

DISSERTATION

Small-Signal Device and Circuit Simulation

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

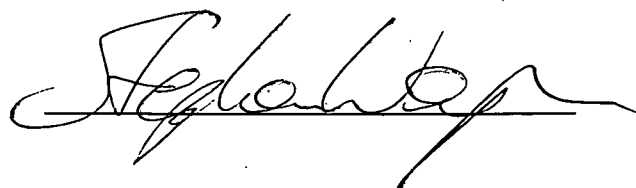
eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
von

STEPHAN WAGNER

Gauermannngasse 3
A-3003 Gablitz, Österreich

Matr. Nr. 9525168
geboren am 15. Oktober 1976 in Wien

Wien, im April 2005

A handwritten signature in black ink, appearing to read 'Stephan Wagner', written over a horizontal line.

Kurzfassung

Die Entwicklung der letzten Jahrzehnte hat gezeigt, dass die numerische Bauelementsimulation einen wichtigen Beitrag zur Charakterisierung von Halbleiterstrukturen leistet. Der wirtschaftliche Vorteil des Einsatzes von modernen Simulationsprogrammen ist enorm, da Resultate von aufwendigen und teuren Experimenten durch numerische Berechnungen vorausgesagt werden können. Darüberhinaus können diese Resultate für die Optimierung der Strukturen herangezogen werden, wodurch nur die vielversprechendsten Varianten hinsichtlich zuvor bestimmter Kenngrößen tatsächlich hergestellt werden brauchen. Neben diesen Vorteilen führte auch die breite Verfügbarkeit und großzügigere Speicherausstattung eines durchschnittlichen Computers heutzutage zu einer stark vermehrten Anwendung dieser Simulationsprogramme.

Um die Voraussetzungen für die Simulation von modernen Halbleiterbauelementen zu erfüllen, müssen die Simulationsprogramme stets erweitert werden, um alle notwendigen physikalischen Effekte berücksichtigen zu können. Neben immer genaueren und aufwendigeren Modellen werden auch neue Simulationsmodi benötigt, um die Menge an berechenbaren Kenngrößen zu erweitern. Im Zuge dieser Arbeit wurden die Anforderungen an eine numerische Kleinsignalanalyse definiert und die zugrundeliegenden Konzepte in den allgemeinen Bauelement- und Schaltungssimulator MINIMOS-NT implementiert.

Nach einer kurzen Einleitung über die Motivation dieser Arbeit und einen Überblick über die derzeitige Marktsituation der Bauelemente und Simulatoren betreffend, wird das analytische Problem der Simulationen hergeleitet. Für stark miniaturisierte Feldeffekt-Bauelemente verliert das wegen seiner numerischen Robustheit nachwievor sehr stark eingesetzte Drift-Diffusionsmodell für den Ladungsträgertransport an Genauigkeit, weshalb erweiterte Transportmodelle, die mehr als zwei Momente berücksichtigen, herangezogen werden sollten. Die betreffenden Systeme nichtlinearer partieller Differentialgleichungen werden zusammen mit den Systemen für die Kleinsignalanalyse hergeleitet. Danach werden die implementierten neuen Fähigkeiten des Simulators erklärt und anhand von typischen Simulationaufgaben vorgestellt.

Alle Fähigkeiten werden in einem eigenen Kapitel mit Beispielen eingesetzt, um moderne Bauelemente zu charakterisieren, zum Beispiel InGaP/GaAs und SiGe Heterostrukturbipolartransistoren, einen SiC MESFET sowie Doppelgate-Feldeffekttransistoren. Alle Fähigkeiten zur Simulation von Schaltungen werden anhand eines *Colpitts*-Oszillatorbeispiels gezeigt.

Da der gewählte Kleinsignalansatz direkt im Frequenzbereich arbeitet, muss pro Frequenzschritt ein komplexwertiges Gleichungssystem gelöst werden. Da zuvor nur reellwertige Module zur Assemblierung und Lösung von Gleichungssystemen zur Verfügung standen, wurde beschlossen, diese durch neue Module zu ersetzen, welche sowohl reell-, als auch komplexwertige Systeme behandeln können. Darüberhinaus bieten die neuen Module auch eine neue simulatorunabhängige Programmierungsschnittstelle, die es erlaubt, einfach und benutzerfreundlich beliebige Gleichungssysteme zu assemblieren und lösen.

Um von der fortschreitenden Entwicklung von linearen Gleichungslösern zu profitieren, wurde das Lösermodul um eine Schnittstelle zu externen Modulen erweitert. Mehrere Module mit Vorteilen für bestimmte Simulationen wurden bereits angebunden. Zur Analyse dieser Vorteile wurden alle internen und externen Löser einer Beurteilung anhand von zahlreichen Simulationsbeispielen unterzogen, deren Ergebnisse es erlaubten, automatische Hierarchien aus mehreren Gleichungslösern in den Simulatoren zu entwickeln.

Abstract

During the last decades numerical device simulation has proven to be invaluable for characterizing semiconductor devices. The economic impact is enormous because results of expensive experiments can be predicted by employing one or more simulation tools. Furthermore, the results can be used to optimize the investigated device structures and to single out unpromising variations in advance. The application of simulation tools has been significantly intensified because of the computational power and the available memory in today's average computers.

To meet the requirements for the simulation of advanced devices, ongoing effort is put into extension of these tools by implementations of state-of-the-art descriptions of all relevant physical effects. Besides of that modeling the simulators have also to be extended by new simulation modes. In course of this work, the requirements for small-signal simulations have been identified and new features have been added to the general-purpose device and circuit simulator MINIMOS-NT.

After a short introduction including a motivation and overview of the current market situation, the analytical problem of the respective simulations is derived. As advanced device structures cannot be correctly simulated by the drift-diffusion transport model any more, higher-order transport models such as four- and six moments models are also taken into consideration. In addition, the small-signal systems are derived which are based on the sinusoidal steady-state approach. Afterwards, the various new capabilities of the simulator are presented and applied for typical simulation tasks.

All features have been used to characterize advanced devices, such as InGaP/GaAs and SiGe HBTs, a wide-bandgap SiC MESFET, and double gate MOSFETs. The latter require higher-order transport models in order to accurately extract the steady-state and small-signal device quantities. Furthermore, the mixed-mode small-signal features have been used to simulate a *Colpitts* oscillator.

Since the small-signal simulation mode is directly based in the frequency domain, the solution of one complex-valued equation system per frequency step is required. For that reason, the numerical core modules have been extended to handle both real-valued and complex-valued quantities. The application programming interface of the new modules allows an efficient and user-friendly assembling and solving of linear equation systems.

In order to profit from new developments of mathematical code, the solver module has been extended by an interface to external solvers. Several promising linear solvers which have particular strengths for different kinds of simulations have been coupled to the solver module. In the course of a subsequent performance evaluation, these strengths have been identified and efficient solver hierarchies have been constructed.

