the UNIX *make* utility.

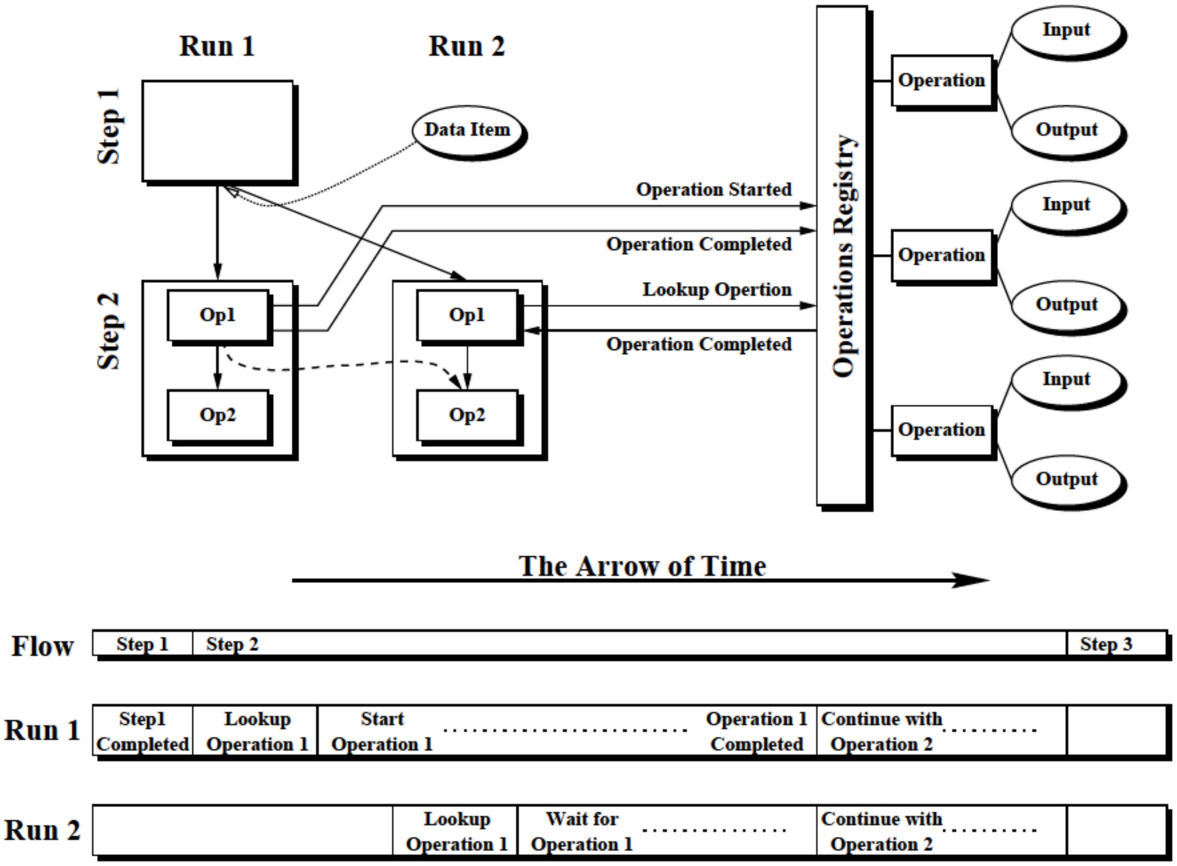


Figure 6.4: Detailed view of the events taking place during data sharing between split branches.

6.3 Incremental and Parallel Simulation

Exploring the design space of a new process usually takes place in an iterative fashion, with one run after the other being submitted by the user for execution. Unless split point generation is disabled by the user, every time a new version of a process flow is submitted, split points are generated automatically at the locations of differences in the process flow; there is no need to specify split points explicitly or in advance.

Due to the asynchronous operation of the run control module, it is possible to submit new process flows while others are still in progress. If a split point is detected at a location that has not yet been computed, the split branch is set to a *waiting* state and continues after the split parent has reached and completed the split point. If a number of runs are started simultaneously, split points are determined, and only completely independent runs start execution.

Once more than one run is active, an automatic load-balancing mechanism [KPR+96] ensures optimum load distribution and minimum total execution time for parallel experiments. All run scheduling information remains consistent across sessions.

6.4 Robust Operation and Error Handling

To achieve smooth and stable execution of large-scale experiments and complex analysis tasks, fault-tolerance is a top priority for a simulation environment. The effect of an error, e.g., a failing system call or a numerical error, has to be limited to the affected experiment, with all other tasks in progress remaining undisturbed[4].

For the integration of external tools, the main concern lies on the detection of errors. To this end, several standard procedures are available in the VISTA/SFC environment to check the return code of system calls, to verify the existence of output files, and to examine the data generated. For instance, generated PIF files are automatically checked for containing the desired dopant distributions or geometry data. These procedures are encapsulated in a class of *system tool* objects automatically employed on every system call issued by the framework. If necessary, additional error handling routines can be defined for each system executable as well as for each application.

Robust operation is a an indispensable prerequisite for taking full advantage of CAD support for technology development and analysis. The reluctance usually encountered when proposing to perform several hundred (or thousand) process flow simulation runs, each involving a number of process steps and leading to the invocation of a variety of simulation tools, is very much based on the fact that in most cases, human interaction is required at many stages of the simulation, making large experiments cumbersome and time-consuming[5]. Therefore, response-surface modeling plays a very important role in technology design and optimization, as it allows to reduce the numbers of actually simulated design space points

[4] One invaluable advantage of subdividing numerical modules and controlling instances into separate executables lies in the fact that segmentation faults, floating point exceptions, and so forth, which otherwise would have a detrimental effect on the framework, affect only the tool they occur in.

[5] In terms of time spent by the user. CPU time, in general, is available in plenty, although shortages occur during peak hours.

[AAY+88] [MOA+91] [CBK+93] [KKW93] [BM94] [GW94] [STA+95] [SLW96]; in Chapter 8, the integration of RSM techniques is treated in more detail.

6.4.1 Error Recovery

Error recovery plays a very crucial role for the convenient and efficient operation of the TCAD environment. A number of error modes can be selected from to determine the behavior of the framework on the occurrence of errors. The options include halting execution of all framework operation, halting the affected run, automatically retrying the erroneous step for a specified number of times, and ignoring the error condition. Furthermore, great care has been exerted to design framework modules as fault tolerant as possible. As a consequence, the user is liberated from almost all manual interference on errors and malfunctions; even after a crash in full operation the session can be resumed by just restarting the framework.

6.5 Development and Debugging

During the early stages of TCAD support for a new technology, the point of main effort shifts from the adaption of existing tools or the development of new tools to the application of these tools to technology analysis and development[6]. Errors, flaws, and mistakes occur in both realms. The VISTA/SFC TCAD environment is designed as a development and production platform and tries to cover all aspects of technology development equally.

6.5.1 Process Development

When designing a new process flow for technology modeling, often several trials have to be made before a process completely runs without any user interference. This is especially true when using different simulation tools together, as they are usually very intolerant with respect to even the slightest flaws in their input data. Critical settings have to be chosen carefully to generate a stable configuration capable of modeling a process for a wide variety of parameter settings[7].

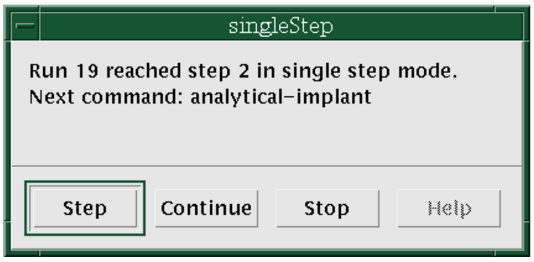
From the simulation environment point of view, some helpful support can be provided to ease the pains of these design stages. Keeping all temporary simulation results available for back-stepping purposes aims at efficiently supporting the early stages of TCAD process design; see Section 7.2 for details. Single-step operation, stop-and-resume operation, and simple reworking further add in exploring possible solutions.

[6] A tendency exists with some companies to delegate some of these aspects, e.g., the tool development part, to external institutions, arguing that the expert status of such dedicated providers increases the quality of the service rendered and decreases its costs. Nevertheless, we feel that the close interaction between technology and TCAD groups is of vital interest for both parties, because it minimizes reaction time and fosters each side's insight into the other one's needs and concerns.

[7] Variations of the device geometry by changing lithography mask dimensions exert the most harmful influence on the stability of the final simulation results as they induce changes in the grids used by process and device simulation tools.

6.5.2 Single-Step Operation

A *single-step* mode allows walking through a process flow on a step-by-step basis, examining the computed results and applying changes to parameter settings as necessary. If a graphical user interface is available, a confirmation dialog prompts the user before each step (Figure 6.5). The available choices are to start the next step, to continue in normal operation mode, and to stop the run.



Run 19 reached step 2 in single step mode.
Next command: analytical-implant

Figure 6.5: GUI dialog for single-step operation.

6.5.3 Stop and Resume

The *stop* and *resume* commands provide a convenient way to halt execution of selected runs or of all active runs, and to continue at a later time without any inconsistencies in the data or control information. When a run is stopped, if waits for all its active system processes to terminate and halts before the next step is started.

6.5.4 Rework

The *rework* command initiates reworking of a run from a specified step on with optionally modified parameters, providing the quickest way possible to explore the effects of code alterations on otherwise unchanged tasks, or to quickly apply variations to parameter values[8].

6.6 Graphical User Interface

For interactive operation, a graphical user interface has been designed and implemented that intends to present all important information on work-in-progress, data base contents, system status, etc., in an intuitive and convenient way and to support all available functionality without imposing restrictions or enforcing “unnatural” procedures of operation[9].

[8] Note that code modifications can be quickly applied to the VISTA/SFC environment by loading appropriate LISP files. Being read by the interpreter, such modifications have immediate effect; there is no need to leave a session for this purpose.

[9] A major complication in building graphical user interfaces atop any application arises from the inherently parallel nature of access through the GUI. Strict and rigorous *state checking* is indispensable to ensure robust and reliable operation.

Great care has been exerted to design the visual user interface as ergonomic and unobtrusive as possible[10], refraining especially from using a multitude of pop-up windows, but using a rather rigid screen layout with fixed locations for most widget items. Nevertheless, the screen layout is customizable to fit all applications' needs. Adding functionality to the visual user interface and changing its look–and–feel are simplified by a set of functions that provide simplified access to the framework's widget library. User interface reactions may be modified at run time without need for interrupting a session.

Figure 6.6 shows the visual user interface during parallel execution of a split-lot experiment on a UNIX workstation cluster. The upper part of the screen displays the experiment split tree generated for all known experiments for the selected process flow. The middle part of the screen shows a monitor window listing all queued and active system jobs for external simulators and auxiliary tools. The lower part indicates the states of all available machines that are being used by the simulation environment.

In the upper part, runs appear from left to right, the step sequence flows from top to bottom. The topmost entry of each column, carrying a number, represents the entire experiment, the leftmost column shows the step names as defined in the flow description. The remaining matrix has an entry for each step of each experiment. Split branches are shown to the right of the split parent. To find the parent run of a split run, move left until you run into a step entry.

6.6.1 Color Coding

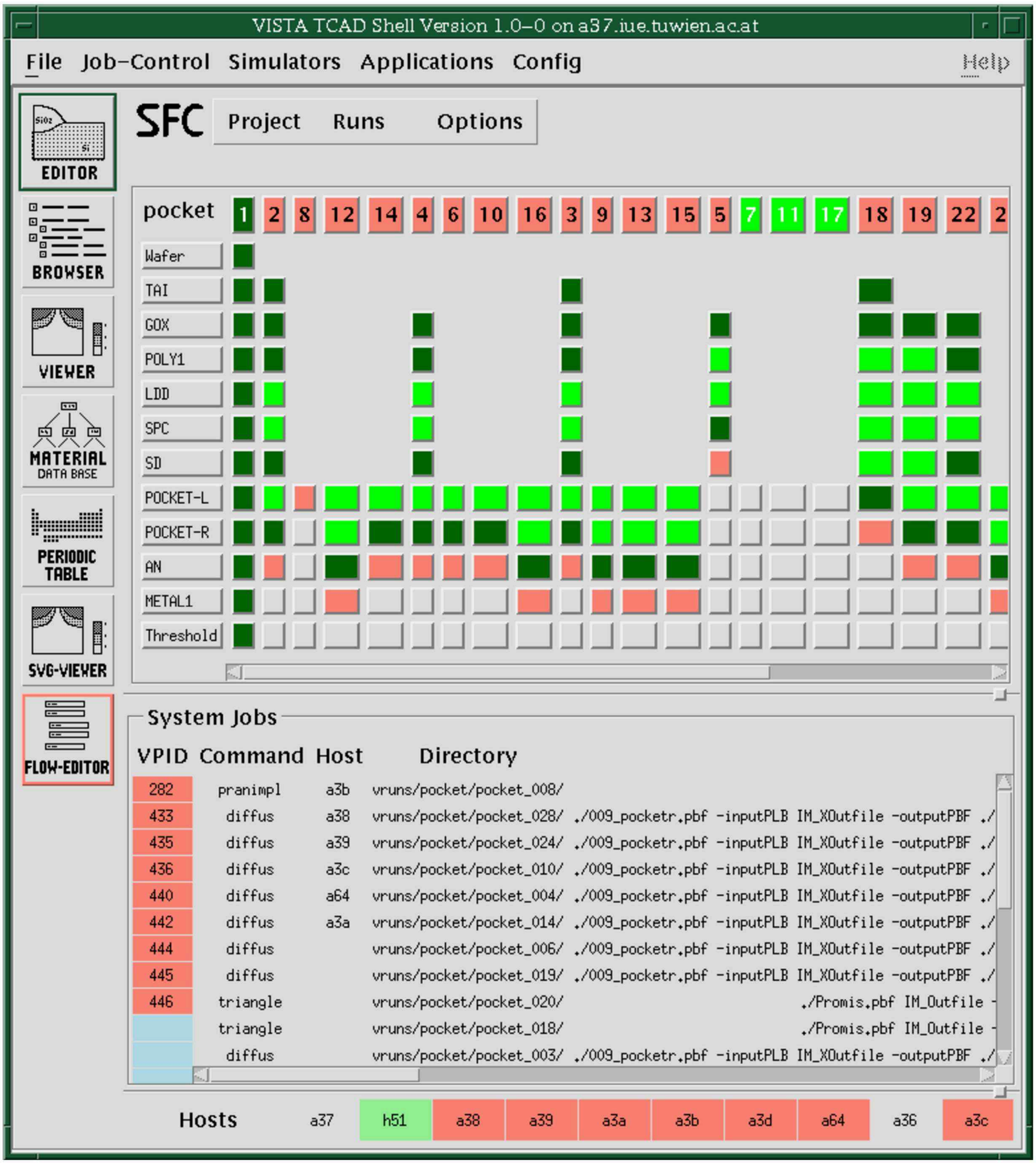
Widget color-coding capability has been added to the framework's GUI functionality, because it has become clear that the consistent use of colors for the indication of system states is very helpful in quickly assessing the state and operation of TCAD tools and framework components. For the run table display, white is used for *waiting* runs, red for *active*, and dark green for *done* runs. *Aborted* runs are shown in an aggressive *orange* tone. Additional states not displayed are *stopped*, indicated by a dark blue appearance, and *empty* (light green), the latter indicating a successful completion with output data already purged to save disk space. Table 6.1 summarizes the color assignments chosen[11].

6.6.2 Managing Runs

At each run entry, a pop-up menu is available to directly access run information, to remove entire runs, and to stop and resume execution. All operation modes and debugging aids as described in section 6.5 are directly accessible in the GUI.

[10] See [KGMB94] for a detailed account of import design aspects of graphical user interfaces for CAD frameworks.

[11] As a matter of fact, the actual value of the colors used for displaying states is specified in the user's X resource file, e.g., `.Xresources`. The environment uses names like `"active"` or `"busy"` to modify the display state of visible objects. The colors presented here have proven to convey most concisely the intended meaning.



pocket	1	2	8	12	14	4	6	10	16	3	9	13	15	5	7	11	17	18	19	22	2
Wafer	■																				
TAI	■	■								■								■			
GOX	■	■				■				■				■				■	■	■	
POLY1	■	■				■				■				🟩				🟩	🟩	■	
LDD	■	🟩				🟩				🟩				🟩				🟩	🟩	🟩	
SPC	■	🟩				🟩				🟩				■				🟩	🟩	🟩	
SD	■	■				■				■				🟥				🟩	🟩	■	
POCKET-L	■	🟩	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	□	□	□	□	■	🟩	🟩	🟩
POCKET-R	■	■	□	🟩	■	■	■	■	🟩	■	🟩	🟩	🟩	□	□	□	□	🟥	■	■	🟩
AN	■	🟥	□	■	🟥	🟥	🟥	🟥	■	🟥	■	■	■	□	□	□	□	□	🟥	🟥	■
METAL1	■	□	□	🟥	□	□	□	□	🟥	□	🟥	🟥	🟥	□	□	□	□	□	□	□	🟥
Threshold	■	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□

System Jobs

```
VPID Command  Host  Directory
282  pranimpl  a3b  vruns/pocket/pocket_008/
433  diffus    a38  vruns/pocket/pocket_028/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
435  diffus    a39  vruns/pocket/pocket_024/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
436  diffus    a3c  vruns/pocket/pocket_010/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
440  diffus    a64  vruns/pocket/pocket_004/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
442  diffus    a3a  vruns/pocket/pocket_014/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
444  diffus         vruns/pocket/pocket_006/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
445  diffus         vruns/pocket/pocket_019/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
446  triangle       vruns/pocket/pocket_020/                              ./Promis.pbf IM_Outfile -
     triangle       vruns/pocket/pocket_018/                              ./Promis.pbf IM_Outfile -
     diffus         vruns/pocket/pocket_003/ ./009_pocketr.pbf -inputPLB IM_XOutfile -outputPBF ./
```

Hosts	a37	h51	a38	a39	a3a	a3b	a3d	a64	a36	a3c

Figure 6.6: GUI for work-in-progress and status information for runs, system jobs, and hosts.

State	Color	Description
`aborted`	orange	An error has occurred during execution and the the run has been aborted.
`active`	red	*Run:* Currently in progress, submitting system jobs and stepping down the flow. *Step:* Handling system jobs and data for this step.
`done`	dark green	*Run:* All steps completed successfully. *Step:* Successfully completed, data available.
`empty`	light green	*Step:* Successfully completed, data has been purged.
`stopped`	dark blue	*Run:* Stopped by the user and deactivated.
`stopping`	light blue	*Run:* Stopped by the user and waiting for system jobs to complete.
`waiting`	light green	*Run:* Activated and waiting for another run's results at the split point to become available.

Table 6.1: Summary of color-coding for state display.

6.6.3 Viewing Output Data

At each step entry, all available output can be directly accessed and displayed in accordance with its data type. For example, Figure 6.7 shows the available choices after a MINIMOS step. The highlighted selection `DATA DRAINCURRENT 1.18E-11 ...` identifies a series of drain current values of a NMOS transistor calculated by MINIMOS. On selecting this entry, a plotting tool[12] is activated to display a graph of the the drain current values. Figure 6.8 shows the interactive plotting tool `xpif1d`[13] displaying the set of drain current values generated by the MINIMOS step.

For wafer data that is stored in the PIF format, not only the PIF file, but also the different data sets contained in the file are directly accessible[14].

In this manner, at a simple glance one can check, e.g., what kind of dopants are present at a given position of a run, and invoke the `xpif2d` visualization program for further examination of the dopant distribution in the wafer.

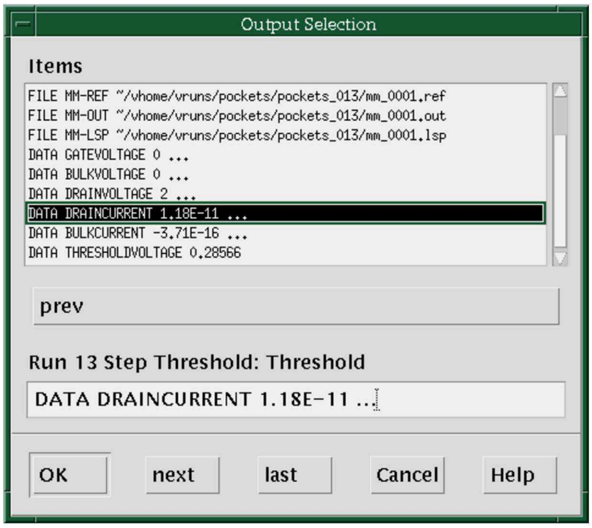
6.7 Text-Terminal and Batch-Mode Operation

Although most of the concerns employed in the design of the VISTA/SFC simulation environment are geared towards easy, convenient, and robust interactive operation, TCAD applications exist where a graphical user interface is of no help, or even detrimental to smooth operation.