Danksagung

Für die Betreuung und die Zusammenarbeit während der Erstellung dieser Arbeit sowie für die Organisation der finanziellen Unterstützung in dieser Zeit möchte ich mich bei Herrn Prof. Dr. Siegfried Selberherr und Herrn Prof. Dr. Tibor Grasser bedanken. Mein Dank vor allem finanzieller Hinsicht gilt der *Christian Doppler Gesellschaft* sowie unseren Industriepartnern *austriamicrosystems AG* in Unterpremstätten und *Infineon Technologies AG* in Villach.

Herrn Prof. Dr. Emmerich Bertagnolli bin ich zu Dank verpflichtet, der sich relativ kurzfristig bereit erklärte, als Zweitprüfer zur Verfügung zu stehen. Stellvertretend für die ausgezeichnete Administration bedanke ich mich bei unserem Institutsvorstand Herrn Prof. Dr. Erasmus Langer. Für die Aufnahme und Unterstützung während meines interessanten Aufenthalts bei *Texas Instruments, Inc.* in Dallas, Texas, bin ich darüberhinaus Herrn Dr. Christoph Wasshuber zu Dank verpflichtet.

Für die anfängliche, sehr intensive Zusammenarbeit möchte ich mich bei Herrn Dr. Vassil Palankovski bedanken. Mein besonderer Dank gilt natürlich auch Herrn Dipl.-Ing. Wilfried Wessner für die Zusammenarbeit bei verschiedensten Projekten. An die steten Diskussionen mit meinen zahlreichen Zimmerkollegen über alltägliche Probleme, die die wichtige Wahl des Restaurants für das Mittagessen einschloss, werde ich mich immer gerne erinnern.

Anstatt mit einer langen Liste von Kollegen fortzusetzen, mit denen ich in den vergangenen Jahren zusammenarbeiten und das eine oder andere Problem besprechen durfte, möchte ich allgemein feststellen, dass die Zeit am Institut für mich gerade wegen dieser interessanten Gruppe eine sehr fruchtbare gewesen ist. Natürlich trägt dazu auch ein persönlicher Freiraum bei, den ich sehr zu schätzen gewusst habe, doch ein gutes Arbeitsklima mit ins Private reichenden Freundschaften ist ungeachtet dessen ein wichtiger Bestandteil erfolgreicher Arbeit. In diesem Sinne bedanke ich mich für die zahlreichen Nachbereitungen nach anstrengenden Arbeitstagen, die durchaus auch den Charakter eines privaten Treffens erreicht haben könnten.

Auf diesem Wege möchte ich aber auch nicht unterlassen, meiner langjährigen Freundin Sandra, meinen Eltern und Großeltern, meiner Schwester sowie meinem Schwager für so manche Anregung und moralische Unterstützung in den verschiedensten Zeiten des Dissertationsstudiums zu danken.

Contents

1	Introduction	1
1.1	Radio Frequency	2
1.2	Devices for RF Applications	3
1.2.1	Devices Based on the III-V Material System	3
1.2.2	Devices Based on Silicon	4
1.3	Numerical Simulators	5
1.3.1	The Commercial Simulators	6
1.3.2	The Device and Circuit Simulator Minimos-NT	7
2	Device Simulation	8
2.1	The Analytical Problem	9
2.1.1	The Drift-Diffusion Transport Model	10
2.1.2	Types of Partial Differential Equations	11
2.1.3	Systematic Derivation of the Macroscopic Transport Models	12
2.1.4	The Six Moments Transport Model	15
2.1.5	The Energy-Transport Model	16
2.1.6	The Drift-Diffusion Transport Model	16
2.2	The Discretized Problem	16
2.2.1	Interface Conditions	18
2.2.2	Boundary Conditions	20
2.3	Steady-State and Transient Analysis	21
2.3.1	Solving the Nonlinear System	21
2.3.2	Transient Simulation	24
2.4	Small-Signal Simulation	24
2.4.1	Equivalent Circuits	25
2.4.2	Sinusoidal Steady-State Analysis	25
2.5	Derivation of the Complex-Valued Small-Signal System	26
2.5.1	Boundary Conditions and the Complete System	28
2.5.2	Extension for Higher-Order Transport Models	28

2.6	Concluding Remarks	29
2.6.1	Harmonic Balance	30
2.6.2	Complex-Valued Equation Systems	31
3	Small-Signal AC Analysis	32
3.1	Introduction	32
3.1.1	Admittance, Impedance, Hybrid Matrices and Parameters	33
3.1.2	Scattering Parameters	34
3.1.3	Extraction of the Cut-Off and Maximum Oscillation Frequency	36
3.2	Overview of the Minimos-NT Small-Signal Capabilities	38
3.3	Standard Single-Mode AC Analysis	39
3.3.1	Introduction	39
3.3.2	Simulation Example	39
3.3.3	Extraction of the Cut-Off Frequency	42
3.4	Extended Single-Mode AC Analysis	44
3.4.1	Capacitance Matrix	45
3.4.2	Simulation Example	45
3.5	Transformation to Extrinsic Parameters	47
3.6	Small-Signal Capabilities for Mixed-Mode Device/Circuit Simulations	48
3.6.1	Mixed-Mode Simulation	48
3.6.2	The Nodal Approach and Modified Nodal Approach	49
3.6.3	Two-Level Newton and Full Newton Methods	50
3.6.4	Iteration Schemes	50
3.6.5	The Mixed-Mode AC Capabilities	51
3.7	Concluding Remarks	51
4	The Assembly Module	52
4.1	Key Demands on the Assembly Module	52
4.2	Approaches to Meet these Demands	53
4.2.1	Third Party Modules and Packages	53
4.2.2	A New Generally Applicable Assembly Module	55
4.3	Refined Key Demands on the Assembly Module	56
4.4	Condition of the Linear System	58
4.5	The Parameter Administration	59
4.6	Assembling the Complete Linear Equation System	59
4.7	Compiling the Complete Linear Equation System	61
4.7.1	Row Transformation	62
4.7.2	Variable Transformation	62

4.8	The Pre-Elimination	63
4.9	Sorting the Inner Linear Equation System	64
4.10	Scaling the Inner Linear Equation System	66
4.11	Solving and Back-Substitution	67
4.12	Newton Adjustment	68
4.12.1	The Administration Scheme	69
4.12.2	Newton Adjustment Levels	70
4.12.3	Improved Sorting Feature	70
4.13	The Transferred-Transformation Problem	71
4.13.1	Mathematics	73
4.13.2	The Basic Correction Algorithm	73
4.13.3	The Advanced Algorithm	75
4.14	Concluding Remarks	77
5	The Solver Module	79
5.1	Third Party Modules and Libraries	80
5.1.1	Commercial Libraries	80
5.1.2	Libraries of Hardware Vendors	80
5.1.3	Recent Developments	81
5.1.4	Solver Frameworks	82
5.1.5	Special-Purpose Libraries	83
5.1.6	Discussion	83
5.2	Overview of the Solver Module	84
5.2.1	Solver Selection	85
5.2.2	In-House Direct Solvers	86
5.2.3	In-House Iterative Solvers	86
5.2.4	The Generalized Minimal Residual Method	87
5.2.5	Preconditioner	87
5.3	External Solvers	89
5.3.1	Parallelization Issues	89
5.3.2	Pardiso	91
5.3.3	Algebraic Multigrid Methods for Systems	92
5.4	Solver Hierarchy	93
5.5	Practical Evaluation of the Solvers	93
5.5.1	Test Examples and Processing	94
5.5.2	General Quantitative Comparisons	94
5.5.3	Simulation Results of the GMRES(m) Evaluation	96