[12] The actual program to invoke is determined by the user as a configuration variable of the framework. Available plotting tools on UNIX include `gnuplot`, `xmgr`, `xpif1d`, and a built-in visualization module based on the SVG library [Hal94]; additional plotting tools can be added to this list easily.

[13] Xpif1d is a one-dimensional visualization tool based on Sander's XgPlot [San92].

[14] One invaluable advantage of the PIF data format lies in the random access it provides. Therefore, names and types of all data items present in a PIF file can be easily inquired. The handles returned may then be used for reading the data at a later time.



```
FILE MM-REF ~/vhome/vruns/pockets/pockets_013/mm_0001.ref
FILE MM-OUT ~/vhome/vruns/pockets/pockets_013/mm_0001.out
FILE MM-LSP ~/vhome/vruns/pockets/pockets_013/mm_0001.lsp
DATA GATEVOLTAGE 0 ...
DATA BULKVOLTAGE 0 ...
DATA DRAINVOLTAGE 2 ...
DATA DRAINCURRENT 1.18E-11 ...
DATA BULKCURRENT -3.71E-16 ...
DATA THRESHOLDVOLTAGE 0.28566
```

Run 13 Step Threshold: Threshold

DATA DRAINCURRENT 1.18E-11 ...

Figure 6.7: Output data selection dialog. Direct access is provided to all data generated by a step. The data type determines the viewer that is used for displaying the data item selected.

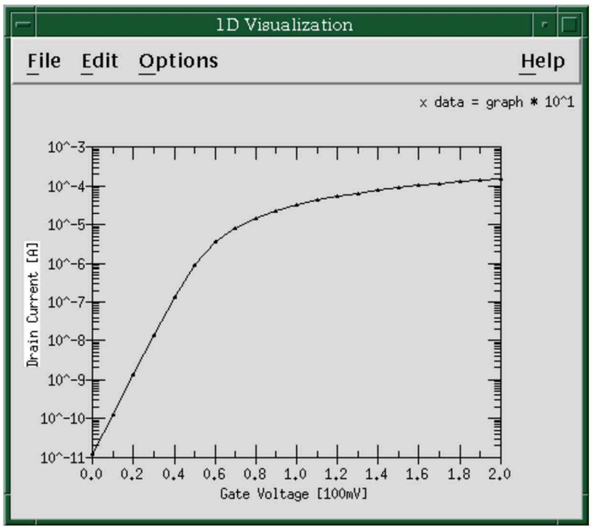


Figure 6.8: **Xpif1d** is one of a number of available plotting tools for viewing one-dimensional data sets.

6.7.1 Why a GUI is not enough

Whereas the GUI provides valuable support for the interactively defining new tasks and for viewing and analyzing simulated results, it imposes some restriction on the versatility of the simulation environment.

- First of all, it requires a connection to a display, which makes it impractical for use across phone lines. Furthermore, once a task has been defined, the user might wish to leave the terminal and have the task completed autonomously. With a permanent GUI, the display has to remain active all the time, and the user cannot log out of the session without stopping the application.
- Secondly, the representation of the functional components at the GUI level, in general, cannot completely reflect all possible operations, but supports only a subset of possible applications. One can imagine to define tasks and applications that are very difficult to support on the GUI level.
- Thirdly, when intending to integrate the VISTA/SFC environment with other tools in a client–server fashion, a permanent GUI is a severe impediment for all automation tasks. For instance, when operated as a simulation server for a larger TCAD group, the GUI is provided by some client application that goes in between the user and the server process; Figure 6.9 sketches the situation.

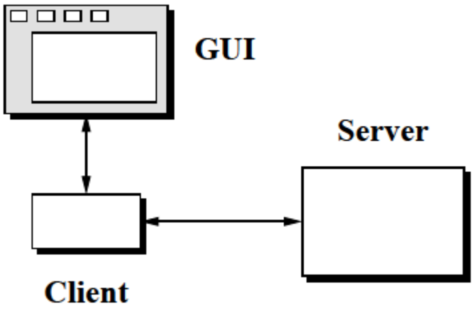


Figure 6.9: When operated as a server for other applications, VISTA/SFC is run in text-terminal mode, the GUI being provided by the client.

Therefore, emphasis has been put on completely separating the graphical user interface from the rest of the simulation environment. Both text-terminal and batch-mode operations are fully supported.

6.7.2 Text-Terminal Operation

All functionality of the simulation environment is always accessible by entering commands at the prompt of the VLISP interpreter. If a GUI is present, it correctly reflects the internal state produced by all commands entered, i.e., the text-input mode may be used simultaneously with the graphical user interface. In a typical application, a file is loaded at the beginning of the session that describes the tasks to perform, and the user enters commands to start and stop jobs, to view work-in-progress information, to switch between projects, etc.

6.7.3 Batch-Mode Operation

Starting the simulation environment with an input deck allows the evaluation of arbitrary tasks, taking full advantage of the functionality of the VLISP interpreter and all other modules. The input deck contains LISP code to define process flows (see Section 5.9), to create parameterized tasks (see Section 8.9), and to submit simulation jobs.

To establish a communication mechanism with the VISTA/SFC environment even in detached operation, i.e., without connection to a terminal or a display, and without using socket connections, a simple, file-based, one-directional data transfer mechanism has been implemented. A temporary input file is loaded by the environment upon reception of a signal (Figure 6.10). The input file can contain any statements valid in an input deck and provides a simple means of inquiring status information, viewing available results, and starting or halting operation of the detached process.

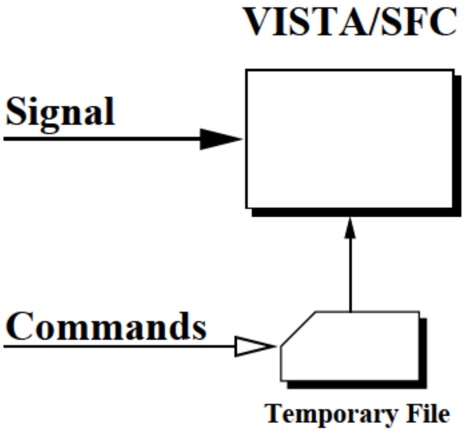


Figure 6.10: Simple signal-based remote control mechanism. Signals are used to initiate reading input files during detached operation.

Chapter 7

Data Management

Automatic execution of multiple versions of large process flow and device simulation tasks generates great amounts of data that have to be managed by the simulation environment in order to liberate the user from tedious file and directory manipulations. Persistent and failure-save storage as well as simple and straightforward retrieval and presentation of existing results to avoid unnecessary re-computations during sessions as well as across sessions form the key issues of this chapter.

7.1 Simulation Data

Semiconductor process and device simulation typically generates large amounts of data at each step. During process simulation, each step produces a two-dimensional[1] representation of the semiconductor wafer that includes a boundary representation of the wafer domain under consideration, material data for different areas of the wafer, grids for carrying distributed quantities along boundaries and in the wafer bulk, and the values of these distributed quantities on their respective grids. In process simulation, distributed quantities include the concentrations of total and electrically active dopants, of silicon interstitials, of dopant clusters, and the distributions of amorphization, strain, and stress in the material; in device simulation, the distributions of the carrier velocities, of the current density, of the electrical potential, and of the temperature.

During device simulation, electrical quantities – either as lump values or as distributed values on grids – are generated. A MINIMOS [SSP80] [FHH$^+$94] simulation of a MOS transistor, e.g., calculates voltages and currents for all electrodes, parameters such as the maximum values of mobility and drift velocity of electrons and holes, and their respective distributions. All these data are produced at each operating point and make for an impressive pile of numbers.

[1] Although the operation of the VISTA/SFC environment is independent of the geometrical dimension of the simulation, two-dimensional applications are the most common ones. One-dimensional simulations of modern VLSI technology cannot accurately enough model the complex topological situation found in state-of-the-art devices, three-dimensional simulations are still too slow and produce too much data to be applied routinely in an automatic fashion.

In general, the number and diversity of physical quantities computed during simulation depends only on the sophistication of the models employed. Therefore, no restrictions should be established or enforced with respect to the nature of simulation data handled by a TCAD environment.

For the sheer amount of data generated during simulation, it is not feasible to store tool output data in a database and extract it again when necessary, but it was decided to use a file-based concept to keep data transfer overhead as small as possible.

7.1.1 Data Formats

Designed to operate independently of any given data format, the VISTA/SFC simulation environment supports arbitrary types of data files.

Wafer Representation

For wafer representation, the Profile Interchange Format PIF[Duv88] [FFS+91][2] and the Technology Interchange Format (TIF) [Tec95] are the most commonly used within VISTA/SFC[3].

The PIF format is considered the native data format of the framework; built in pre-processors and post-processors for geometry editing and viewing of grids and distributed quantities are provided [Rie96] [Ins96c], and VLISP function calls are available to write and read data in PIF format.

7.1.2 Data Files

By default, all files generated by tools are kept on the file system until they are purged explicitly by the user. These files include output data files, log files, and others. In situations where disk space is an issue, automatic purging can be enabled. In this case, output files that are no longer needed are deleted automatically.

7.1.3 Tool Output Data

In addition to wafer data, tools may generate any number of data items of arbitrary types. These data are made available to the controlling environment by registering them after termination of the tool run, specifying a name and a type. Types either belong to a group of predefined, standard data types (Table 7.1), or are defined by the user. For instance, a *threshold* type would denote a data item that contains the extracted threshold voltage after a device simulation step[4]. Type specifiers are used to search for required input data during run-time of simulation, to select appropriate post-processing and display tools,

[2] Other implementations exist apart from [FFS+91]. See [WN91] [BHW91] for details. .

[3] The Semiconductor Wafer Representation (SWR) [otCTT92a] [otCTT92b] offers an alternative that has not been used too widely yet.

[4] A type specifier not only determines the data format of a data item, but may also contain information on its meaning.

DATA	Generic LISP data
DOPING	PIF dopant data
FILE	Text file
GEOMETRY	PIF geometry data
MEDICI-FILE	MEDICI output file
PLB	PIF logical
PS-FILE	Postscript file
STR-FILE	SUPREM structure file

Table 7.1: Examples of available tool output data types.

7.1.4 Meta-Data

Meta-Data holds information about the contents of data and about their relation with other data. E.g., the fact that a given file on the file system has been generated by a certain simulation tool on behalf of a certain process step in a process flow is called *meta-information*; all meta-data are stored in a data-base implemented as a class of VLISP objects.

7.1.5 Persistence and Robustness

While in principle all data generated by simulation tools is completely derivable from a simulator-specific flow description, computation costs (in terms of time and efforts) demand that all results be kept accessible from the moment they are produced on until they are explicitly removed. This holds for mere sequential operation as well as for parallel computation of split-lot experiments. More precisely, it is not acceptable to lose precious data due to malfunctions of a tool, the framework, the operating system, the network, or some other component of the computing environment. In VISTA/SFC, great care has been exerted to ensure a continual update of information stored on the file system to reflect the current state of all tasks in progress.

7.2 Splitting and Data Dependency

During multi-step simulation, a step uses its predecessor's output to generate output on its own. Therefore, all data produced by a step depend on the data produced by previous step. For split-experiments, all runs – and all data of all pertaining steps – that branch after a given step depend on the the parent's step data. Figure 7.1 shows three runs created from a flow with three steps, with runs number 2 and 3 splitting from run 1 after step 1. If run 1 is removed, runs 2 and 3 are said to be *orphaned*, indicating the fact that their split parent is not available any more . Note that only data from the second step on are available in runs 2 and 3.

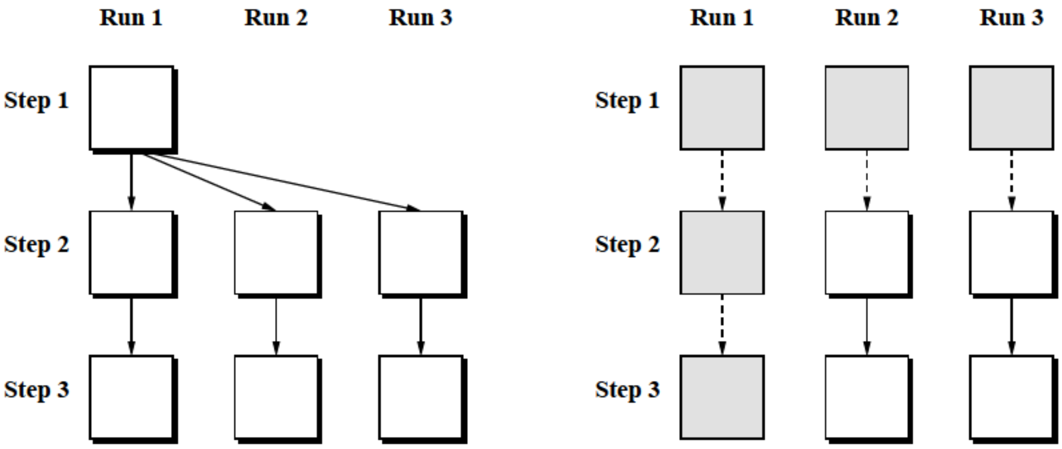


Figure 7.1: Runs 2 and 3 split from run 1 after step 1. Data for step 1 is only available in run 1. After removing run 1, runs 2 and 3 are orphaned. Information about how step 1 was produced is maintained, but output data are not available.

Chapter 8

Task–Level TCAD Applications

The process flow simulation capabilities presented in the previous chapters provide a sound basis for applications that use results from simulation to model and analyze semiconductor technology. The term *task level* carries different meanings in the literature, depending on the point of view and the objectives in a particular context. See [KGMB94] for an extensive discussion of the subtle differences in terminology, approach, and philosophy regarding the definition of *task* in the CAD domain[1]. In this work, it refers to TCAD applications that use results obtained from process and device simulation for further analysis. Figure 8.1 shows the relation between tasks, tools, and tool control level in the VISTA/SFC environment.

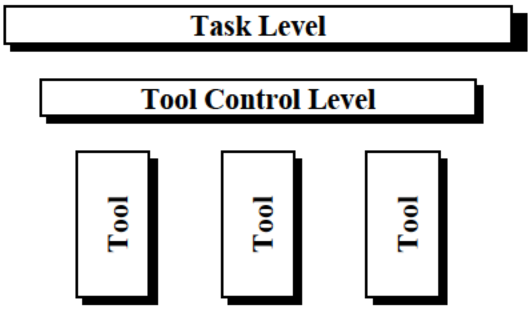


Figure 8.1: Tasks, tools, and tool control level in VISTA/SFC.

Process simulation, thus, is not regarded as a task-level application on its own, but just delivers the necessary results for more complex analyses. All of these applications seek to gain insight into a particular process or process technology by sampling the design space and identifying and modeling the underlying mechanism. Simulation[2] is used as a substitute for measurements and real-observations for cost and time reasons (Figure 8.2).

[1] Agreement exists as to the meaning of the word itself in the English language. In [AHD83], a task means a *difficult or tedious undertaking*, [Web81] defines it as *something hard or unpleasant that has to be done*.

[2] Almost all process and device simulators expect exact values as inputs and generate exact values as outputs, in stark contrast with physical reality. See Section 2.1.8 for methodologies employed to represent nature's fuzziness in the virtual world of TCAD.

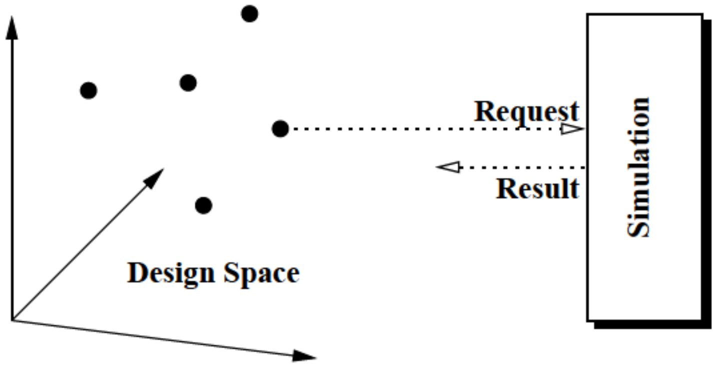


Figure 8.2: Design-space probing and simulation. Only three dimensions of the n-dimensional design space are shown for clarity.

Design space probing does not necessarily lead to a valid result, but there exists always the possibility of selecting a design point that is not allowed for a given design. Consequently, the failure of a simulation tool should not be regarded as an unlikely event, but rather as a regular case, indicating a range fault in the design space. If this fundamental insight is applied properly to the design of tool-control frameworks, a more stable and robust operation is achieved.

8.1 Design Parameterization and Variational Design

As a prerequisite for all further analysis, a design has to be able to be modified in order to be subjected to automatic exploration of the design space. If the modification affects only the value of certain parameter settings while leaving the structure of the problem untouched, we speak of a *parameterized* design. In the case where the structure of the problem is modified, e.g., the sequence of process steps is altered or different paths are taken from the beginning to the end of the flow, we call it *structurally variational*, or *variational* for short.

Obviously, this difference depends on the level of abstraction at which the design is described. In many cases, and, most prominently, in geometry modeling, the modification of parameter values leads to structurally different results, e.g., changing layout parameters such as line widths or distances between lines may lead to completely different lithography masks, resulting in wafer geometries that are topologically dissimilar (Figure 8.3). In an analogous fashion, the modification of an etch rate in an etch process step, e.g., may lead to completely distinct wafer topologies.

For process-flow related task-level applications, no structural changes in the process flow, i.e., the structure of the design, are admitted. In the case where comparisons between process flows with different step sequences have are to be made, the difference in the sequence has to be mapped to differences in some parameter values that modulate the impact of a given step. For example, if a comparison is to be made between two different process options, one including an additional implantation step that the other one is lacking, the problem is most

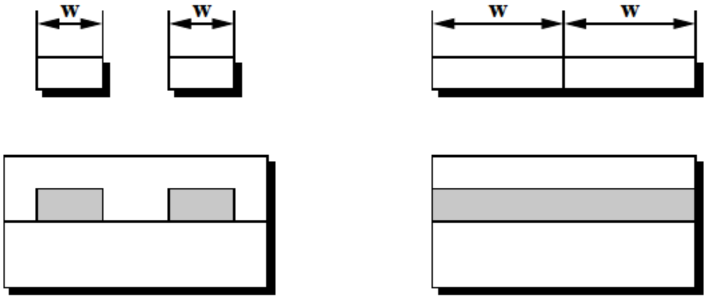


Figure 8.3: Layout parameter modifications cause topologically distinct wafer geometries.

conveniently modeled by using a very low value for the dose parameter of the implantation step in the flow that, in reality, entirely lacks that step.

There are two justifications for this approach. Firstly, instabilities of the numerical solution of the overall problem are likely to blur the results due to different grids generated for the two different sequences[3]. Secondly, most task-level applications are aimed at adjusting parameter settings of a given process sequence to meet a given target rather than changing the sequence itself.

8.2 Task Encapsulation

When focusing on the system responses of process flows and devices, all simulation-related concerns should not be dealt with, but rather be handed down to some service that delivers the requested results. Furthermore, no distinction should be made with respect to the kind of procedure to invoke to get these results. More precisely, task-level applications should be liberated as far as possible from subtleties regarding the invocation of simulators, the precise format of input decks, the proper interpretation of generated files and return values, etc. To this end, the VISTA/SFC simulation environment provides a class of objects that encapsulate all evaluation tasks on the task level and establish a uniform interface between task-level modules such as optimizers, response surface models, database queries, etc.

8.3 EVE – Evaluable Entities

A class of *evaluable entity* (EVE) objects has been defined to provide uniform access to basic services like process flow simulation and RSM evaluation as well as to more complex, user-defined tasks. For example, the minimization of the bulk current for a given process could be encapsulated in an object that is evaluated for a set of initial value vectors for the optimizer generated by a DoE module or by a simple LISP loop. Figure 8.4 shows the basic idea of an

[3] These fluctuations, in general, are far below the tolerances and measurement limits of all factory settings and measurement procedures. Nevertheless, as simulators produce *exact* numbers, grid alterations or other numerical phenomena may introduce artificial effects that lead to erroneous results.

EVE object representing a process flow simulation task. In order to avoid any ambiguity, the left side of the EVE object in Figure 8.4 is called *client side*, the right one *server side*.

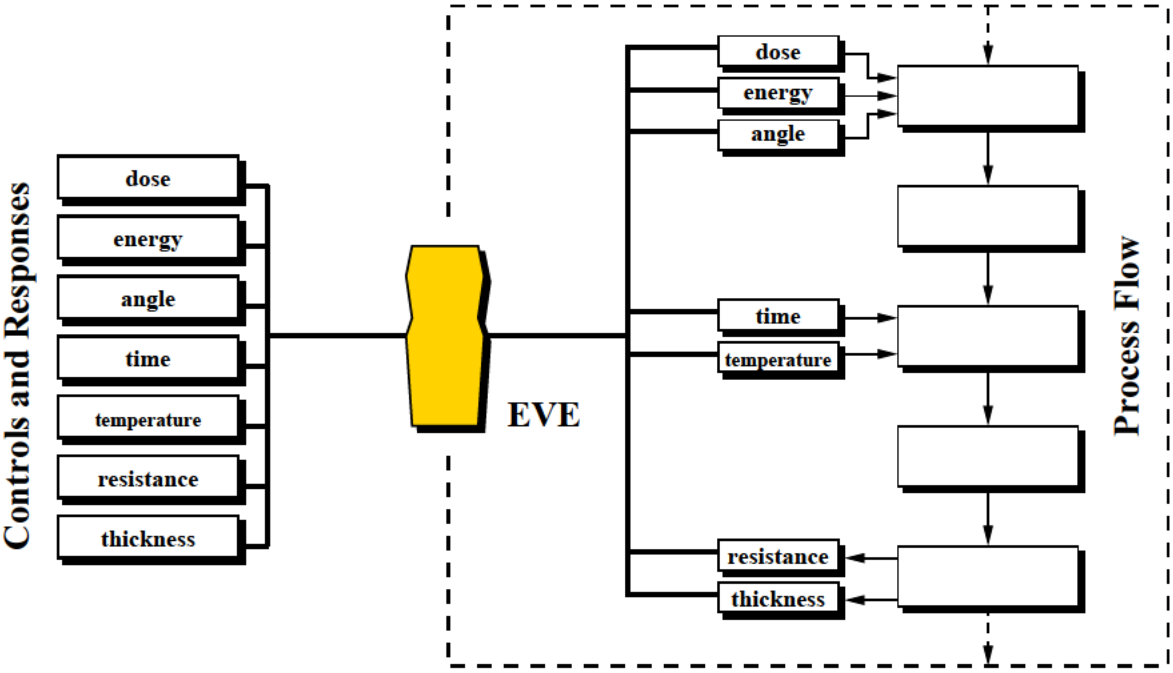


Figure 8.4: Process flow encapsulation with evaluable entity (EVE) objects.

After specifying those process parameters and measurements that are to be used as input variables (controls) and output variables (responses)[4], respectively, for subsequent analysis tasks, the EVE object hides all evaluation details of the underlying simulation. Different classes of EVE objects exist for different purposes; see Section 8.4 for a detailed description.

8.3.1 Basic Functionality

An EVE object offers a set of basic functions that can be invoked uniformly across all classes of EVE objects. Table 8.1 gives a summary of their names together with short explanations.

The `define-control` and `define-response` methods of an EVE object usually are invoked only once to establish a link with the server-side module. In general, a single control variable

[4] Experience shows that it is a good idea to refrain from using case-sensitive names at all, considering the difficulty in communicating verbally the cases of letters. As a matter of fact, the advantage of a greater number of available names due to case-sensitivity is more than compensated for by the increase in typing and reading efforts. Within the TCAD world, only chemical formulas seem to justify a case-sensitive interpretation of names. Wherever a significant distinction between the cases of letters is made, ambiguity and confusion arise. Furthermore, I would go as far as to claim that *significance* should only be attributed to the sequence of letters and digits in a name, disregarding their cases. In other words, two names should be considered equal if they agree in the sequence of their constituting letters and digits, skipping all other characters such as spaces, dashes, underscores, etc. The VLISP functions `name` and `name=` implement this philosophy.

Symbolic Name	**Description**
`define-control`	Defines a new control variable. All variables are identified by case-insensitive names that are unique in the context of an EVE object. Used to define the position of a process parameter in a process flow or to select one out of a number of available parameters.
`set-control`	Defines a default value and ranges for a control variable. Additionally, a control variable may be marked as `internal` if it is not to be accessed from the client side.
`set-control-expression`	Defines a LISP expression used to derive a control variable's value from other control variables (Figure 8.5).
`set-control-conversion`	Defines a LISP expression used to convert a control variable's value to a different representation before handing down to the server side.
`eval`	Request the evaluation of the underlying model for a given set of control values. If the set of control values is not complete, missing values are taken from the default values of the respective control variables. After termination of the evaluation, the responses are returned in a callback.

Table 8.1: Summary of basic EVE functionality. For the sake of conciseness, operations that are equivalent for both controls and responses are not listed separately.

```
(Eve-Define-Control "alpha")

(Eve-Define-Control "beta" :eval-expr '(if (plusp alpha) (sqrt alpha) 0.)
                           :internal T)

(Eve-Define-Control "gamma" :conversion '(string gamma))
```

Figure 8.5: Three examples of the command `Define-Control` for defining control variables. `alpha` is declared as a control variable. `beta`'s value is derived from a LISP expression using `alpha`. The `internal` attribute marks the `beta` to be inaccessible to the client side. `gamma`'s value is converted to a string on the server side, but not on the client side.

can have any number of connections to the server side. This mechanism is particularly useful when encapsulating process flows to map a single control variable to all appearances of a certain class of process or simulation parameter.

8.3.2 Nested Tasks

Given a set of EVE objects, they can be linked together to represent more complex applications. Figure 8.6 shows in principal how a process simulation module and a device simulation module are encapsulated to appear as a single task on the client side of the leftmost EVE.

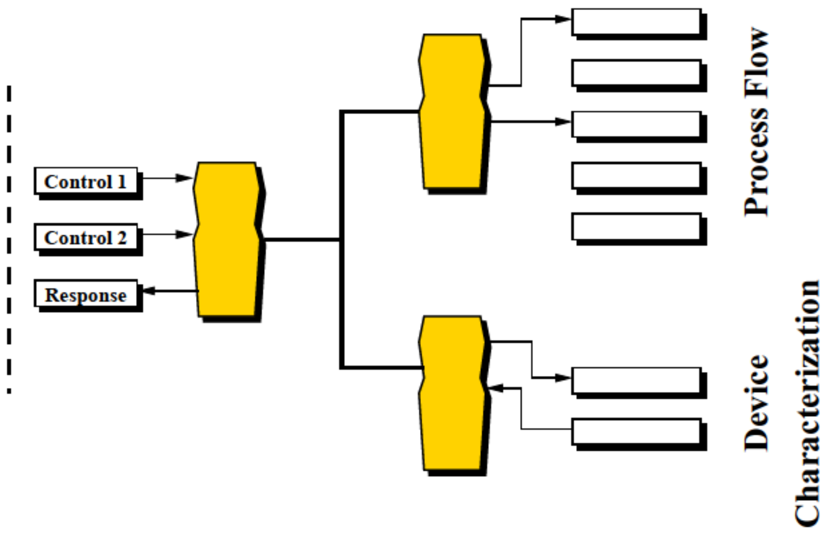


Figure 8.6: EVE objects are combined to build complex tasks from simpler ones.

8.3.3 Interface with Tool Agents

A variety of clients can request evaluations for certain input parameter sets via the uniform interface established by EVE objects (Figure 8.7) For instance, an optimizer sends requests to an EVE object without having to care about the class of the underlying module on the server side. External task-level clients are interfaced by tool agents; see Section 3.5.1 for details.

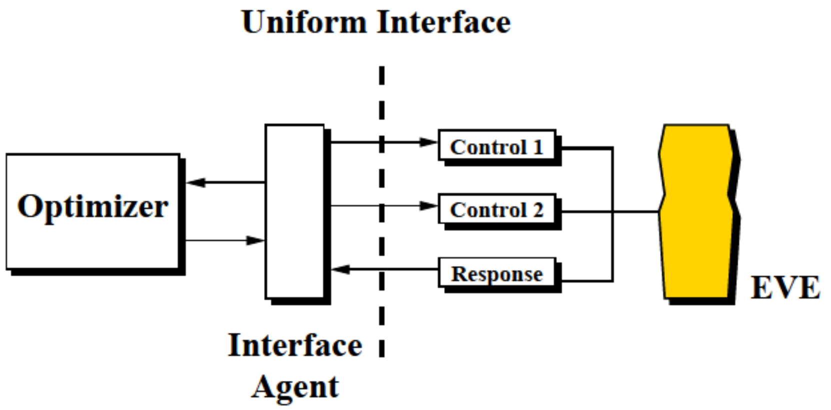


Figure 8.7: EVE objects establish a uniform interface for evaluation requests by internal and external clients and applications.

8.4 EVE Classes

For different applications, different classes of EVE objects exist that all share the basic functionality described in Section 8.3. The most important ones for many TCAD applications are the Flow EVE class and the RSM EVE class.

8.4.1 Flow-EVE

A complete process flow contains several thousands of potential design parameters. For practical purposes, a small number has to be selected for TCAD analysis. Defining the position of control variables in the process flow is accomplished by specifying a pattern for the labels of the affected steps and parameters. Built-in as well as user-defined parameters of steps and flow modules are accessible. When using the graphical flow editor (Section 5.10), a parameter selection mode allows direct access to all possible choices. In batch-mode or text-terminal operation, a short command establishes the link between a flow EVE variable and variable in the process flow. Figure 8.8 shows the definition of two control parameters for a flow EVE. The control `ntai-dose` is assigned to the process parameter `dose` in the step labeled `N TAI`. The control `grid-lines` is assigned to the parameter `nxm` in all steps of type `analytical-implant`. For implementing conditional execution of a process flow, the symbolic parameter `skip` can be used at any step. Assigning an expression, a simple mapping mechanism from numerical values to conditional branching can be realized.

```
(Eve-Define-Control "ntai-dose"     ;; symbolic name
                    "N TAI"         ;; step label
                    "dose"          ;; process parameter name
)
(Eve-Define-Control "grid-lines"    ;; symbolic name
                    '(:type "analytical-implant") ;; step type
                    "nxm"           ;; step parameter name
)
```

Figure 8.8: Text-mode commands for defining control variables in a flow EVE.

8.4.2 RSM-EVE

The class of RSM EVE objects encapsulates response surface models built from a set of experiments (see Section 8.7) and hides the RSM evaluation from the client side. In VISTA/SFC, RSM evaluation is performed by an external tool that communicates via a tool-agent with the simulation environment (see Section 3.5.1).

8.4.3 RSM-EVE Calibration

In order to calibrate an RSM with measured data from a fabrication unit, not only its control and response variables have to be accessible, but also internal parameters of the RSM need to be accessible. For this purpose, the RSM EVE class provides access to the coefficients of the RSM in terms of names that are constructed from the control variables they are attached to. For example, when modeling the threshold voltage (`vth`) as a function of the gate length (`lg`) and the threshold adjust implant dose (`dose`) with a polynomial model of second order, possible control names would be `c-lg-2`, `c-lg-dose`, `c-dose-2`, and `c-vth-0`. If a conflict arises with names already used as control or response variable names of the model, these standard assignments can be overwritten.

Fitting the coefficients of a RSM to experimental data is a powerful method for modeling the overall behavior of a fabrication process [Mas95].

8.4.4 Opt-EVE

A special class of EVE objects has been defined that encapsulates an optimization task, presenting the initial values for the optimizer run as control variables to the outside world. In a fashion similar to the previous section, initial values are accessed by names constructed from the names of the variables they correspond to. For instance, if the system to be optimized has a parameter **`alpha`**, the corresponding initial value is accessible as **`alpha-init`**. As a response, the optimum found by the optimizer is returned. Communication with the external optimizer executable is taken care of by tool agent as described in Section 3.5.1. Figure 8.9 shows an optimizer working on an RSM model. The initial values of the optimization loop are available as control variables in the OPT EVE.

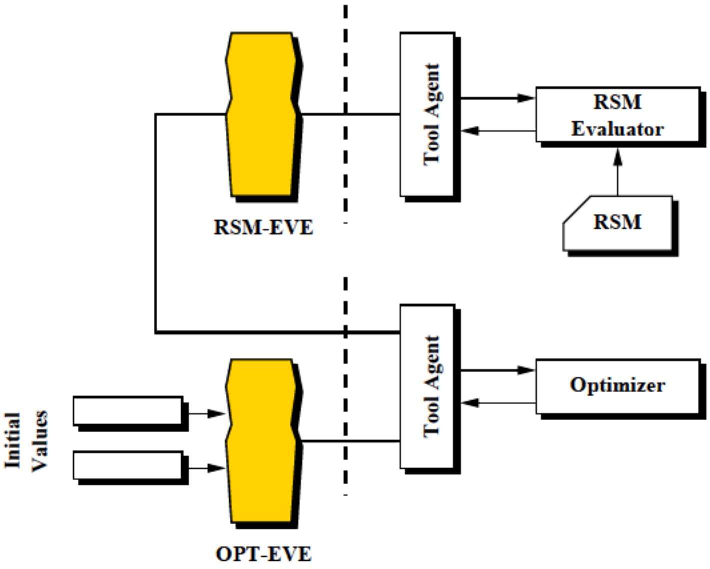


Figure 8.9: Optimizer and RSM connected with EVE objects.

8.4.5 LISP-EVE

The most generic class of EVE objects allows to specify arbitrary LISP expressions to be evaluated upon an evaluation request. It serves as a simple interface for all kinds of applications without a dedicated EVE class. The names of the control variables are automatically derived from the names of the arguments of the LISP functions if a function is passed; otherwise, they have to be specified by the user.

8.5 EVI- Evaluation Instances

During the process of trying different design paths, a lot of intermediate data is generated, that is too volatile to be stored in a full-fledged data base, but nevertheless has to be maintained for a certain period of time during the exploration of the design space. These temporary – with respect to the time frame of the whole project – data are stored as evaluation instances, belonging to a given EVE. In addition to defining a standardized interface, EVE objects take care of some simple data management services for all *evaluation instances* (EVI) that have been created (Figure 8.10). Each EVI represents a point in the design space. Individual EVIs can be assigned to groups for convenient access for further processing. Groups are maintained by the EVE object and can be overlapping. Both groups and individual instances offer slots for recording the history of creation of the data point; slots are set automatically by design generators, or manually by the user. Together with its evaluation instances, an EVE offers rudimentary spread-sheet capabilities. In Section 8.10, a graphical user interface for accessing and managing EVIs is shown.

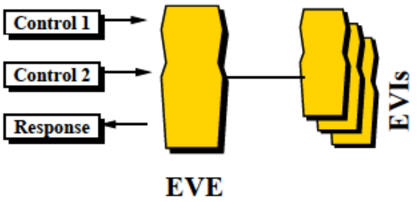


Figure 8.10: Evaluation instance (EVI) objects are managed by EVE objects.

8.6 Design of Experiments

The automatic generation of experiments is taken care of by a DoE module that runs as an external executable and communicates with the environment using a tool agent. A number of design types is available (Table 8.2)[KPR$^+$96]. The screening analysis design SA provides a simple means to generate a number of experiments along each axis of the design space with all other coordinates remaining at their respective default values. The supplementary design SUP is an implementation of [STA$^+$95], aiming at minimizing the number of additional experiments required when adding a new control variable to an existing set of experiments at the cost of a slightly sub-optimum design with respect to the standard CCC design. See [LA91] for an extensive treatment of the design of experiments. For each experiment generated by the DoE module, an EVI is added to the active EVE and submitted for evaluation. On return, the computed results are read and written to the EVI. Optionally, control values generated automatically may be modified interactively before starting an evaluation.

NOM	Nominal Design
SA	Screening Analysis
FUL	Full Factorial Design
CCF	Central Composite Face-Centered Design
CCC	Central Composite Circumscribed Design
CCI	Central Composite Inscribed Design
RAN	Random Design
DIA	Diagonal Design
GRI	2D - Grid Design
LAT	Latin Hypercube Design
FRA	Fractional Factorial Design
PLA	Plackett-Burman Design
OME	Orthogonal Main Effect Design
SUP	Supplementary Design

Table 8.2: Summary of design types provided by the DoE module.

8.7 Response Surface Modeling

Given a number of completed experiments in an EVE, a response surface model can be constructed to represent the system's behavior analytically with – mostly polynomial – functions by fitting the polynomial coefficients to the data points [BD87]. The resulting coefficient values, together with information on the valid ranges for all controls and on the quality of the fit achieved, are stored as a response surface model for evaluation by the RSM evaluator (cf. Figure 8.9). In Section 8.10, glimpses of the GUI module for interactive exploration response surface models are shown.

8.7.1 System Linearization

Both DoE and RSM modules work best if all system responses behave more or less linearly with respect to the control variables. Unfortunately, this behavior is very rarely encountered in real applications, To accurately model the system behavior, transformations of the parameter space are used to to linearize the dependence of the output variables on the transformed input parameters. Subdivision of the parameter space as well as fitting of the response surfaces takes place in transformed space.

For each input parameter, a transformation function can be selected from a set of predefined transformations. If the transformation function needs parameters (*transformation parameters*), these parameters may either be specified explicitly, e.g., in the case when a physical formula has been established, or they may be determined automatically from a set of sample points. Additionally, it is also possible to select the best one of a given set of transformation functions for a given set of sample points. Thus, the user does not need to specify the transformation to use.

It is important to note that all transformation functions have to be defined by specifying code for both the forward and reverse directions and assigning a reference name to the transformation before they can be used. All information on transformations is stored centrally and accessed exclusively by the reference name. E.g., for a given technology, a transformation called `vth-lg` can be defined, which analytically reflects the short-channel effect and is used to linearize the dependence of the threshold voltage on the gate length for DoE and RSM.

8.8 Optimization and Calibration

Being the most ubiquitous, and at the same time the most mystic, terms in TCAD applications, they refer essentially to the same thing: the minimization of the value of a target function that is constructed from the inputs and outputs of an arbitrary system[5]. (Figure 8.11). The minimization starts at a given set of initial values for the system inputs. The result of the minimization task are the values of the inputs and the outputs at the minimum value of the target function.

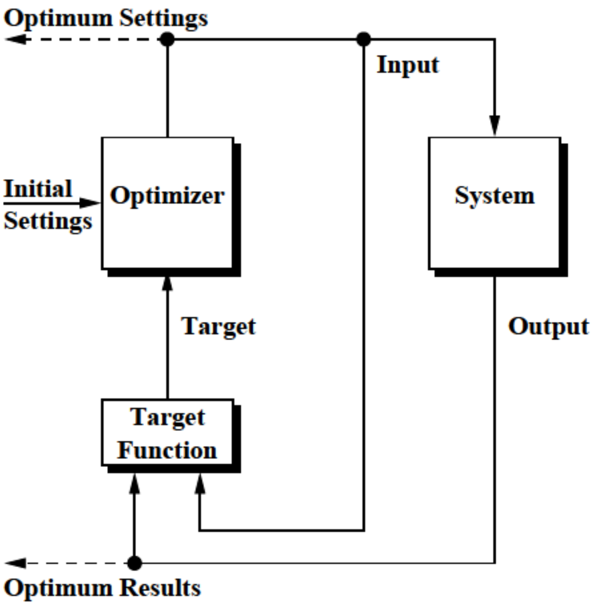


Figure 8.11: Optimization as minimization of target function.

For calibration, the output values for the minimization tasks are computed as the fit error that describes the quality of the correspondence of the system with a set of fit data. The fit

[5] In principal, the inputs need not be explicitly dealt with but could be modeled as additional system outputs. They appear here for the sake of clarity only.

error is usually defined as the sum over all fit data points of the squares of the differences of fit data and system output data. The input values for the calibration task are the fit parameters of the system to calibrate (Figure 8.12).