5.5.4	Simulation Results of the Performance Evaluation	100
5.5.5	Evaluation of Solvers for Higher-Order Transport Models	101
5.6	Concluding Remarks	103
6	Examples	104
6.1	Simulation of an InGaP/GaAs Heterojunction Bipolar Transistor	104
6.2	Simulation of a SiGe Heterojunction Bipolar Transistor	106
6.3	Simulation of a 4H-SiC MESFET	109
6.4	Oscillator	111
6.4.1	Amplifier	111
6.4.2	Amplifier with Resonant Circuit	112
6.4.3	Colpitts Oscillator Circuit	113
6.5	Simulations with Higher-Order Transport Models	114
6.5.1	Simulation Results	114
6.5.2	Conclusions	115
7	Summary and Outlook	118
A	Input-Deck Interface to the New Simulator Features	119
A.1	Activation of the Transient and Small-Signal Mode	119
A.2	Activation of the Admittance Matrix Calculation Feature	120
A.3	Simulation Setup of the Diode Example	120
A.4	Inquiring Capacitances	121
A.5	Configuration of the Two-Port Features	121
A.6	Output Functions	122
A.7	Setup for the Cut-Off Frequency Extraction	123
A.8	Setup of the Resonant Circuit and the Band Rejection Filter	125
A.9	Inquiring Complex-Valued Node Voltages and Terminal Quantities	126
A.10	Setup for the Extended Mixed-Mode AC Simulation	126
A.11	Inquiring Circuit Quantities	127
A.12	Selection of External Solvers	127
A.13	Netlist Definition of the Oscillator Example	128
B	The Stepping Module of Minimos-NT	130
B.1	The Basic Stepping Functions	131
B.2	The Special-Purpose Stepping Functions	131
B.3	Conditional Stepping	132
B.4	Stepping Control	133
B.5	Additional Stepping Control Keywords	136

C	Miscellaneous Projects	137
C.1	The Minimos-NT Post-Processing System	137
C.2	The Minimos-NT Interactive Mode	139
C.3	SEILIB - The Simulation Environment Interaction Library	141
C.3.1	Motivation	141
C.3.2	Getting Started	141
C.3.3	Basic Nomenclature and Definitions	142
C.3.4	The Argument System	144
C.3.5	Process Management System	145
C.3.6	The Host Management System	148
C.3.7	Class Diagram	148
C.3.8	Example Application: The Minimos-NT Test	149
C.3.9	The Optimization System	150
C.4	The Minimos-NT Test	152
C.4.1	Numerics	152
C.4.2	Nomenclature and Definitions	153
C.4.3	Test Levels	154
C.4.4	File Structure	154
C.4.5	Model Identification System	156
C.4.6	The References	156
D	Calculation of Additional Extrinsic Parameters	157
E	Matrix Storage Formats	159
E.1	Modified Compressed Sparse Row Matrices	159
E.2	Compressed Sparse Row - Variant 1	160
E.3	Compressed Sparse Row - Variant 2	161
E.4	Matrix Storage Format Conversion	161
	Bibliography	162
	Own Publications	176
	Curriculum Vitae	178

List of Tables

2.1	Ignored derivatives during the first steps of the <i>Newton</i> iteration.	23
3.1	Admittance matrix calculated by MINIMOS-NT for a frequency $f = 100$ MHz and terminal voltages $V_{GS} = 0.0$ V and $V_{DS} = 0.0$ V.	45
4.1	Eigenvalues of the compiled, pre-eliminated, sorted, and scaled system matrix.	68
4.2	<i>Newton</i> adjustment levels.	70
4.3	Comparison of terminal quantities with and without the correction. Note that the boundary charges are significantly different.	72
4.4	The transposed and untransposed transformation matrix before and after the correction. The column index stands for the source or the target of the transformation, respectively. . .	77
5.1	Six two-dimensional, five mixed-mode device/circuit (the number of devices is given in parenthesis), and five three-dimensional simulations were used for evaluating the solver performance. The dimension of the linear equation system, the number of non-zero entries, and the typical number of DC <i>Newton</i> iterations are given.	97
5.2	This table provides general quantitative information about the simulations using the in-house ILU-BICGSTAB on the IBM cluster. After the index of the simulation, the dimensions of the inner and complete equation system is given. The next column shows the share of the inner in respect to the complete system. In order to analyze how the simulation time is spent, the remaining columns contain the shares of the initialization, pre-processing, assembling, solving (includes all steps shown in Table 5.3), quantity update, and post-processing, respectively.	97
5.3	This table provides general quantitative information about the solving process which includes the MCSR conversion, compiling of the complete linear system, pre-elimination, sorting, scaling, preconditioning (ILU), and solving with the BICGSTAB. The simulations were started on the IBM cluster.	98
5.4	This table shows the same information as Table 5.3, but without the <i>Newton</i> adjustment. . .	98
5.5	Three two-dimensional device structures with the given gate lengths were simulated. The dimensions of the linear equation systems are given depending on the transport model used. For the drift-diffusion simulation and BICGSTAB, the CPU time as well as the share of the simulation time spent for compiling, pre-eliminating, sorting, scaling, and solving are specified.	101
A.1	Keywords provided in the Contact section.	121
A.2	Valid selections for external solver modules.	128

B.1	Stepping functions provided by MINIMOS-NT.	130
B.2	Information provided to delta stepping control algorithms.	134
B.3	Information expected from delta stepping control algorithms.	134
B.4	Information provided to conditional stepping control algorithms.	135
B.5	Information expected from conditional stepping control algorithms.	135
B.6	General stepping control keywords.	136
C.1	List of all <i>special-purpose arguments</i> , which are automatically generated by the library. . .	146
C.2	Definition of the test levels.	154

List of Figures

2.1	Box i with six neighbors	18
2.2	Splitting of interface points: Interface points as given in a) are split into three different points having the same geometrical coordinates b)	19
2.3	The complete equations are a combination of the boundary and the segment system. This combination is controlled by the transformation matrix and depends on the interface type. In the upper figure, the case for <i>Dirichlet</i> boundary conditions including substitute equations is shown, in the lower figure for all other cases.	20
2.4	Comparison of transient and frequency-domain-based approaches [233]. The dashed rectangles of the S ³ A approach symbolize complex-valued equation systems, the other real-valued ones.	25
2.5	These figures [160] show a standard Π -type small-signal equivalent circuit of a HEMT (left) and a T-type eight-element small-signal equivalent circuit of an HBT. The dashed rectangles denote the intrinsic devices, the terminal resistances can be additionally included in the simulation.	26
2.6	Comparison of small-signal and large-signal simulations. In the case of too high RF power, harmonics are generated within the non-linear devices. These additional voltage and current vectors cannot be taken into account by linearized small-signal approximations [45].	30
3.1	Voltages and currents of a two-port device/network.	33
3.2	Traveling waves at a two-port device/network.	34
3.3	Definition of S-parameters.	35
3.4	Complete f_T curve of a bipolar junction transistor (left). Slope of the absolute value of the short circuit current gain and the cut-off frequency at the unity gain point at $I_C = 0.865$ mA (right).	37
3.5	Overview of the MINIMOS-NT small-signal capabilities.	38
3.6	Simple diode structure under investigation. The boron concentration in the p-doped half on the left is $1 \times 10^{17} \text{ cm}^{-3}$ equal to the phosphorus concentration in the n-doped part of the diode.	40
3.7	The left figure shows the results of the transient simulations for the three amplitudes $A = 10$ mV, $A = 250$ mV, and $A = 1$ V. The small figures depict the results of the <i>Fast Fourier Transformation</i> of the respective cathode currents.	40