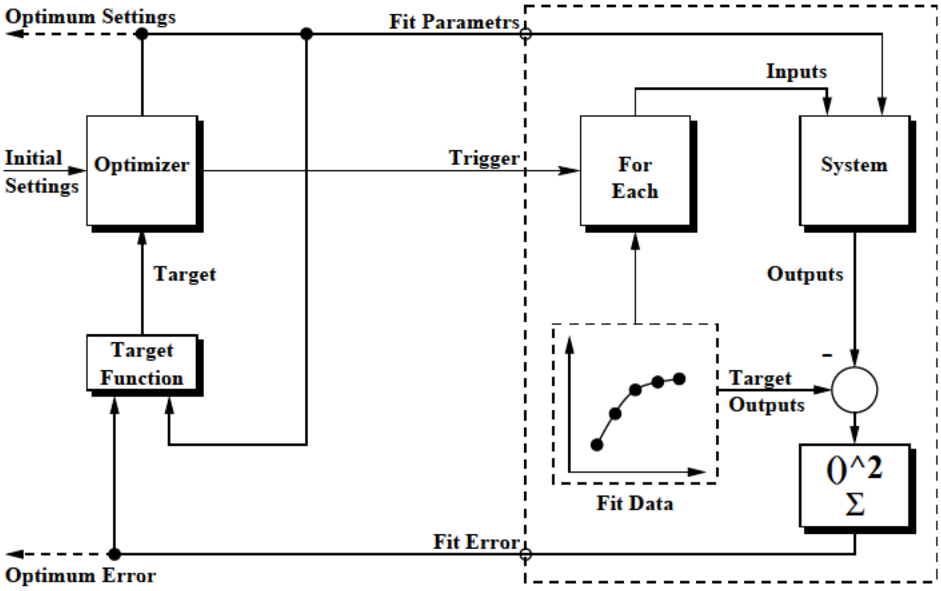


Figure 8.12: Calibration as minimization of the fit error.

If a system does not have any parameters to use as fit parameters, it is possible to apply a parameterized transformation of the input variables to calibrate the new system consisting of the original system and transformation (Figure 8.13). The result of the calibration, or the *calibration* for short, is the set of optimum fit parameters obtained. In the second case, also the optimum transformation pertains to the calibration.

In TCAD, the term *optimization* refers to the search of a set of process parameters that lead to a desired device behavior. In most cases, the system used for optimization is represented as a response surface model generated from a standard design of experiments, as direct evaluation of a process and device simulation sequence for each step of the optimizer is not feasible.

On the other hand, the term *calibration* is used to indicate that a set of parameters has to be computed that optimally fits a system to a set of measured data. For tool calibration, the system is represented by a single simulator run including pre-processing and post-processing; for global process calibration, the system is a response surface model the coefficients of which are used as parameters (cf. Section 8.4.3).

8.8.1 Combining Tool and Calibrations

After successful calibration, the parameter values obtained have to stay with the calibrated tool. Several options exist:

1. Generating a calibrated executable by compiling the parameter values into the tool.

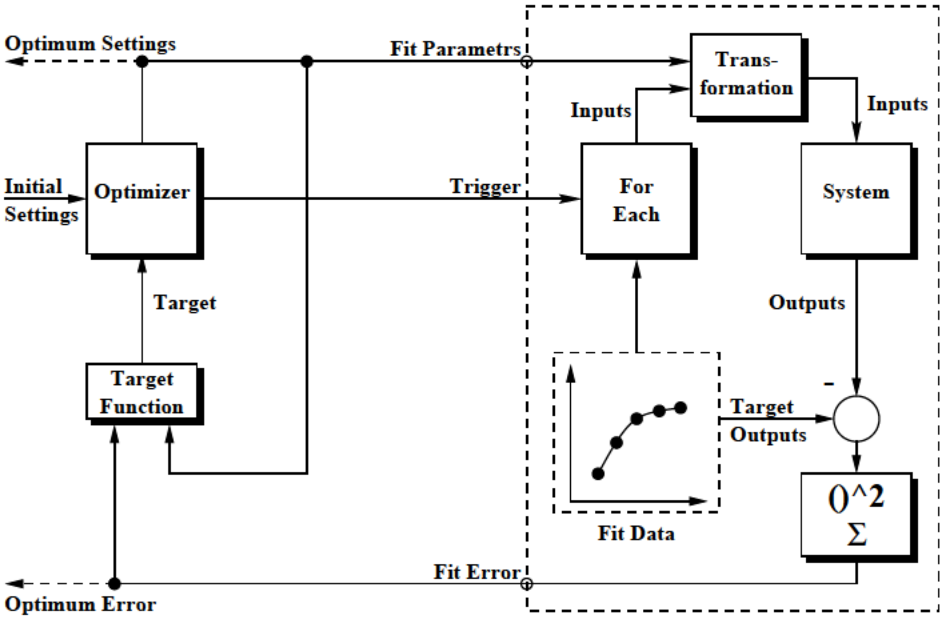


Figure 8.13: Calibration by transformation of input variables.

2. Storing the parameter values in an input deck include file to be loaded by the tool each time the calibration is required.

3. Writing a shell script that sets the parameter values on the tool's command line on invocation.

4. Defining a new application in VISTA/SFC as a combination of the tool and the calibrated parameter values.

8.8.2 Available Optimizers

In VISTA/SFC, different external optimizers can be integrated using tool agents as described in Section 3.5.1. For optimizing device performance parameters over a given input variable space, a constrained optimizer with sequential quadratic approximations has been integrated. It minimizes the target function, which can be assembled out of the input and output values. The gradient is calculated by evaluating finite differences, and the Hessian matrix is built by an BFGS update.[6] For the calibration and fitting applications, an optimizer based on the Levenberg-Marquardt algorithm[7] [Kha95][PKSS95] is available.

[6] See [GMW95] for optimization techniques and algorithms.

[7] The Levenberg-Marquardt algorithm includes the minimization task as well as the fit error calculation in Figure 8.12.

8.9 Programming Interface

The programming interface is directly based on the VLISP interpreter, providing access to a variety of framework services on different levels of abstraction. Besides using all services directly as function calls, it is possible to define new operations or customize existing ones, store them in a file, and retrieve them at a later time, or add them to a list of startup files to permanently enhance the environment's functionality. In Section 9.2, a detailed example of the capabilities of the programming interface is given. Table 8.3 gives a summary of commands available for defining task-level applications.

Symbolic Name	**Description**
`Eve-Eval`	Generates an experiment for the values given as arguments.
`Eve-Doe`	Generates experiments with the DoE module.
`Eve-Plot`	Displays a 1-D plot of selected data entries.
`Eve-Rsm`	Generates a response surface model from the values present in the EVE object.
`Eve-Stop`	Prevents the evaluation of new EVIs.
`Eve-Start`	Enables the evaluation of new EVIs and starts evaluation of all waiting ones.
`Flow-Eve`	Creates a flow EVE in the current project.

Table 8.3: Summary of EVE commands for task-level applications.

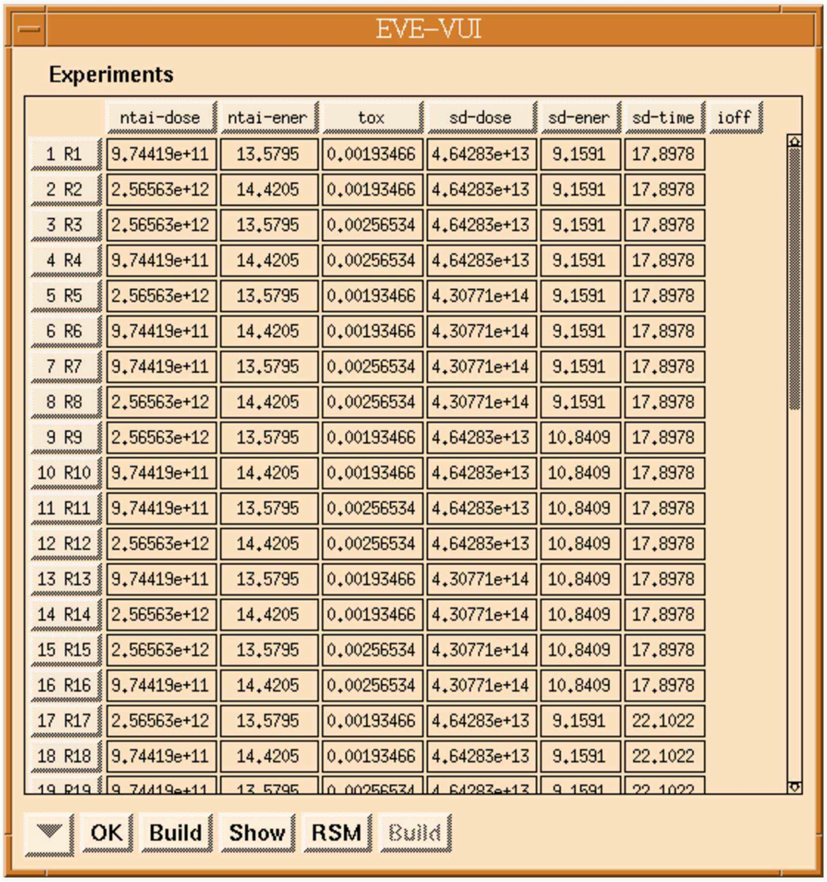
8.10 Graphical User Interface

On the GUI level an EVE and its pertaining evaluation instances are represented as a spreadsheet with direct access to plot tools, DoE generation, and RSM generation (Figure 8.14). Furthermore, new evaluations can be requested or removed from the list of existing ones, and all kinds of editing tasks with respect to the definition of control and response variables can be performed.

Figure 8.15 shows the GUI for exploring an RSM built from data points in an EVE object.

One-dimensional and two-dimensional cuts of the N-dimensional parameter space can be selected and displayed using the `gnuplot` tool (Figure 8.16) or any other plot program[8].

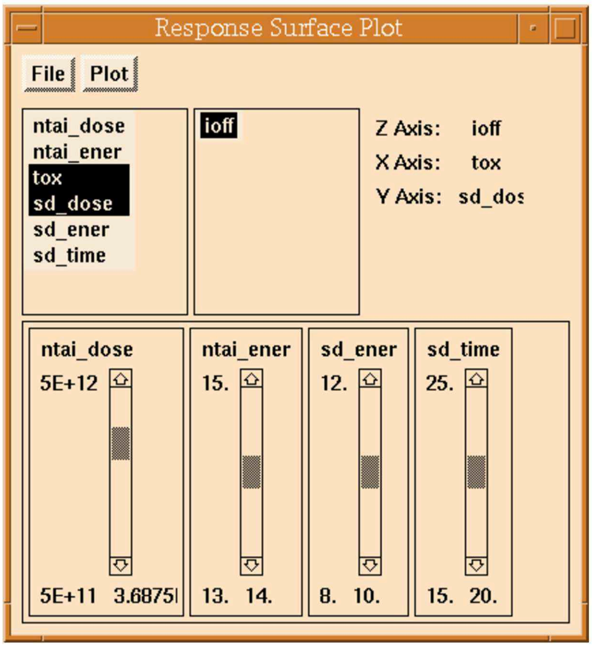
[8] Access to an external plot tool is again encapsulated by an agent object; therefore, any number of different programs can be easily integrated using the standardized agent interface.



Experiments

	ntai-dose	ntai-ener	tox	sd-dose	sd-ener	sd-time	ioff
1 R1	9.74419e+11	13.5795	0.00193466	4.64283e+13	9.1591	17.8978	
2 R2	2.56563e+12	14.4205	0.00193466	4.64283e+13	9.1591	17.8978	
3 R3	2.56563e+12	13.5795	0.00256534	4.64283e+13	9.1591	17.8978	
4 R4	9.74419e+11	14.4205	0.00256534	4.64283e+13	9.1591	17.8978	
5 R5	2.56563e+12	13.5795	0.00193466	4.30771e+14	9.1591	17.8978	
6 R6	9.74419e+11	14.4205	0.00193466	4.30771e+14	9.1591	17.8978	
7 R7	9.74419e+11	13.5795	0.00256534	4.30771e+14	9.1591	17.8978	
8 R8	2.56563e+12	14.4205	0.00256534	4.30771e+14	9.1591	17.8978	
9 R9	2.56563e+12	13.5795	0.00193466	4.64283e+13	10.8409	17.8978	
10 R10	9.74419e+11	14.4205	0.00193466	4.64283e+13	10.8409	17.8978	
11 R11	9.74419e+11	13.5795	0.00256534	4.64283e+13	10.8409	17.8978	
12 R12	2.56563e+12	14.4205	0.00256534	4.64283e+13	10.8409	17.8978	
13 R13	9.74419e+11	13.5795	0.00193466	4.30771e+14	10.8409	17.8978	
14 R14	2.56563e+12	14.4205	0.00193466	4.30771e+14	10.8409	17.8978	
15 R15	2.56563e+12	13.5795	0.00256534	4.30771e+14	10.8409	17.8978	
16 R16	9.74419e+11	14.4205	0.00256534	4.30771e+14	10.8409	17.8978	
17 R17	2.56563e+12	13.5795	0.00193466	4.64283e+13	9.1591	22.1022	
18 R18	9.74419e+11	14.4205	0.00193466	4.64283e+13	9.1591	22.1022	
19 R19	9.74419e+11	13.5795	0.00256534	4.64283e+13	9.1591	22.1022	

Figure 8.14: EVE spreadsheet–GUI provides a convenient interface for initiating new evaluations and experiments, generate one-dimensional plots, and build and view response surface models.



Z Axis: ioff
X Axis: tox
Y Axis: sd_dos

Figure 8.15: Graphical user interface for viewing response surface models. One or two parameters can be selected as plot axes, the remaining ones are parameters of the plot, used to interactively explore the model.

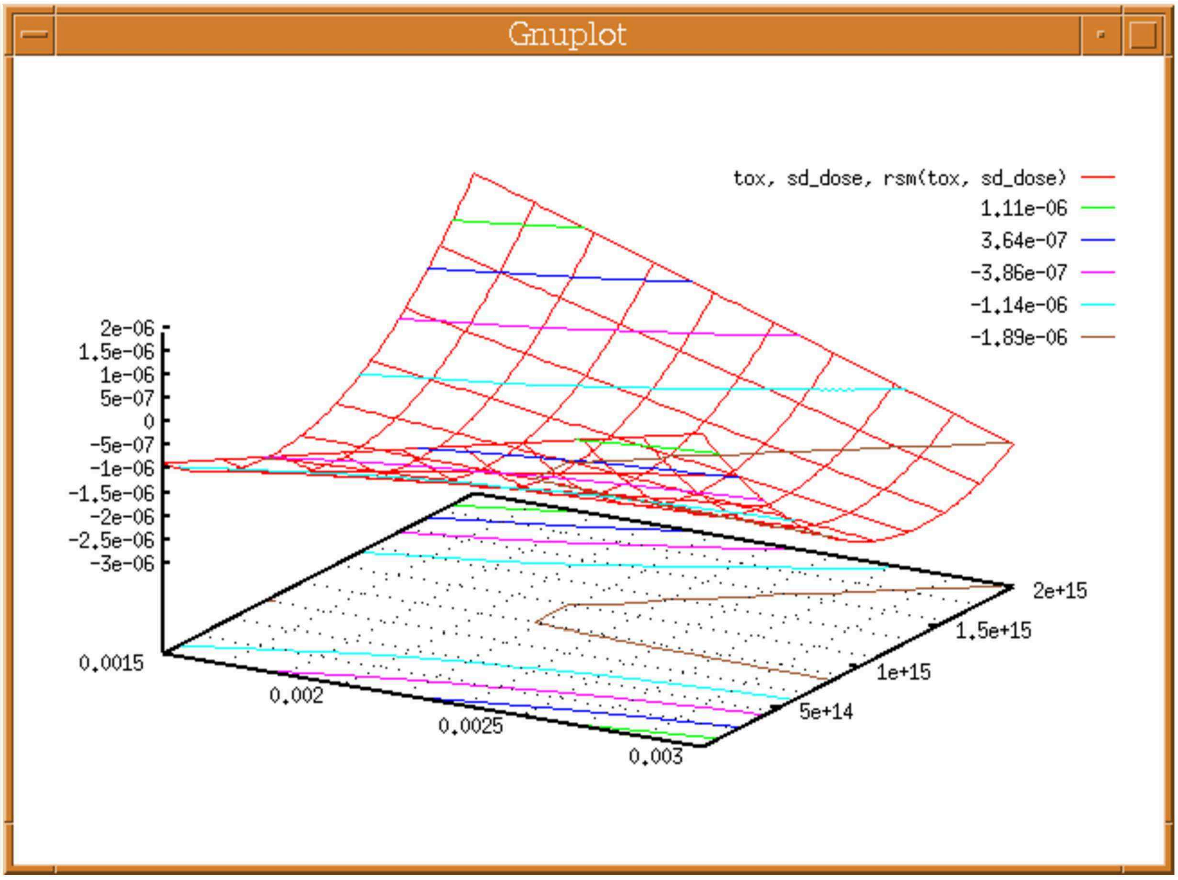


Figure 8.16: Two-dimensional plot of response surface model.

Chapter 9

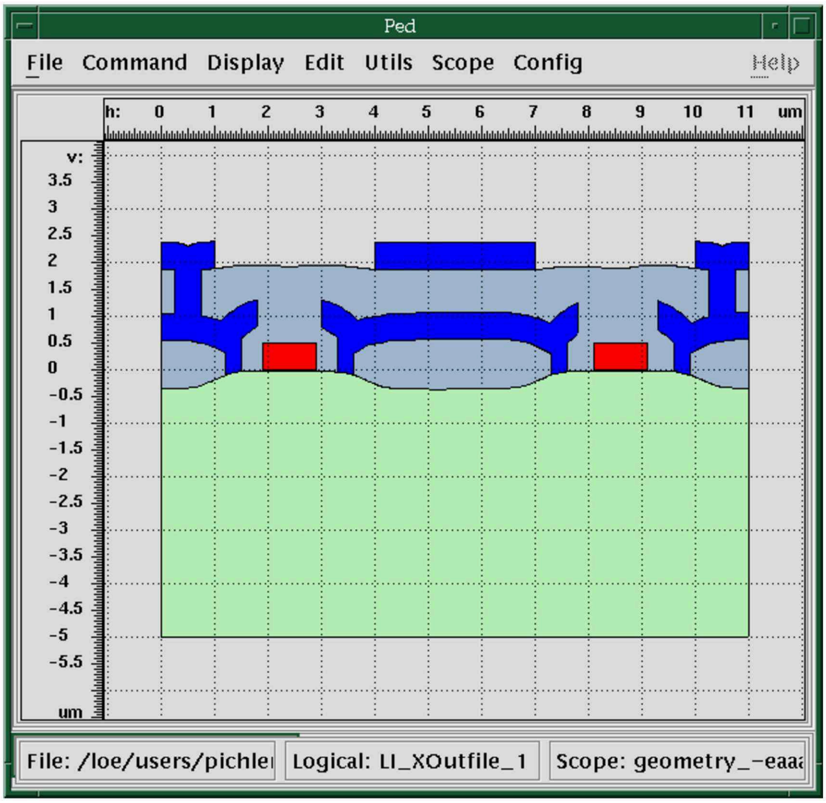
Application Examples

9.1 Standard CMOS Process

To demonstrate the application of the VISTA/SFC environment to the modeling and analysis of VLSI fabrication processes, a complete standard silicon-gate LOCOS CMOS process [SS91] was simulated using a set of heterogeneous simulation tools. The process flow contains all types of fabrication process steps occurring in modern VLSI technology and uses eleven lithography masks. Simulation was carried out up to the second metalization layer, which forms the gate lines, using 55 tool steps. The simulated structure is a CMOS inverter, containing most of the relevant intrinsic and parasitic devices. The final topography is shown in Figure 9.1. An overview of the process is given in Table 9.1, a detailed description can be found in [SS91], p. 370.

To achieve accurate and realistic modeling of etching and deposition processes, the PROMIS etch module [SS93] [SS95] was used. It is based on extremely stable cell-based algorithms and contains a number of physically sound models for a variety of topography processes. Lithography steps are performed by the SKETCH tool which operates as a simple geometry manipulator and provides mask pattern transfer and material strip capabilities where no accurate simulation of optical phenomena is needed. All implantation steps have been carried out with the PROMIS implantation module [Sti93] [Boh96]. Diffusion has been realized with the PROMIS-NT diffusion module [PS95] [Puc96], which offers the possibility of solving different models on each material segment and therefore greatly facilitates the investigation of advanced effects in state-of-the-art devices. Oxidation and diffusion in reactive environment have been carried out with TSUPREM4 [Tec95].