3.8	The upper left figure compares the small-signal results of MINIMOS-NT and DESSIS. The other three figures compare the small-signal results of MINIMOS-NT with its transient results: In the upper right figure the capacitance versus the frequency is shown. The argument versus the frequency is given in the lower left figure. In both figures, different number of periods (P) and number of steps per period (S) are compared. Finally, the dependence of the relative errors and the time scaled to the small-signal result (time ratio) on the number of transient steps is illustrated in the lower right figure. The number of periods is 2, the frequency is 1 MHz and the transient data is compared with the small-signal results. The trade-off between the reduction of the relative errors and the increasing computational effort can be clearly seen.	41
3.9	These figures analyze the effort for the extraction of f_T by conditional stepping. On the left side, two different sets of lower and upper boundaries are compared for an error value of 10^{-3} . Whereas for the wide boundaries of 10 GHz and 100 GHz 24 steps are required, the narrow boundaries of 30 GHz and 40 GHz reduce this effort down to five steps. On the right side, the number of steps depending on the narrowness of the boundaries for different errors are shown.	43
3.10	In the left figure, the three transport models are compared with quasi-static simulation results of MINIMOS-NT as well as with Monte Carlo data. The right figure shows the cut-off frequency depending on the gate voltage. Whereas for larger devices, the difference is again minimal, the superiority of the six moments transport model for smaller devices can be clearly seen.	44
3.11	Part of the investigated device structure with a depth of $1\text{ }\mu\text{m}$	46
3.12	Gate drain capacitance versus gate voltage at $V_D = 0\text{ V}$: comparison of simulation results of MINIMOS-NT and DESSIS.	46
3.13	Two-Port parasitic equivalent circuit.	47
3.14	Results of mixed-mode AC simulations with compact models only: the resonant circuit on the left side and the band rejection filter on the right side.	51
4.1	All equations marked for pre-elimination (*) are moved to the outer system matrix, the others remain in the inner one [65, 228].	64
4.2	On the left the completely compiled system matrix of a discretized two-dimensional MOS transistor structure assembled by MINIMOS-NT is shown. The magnitude of the entries are encoded by the colors according to the legend in the right. Some regions with problematic equations are indicated by red dashed rectangles. After the pre-elimination, the inner system matrix (right) does not contain the problematic equations any more. The dimension is not significantly reduced since the majority of equations is not affected. . . .	65
4.3	In comparison to the pre-eliminated structure, the reordering algorithm significantly reduces the bandwidth from 2,867 to 102 in order to reduce the factorization fill-in (left). The circle indicates the range of the cut-out of the scaled matrix shown in the right figure. The scaled inner system matrix has diagonal entries equal unity, which is demonstrated by the red color. Since only the values are changed, no structural difference can be seen in comparison to the sorted matrix.	66
4.4	Schematic assembly overview.	67
4.5	Former (left), refined (center), and final (right) administrative scheme.	69
4.6	Transformations involved if three segments share one physical grid point.	71

4.7	Graphical representation of the multiple transfer problem.	76
4.8	Comparison of the one-phase (upper) and four-phases (lower) approach. In the latter case the implementation of the simulator is much more complicated as the assembly module requires a specific assembling sequence. Whereas in the one-phase approach the loop over all models is processed only once, it has to be processed four times in the four-phases approach. In addition it is necessary to call specific preparation functions and each model implementation has to take the current phase into account, which leads to complicated codes.	78
5.1	Schematic overview of the linear modules [230].	85
5.2	The hierarchical concept [65, 228].	88
5.3	Concept of single- (left) and multi-threaded (right) stepping algorithm. Whereas the former calculates all 55 steps subsequently, the latter uses five processes to calculates eleven steps each.	90
5.4	Comparison of a canonical shared memory (left) and message passing non-shared memory (right) architectures [34].	91
5.5	User time, number of operations (left), and memory consumption ratios (right) depending on the GMRES(M) restart factor m for the two-dimensional (above), three-dimensional (center), and mixed-mode simulations (below).	99
5.6	Solving times (average, minimum, maximum) on the <i>Intel</i> computer. All times are scaled to the in-house ILU-BICGSTAB in the center.	100
5.7	In the left figure, the relative PARDISO real/wall clock times (average, minimum, maximum) versus the number of processors/threads on the IBM cluster. The user time in the right figure increases due to the parallelization overhead.	101
5.8	Scaled results for the six moments transport models (upper figure). The lower figures show scaled results for the energy-transport (left) and the drift-diffusion (right) transport models.	102
6.1	Simulated device structure together with pad parasitics used for S-parameter calculation [161].	104
6.2	Comparison of simulated and measured AC collector current i_C over AC input power P_{IN} (left). Comparison of simulated and measured AC output power P_{OUT} over AC input power P_{IN} (right).	105
6.3	S-parameters in a combined <i>Smith</i> /polar chart with a radius of one from 50 MHz to 10 GHz at $V_{CE} = 3$ V, $J_C = 2$ kA/cm ² (upper left), $J_C = 8$ kA/cm ² (upper right), and $J_C = 15$ kA/cm ² (lower left). In the lower right figure the short-circuit current gain and matched gain versus frequency at $J_C = 15$ kA/cm ² is shown.	106
6.4	Active antimony concentration of the investigated SiGe Heterojunction Bipolar Transistor (large device).	107
6.5	Comparison of simulated and measured forward <i>Gummel</i> plots at $V_{CE} = 1$ V.	108
6.6	The figures compare S-parameters in a combined <i>Smith</i> /polar chart with a radius of one from 50 MHz to 31 GHz at $V_{CE} = 1$ V for $J_C = 28$ kA/cm ² (left) and $J_C = 76$ kA/cm ² (right) for a large device structure and a small one embedded in a circuit.	108
6.7	The cut-off frequency f_T versus collector current I_C at $V_{CE} = 1$ V (left) and the short-circuit current gain versus frequency (right) is depicted [233].	108