Figure 9.2 shows the resulting net doping concentration in both the N and P devices. Despite the interaction of fundamentally different simulation tools and the large number of simulation steps, fully automatic simulation has been performed without the necessity for user interaction at any point of the simulation. Figure 9.3 shows a detail of the resulting triangular grid after the last process step. 13000 grid nodes are used to accurately resolve arsenic, boron, and phosphorus distributions generated by PROMIS, PROMIS-NT, and TSUPREM4. All grid operations, from the generation of the initial grid to merging and updating grid data at various points in the process flow, have been carried out by the TRIANGLE [She96] gridding


Ped
File
Command
Display
Edit
Utils
Scope
Config
Help
h: 0 1 2 3 4 5 6 7 8 9 10 11 um
v: 3.5 3 2.5 2 1.5 1 0.5 0 -0.5 -1 -1.5 -2 -2.5 -3 -3.5 -4 -4.5 -5 -5.5 um

File: /loe/users/pichlei Logical: LI_XOutfile_1 Scope: geometry_-eaa

Figure 9.1: The final CMOS structure including two metal layers.

tool. Using the framework's built-in dynamic load balancing mechanism on a DEC 3000-600 workstation cluster, the total computation time for all process simulation and regrid operations amounts to less than 17 hours of real time.

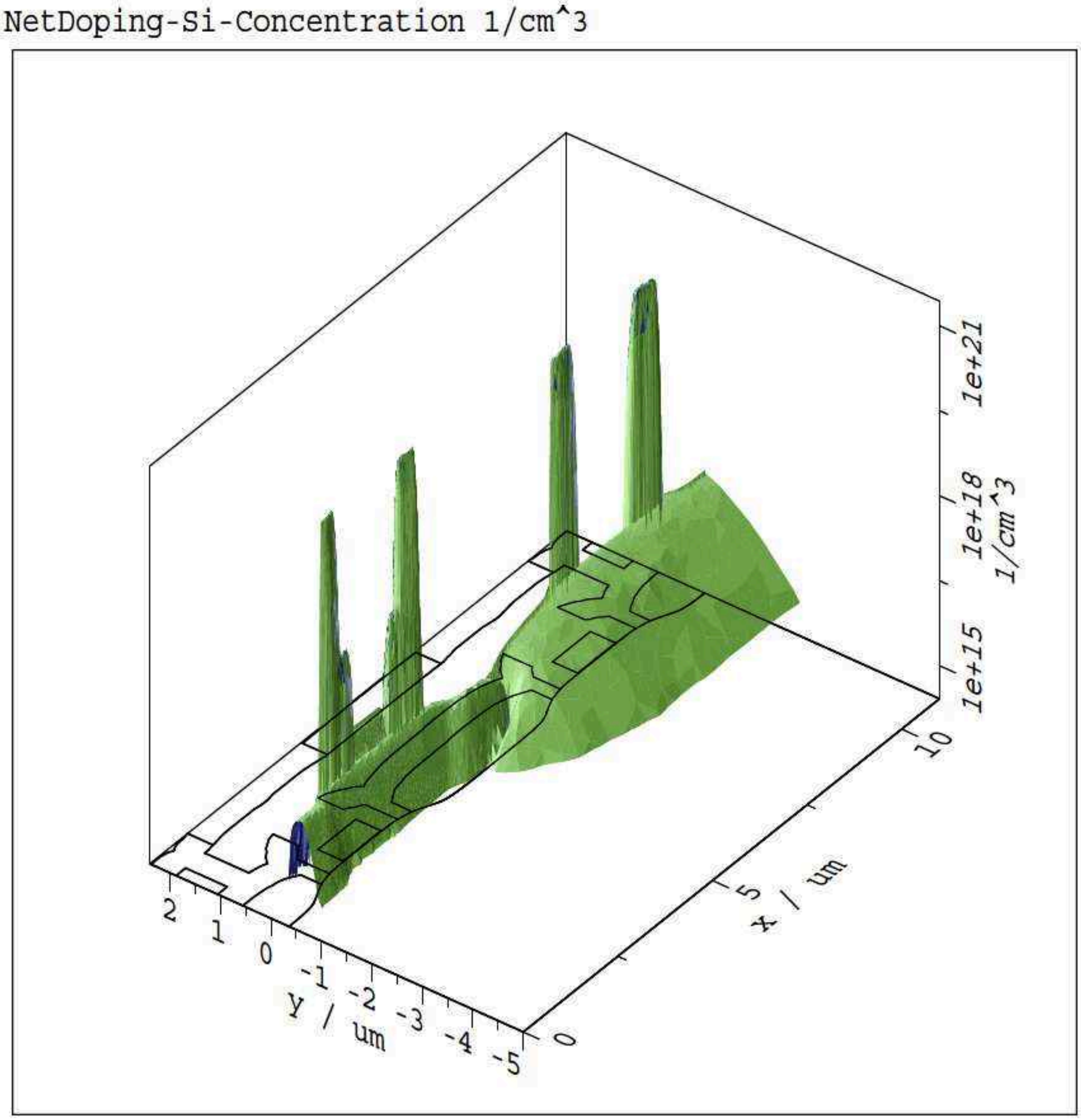


Figure 9.2: Net doping distribution in CMOS inverter after simulation of second metal layer. The N-device on the left side uses an LDD structure and an anti-punch-through implant.

Module	Mask	Process Step		Tool
Substrate			(100)-oriented, p-doped silicon (ρ = 10mΩcm) with a 12μm epi layer (ρ = 10Ωcm)	SKETCH
N-Tub	NW	Lithography		SKETCH
		N-Tub Etch		PROMIS
		Strip Resist		SKETCH
		Screening Oxide	$t_{ox} = 50nm$	PROMIS
		N-Tub Implant	Phosphorus (150keV,$5 \cdot 10^{12}$cm^{-2})	PROMIS
		N-Tub Diffusion	1150°, 150min, N$_2$	PROMIS
		Strip Oxide		SKETCH
Nitride		Oxide	O_2, $t_{ox} = 25nm$	TSUPREM4
		Nitride deposition	$t_{nitride} = 150nm$	PROMIS
	OD	Lithography		SKETCH
		Nitride Etch		PROMIS
		Strip Resist		SKETCH
Channel Stop	NWI	Lithography		SKETCH
		Oxide Etch		PROMIS
		Channel Stop Implant	Boron (16keV, $5 \cdot 10^{13}$cm^{-2})	PROMIS
Anti Punch-Through		Anti-Punch-Through Implant	Boron (180keV, $8 \cdot 10^{11}$cm^{-2})	PROMIS
		Strip Resist		SKETCH
Field Oxide		Field Oxidation	960°, H_2, O_2, $t_{ox} = 850nm$	TSUPREM4
		Etchback	t_{ox} = 500nm	PROMIS
Gate		Gate Oxidation	t_{ox} = 25nm	TSUPREM4
		Screening Poly	$t_{poly} = 100nm$	PROMIS
		Threshold Adjust Implant	Boron (45keV, $9 \cdot 10^{11}$cm^{-2})	PROMIS
		Gate Poly	t_{poly} = 400nm	PROMIS
	PS	Lithography		SKETCH
		Poly Etch		PROMIS
		Strip Resist		SKETCH
N-S/D	SN	Lithography		SKETCH
		N-LDD Implant	Phosphorus (100keV, $2 \cdot 10^{13}$cm^{-2})	promis
Spacer		TEOS Oxide	t_{ox} = 300nm TEOS-oxide	PROMIS
		Anisotropic Etch		PROMIS
		Screening Oxide	$t_{ox} = 30nm$	PROMIS
	SN	Lithography		SKETCH
		N Source/Drain-Implant	Arsenic (100keV, $5 \cdot 10^{15}$cm^{-2})	PROMIS
		Strip Resist		SKETCH
P-S/D	SP	Lithography		SKETCH
		P Source/Drain-Implant	Boron (30keV, $3 \cdot 10^{15}$cm^{-2})	PROMIS
Contacts		TEOS oxide	(t_{ox} = 800nm)	PROMIS
		Reflow	900°, H_2, O_2	TSUPREM4
	CO	Lithography		SKETCH
		Oxide Etch		PROMIS
		Strip Resist		SKETCH
Metal 1		Al Deposition		PROMIS
	IN	Lithography		SKETCH
		Al Etch		SKETCH
Planarize		Plasma Oxide	$t_{ox} = 1.2\mu m$	PROMIS
		Resist Spin-On		SKETCH
		Etchback		PROMIS
Vias	COS	Lithography		SKETCH
		Oxide Etch		PROMIS
		Strip Resist		SKETCH
Metal 2		Al Deposition		PROMIS
	INS	Lithography		SKETCH
		Al Etch		PROMIS
		Strip Resist		SKETCH

Table 9.1: CMOS process sequence. 55 simulation tool steps are used to simulate the entire fabrication process.

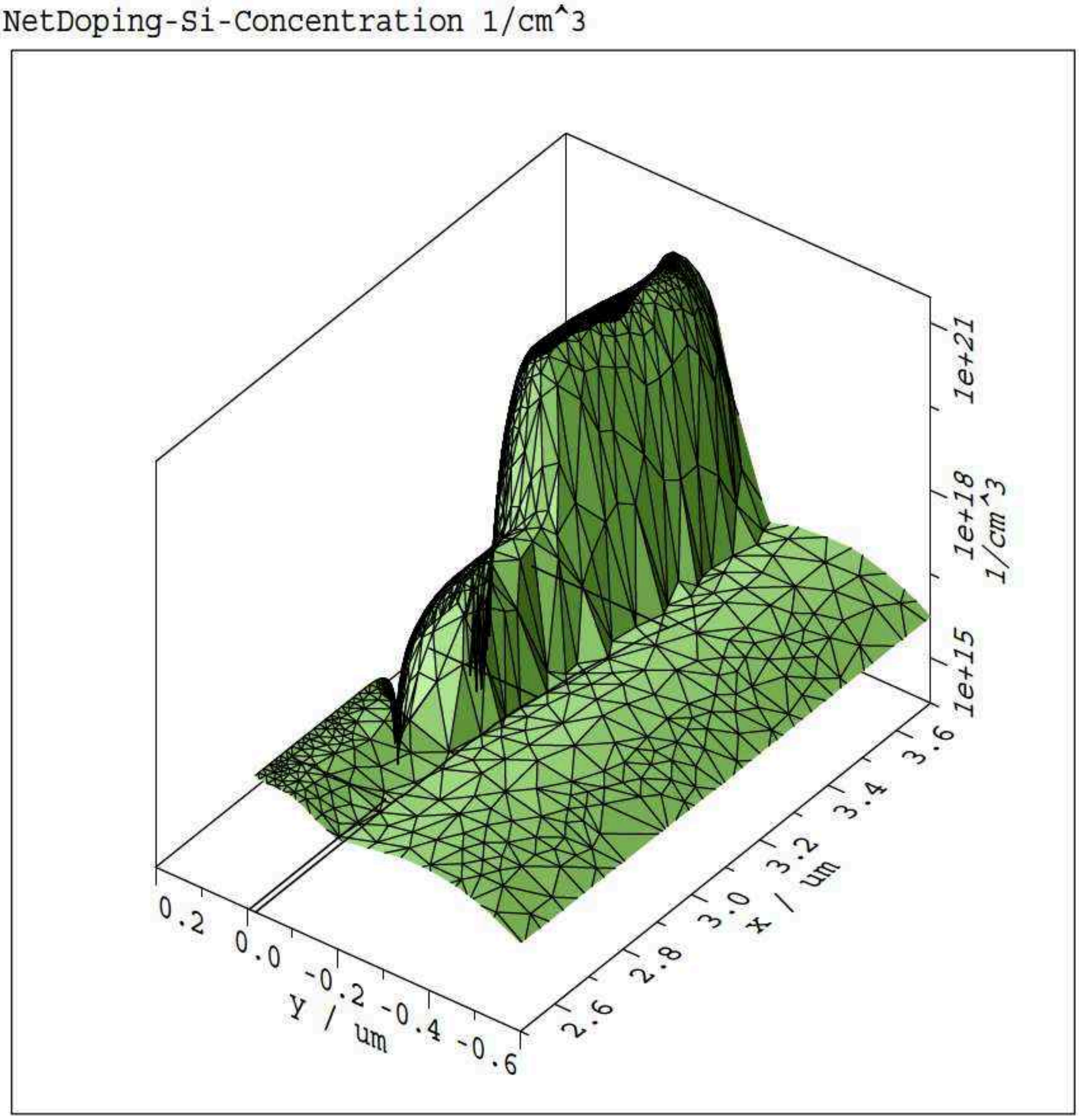


Figure 9.3: Grid detail of N-device spacer region with LDD implant, source/drain implant, channel implant, and anti-punch-trough implant after simulation of metal 2.

9.2 Short-Channel-Effect Reduction with Pocket Implants

Reducing the gate-length of a MOS transistor to dimensions smaller than about 2 μm alters the electrical behavior of the device in a characteristic way. The influence of the source and drain regions on the channel grow, i.e., the fraction of the channel that is controlled by the gate decreases with respect to the long channel device[1], where the transition region from the channel to the source and drain regions is small in comparison to the gate length. These *short channel effects*[2] cause considerable distress for the device designers, because one-dimensional device equations – widely used in circuit-design applications – do not any more give accurate results. In consequence, it is a declared goal of VLSI process design to sustain long-channel behavior even for short-channel devices.

9.2.1 Threshold Voltage Reduction

One well understood effect found at decreasing channel lengths is a reduction of the threshold voltage[3].

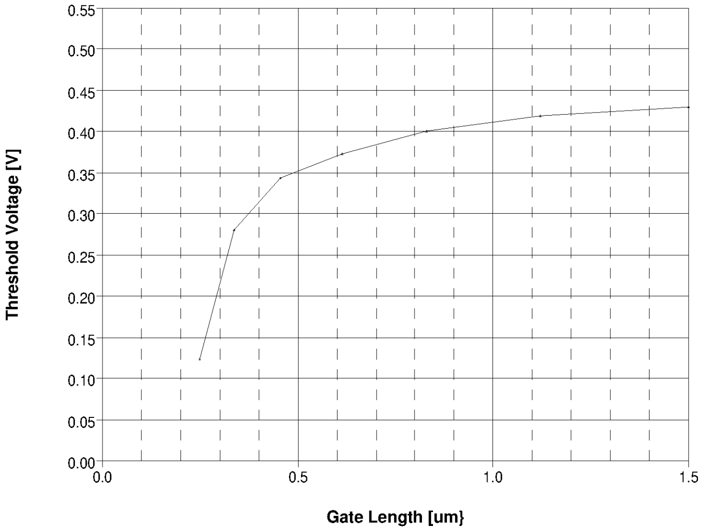


Figure 9.4: Short channel effect in a simulated NMOS transistor.

Figure 9.4 shows the threshold voltage of an NMOS transistor versus the gate length, with the gate length being varied from 0.25μm to 1.5μm. For lengths smaller than a quarter of

[1] A transistor which can be sufficiently described by one-dimensional equations.

[2] A concise treatment of short channel effects can be found in [Wol95b].

[3] The threshold voltage itself represents a rather elusive term, as no universal agreement exists as to a single – physically motivated – definition of such voltage. In this example, it is pragmatically defined as the value of the gate voltage at a drain current of 1nA.

a micron, the device shown does not turn off any more and no threshold voltage can be established. The decrease of the threshold voltage is caused by the increased fraction of the channel-depletion region that is occupied by the space charge in the junction depletion regions between source and channel, and channel and drain, respectively, and the resulting decrease in gate charge needed to cause inversion in the channel.

9.2.2 Large-Angle-Tilt Implant Optimization

To compensate the influence of the source/drain regions on the channel below the gate edges, it is possible to artificially raise the p doping level below the gate edges by local boron implants into the transition region between the source/drain regions and the channel. One way of realization is to use a large-angle tilt (LAT) implant of boron ions into the NMOS device after the formation of the LDD structure [HKS+91].

A frequently encountered process engineering task deals with determining the optimum parameter settings for the LAT implant. The procedure for establishing these values consists of several tasks:

1. For the nominal device, screen the influence of the three LAT parameters to determine ranges for the optimization.

2. Select nominal parameter settings from the results of the screening analysis.

3. Create a response surface model for the dependence of the device behavior on the LAT parameters in the vicinity of the nominal point.

4. Use the RSM model to optimize the LAT parameter values.

The creation of the response surface model – or any other analytical model the can be evaluated sufficiently fast – is not a necessity, as the optimization process could also be carried out on the process and device simulations directly[4].

9.2.3 Process Simulation

To demonstrate the applicability of the VISTA/SFC simulation environment to the optimization of the LAT implant parameters, an NMOS device was designed for a nominal gate length of $0.5\mu m$ and simulated using two-dimensional process and device simulation. Table 9.2 gives the main process parameters for the nominal device, Figure 9.5 shows the doping profiles in the nominal device for the three dopant species boron, arsenic, and phosphorus.

[4] Given the high degree of autonomy, robust operation, efficient disk space usage, and automatic load balancing as provided by VISTA/SFC, running an optimizer directly on the simulators is not an infeasible task and avoids many drawbacks associated with derived models.

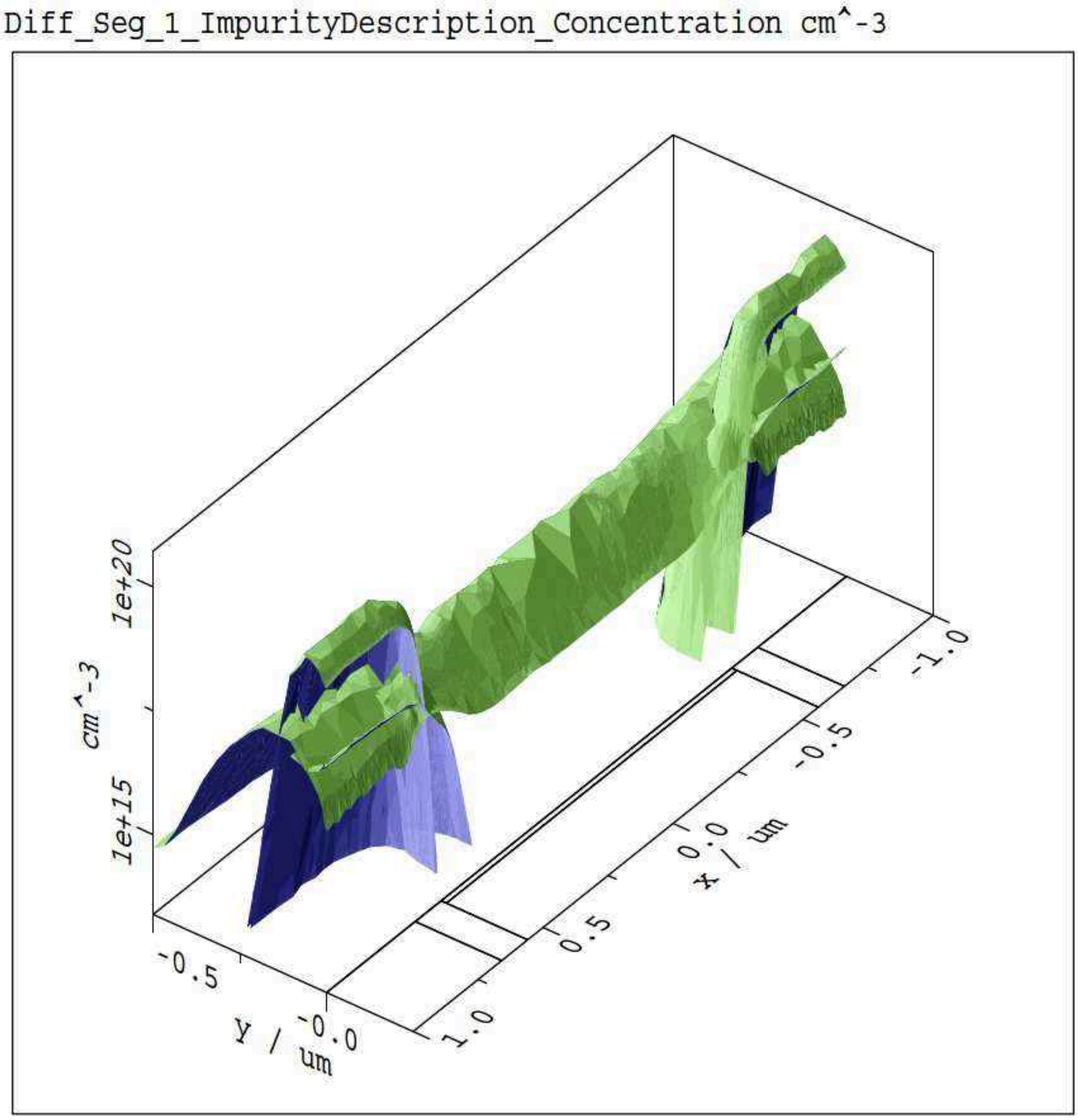


Figure 9.5: Nominal NMOS device doping profile without pocket implants for a gate length of 1.12μm, including threshold-adjust implant, LDD implant, and source/drain implant.

Substrate	p, $5e^{14}cm^{-3}$
Threshold adjust implant	Boron, $1e^{13}cm^{-2}$, $40keV$
Gate oxide	$10nm$
LDD	Phosphorus, $1e^{15}cm^{-2}$, $45keV$
Spacer	$100nm$
Source/Drain	Arsenic, $1e^{15}cm^{-2}$, $80keV$
Anneal	$30min$, 900^oC
U_{SD}	2 V

Table 9.2: Process parameters for NMOS device.

9.2.4 Task Definition

Using the task-level encapsulation mechanisms presented in Chapter 8, the complete analysis and optimization task outline above can be defined in VISTA/SFC by a list of statements. Figures 9.6 through 9.10 show the code for batch-mode operation[5].

In Figure 9.6, the first statement defines a new project that uses a process flow description file. (The LAT implant is realized by using two subsequent implantation steps, POCKET-L and POCKET-R, of opposite implantation angles). Table 9.3 lists the relevant process parameters and responses. The second statement creates an EVE object to manage all experiments of the optimization task. The symbolic name "flow-eve" is assigned to identify this object unequivocally in the case several EVE instances exist simultaneously. The Eve-Define-Control statements (cf. Section 8.4.1) define the position of the experimental control variables in the process flow. For example, the implantation angle anginc in the process step labeled POCKET-R is assigned the symbolic name PAR. The value of PAR is derived from the value of

Step	Operation	Parameter	Description
POCKET-L	Ion implantation	PD	Implantation dose
		PE	Implantation energy
		PA	Implantation angle
POCKET-R	Ion implantation	PD	Implantation dose
		PE	Implantation energy
		PAR	Implantation angle with opposite sign
DEVICE	Device simulation	vd	Drain voltage
		vth	Threshold voltage
		ib	Maximum bulk current
		id-on	Maximum drain current
		id-off	Minimum drain current

Table 9.3: Optimization parameters and extracted responses for LAT implant.

[5] All parameter definition, experiment generation and management, response-surface modeling, and optimization operations are also supported by the GUI.

```
;; define a new project for this task
(Project "pockets" :flow-file #"~/processes/pockets.sfe")

;; create a new EVE object for data and experiment management
(Flow-Eve "flow-eve")

;; define control variables in process flow

(Eve-Define-Control "L"   "INIT" :conversion '(string L))

(Eve-Define-Control "PD"  "POCKET-L" "dose")     ;; LAT implant dose
(Eve-Define-Control "PD"  "POCKET-R" "dose")     ;; same for negative
                                                 ;; angle
(Eve-Define-Control "PE"  "POCKET-L" "rmoden")   ;; LAT implant energy
(Eve-Define-Control "PE"  "POCKET-R" "rmoden")   ;; LAT implant energy

(Eve-Define-Control "PA"  "POCKET-L" "anginc")   ;; LAT implant angle
(Eve-Define-Control "PAR" "POCKET-R" "anginc"    ;; negative implant angle
                    :eval-expr '(and PA (* -1 PA)))

(Eve-Define-Control "vd"  "DEVICE" "drain-bias") ;; drain bias for
                                                 ;; device simulation

(Eve-Set-Control "L"   0.5  :min 0.25 :max 1.5  :trans 'log)
(Eve-Set-Control "PD"  1e13 :min 1e12 :max 3e13 :trans 'log)
(Eve-Set-Control "PE"  50.  :min 40.  :max 60.)
(Eve-Set-Control "PA " 25.  :min 20.  :max 30.)
(Eve-Set-Control "vd"  2.)

;; define response variables in process flow

(Eve-Define-Response "vth"    "DEVICE" "THRESHOLDVOLTAGE")
(Eve-Define-Response "ib"     "DEVICE" "BULKCURRENT"
                     :eval-expr '(* 1e10 (last ib)))

(Eve-Define-Response "id-on"  "DEVICE" "DRAINCURRENT"
                     :eval-expr '(last id-on))

(Eve-Define-Response "id-off" "DEVICE" "DRAINCURRENT"
                     :eval-expr '(first id-off))
```

Figure 9.6: Definition of process control and response variables for a process-flow EVE.

the control variable `PA` by multiplying with `-1`, leading to symmetric implantation profiles. The `Eve-Set-Control` statements define default values and ranges for the control variables.

Responses are defined with the `Eve-Define-Control` statement. In the example the device simulation generates sets of data points for gate voltages between $0V$ and the drain bias `vd`. Using the LISP functions `last` and `first` to derive the values of the response variables from these sets is a simple way to get the minimum and maximum drain current and bulk current values.

Once the controls and responses are defined, the `Eve-Eval` statement is used to simulate the nominal device. After termination of the simulation, the results can be examined (Figure 9.7). Parameter value settings can be given to study interactively the behavior of the design.

```
 ;; simulate nominal device
 (Eve-Eval)
 ;; show all evaluation instances
 (Eve-Show)
         L  TAI       PD PE  PL   PAR vd   vth      ib    id-on   id-off

DEFAULT: 0.5 10.   1E+10 50. 25.   -- 2.
 1 R  1: 0.5 10.   1E+10 50. 25. -25. 2. 0.463 2.5E-05 9.35E-05 2.87E-13

 ;; simulate device with modified LAT implant dose
 (Eve-Eval :LD 2e12)

 (Eve-Show)
           L    PD PE  PA   PAR vd   vth       ib    id-on   id-off

DEFAULT: 0.5 1E+10 50. 25.   -- 2.
 1 R  1: 0.5 1E+10 50. 25. -25. 2. 0.463  2.5E-05 9.35E-05 2.87E-13
 2 R  2: 0.5 1E+12 50. 25. -25. 2.  0.36 4.11E-06 0.000118 2.08E-12
```

Figure 9.7: The simulation of the nominal device is started by using the `Eve-Eval` command. Parameter value settings can be given. Note that `ib` represents 10^{10} times the maximum bulk current (see Figure 9.6).

Figure 9.8 shows the VISTA/SFC `sequence`[6] statement calling three DoE steps to create two times 7 experiments, varying the gate length `L` from $0.25\mu m$ to $1.5\mu m$ in equidistant steps for `PD` set to 10^{10} and 10^{12}, respectively, plus 15 experiments from a full-factorial CCI design for the three process parameters. The `Eve-Stop` command prevents the new experiments from being submitted for simulation immediately; `Eve-Start` starts evaluation for all 29 scheduled instances. They are run simultaneously by the run controller and distributed across the network.

After computation of the sample points, a response surface is built from the data generated[7] for the three responses `vth`, `ib`, and `id-on`. Figure 9.9 shows the sequence of statements to

[6] Both the `sequence` and `parallel` statements create objects that provide basic control capabilities for executing a set of asynchronous VLISP statements – user-defined or built-in – sequentially or in parallel, respectively. They liberate the user from explicitly taking care of connecting the statements by means of callbacks. The `sequence` and `parallel` statements can be arbitrarily nested.

[7] Simulations are only started if no results exist in the run data base for the given process flow and parameter settings.

```
(sequence
  '(Eve-Stop)
  '(Eve-Doe 'SA :nsp 7 :select "L")
  '(Eve-Doe 'SA '(("PD" 1e12)) :nsp 7 :select "L")
  '(Eve-Doe 'CCI :cube "FF" :select '("PD" "PE" "PA"))
  '(Eve-Start)
)
```

Figure 9.8: Design of experiments for screening analysis of LAT implant. The DoE module is called for the design type SA and override nominal settings for the dose parameter `PD`. The **`sequence`** operator is used to chain a sequence of asynchronous LISP commands.

run the RSM generator, define a target function for the optimization process, and start the optimization for a set of initial values. From the available optimizers, the **`donopt`** program is selected. The target function for the optimizer is constructed from the threshold and maximum bulk current values to raise the threshold voltage at L=0.5μm and keep the bulk current as low as possible. If one wishes to run the optimizer directly on the simulation server, the RSM generation step is skipped and the target function is defined directly in the **`"flow-eve"`** object to provide input for the optimizer (Figure 9.10).

```
(sequence
  '(Opt-Rsm-Eve "rsm-eve" :select '("PD" "PE" "PA" "vth" "ib" "id-on")
                          :eve "flow-eve")

  '(Eve-Define-Response "target"
                        :eval-expr '(and vth ib (abs (/ (* 1e3 ib) vth)))
                        :eve "rsm-eve")

  '(Eve-Optimize "target" :eve "rsm-eve"
                          :range-eve "flow-eve"
                          :optimizer 'donopt
                          :init '(("PD" 1e12)("PE" 20.)("PA" 0.))
)
```

Figure 9.9: VISTA/SFC statements for RSM model generation and optimization.

```
(sequence
  '(Eve-Define-Response "target"
                        :eval-expr '(and vth ib (abs (/ (* 1e3 ib) vth)))
                        :eve "flow-eve")
  '(Eve-Optimize "target" :eve "flow-eve"
                          :optimizer 'donopt
                          :init '(("PD" 1e12)("PE" 20.)("PA" 10.))
)
```

Figure 9.10: Calling the optimizer directly on the process flow simulation task encapsulated in the **`"flow-eve"`** EVE object.

9.2.5 Optimization Result and Verification

Table 9.4 gives the parameters found by the procedure shown in Figure 9.9, Figure 9.11 depicts the threshold-voltage vs. gate-length characteristic for the optimum device.

Dose `PD`	$9.64e^{12}cm^{-2}$
Energy `PE`	$42.9keV$
Angle `PA`	21.9^o

Table 9.4: Optimum implantation parameters for pocket implants.

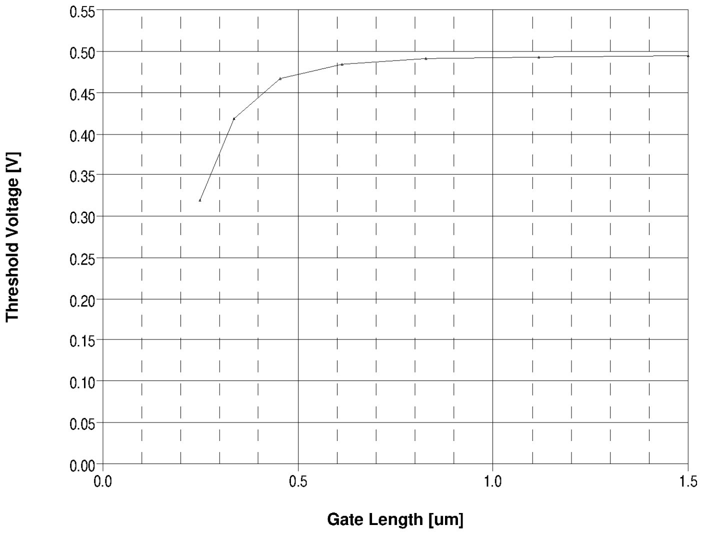


Figure 9.11: V_{th}–L behavior after optimum pocket implants. The onset of the V_{th} roll-off shifts towards smaller gate lengths in comparison with the original device (Figure 9.4).

For comparison with the nominal device, Figure 9.12 shows the LDD, source/drain, and threshold-adjust implants around the gate edge (at $0.55\mu m$) of the 1.1μ NMOS device, looking from the surface into the bulk. The (n-type) LDD doping compensates the (p-type) threshold-adjust implant inside the channel region and causes a reduction of the effective channel length. Figure 9.13 shows the channel region around the gate edge with a pocket implant, using the parameters established by the optimization procedure. In the critical region between $0.5\mu m$ and $0.6\mu m$, the threshold-adjust implant is enhanced by the pocket implant; the total p-type doping is high enough to compensate the influence of the LDD and source/drain implants around the gate edge.

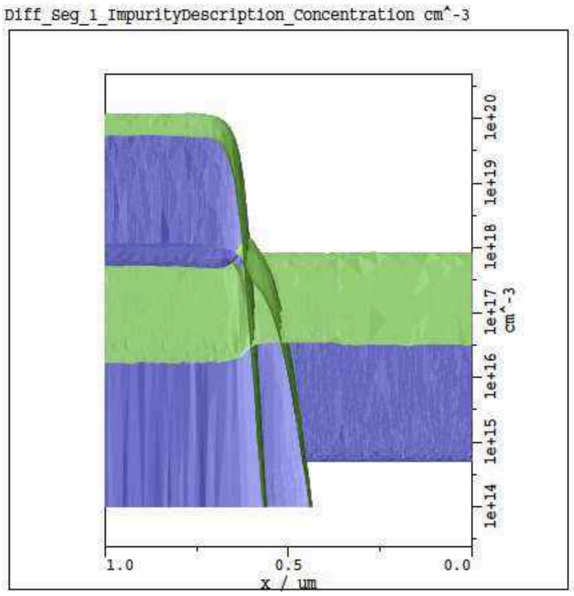


Figure 9.12: Short-channel device doping profile without pocket implants.

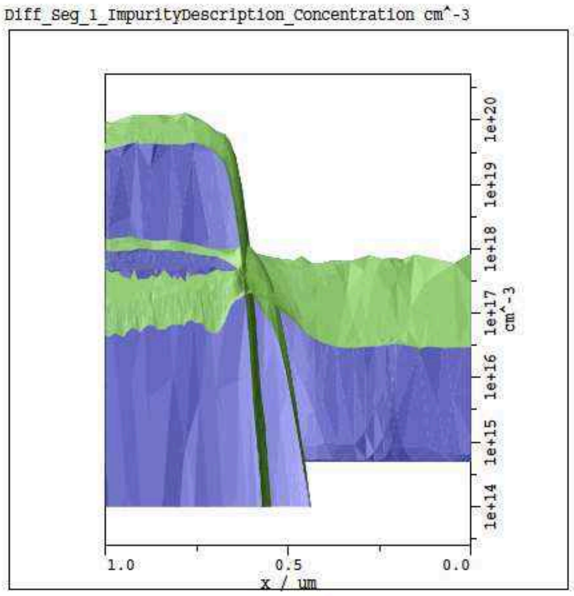


Figure 9.13: Short-channel device doping profile with pocket implants.

In many cases, an optimization task gets stuck in a local minimum and does not reach the global optimum. Starting the optimizer at different initial points is a helpful strategy to find the global optimum. Taking advantage of VISTA/SFC's task-level programming facilities, the dependence of the optimization result on the choice of initial values for the optimization problem can be easily investigated by specifying the optimization problem itself as en EVE object and submitting it to a set of experiments. Figure 9.14 shows the list of statements used for running the optimization task for three values for each of the optimization variables, generating a total of nine optimization runs that are executed in parallel by VISTA/SFC.

```
(sequence
  '(Eve-Define-Response "target"
                        :eval-expr '(and vth ib (abs (/ (* 1e3 ib) vth)))
                        :eve "rsm-eve")

  '(Lisp-Eve "lisp-eve" :control '("PD0" "PE0" "PA0")
                        :expression
                         '(Eve-Optimize "target" :eve "rsm-eve"
                                                 :range-eve "flow-eve"
                                                 :optimizer 'donopt
                                                 :init `(("PD" ,PD0)
                                                         ("PE" ,PE0)
                                                         ("PA" ,PA0))))

  '(Select-Eve "lisp-eve")
  '(Eve-Set-Control "PD0" 1e12 :min 1e10 :max 1e14 :trans 'log)
  '(Eve-Set-Control "PE0" 30.  :min 10.  :max 50.)
  '(Eve-Set-Control "PA0" 20   :min 5.   :max 35.)
  '(Eve-Doe 'SA :nsp 3)
)
```

Figure 9.14: Verification of optimization by sampling the initial value space for the optimization task using three experiments each along the axes of the Lisp-EVE object encapsulating the optimization task.

9.3 Summary

Given the capability of analyzing complete VLSI designs by means of process and device simulation using heterogeneous tools, more complex tasks can be encapsulated by objects that provide basic flow control and data management functionality. Additionally, the encapsulation defines standard interfaces with optimization and RSM tools that can be used for a variety of applications.

Chapter 10

Discussion

10.1 Benefits of TCAD Environments

With respect to simulation environments, the question is frequently raised if they are at all necessary, if they are beneficiary for industrial applications, or if they present merely a nicely designed cover-up for missing functionality and methodological flaws. Should VLSI engineers bother to get accustomed to a new environment, with new concepts, procedures, and possible drawbacks ? Or should they rather stay with existing solutions that have proven viable ?

The costs of migrating to new procedures and employing new methodologies should not be neglected. As with any transition process, failures and frustrations are to be expected. However, the benefits gained from such transition should outweigh the costs incurred. In general, a software environment should guide the user through a large number of complex decisions and operations, providing abstractions of frequently encountered tasks and limiting the number of erroneous design paths taken without imposing unjustified restrictions.

VISTA/SFC helps to liberate the users' time and energy to focus on creative design tasks by offering full programmability atop of a wide range of built-in functionality.

10.2 Side Effects on Simulators

Automatic control and execution of heterogeneous simulation tools does not only concern the controlling environment, but also the tools themselves. Aside from the basic need to accept initial data from other tools in some form[1], a certain conduct has to be followed in order to cooperate with other simulators. The reasonable treatment of error conditions, including the generation of concise and helpful error messages, returning appropriate exit codes, and closing files that have been opened during the program run before the program exits, as well as refraining from producing large amounts of unsolicited output that is only helpful for the tool developer, are to be mentioned. Moreover, a certain degree of tolerance and flexibility with respect to the input data and to the class of problems to solve proved to be

[1] As opposed to solve problems that are hard-coded in the program-m code and can only be altered by re-compilation.

extremely important. Given the ability to automatically control large sets of experiments, many deficiencies and nuisances have been eliminated from the tools used as a result of deploying them in a greater context.

10.3 Implementation

The VLISP interpreter has proven to be a powerful, convenient, and flexible programming environment very well suited for highly dynamic applications. As a prototype development tool, it probably has no peer. For industrial applications, though, two drawbacks can be identified, which potentially limit the widespread use of VLISP-based applications:

1. The general reluctance of most people not familiar with LISP and LISP-like syntax to get involved with the language.
2. The fact that, in comparison to a compiled language, consistency checks automatically applicable to the body of code loaded by applications are limited in their effectiveness and number.

Although these drawbacks are more than compensated for by the flexibility and ease of use during code developing stages, the production use of the VISTA/SFC simulation environment might gain considerably in long-term stability from using compiled versions of critical modules and routines[2].

10.4 Development Process

The Simulation Flow Control Module as well as the VISTA/SFC environment have been in active use from the very beginning of the project on. Therefore, all modifications of interfaces and file formats that go along with the maturation of a software product had to keep the compatibility with previous versions in order not to render existing results generated with these tools obsolete. Up to three different versions of a submodule were found to simultaneously exist at a given time in VISTA/SFC. Although the continuing support of different versions did not add to the maintainability of the system during development, it was crucial for acceptance by the users who did not want to throw away existing data. Changing critical definitions and forcing all existing data to be rebuilt is the most simple, but not the most user-friendly solution.

10.5 The Road Ahead

VISTA/SFC has reached an intermediate state with respect to unrestrained industrial application. A high level of stability and robustness of operation has been achieved. Due the

[2] The VLISP mechanisms to automatically interface with C or C++ code, making it available to the LISP user, seems to provide a sound basis for translating to C or C++ much of the LISP-code presently used in VISTA/SFC.

openness for new tools and applications, a reasonable degree of penetration of the TCAD world can be expected. As work is still in progress, a couple of issues remain to be addressed by future developments. Comparing VISTA/SFC with a generic framework like Nelsis [Dim96] gives some hints for the direction to go.

10.5.1 Tool Integration and Task Abstraction

The integration of new tools still requires some manual coding, and the ones already integrated stem from different epochs of development and follow slightly different lines of thoughts. To make the integration procedure less cumbersome, some interactive support has to be added to the environment to at least automate the interfacing of *standard* tools, i.e., tools that follow a standard scheme for data input and output, input deck format, and invocation. An added benefit of this undertaking would be the definition of standard tool development guidelines to aid in the design of the data and control interfaces of new simulators.