6.8	On the left, the cross section of a MESFET in SiC is shown and on the right a comparison of measured and simulated DC IV characteristics [12].	109
6.9	Comparison of measured and simulated S-parameters in a combined <i>Smith</i> /polar chart with a radius of one (left). Small-signal current and power gain (right) [12].	110
6.10	The three subcircuits which are used in the example circuits are shown on the left side. They are parts of the three circuits depicted on the right [231].	111
6.11	Result of transient simulation of the amplifier circuit with $V_{ac} = 10$ mV and $f = 2.4$ GHz [231].	112
6.12	Results of small-signal simulations of the resonant circuit: absolute value (left) and argument (right). The results are compared with ADS simulations using a VBIC95 model of a similar transistor [231].	112
6.13	Result of the transient simulation of the oscillator: output V_{pin2} in the initial phase (left) and in the state of equilibrium (right) [231].	113
6.14	Structure of the simulated double-gate MOSFET devices. The gate length is varied from 250 nm down to 25 nm [83].	114
6.15	These four figures show the increasing errors of the macroscopic transport models with decreasing gate lengths. Whereas at $L_g = 250$ nm in the upper left figure the difference of the drain currents is minimal, it can be clearly seen that for $L_g = 50$ nm the six moments transport model delivers the best results. However, for extremely small gate length, it loses its advantages and even more moments would be necessary. Note that the drift-diffusion model results in terminal quantities which underestimates the Monte Carlo results.	116
6.16	These four figures show the cut-off frequency versus the drain current and the much higher sensitivity of that small-signal figure of merit is demonstrated.	117
C.1	SEILIB class diagram.	149
C.2	Class diagram of the MINIMOS-NT test system based on the SEILIB library.	150
C.3	Class diagram of the SEILIB optimization system.	151

List of Symbols and Acronyms

A	... System matrix of a linear equation system
\tilde{A}	... System matrix of the complete linear equation system
A_i	... System matrix of the inner equation system
A_b	... System matrix of the boundary system
A_s	... System matrix of the segment system
BV_{CEO}	... Collector breakdown voltage
β_n	... Electron kurtosis
β_p	... Hole kurtosis
b	... Right-hand-side vector of a linear equation system
\tilde{b}	... Right-hand-side vector of the complete linear equation system
b_i	... Right-hand-side vector of the inner equation system
b_{avg}	... Average bandwidth
\underline{b}	... Complex-valued right-hand-side vector of the small-signal system
b_I	... Imaginary part of the complex-valued right hand side vector
b_R	... Real part of the complex-valued right-hand-side vector
b_b	... Right-hand-side vector of the boundary system
b_s	... Right-hand-side vector of the segment system
c_L	... Heat capacity
C_{ij}	... Capacitance between ports i and j
D_n	... Electron diffusion coefficient
f_{max}	... Maximum oscillation frequency
f_T	... Cut-off or transit frequency
f_{op}	... Operating frequency
$F_{x_i,j}$... Flux between points i and j
g_m	... Matched gain
G_i	... Source term
i_B	... Base Current
i_C	... Collector Current
J	... <i>Jacobian</i> matrix
J_C	... Density of the collector current
κ_L	... Thermal conductivity
κ_s	... Spectral condition number
k_B	... <i>Boltzmann's</i> constant
L	... Lower triangular matrix
L_g	... Gate Length
λ_{min}	... Smallest eigenvalue
λ_{max}	... Largest eigenvalue
μ_n	... Electron mobility
μ_p	... Hole mobility

M	... Preconditioner
n	... Dimension of a linear equation system
n	... Electron concentration
P_{IN}	... Input power
P_{OUT}	... Output power
ψ	... Electrostatic potential
ψ_{C}	... Contact potential
p	... Hole concentration
q	... Elementary charge
ρ	... Space charge density
ρ_{L}	... Mass density
R	... Recombination rate
σ_n	... Electron conductivity
σ_p	... Hole conductivity
\mathbf{S}_{c}	... Scaling vector (diagonal matrix) for columns
\mathbf{S}_{r}	... Scaling vector (diagonal matrix) for rows
\mathbf{T}_{b}	... Transformation matrix of boundary conditions
\mathbf{T}_{v}	... Transformation matrix of variables
T_{L}	... Lattice temperature
T_n	... Electron temperature
T_p	... Hole temperature
U	... Upper triangular matrix
v	... Variable vector
V	... Velocity
V_{sat}	... Saturation velocity
V_{CE}	... Collector Emitter Voltage
V_{DS}	... Drain Source Voltage
V_{GS}	... Gate Source Voltage
W_{g}	... Gate Width
x	... Solution vector of a linear equation system
$\tilde{\mathbf{x}}$... Solution vector of the complete linear equation system
\mathbf{b}_i	... Solution of the inner equation system
$\underline{\mathbf{x}}$... Complex-valued solution vector of the small-signal system
\mathbf{x}_{I}	... Imaginary part of the complex-valued solution vector
\mathbf{x}_{R}	... Real part of the complex-valued solution vector
y	... Forward substitution vector
AC	... Alternating current
ACML	... AMD Core Math Library
AlAs	... Aluminum Arsenide
A-parameter/matrix	... Chain parameter/matrix (also ABCD)
ADS	... Advanced Design System
AINV	... Approximate Inverse (Preconditioner)
AIX	... Advanced Interactive eXecutive
AMG	... Algebraic Multi-Grid
AM	... Amplitude Modulation
AMD	... Advanced Micro Devices
API	... Application Programming Interface
ARPACK	... <i>Arnoldi</i> Package

ASCII	... American Standard Code for Information Interchange
ATLAS	... Automatically Tuned Linear Algebra Software
BICG	... Bi-Conjugate Gradient
BICGSTAB	... Bi-Conjugate Gradients Stabilized
BiCMOS	... Bipolar Complementary Metal Oxide Semiconductor
BJT	... Bipolar Junction Transistor
BLAS	... Basic Linear Algebra Subprograms
BSIM	... Berkeley Short-Channel IGFET Model
CB	... Consumer Band
CGS	... Conjugate Gradient Squared
CG	... Conjugate Gradient
CMOS	... Complementary Metal Oxide Semiconductor
CPU	... Central Processing Unit
CSR	... Compressed Sparse Row
CVS	... Concurrent Version System
Cv	... Capacitance (C) – Voltage (V) plot
DC	... Direct current
DANSYS	... Distributed ANSYS
DESSIS	... Device Simulation for Smart Integrated Systems
EAS	... Equation Assembly System
EISPACK	... Eigensolver Package
ESSL	... Engineering and Scientific Subroutine Library
FEDOS	... Finite Element Diffusion and Oxidation Simulator
FET	... Field Effect Transistor
FFT	... Fast Fourier Transformation
FM	... Frequency Modulation
FORTRAN	... Formula Translator
GaAs	... Gallium Arsenide
GaP	... Gallium Phosphide
Gb	... Gigabit
GB	... Gigabyte
GHz	... Gigahertz
GMRES	... Generalized Minimal Residual
GNU	... GNU's not Unix
GUI	... Graphical User Interface
HBT	... Heterojunction Bipolar Transistor
HEMT	... High Electron Mobility Transistor
HF	... High Frequency
H-parameter/matrix	... Hybrid parameter/matrix
HPF	... High Performance Fortran
HSL	... Harwell Subroutine Library
IGFET	... Insulated Gate Field Effect Transistor
IBM	... International Business Machines
IEEE	... Institute of Electrical and Electronics Engineers
IISS	... Intelligent Iterative Solver Service
ILU	... Incomplete LU-factorization
IMSL	... International Mathematical and Statistical Library
InAs	... Indium Arsenide
InGaP	... Indium Gallium Phosphide
InP	... Indium Phosphide