The EVE concept introduces a certain degree of abstraction on the task level, but a lot of work remains to be done to define a terminology and supporting object classes. Furthermore, the different tasks encapsulated by EVE objects have to be provided with representations on the GUI level to be more easily accessible without losing the flexibility of the programming interface.

10.5.2 Concurrent Design

Concurrent engineering and design becomes more and more important as design tasks grow more complex and an increasing number of people have to cooperate. Shared access to design data plays a key role for efficient interaction if the access mechanisms are realized properly. Supporting the notion of a design repository with import and export procedures available for obtaining and providing data that can be used within one's own design could be a first step. In the long run, a separation of different tasks into different applications acting as servers for user clients seems to be the preferred solution.

10.5.3 Version Control

Both for stand-alone operation as well – and even more so – in a concurrent design environment, the management of different versions of design data should be supported by the simulation environment, instead of relying on file-oriented tools like RCS [Tic85] being directly accessed by the user.

10.5.4 PIF Wafer Processor

One of the applications that could potentially spawn from VISTA/SFC at its present state would take care of all simulation and data manipulation carried out on PIF wafer data. Conceptually, all simulation tools, gridding tools, and auxiliary tools (cf. Appendix C) operating on PIF data form a PIF *wafer processor* that encapsulates wafer model manipulations and mediates between process simulation requests and wafer data manipulation. It acts as a server for all physical and non-physical wafer model operations (Figure 10.1). The server paradigm[3] effectively enhances the file-based tool integration and data exchange approach as it is able to maintain context information during sequences of operations that would otherwise be lost.

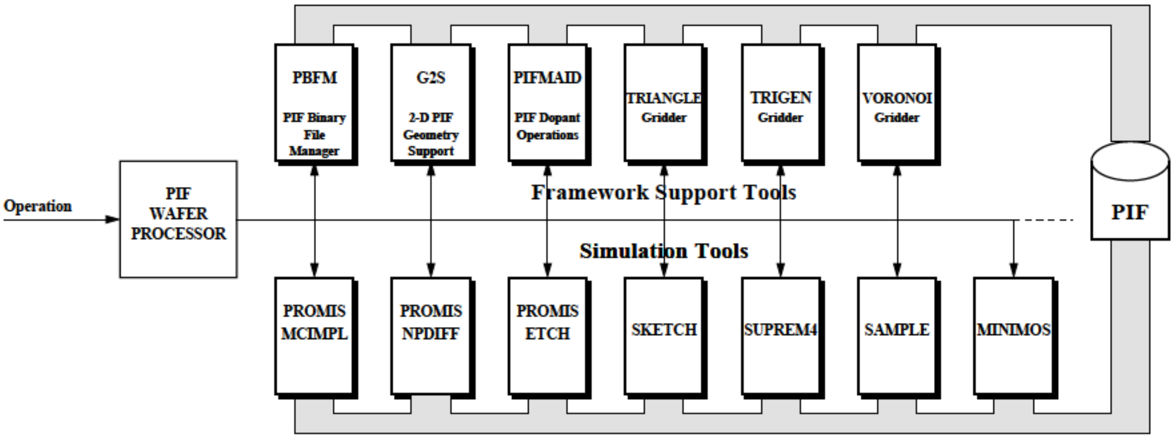


Figure 10.1: PIF Wafer Processor.

Furthermore, calibration data can be down-loaded to the wafer processor to tune it to measurements obtained from the fab. For practical purposes, inserting calibration data into the data flow to the simulation tools could be done in a pre-processing module (Figure 10.2). Calibration can be implemented as outlined in Sections 4.5 and 8.8.

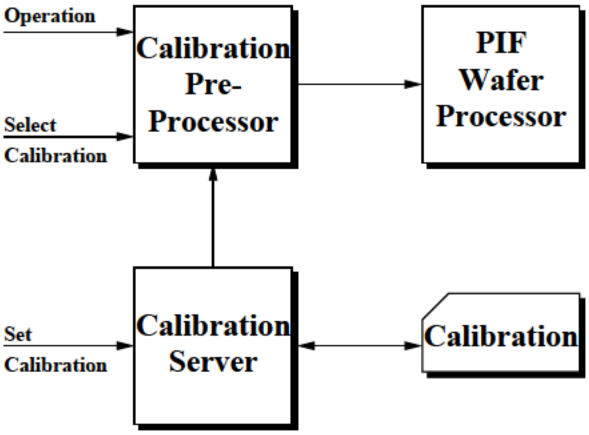


Figure 10.2: Calibration pre-processor for PIF Wafer Processor.

[3] As VLISP intrinsically supports multiple parallel processes and contexts, the *server* need not necessarily be implemented as a program separate from the framework.

10.5.5 Internet Collaboration and WWW

As pointed out in [Los96], the Internet in its incarnation as World Wide Web offers an attractive means of cooperation between designers and design groups at distant locations. The architecture of VISTA/SFC has taken into account many aspects prerequisite for interfacing with all kinds of clients, as the built-in GUI is completely independent of the rest of the framework, which can run as background process servicing requests from other processes. The implementation of network functions consistent with the overall architecture is currently being addressed.

Appendix A

LISP, XLISP, and VLISP

For the reader unfamiliar with the LISP programming language, the following few paragraphs give a brief overview of the peculiarities of the language. For more detailed information, the literature should be consulted [WH89] [Bet89] [Ins96a].

The programming language LISP has been designed primarily for use in artificial intelligence research and applications. It is an interpreting language, byte-code compilers exist. LISP provides automatic memory management and dynamic type binding. Due to its unique – some say, ugly – syntax, LISP, despite its many advantages, has not grown very popular with the scientific community. LISP uses operators in prefix notation and is notorious for the multitude of parentheses that intersperse the code. Figure A.1 shows the same statement twice, on the left side coded in C [KR88], on the right side coded in LISP, Figure A.2 shows the definition of a simple function in the C and in LISP.

```
{
  integer x;
  x = 666;           (setq x 666)
}
```

Figure A.1: Value assignment in C and LISP. **setq** is the LISP assignment statement and reads *set quote* to indicate that **x** should not be evaluated but should be used as a name.

```
integer times10 (integer x)        (defun times10 (x)
{                                    (* 10 x)
  return(10 * x);                  )
}
```

Figure A.2: Function declaration and definition in C and LISP. Note that no type declaration is provided in LISP, allowing values of any types to be passed to the function. If necessary, the type of an argument can be checked. A LISP function automatically returns the result of the last statement of the function code.

A LISP function declaration may contain various classes of function arguments as shown in Figure A.3. The list of arguments starts with required symbols that have to be assigned when the function is invoked. After the `&optional` statement, optional arguments may be passed. The `&key` statement starts the list of `key` parameters which are passed by specifying the name together with the value. `&optional` and `&key` parameters can also carry default values to be used when they are not specified by the caller.

```
(defun print-time (time
                   &optional (destination *standard-output*)
                   &key pretty)

  ;;; printing function code ...
)

(print-time time)
(print-time time file)
(print-time time :pretty T)
(print-time time file :pretty T)
```

Figure A.3: LISP function header. A LISP function accepts required, optional, and key arguments. The given function `print-time` prints the time information passed in `time` to the standard output or to an optionally specified file. The `pretty` key selects a pretty-print mode. The semicolon is used as the comment character in LISP.

All LISP data are stored in *nodes*, which are allocated, collected, and reused automatically by the interpreter; no explicit memory management is required. As long as a node is visible to any procedure it is protected from garbage collection. LISP uses lexical scoping to determine the visibility of a symbol.

LISP nodes carry type information, which is established at the moment when a symbol is bound to a specific value. In the example of Figure A.3, the symbol `time` of the function `print-time` accepts a value of any type. In order to work properly, however, a value of the correct type has to be passed to the print routine (not shown in the figure). LISP provides predicate functions to inquire the type of a symbol at run-time.

XLISP by D. Betz [Bet89] is a publicly available LISP implementation including object-oriented extensions completely written in the programming language C; it is easily extended with user written built-in functions and classes.

Vienna LISP (VLISP)is based on XLISP version 2.1 and has been enhanced by various extensions of the original implementation. New data types and procedure libraries have been added to support handling of system processes, regular expressions, operating-system independent file-specifications, timeouts, and time stamps (cf. Tables 3.2 and 3.3). The main event loop has been adapted to accommodate the callback mechanism (cf. Appendix B) used throughout VLISP as means of asynchronous communication with system processes, the graphical user interface, and timers. It runs on a variety of operating systems and platforms and has proven a very reliable and robust basis of implementation even for very large systems.

Appendix B

Callback Mechanism

The callback concept as a means of asynchronous communication in a computing environment plays a very crucial role for the operation of the VISTA/SFC environment. All modules communicate with the user interface, with external programs, and with each other by using VLISP's callback mechanism to synchronize asynchronous operations on the task level and on the tool level. Figure B.1 shows the basic principle. After calling the callee, e.g., an external executable or a pop-up window, the caller remains idle until the callee signals termination by calling the callback function passed as an argument by the caller. The callback function returns exit code, output data, and diagnostic information in a uniform format. In the time between call and callback, the caller is able to service other callbacks to initiate new calls. The callback mechanism forms the basis for the parallelization of tool calls, process runs, and projects.

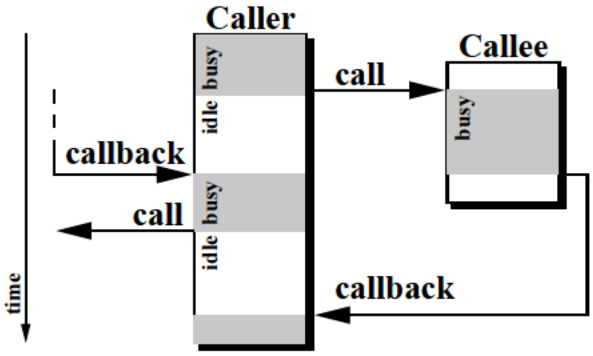


Figure B.1: Callback concept for asynchronous communication between modules.

Appendix C

Auxiliary PIF Tools

The manipulation of large PIF files in VISTA/SFC is supported by a number of auxiliary tools to perform operations that are time-consuming or computationally intensive. Table C.1 gives a brief summary of tools and purposes.

`pifcopy`	Copies and renames logical PIF files.
`pifmaid`	Generates net doping and total doping distributions; generates grids; removes doping information from segments; deletes PIF objects from a logical file; merges dopant data; analyzes contents of logical PIF files and prints wafer information (Figure C.1).
`pifrm`	Deletes logical PIF files.
`pifstat`	Checks accessibility of PIF files.

Table C.1: Auxiliary PIF tools.

```
X-Min: -5.000000e+00, Y-Min: -2.000000e+00
X-Max: 5.000000e+00, Y-Max: 3.660000e-01
Units: um
POINTS: 18 in pointList_-paaa
LINES: 22 in lineList_-qaaa
FACES: 2 in faceList_-raaa
SEGMENTS: 2 in segmentList_-saaa
GRIDS: 2
 GRID: Diff_Grid, TYPE: TRI3, DIMENSION: 2, POINTS: 4851
 GRID: Diff_Grid_-faaa, TYPE: TRI3, DIMENSION: 2, POINTS: 61
IMPURITIES: 1
 IMPURITY: Diff_Seg_1_ImpurityDescription
  MATERIALTYPE: Boron, ACTIVITYTYPE: active, GRID: Diff_Grid
SEGMENTS: 2
 SEGMENTNAME: Diff_Seg_1_SegmentDescription
  MATERIALTYPE: Si, FACES: 1
 SEGMENTNAME: Diff_Seg_2_SegmentDescription
  MATERIALTYPE: SiO2, FACES: 1
BOUNDARIES: 0
```

Figure C.1: Summary of PIF wafer information generated by `pifmaid`.

Bibliography

[AAY+88] A.R. Alvarez, B.L. Abdi, D.L. Young, H.D. Weed, J. Teplik, and E.R. Herald. Application of Statistical Design and Response Surface Methods to Computer-Aided VLSI Device Design. *IEEE Trans.Computer-Aided Design*, 7(2):272–288, 1988.

[AGJ93] V. Axelrad, Y. Granik, and R. Jewell. CAESAR: The Virtual IC Factory as an Integrated TCAD User Environment. In Fasching et al. [FHS93], pages 293–307.

[AHD83] *The American Heritage Dictionary.* Houghton Mifflin, second edition, 1983.

[Ass94] Semiconductor Industry Association. National Technology Roadmap for Semiconductors, 1994.

[Ban90] R.E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations*, volume 7 of *Frontiers in Applied Mathematics.* SIAM, Philadelphia, 1990. Users' Guide 6.0.

[BCDH94] R. Beaver, A. Coleman, D. Draheim, and A. Hoffman. Architecture and Overview of MMST Machine Control. *IEEE Trans.Semiconductor Manufacturing*, 7(2):127–133, 1994.

[BCF+95] B. Baccus, D. Collard, P. Ferreira, V. Senez, and E. Vandenbossche. *IMPACT-4 User's Guide*, February 1995. Version 4.7.

[BD87] G.E.P. Box and N.R. Draper. *Empirical Model-Building and Response Surfaces.* Wiley, 1987.

[Bet89] D.M. Betz. *XLISP: An Object-Oriented Lisp, Version 2.1.* Apple, Peterborough, New Hampshire, USA, 1989.

[BHW91] D.S. Boning, M.L. Heytens, and A.S. Wong. The Intertool Profile Interchange Format: An Object-Oriented Approach. *IEEE Trans.Computer-Aided Design*, 10(9):1150–1156, September 1991.

[BM94] D.S. Boning and P.K. Mozumder. DOE/Opt: A System for Design of Experiments, Response Surface Modeling, and Optimization Using Process and Device Simulation. *IEEE Trans.Semiconductor Manufacturing*, 7(2):233–244, 1994.

[BMPS92] D.S. Boning, M.B. McIlrath, P. Penfield, Jr., and E.M. Sachs. A General Semiconductor Process Modeling Framework. *IEEE Trans.Semiconductor Manufacturing*, 5(4):266–280, 1992.

[Boh96] W. Bohmayr. *Simulation der Ionenimplantation in kristalline Siliziumstrukturen.* Dissertation, Technische Universität Wien, 1996.

[Bon91] D.S. Boning. *Semiconductor Process Design: Representation, Tools, and Methodologies.* PhD thesis, Massachusetts Institute of Technology, 1991.

[CBF$^+$90] D.C. Cole, E.M. Buturla, S.S. Furkay, K. Varahramyan, J. Slinkman, J.A. Mandelman, D.P. Foty, O. Bula, A.W. Strong, J.W. Park, T.D. Linton Jr., J.B. Johnson, M.V. Fischetti, S.E. Laux, P.E. Cottrell, H.G. Lustig, F. Pileggi, and D. Katcoff. The Use of Simulation in Semiconductor Technology Development. *Solid-State Electron.*, 33(6):591–623, 1990.

[CBK$^+$93] R. Cartuyvels, R. Booth, S. Kubicek, L. Dupas, and K.M. De Meyer. A Powerful TCAD System Including Advanced RSM Techniques for Various Engineering Optimization Problems. In Selberherr et al. [SSS93], pages 29–32.

[Dav96] B. Davari. CMOS Technology Scaling, 0.1μm and Beyond. In *Int.Electron Devices Meeting*, pages 555–558, 1996.

[DCB93] D. Durbeck, J.-H. Chern, and D.S. Boning. A System for Semiconductor Process Specification. *IEEE Trans.Semiconductor Manufacturing*, 6(4):297–305, 1993.

[Dim96] Dimes Design and Test Centre, Delft University of Technology, The Netherlands. *Introduction to the NELSIS CAD Framework*, September 1996. http://www.ddtc.dimes.tudelft.nl/docs/frame.intro/fintro-toc.html.

[Duv88] S.G. Duvall. An Interchange Format for Process and Device Simulation. *IEEE Trans.Computer-Aided Design*, 7(7):741–754, 1988.

[DY93] R.W. Dutton and Z. Yu. *Technology CAD – Computer Simulation of IC Processes and Devices.* Kluwer, 1993.

[Ele91] Electronics Research Laboratory, University of California, Berkeley. *SAMPLE 1.8a User Guide*, 1991.

[Eng86] W.L. Engl, editor. *Process and Device Modeling*, volume 1 of *Advances in CAD for VLSI.* North-Holland, 1986.

[Fas94] F. Fasching. *The Viennese Integrated System for Technology CAD Applications–Data Level Design and Implementation.* Dissertation, Technische Universität Wien, 1994.

[FFS$^+$91] F. Fasching, C. Fischer, S. Selberherr, H. Stippel, W. Tuppa, and H. Read. A PIF Implementation for TCAD Purposes. In W. Fichtner and D. Aemmer, editors, *Simulation of Semiconductor Devices and Processes*, volume 4, pages 477–482, Konstanz, 1991. Hartung-Gorre.

[FHH$^+$94] C. Fischer, P. Habas, O. Heinreichsberger, H. Kosina, Ph. Lindorfer, P. Pichler, H. Pötzl, C. Sala, A. Schütz, S. Selberherr, M. Stiftinger, and M. Thurner. *MINIMOS 6.0 User's Guide.* Institute for Microelectronics, Technical University Vienna, March 1994.

[FHS93] F. Fasching, S. Halama, and S. Selberherr, editors. *Technology CAD Systems*, Wien, 1993. Springer.

[Fis94] C. Fischer. *Bauelementsimulation in einer computergestützten Entwurfsumgebung.* Dissertation, Technische Universität Wien, 1994.

[FTS94] F. Fasching, W. Tuppa, and S. Selberherr. VISTA-The Data Level. *IEEE Trans.Computer-Aided Design*, 13(1):72–81, 1994.

[GG91] TCAD Framework Group and SPR Working Group. Semiconductor Process Representation Information Model V0.2.2. Technical Report Document Number 71, CAD Framework Initiative, Austin, Texas, August 1991.

[GG92] TCAD Framework Group and SPR Working Group. Semiconductor Process Representation Requirements V0.4.1. Technical Report CFI Doc. TCAD-91-G-4, CAD Framework Initiative, Austin, Texas, April 1992.

[GMW95] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, 1995.

[GW94] G.J. Gaston and A.J. Walton. The Integration of Simulation and Response Surface Methodology for the Optimization of IC Processes. *IEEE Trans.Semiconductor Manufacturing*, 7(1):22–33, 1994.

[Hal94] S. Halama. *The Viennese Integrated System for Technology CAD Applications–Architecture and Critical Software Components*. Dissertation, Technische Universität Wien, 1994.

[HB93] P.J. Hopper and P.A. Blakey. The MASTER Framework. In Fasching et al. [FHS93], pages 275–292.

[HFF^{+}93] S. Halama, F. Fasching, C. Fischer, H. Kosina, E. Leitner, Ch. Pichler, H. Pimingstorfer, H. Puchner, G. Rieger, G. Schrom, T. Simlinger, M. Stiftinger, H. Stippel, E. Strasser, W. Tuppa, K. Wimmer, and S. Selberherr. The Viennese Integrated System for Technology CAD Applications. In Fasching et al. [FHS93], pages 197–236.

[HH83] J.M. Hammersley and D.C. Handscomb. *Monte Carlo Methods*. Chapman & Hall, London, 1983.

[HHW^{+}90] G. Hobler, S. Halama, W. Wimmer, S. Selberherr, and H. Pötzl. RTA-Simulations with the 2-D Process Simulator PROMIS. In *Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits NUPAD III*, pages 13–14, Honolulu, 1990.

[HKS^{+}91] A. Hori, S. Kameyama, M. Segawa, H. Shimomura, and H. Ogawa. A Self-Aligned Pocket Implantation (SPI) Technology for 0.2μm-Dual Gate CMOS. In *Int.Electron Devices Meeting*, pages 641–644, 1991.

[HM96] M. Hasanuzzaman and C.H. Mastrangelo. Process Compilation of Thin Film Microdevices. *IEEE Trans.Computer-Aided Design*, 15(7):745–764, 1996.

[HPR^{+}95] S. Halama, Ch. Pichler, G. Rieger, G. Schrom, T. Simlinger, and S. Selberherr. VISTA—User Interface, Task Level, and Tool Integration. *IEEE Trans.Computer-Aided Design*, 14(10):1208–1222, 1995.