IPL	... input-deck programming language
IR	... Infrared
ISE	... Integrated Systems Engineering
IV	... Current (I) - Voltage (V) plot
LF	... Low Frequency
LINPACK	... Linear Package
MAG	... Maximum Available Gain
MCSC	... Modified Compressed Sparse Column
MCSR	... Modified Compressed Sparse Row
MDI	... Multiple Document Interface
MESFET	... Metal Semiconductor Field Effect Transistor
MF	... Middle Frequency
MHz	... Megahertz
MODFET	... Modulation-Doped Field Effect Transistor
MOS	... Metal Oxide Semiconductor
MOSFET	... Metal Oxide Semiconductor Field Effect Transistor
MPI	... Message Passing Interfaces
MSG	... Maximum Stable Gain
MUMPS	... Multifrontal Massively Parallel Sparse Direct Solver
NAG	... Numerical Algorithms Group
NaN	... Not a Number
OPENMP	... Open Multi Processing
PARDISO	... Parallel Direct Solver
PSBLAS	... Parallel Sparse Basic Linear Algebra Subroutines
PVM	... Parallel Virtual Machines
RFIC	... Radio Frequency Integrated Circuits
RF	... Radio Frequency
SAMG	... Algebraic Multi-Grid Methods for Systems
SEILIB	... Simulation Environment Interaction Library
SIESTA	... Simulation Environment for Semiconductor Technology Analysis
SiC	... Silicon Carbide
SiGe	... Silicon Germanium
SMP	... Shared Memory Parallelization
SOI	... Silicon-On-Insulator
SOR	... Successive Over-Relaxation
SPAI	... Sparse Approximate Inverse Preconditioner
S-parameter/matrix	... Scattering parameter
SPICE	... Simulation Program with Integrated Circuit Emphasis
STL	... Standard Template Library
SSOR	... Symmetric Successive Over-Relaxation
S ³ A	... Sinusoidal Steady-State Analysis
TCAD	... Technology Computer Aided Design
UHF	... Ultra High Frequency
UMFPACK	... Unsymmetric Multi-Frontal Package
VBIC95	... Vertical Bipolar Inter-Company Model 1995
VHF	... Very High Frequency
VTK	... Visualization Toolkit
WSMP	... Watson Sparse Matrix Package
Y-parameter/matrix	... Admittance parameter/matrix
Z-parameter/matrix	... Impedance parameter/matrix

Chapter 1

Introduction

Numerical device simulation has proven to be invaluable for characterizing the specific properties of semiconductor devices under different operating conditions. Advanced simulation tools are able to extract various figures of merit and allow detailed insight into the physical processes taking place in all regions of the devices. Furthermore, they can be finally employed to optimize the device structures regarding various targets.

Due to the computational power and the available memory in today's average computers, device simulation and optimization can be rigorously employed to pursue these goals. The economical impact is enormous, considering only the cost of one test wafer in comparison to the costs of one average workstation and the maintenance of application setups. For that reason, the development of these tools, both commercially and academically, will be continued in order to provide state-of-the-art models, simulation modes, and user-interaction capabilities.

Such extended features are necessary to meet the requirements for today's advanced radio frequency (RF) device structures, which are supposed to be characterized also by efficient small-signal capabilities. The basic idea of a small-signal simulation mode is to linearize the transport equations around a steady-state operating point and apply sinusoidal contact signals. Various important figures of merit, such as S-parameters or the cut-off frequency f_T of a transistor, can be extracted by means of the small-signal simulation mode.

In this introductory chapter the term RF is discussed first, followed by an overview of state-of-the-art RF devices. In addition, the respective capabilities of commercial simulators are presented and compared with academic codes, especially with the newly implemented features of MINIMOS-NT.

The second chapter contains the derivation of the analytical problem and its discretization including the nonlinear solution technique. Since the accurate simulation of advanced MOS devices requires a higher-order transport model as a replacement for the well-known drift-diffusion model, the energy-transport and six moments model provided by MINIMOS-NT are discussed. Furthermore, the small-signal systems for all three transport models are derived. The third chapter deals with the identified concepts of all small-signal capabilities which have been implemented in MINIMOS-NT based on the chosen small-signal approach. The various figures of merit are described and their results presented.

Since the small-signal simulation mode requires the ability to handle complex-valued quantities, the topic of linear equation systems becomes important. As discussed in the fourth chapter, it has been decided to completely replace the formerly applied modules handling the real-valued equation systems assembled by the steady-state and transient simulation mode. Instead, new modules have been introduced which are able to handle both real-valued and complex-valued equation systems. As this decision raised various questions regarding the design, implementation, and application of the new modules, their solutions are a fundamental part of the discussion of a small-signal simulation mode. Thus, the fourth chapter deals

with the assembly module, whereas the fifth one discusses the solver modules. Both chapters include an overview of existing modules and the motivation behind each measure taken during assembling and solving linear equation systems.

The simulations presented in the sixth chapter finally demonstrate all of the capabilities discussed before. Different kinds of examples show how well these features can be used to perform advanced device and mixed-mode device/circuit simulations. In addition, higher-order transport models are evaluated for a set of devices. All simulation results are compared either with measurements or with reference results from other simulators.

A final summary and the outlook for future developments regarding the two main topics of this thesis are given in the last chapter. The appendices consist mostly of topics regarding the usability of the newly implemented capabilities. After the documentation of the input-deck interface to all features, the powerful and convenient stepping module of MINIMOS-NT is presented, which has been significantly extended in the course of this work. The third appendix discusses miscellaneous projects such as the interactive and post-processing mode of MINIMOS-NT, the newly designed and implemented library SEILIB for advanced and efficient processing of parameterized input-decks, and eventually one of its most prominent applications, the MINIMOS-NT test. The fourth appendix summarizes the formulae of two-port parameter conversions and the last one explains three different sparse matrix formats.

1.1 Radio Frequency

The abbreviation of *radio frequency* RF is used both as noun and as a qualifier as seen for example in RF devices. Actually, the distinction between RF and other objects is based on different historical considerations, such as the bandwidth-based, frequency-based, application-based, and size-based definition [95]:

Bandwidth-Based Definition: RF amplifiers are treated synonymously with tuned amplifiers, implicating that RF circuits are necessarily narrow-band ones with bandwidths of a small-fraction of the center frequency. This definition is not much in use any more today.

Frequency-Based Definition: RF is frequently defined as the range of electro-magnetic waves lying between the low-frequency and the microwave frequency bands, thus consisting of the three frequency bands:

- MF (Middle Frequency) refers to the frequency band between 300 kHz – 3 MHz. The waves are propagated in the troposphere and absorbed in the ionosphere. They are used for AM radio, maritime radio, radio direction finding, and emergency applications.
- HF (High Frequency): The band between 3 MHz – 30 MHz is propagated in the ionosphere and used for amateur radio, CB radio, international broadcasting, military communication, long-distance aircraft and ship communication, telecommunication.
- VHF (Very High Frequency): The frequencies between 30 MHz – 300 MHz are characterized by line-of-sight propagation and are used for VHF television, FM radio, aircraft AM radio, and aircraft navigation.