[HPW91] G. Hobler, P. Pichler, and K. Wimmer. *PROMIS 1.5 User Guide*. Institut für Mikroelektronik, Technische Universität Wien, 1991.

[HRW90] Ch.J. Hegarty, L.A. Rowe, and Ch.B. Williams. The Berkeley Process-Flow Language WIP System. Technical Report UCB/ERL M90/77, University of California, Berkeley, September 1990.

[Ins96a] Institute for Microelectronics, Technical University Vienna. *VISTA Documentation 1.3-1, VLISP Manual*, January 1996.

[Ins96b] Institute for Microelectronics, Technical University Vienna. *VISTA Documentation 1.3-2, PIF Application Interface Manual*, June 1996.

[Ins96c] Institute for Microelectronics, Technical University Vienna. *VISTA Documentation 1.3-2, VISTA Visualization Tool*, June 1996.

[Jon89] O. Jones. *Introduction to the X Window System*. Prentice-Hall, 1989.

[KFBS95] N. Khalil, J. Faricelli, D. Bell, and S. Selberherr. The Extraction of Two-Dimensional MOS Transistor Doping via Inverse Modeling. *IEEE Electron Device Lett.*, 16(1):17–19, 1995.

[KGMB94] S. Kleinfeldt, M. Guiney, J.K. Miller, and M. Barnes. Design Methodology Management. *Proc.IEEE*, 82(2):231–250, 1994.

[Kha95] N. Khalil. *ULSI Characterization with Technology Computer-Aided Design*. Dissertation, Technische Universität Wien, 1995.

[KKW93] W. Kanert, N. Krischke, and K. Wiesinger. Optimization of DMOS Transistors for Smart Power Technologies by Simulation and Response Surface Methods. In Selberherr et al. [SSS93], pages 213–216.

[KN95] P.R. Kristoff and D.P. Nunn. The Process Specification System for MMST. *IEEE Trans.Semiconductor Manufacturing*, 8(3):262–271, 1995.

[KPR+96] H. Kirchauer, R. Plasun, M. Radi, S. Selberherr, and R. Strasser. VISTA Status Report. Technical report, Institute for Microelectronics, Technical University Vienna, December 1996.

[KR88] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, 1988.

[KS96a] H. Kirchauer and S. Selberherr. Rigorous Three-Dimensional Photolitography Simulation Over Nonplanar Structures. In G. Baccarani and M. Rudan, editors, *ESSDERC'96 - 26th European Solid State Device Research Conference*, pages 347–350, Gif-sur-Yvette Cedex, France, 1996. Editions Frontieres.

[KS96b] H. Kirchauer and S. Selberherr. Three-Dimensional Photoresist Exposure and Development Simulation. In SISPAD'96 [SIS96], pages 99–100.

[LA91] T.J. Lorenzen and V.L. Anderson. *Design of Experiments*. Marcel Dekker, 1991.

[Law95] M.E. Law. Grid Adaption Near Moving Boundaries in Two Dimensions for IC Process Simulation. *IEEE Trans.Computer-Aided Design*, 14(10):1223–1230, 1995.

[Los96] P. Losleben. A New Semiconductor Research Paradigm Using Internet Collaboration. In SISPAD'96 [SIS96], pages 9–12.

[Mar93] J. Mar. Technology CAD at Intel. In Fasching et al. [FHS93], pages 63–74.

[Mar96] J. Mar. The Application of TCAD in Industry. In SISPAD'96 [SIS96], pages 139–145.

[Mas95] H. Masuda. T^{2}CAD: Total Design for Sub-μm Process and Device Optimization with Technology-CAD. In Ryssel and Pichler [RP95], pages 408–415.

[Mey95] M. Meyyappan, editor. *Computational Modeling in Semiconductor Processing*. Artech House, 1995.

[MHM94] J. McGehee, J. Hebley, and J. Mahaffey. The MMST Computer-Integrated Manufacturing System Framework. *IEEE Trans.Semiconductor Manufacturing*, 7(2):107–116, 1994.

[MOA+91] H. Masuda, F. Otsuka, Y. Aoki, S. Sato, and S. Shimada. Response Surface Methods for Submicron MOSFETs Characterization with Variable Transformation Technology. *IEICE Trans.*, E 74(6):1621–1633, 1991.

[MRC+90] J.P. McVittie, J.C. Rey, L.Y. Cheng, A. Bariya, S. Ravi, and K. Saraswat. SPEEDIE: A Profile Simulator for Etching and Deposition. In *TECHNOCON*, pages 16–19, 1990.

[MS96] R. Martins and S. Selberherr. Layout Data in TCAD Frameworks. In *Modelling and Simulation*, pages 1122–1126. Society for Computer Simulation International, 1996.

[ONS+80] W.G. Oldham, A.R. Neureuther, C. Sung, J.L. Reynolds, and S.N. Nandgaonkar. A General Simulator for VLSI Lithography and Etching Processes: Part II–Application to Deposition and Etching. *IEEE Trans.Electron Devices*, ED-27(8):1455–1459, 1980.

[otCTT92a] SWR Working Group of the CFI TCAD TSC. Semiconductor Wafer Representation Architecture Document V1.0. Technical Report CFI Document TCAD-91-G-1, CAD Framework Initiative, Austin, Texas, July 1992.

[otCTT92b] SWR Working Group of the CFI TCAD TSC. Semiconductor Wafer Representation Procedural Interface V1.0. Technical Report CFI Document TCAD-91-G-2, CAD Framework Initiative, Austin, Texas, July 1992.

[Pim93] H. Pimingstorfer. *Integration und Anwendung von Simulatoren in der CMOS-Entwicklung.* Dissertation, Technische Universität Wien, 1993.

[PKSS95] Ch. Pichler, N. Khalil, G. Schrom, and S. Selberherr. TCAD Optimization Based on Task-Level Framework Services. In Ryssel and Pichler [RP95], pages 70–73.

[Pla86] *The Dialogues of Plato.* Bantam, 1986.

[Pla96] R. Plasun. HDX Library, August 1996. Internal document, Institute for Microelectronics, TU Vienna.

[PS93a] Ch. Pichler and S. Selberherr. Process Flow Representation within the VISTA Framework. In Selberherr et al. [SSS93], pages 25–28.

[PS93b] Ch. Pichler and S. Selberherr. Rapid Semiconductor Process Design within the VISTA Framework: Integration of Simulation Tools. In M.H. Hamza, editor, *Proceedings of the IASTED International Conference*, Modelling and Simulation, pages 147–150, Pittsburgh, PA, USA, 1993. The International Association of Science and Technology for Development.

[PS95] H. Puchner and S. Selberherr. An Advanced Model for Dopant Diffusion in Polysilicon. *IEEE Trans.Electron Devices*, 42(10):1750–1755, 1995.

[Puc96] H. Puchner. *Advanced Process Modeling for VLSI Technology.* Dissertation, Technische Universität Wien, 1996.

[RHS95] G. Rieger, S. Halama, and S. Selberherr. A Programmable Tool for Interactive Wafer-State Level Data Processing. In Ryssel and Pichler [RP95], pages 58–61.

[Rie96] G. Rieger. *Ein graphischer Editor für Entwurf von Halbleiterbauteilen.* Dissertation, Technische Universität Wien, 1996.

[RP95] H. Ryssel and P. Pichler, editors. *Simulation of Semiconductor Devices and Processes*, volume 6, Wien, 1995. Springer.

[RSS95] M. Rottinger, T. Simlinger, and S. Selberherr. Two-Dimensional Transient Simulation of Charge-Coupled Devices Using MINIMOS NT. In Ryssel and Pichler [RP95], pages 440–443.

[Rug84] I. Ruge. *Halbleiter-Technologie*, volume 4 of *Halbleiter-Elektronik.* Springer, 1984. 2. überarb. Aufl.

[San92] J. N. Sanders. XgPlot. Technical report, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, August 1992.

[SC96] W. Schoenmaker and R. Cartuyvels. Theory and Implementation of a New Interpolation Method Based on Random Sampling. In SISPAD'96 [SIS96], pages 157–158.

[Sel84] S. Selberherr. *Analysis and Simulation of Semiconductor Devices.* Springer, Wien, 1984.

[SFF$^+$91] S. Selberherr, F. Fasching, C. Fischer, S. Halama, H. Pimingstorfer, H. Read, H. Stippel, P. Verhas, and K. Wimmer. The Viennese TCAD System. In *Int. Workshop on VLSI Process and Device Modeling (1991 VPAD)*, pages 32–35, Oiso, Japan, 1991.

[SG92] TCAD Technical Subcommittee and SPR Working Group. Semiconductor Process Representation Procedural Interface V1.0. Technical report, CAD Framework Initiative, Austin, Texas, August 1992.

[SGHH91] E. Sachs, R.-S. Guo, S. Ha, and A. Hu. Process Control System for VLSI Fabrication. *IEEE Trans.Semiconductor Manufacturing*, 4(2):134–144, 1991.

[She96] J.R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First Workshop on Applied Computational Geometry*. American Computer Machinery, May 1996.

[Sil94] Silvaco International, Santa Clara, California. *ATHENA 2D Process Simulation Framework User's Manual*, March 1994.

[Sim96] T. Simlinger. *Simulation von Heterostruktur-Feldeffekttransistoren*. Dissertation, Technische Universität Wien, 1996.

[SIS96] *1996 International Conference on Simulation of Semiconductor Processes and Devices*, Tokyo, Japan, 1996. Business Center for Academic Societies Japan.

[SKRS95] T. Simlinger, H. Kosina, M. Rottinger, and S. Selberherr. MINIMOS-NT: A Generic Simulator for Complex Semiconductor Devices. In H.C. de Graaff and H. van Kranenburg, editors, *ESSDERC'95 - 25th European Solid State Device Research Conference*, pages 83–86, Gif-sur-Yvette Cedex, France, 1995. Editions Frontieres.

[SLW96] H. Stippel, Ph. Lindorfer, and J. Weintraub. Response Surface Method for Statistical Process and Device Simulation. In *First Int. Workshop on Statistical Metrology*, Honolulu, 1996.

[SOK87] P. Sutardja, W.G. Oldham, and D.B. Kao. Modeling of Stress-Effects in Silicon Oxidation Including the Non-Linear Viscosity of Oxide. In *Int.Electron Devices Meeting*, pages 264–267, 1987.

[SS91] G. Schumicki and P. Seegebrecht. *Prozeßtechnologie*. Springer, 1991.

[SS93] E. Strasser and S. Selberherr. A General Simulation Method for Etching and Deposition Processes. In Selberherr et al. [SSS93], pages 357–360.

[SS95] E. Strasser and S. Selberherr. Algorithms and Models for Cellular Based Topography Simulation. *IEEE Trans.Computer-Aided Design*, 14(9):1104–1114, 1995.

[SSP80] S. Selberherr, A. Schütz, and H.W. Pötzl. MINIMOS—A Two-Dimensional MOS Transistor Analyzer. *IEEE Trans.Electron Devices*, ED-27(8):1540–1550, 1980.

[SSS93] S. Selberherr, H. Stippel, and E. Strasser, editors. *Simulation of Semiconductor Devices and Processes*, volume 5, Wien, 1993. Springer.

[STA$^+$95] H. Sato, K. Tsuneno, K. Aoyama, T. Nakamura, H. Kunitomo, and H. Masuda. A New Hierarchical RSM for TCAD-based Device Design to Predict CMOS Development. In *Proc. IEEE 1995 Int. Conf. on Microelectronic Test Structures*, volume 8, pages 299–302, March 1995.

[Ste90a] G.L. Steele. *Common LISP – The Language*. Digital Press, second edition, 1990.

[Ste90b] R.W. Stevens. *UNIX Network Programming*. Prentice-Hall, 1990.

[STHN91] E.W. Scheckler, K.K.H. Toh, D.M. Hoffstetter, and A.R. Neureuther. 3D Lithography, Etching, and Deposition Simulation (SAMPLE-3D). In *1991 Symposium on VLSI Technology*, pages 97–98, Oiso, Japan, 1991.

[Sti93] H. Stippel. *Simulation der Ionen-Implantation*. Dissertation, Technische Universität Wien, 1993.

[Str94] E. Strasser. *Simulation von Topographieprozessen in der Halbleiterfertigung*. Dissertation, Technische Universität Wien, 1994.

[Str96] R. Strasser. Object-Oriented GUI Generator, December 1996. Internal document, Institute for Microelectronics, TU Vienna.

[SWH+96] Z.H. Sahul, K.C. Wang, Z.-K. Hsiau, E.W. McKenna, and R.W. Dutton. Heterogeneous Process Simulation Tool Integration. *IEEE Trans.Semiconductor Manufacturing*, 9(1):35–48, 1996.

[Sze88] S.M. Sze, editor. *VLSI Technology*. McGraw-Hill, second edition, 1988.

[Tec95] Technology Modeling Associates, Inc., Palo Alto, CA. *TMA TSUPREM-4, Two-Dimensional Process Simulation Program, Version 6.2*, June 1995.

[Tic85] W.F. Tichy. RCS — A System for Version Control. *Software - Practice and Experience*, 15(7):637–654, July 1985.

[Toh88] K.K.H. Toh. Two-Dimensional Images with Effects of Lens Aberrations in Optical Lithography. Master's thesis, University of California, Berkeley, May 1988.

[Tup96] W. Tuppa. *VMake – A CASE-Oriented Configuration Management Utility*. Dissertation, Technische Universität Wien, October 1996.

[Web81] *Webster's Third New International Dictionary of the English Language*. Merriam, 1981.

[WH89] P.H. Winston and B.K.P. Horn. *Lisp*. Addison-Wesley, 1989.

[Wim93] K. Wimmer. *Two-Dimensional Nonplanar Process Simulation*. Dissertation, Technische Universität Wien, 1993.

[WKKS93] D.M.H. Walker, J.K. Kibarian, Ch.S. Kellen, and A.J. Strojwas. A TCAD Framework for Development and Manufacturing. In Fasching et al. [FHS93], pages 83–112.

[WMF96] D. Widmann, H. Mader, and H. Friedrich. *Technologie hochintegrierter Schaltungen*, volume 19 of *Halbleiter-Elektronik*. Springer, Berlin, 1996.

[WN91] A.S. Wong and A.R. Neureuther. The Intertool Profile Interchange Format: A Technology CAD Environment Approach. *IEEE Trans.Computer-Aided Design*, 10(9):1157–1162, September 1991.

[WN95] A.K. Wong and A.R. Neureuther. Rigorous Three-Dimensional Time-Domain Finite-Difference Electromagnetic Simulation for Photolithographic Applications. *IEEE Trans.Semiconductor Manufacturing*, 8(4):419–431, 1995.

[Wol90] S. Wolf. *Process Integration*, volume 2 of *Silicon Processing for the VLSI Era*. Lattice Press, Sunset Beach, California, 1990.

[Wol95a] S. Wolf. [Wol95c], chapter 3.4.

[Wol95b] S. Wolf. [Wol95c], chapter 5.2.

[Wol95c] S. Wolf. *The Submicron MOSFET*, volume 3 of *Silicon Processing for the VLSI Era*. Lattice Press, Sunset Beach, California, 1995.

[WR87] C. B. Williams and L. A. Rowe. The Berkeley Process-Flow Language: Reference Document. Technical Report UCB/ERL M87/73, University of California, Berkeley, October 1987.

[WT86] S. Wolf and R. N. Tauber. *Process Technology*, volume 1 of *Silicon Processing for the VLSI Era*. Lattice Press, Sunset Beach, California, 1986.

[YGM94] R.W. Yuen, B. Gogoi, and C.H. Mastrangelo. Toward the Automatic Synthesis of Process Flows for Semiconductor Devices. In *Int. Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits NUPAD V*, pages 117–120, Honolulu, 1994.

[†] All PhD theses from the Institute for Microelectronics are available on the World Wide Web at http://www.iue.tuwien.ac.at/. At the same address, links to the VISTA Documentation are provided.

List of Publications

[1] Ch. Pichler and S. Selberherr. Rapid Semiconductor Process Design within the VISTA Framework: Integration of Simulation Tools. In M.H. Hamza, editor, *Proceedings of the IASTED International Conference*, Modelling and Simulation, pages 147–150, Pittsburgh, PA, USA, 1993. The International Association of Science and Technology for Development.

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

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

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

[5] S. Halama, F. Fasching, C. Fischer, H. Kosina, E. Leitner, P. Lindorfer, Ch. Pichler, H. Pimingstorfer, H. Puchner, G. Rieger, G. Schrom, T. Simlinger, M. Stiftinger, H. Stippel, E. Strasser, W. Tuppa, K. Wimmer, and S. Selberherr. The Viennese Integrated System for Technology CAD Applications. *Microelectronics Journal*, 26(2/3):137–158, 1995.

[6] S. Halama, Ch. Pichler, G. Rieger, G. Schrom, T. Simlinger, and S. Selberherr. VISTA—User Interface, Task Level, and Tool Integration. *IEEE Trans.Computer-Aided Design*, 14(10):1208–1222, 1995.

[7] Ch. Pichler, N. Khalil, G. Schrom, and S. Selberherr. TCAD Optimization Based on Task-Level Framework Services. In H. Ryssel and P. Pichler, editors, *Simulation of Semiconductor Devices and Processes*, volume 6, pages 70–73, Wien, 1995. Springer.

[8] G. Schrom, D. Liu, C. Fischer, Ch. Pichler, Ch. Svensson, and S. Selberherr. VLSI Performance Analysis Method for Low-Voltage Circuit Operation. In G.L. Baldwin, Z. Li, C.C. Tsai, and J. Zhang, editors, *Fourth Int. Conf. on Solid-State and Integrated-Circuit Technology*, pages 328–330, Beijing, China, 1995.

[9] S. Selberherr, C. Fischer, S. Halama, C. Pichler, G. Rieger, G. Schrom, and T. Simlinger. The IC Processes of the Future. In E.J.P. Santos and G.A.S. Machado, editors, *Proc. IV Brazilian Microelectronics School*, volume I, pages 87–114, Recife, January 1995.

[10] H. Stippel, E. Leitner, Ch. Pichler, H. Puchner, E. Strasser, and S. Selberherr. Process Simulation for the 1990s. *Microelectronics Journal*, 26(2/3):203–215, 1995.

[11] C. Pichler, R. Plasun, R. Strasser, and S. Selberherr. Simulation Environment for Semiconductor Technology Analysis. In *1996 International Conference on Simulation of Semiconductor Processes and Devices*, pages 147–148, Tokyo, Japan, 1996. Business Center for Academic Societies Japan.

[12] G. Schrom, Ch. Pichler, T. Simlinger, and S. Selberherr. On the Lower Bounds of CMOS Supply Voltage. *Solid-State Electron.*, 39(4):425–430, 1996.

[13] S. Selberherr, C. Fischer, S. Halama, Ch. Pichler, G. Rieger, G. Schrom, and T. Simlinger. Device structures and device simulation techniques. In G.A.S. Machado, editor, *Low-Power HF Microelectronics A Unified Approach*, chapter 2, pages 75–83. IEE London, 1996.

[14] Ch. Pichler, R. Plasun, R. Strasser, and S. Selberherr. Simulation of complete VLSI Processes with Heterogeneous Simulation Tools. Submitted to *IEEE Trans. Computer-Aided Design.*

[15] Ch. Pichler, R. Plasun, R. Strasser, and S. Selberherr. High-Level TCAD Task Representation and Automation. Submitted to *IEEE Trans. Semiconductor Technology Modeling and Simulation.*

Index

Curriculum Vitae

[redacted]	[redacted]
June 1984	High School Graduation, Schottengymnasium, Vienna
October 1984 – May 1985	Military Service
October 1985	Entered course of Electrical Engineering at the Technical University of Vienna
	Summer Positions:
1985	AVL Graz (medical engineering)
1986	AEG Vienna (industrial process engineering)
1987	AKG Vienna (digital audio systems)
1988	Summer School of the University of Vienna, Strobl (international law and politics)
1990	ABB Baden, Switzerland (power electronics)
June 1991	Graduation, Industrial Electronics and Control Engineering (with honors)
November 1991 – May 1992	Visiting Scholar at the Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
November 1992	Entered PhD. program at the Department of Microelectronics, Technical University of Vienna
January 1995 – May 1995	Visiting Researcher at the University of California, Berkeley, California, USA
September 1995 – November 1995	Visiting Researcher at Hitachi Device Development Center, Ome, Japan
September 1996 – November 1996	Visiting Researcher at Philips NatLab, Eindhoven, The Netherlands