Application-Based Definition: Especially in communication system engineering, RF is distinguished by considering the role of the signals at these frequencies. RF signals were historically used as carriers rather than containing information. Whereas an amplifier extending signals from 600 kHz to 1600 kHz in the AM radio band would be an RF amplifier, a video amplifier having a pass-band from 50 Hz to 6 MHz is not, since the information itself is the frequency range.

Size-Based Definition: An RF device is characterized by taking the phase shift of a signal, occurring over the extent of the device, into account. Compared to the wave-length of the electro-magnetic wave, the size of RF devices is not negligible.

The qualifier RF is used to refer to the frequency range lying just below the microwave range. But this definition is inconsistent with the term RFIC (*Radio Frequency Integrated Circuits*) referring to integrated circuits operating at the millimeter and sub-millimeter wave frequencies. Furthermore, signals are referred as RF from the AM band to the sub-millimeter and even IR region. For that reason, microwave would be a sub-range of RF and the distinction between RF and microwave would be obsolete. Though problematic, the phrase *RF and microwave* is very popular [95].

As RF cannot be properly defined by etymological and historical considerations, the most rational basis for defining RF seems to be a feature-based definition. So the distinction between the RF and non-RF objects is based on required design considerations [95], such as phase shift, reactances, dissipation, noise, radiation, reflections, and nonlinearity. That set of issues combines any kind of device employed from LF up to IR ranges.

1.2 Devices for RF Applications

Transistors for RF applications have been mostly a market for devices based on the III-V material system rather than for silicon technologies. These compound semiconductors are based on group III elements, for example aluminum (Al), gallium (Ga), or indium (In), and group V elements, for example arsenic (As), phosphorus (P), or antimon (Sb). III-V semiconductors provide better high-frequency performance, because the inherent physical properties such as the electron mobility enable the components to achieve a much higher performance. Consequently, III-V components are particularly useful for applications at higher frequencies or for higher data rates as required for broadband and RF wireless components as well as for satellite communications. However, as discussed in the sections below, silicon technologies have started to be a major competitor for such applications.

Besides of the material system, a distinction between the device type has to be made. Devices for RF applications can be split into two major groups: the heterojunction bipolar transistors (HBTs) and field effect transistors (FETs) such as MESFETs, high electron mobility transistors (HEMTs) and in recent time the RF MOSFET. A heterostructure device consists of two or more adjacent layers of different semiconductor materials. Due to the different material properties of these layers, there is an abrupt transition in the bandgap and carrier transport. The transit times of vertical bipolar devices such as HBTs mainly depend on the thickness of the base layer, which has to be as thin as possible to achieve an f_T above 200 GHz [113, 211], and on the base-collector space-charge regions. For silicon FETs, the performance depends on the capabilities of the lithography technology [68]. Basically, the most important limiting factor of the response of the transistor is the transit time of the minority carriers across the base region. Due to the performance advantage, the transconductance, high self-gain, low $1/f$ noise and other benefits bipolar transistors are still the device of choice for many applications, for example in the 40 Gb market [68].

1.2.1 Devices Based on the III-V Material System

The GaAs heterojunction bipolar transistor (HBT) has been among the most popular RF devices and was applied in advanced mobile communication applications. Besides the performance, the advantages are a very low off-state power consumption as well as high current amplification [160].

In recent years a new III-V compound semiconductor technology emerged: devices based on indium phosphide (InP), which are able to replace GaAs as the material of choice for high-performance, high-volume commercial applications [210]. InP based technologies have numerous advantages over the GaAs system for many applications. For example, they offer performance advantages in fiber-optic, millimeter-wave and even wireless applications due to the high gain \times breakdown voltage product and thus yield efficiency. In addition, the ability to produce cost-efficient high-volume InP microelectronics enables several markets for government and commercial applications. InP single heterojunction bipolar transistors demonstrated excellent cut-off frequencies f_T and maximum oscillation frequencies f_{max} , but due to the narrow bandgap collector only relatively low collector breakdown voltages BV_{CEO} – conventionally between 0.5 and 2.0 V – are possible.

In order to increase the breakdown voltage, a second heterojunction and thus a double heterojunction bipolar transistor has been introduced. A number of promising results with cut-off frequencies up to 342 GHz and breakdown voltages of the order of 6 V have been demonstrated [102]. A wide bandgap material, for example typically InP or AlInAs, allows higher BV_{CEO} – up to 9 V and more – due to the reduced collector fields and thus reduced impact ionization. In addition, the thermal conductivity is higher and the electron saturation velocity V_{sat} of InP is about two times higher than that of InGaAs resulting in a short collector transit time at high breakdown voltages [184]. The limitation is the collector current blocking due to the conduction band discontinuity. Thus, the design of the layer structure for the second (base collector) heterojunction is of utmost importance. A quaternary material, that is a positionally step graded InGaAsP, is commonly used to lower the conduction band spike [160].

At the moment, the most prominent high electron mobility transistor is the pseudomorphic AlGaAs/InGaAs HEMT, which still provides competitive performance (tough challenged by the SiGe HBT) for low-noise applications in receiver circuits up to 100 GHz [101, 160]. HEMTs based on narrow bandgap materials such as InGaAs and gate lengths below 100 nm show cut-off frequencies beyond 400 GHz [160]. Recent results show a cut-off frequency of 547 GHz for an InGaAs/InAlAs HEMT [199] and 550 GHz for an InP HEMT [198]. Finally it is to note, that devices which incorporate nitrides have become popular in recent time [170]. The so-called III-nitride devices combine different advantages regarding the transport properties, thermal conductivity, and wide bandgaps results in high breakdown fields [160].

1.2.2 Devices Based on Silicon

Heterojunction bipolar transistors (HBT) based on silicon germanium (SiGe) are able to progressively replace devices of the III-V material system, because competitive typical figures of merit are already achieved. For example values for the cut-off frequency of 375 GHz [173] (with associated $f_{max} = 210$ GHz), and for the maximum oscillation frequency of 285 GHz [113] are reported. The major benefit of these devices is their compatibility with the standard CMOS process flow, where well-established sophisticated multi-layer metalization layers and interconnects are available [160].

Integrated circuits for optical transmission and wireless communication systems are based on SiGe HBT or BiCMOS technologies. Among these applications are 40 Gb optical fiber links, 5.8 GHz electronic toll collection transceivers for intelligent transport systems, and integrated frequency divider circuits for wireless local area networks [236].

Development of such advanced devices is based on aggressive device scaling. Thereby, the design focuses separately on the emitter, the base, and the collector. The base transit time is reduced by thinning the base film as deposited, reducing thermal cycles in order to minimize the base dopant diffusion, adding carbons to reduce boron diffusion, and increasing the germanium ramp to accelerate electrons. The concentration in the collector is increased in order to reduce the collector-base space-charge layer and to increase the transconductance [69].

Due to the relatively low electron mobility, compared to III-V materials, and the location of the inversion channel near to the interface between silicon and silicon dioxide, silicon MOS transistors have always been regarded as slow devices. In addition, the relatively large gate length contributed to an inferior RF performance. Due to the aggressive downscaling in the last decade, an advanced RF MOSFET does in fact have a smaller gate length than comparable III-V FETs today. Due to the resulting peak cut-off frequencies up to 100 GHz with relatively low noise figures, complementary metal oxide semiconductor (CMOS) devices can now be employed also for RF applications.

The main reason for this development can be found in the recent advances in CMOS processing, gate length scaling, and progress in SOI technologies [132]. Despite of several scaling limits regarding short channel and hot carrier effects, gate tunneling and minimum oxide thickness the development has been already continued down below the 100 nm feature size. In fact development has sped up, so that the *International Technology Roadmap for Semiconductors* [67] had to be revised twice recently: the 1999 projection for the 2003 technology node was decreased from 120 nm to 100 nm, the gate length for devices in high-performance logic circuits from 85 nm to 45 nm. These developments have also a significant impact on the transistor speed. The continuous scaling has both reduced the chip size and accelerated the transistors.

Furthermore, there has been research in the area of materials compatible with silicon technology. In order to improve the performance of VLSI circuits, silicon germanium is applied due to its material properties. Strained silicon has been started to be used as channel material. This layer, which utilizes an underlying relaxed SiGe layer, allows to enhance both the electron and hole mobilities. In [120], laterally scaled Si-SiGe n-channel FETs are reported with $f_T = 79$ GHz and $f_{\max} = 212$ GHz for $L_g = 80$ nm and $f_T = 92$ GHz for $L_g = 70$ nm. Related results can be found in [57, 121].

1.3 Numerical Simulators

In order to cope with explosive costs, reduced time-to-market figures, and to deal with high competition, Technology Computer-Aided-Design (TCAD) is more and more applied during development and production of new devices [162]. The basic motivation for applying numerical simulation tools is to reduce the number of test wafers, to evaluate a vast number of device variations by numerical means and to optimize devices and processes. Optimization of geometry, doping, materials, and material compositions targets high output power, high breakdown voltage, high speed (high f_T and f_{\max}), low leakage, low noise, and low power consumption. This is a challenging task that can be significantly supported by device simulation. While steady-state simulations are sufficient for optimization targets such as breakdown voltages, turn-on voltages, or leakage currents, small-signal simulations are required for many performance and noise issues [162].

The continuously increasing computational power of the average workstation computer and available cluster computing technologies enables the large-scale application of TCAD software. Several commercial device simulators [4, 20, 42, 111, 200, 214–216] company-developed simulators like [30, 148], and university-developed simulators [38, 62, 104, 105, 116, 153, 203, 205, 207, 237] have already been used for supporting device design and development. The major differences between these simulators are

- the dimensionality: one, quasi-two, two, quasi-three, or three dimensions.
- the choice of the computational approach: partial differential equations or Monte Carlo.
- the choice between microscopic or macroscopic approaches: *Wigner*, *Schrödinger*, *Green's* function or macroscopic transport models.
- the choice of the macroscopic carrier transport model: drift-diffusion, energy-transport, or six moments.
- capabilities of included electrothermal effects.

On the circuit simulation level the approaches mentioned above including the various macroscopic transport models require too much computational resources, especially in terms of time and memory, if large circuits consisting of thousands of transistors have to be simulated. For that reason, circuit analysis tools like the well-known SPICE [168] simulator are based on compact models (see Section 3.6.1). They try to provide a closed form description of the electrical behavior of the devices, which results in far less equations than the approaches based on discretized partial differential equations. However, especially for advanced semiconductor devices the extraction of the various parameters is a cumbersome task. Furthermore, calibration against given experimental data is required as extrapolation from one generation to the next is rarely possible. A possible replacement of normally expensive experimental data can be simulation results from device simulations.

1.3.1 The Commercial Simulators

Tools for Technology Computer Aided Design have been essential for the development of more advanced and sophisticated devices. In the area of device simulation, the major player in the commercial market for simulators is the company *Synopsis* at the moment. Since the acquisition of *Integrated Systems Engineering AG* (ISE) in November 2004, the company provides the well-known TCAD software packages MEDICI/DAVINCI [214, 216] and DESSIS [111]. Competition comes for example from *Silvaco* with the ATLAS framework [200] and *Crosslight* (formerly *Beamtek Software*) with the LASTIP/PICS3D/APSYS simulators for optoelectronics and advances physical modeling [42]. In addition, several general frameworks as presented in Section 4.2.1 are provided for the solution of partial differential equations, which can also be employed for semiconductor modeling and simulation.

S-PISCES of *Silvaco* as part of the ATLAS simulation framework [200] calculates steady-state, small-signal AC, and transient solutions for general non-planar two-dimensional silicon device structures. The related simulator for compound semiconductors is BLAZE2D/3D, which provides a library including also ternary and quaternary materials. The calculated small-signal characteristics are the cut-off frequency f_T , S-, Y-, H-, and Z- parameters, the maximum available gain (MAG), the maximum stable gain (MSG), the maximum frequency of oscillation (f_{max}) and the stability factor.

The multi-dimensional simulator DESSIS [111] provides related features. The small-signal capabilities are incorporated in the mixed-mode, which supports electrothermal netlists with mesh-based device models and SPICE circuit models.

MEDICI of *Synopsis* [216] (a former *Avant!* product) is a two-dimensional simulator. A small-signal analysis can be performed to calculate frequency-dependent capacitances, conductances, admittances, Y-, S-, and H-parameters. DAVINCI [214] is the related three-dimensional device simulation program with a similar set of features. The approach is also based on [127], including the numerical split of real- and complex-valued part.

Applications of advanced RF devices must often be seen in a circuit related context [208]. For that reason, circuit simulation programs such as Spice [147], *Agilent's* ADS [4], or HSPICE of *Synopsis* [215] are employed. Whereas these circuit simulators are based on compact models, device simulators with distributed modeling (solving of a system of partial differential equations) of the transistors offer so-called mixed-modes. Realistic dynamic boundary conditions imposed by a circuit allow to extract circuit-related figures of merit. Although this approach is limited by performance and memory considerations, the highly sophisticated models required for today's advanced device structures can be directly employed for transient or small-signal circuit simulations [231].

It is a well-known fact, that correct steady-state modeling is an important prerequisite for any kind of subsequent simulations. Thus, the advanced simulators incorporate drift-diffusion and advanced transport models such as energy transport models [89] and provide several advanced mobility models. In addition, models for recombination, band-gap narrowing, impact ionization, band-to-band tunneling, hot carrier in-

jection, *Schottky* contacts, and floating gates have to be included to account for the properties of advanced device structures.

1.3.2 The Device and Circuit Simulator Minimos-NT

MINIMOS-NT [105] is a general-purpose semiconductor device simulator providing steady-state, transient, and small-signal analysis of arbitrary two- and three-dimensional device structures. In addition, mixed-mode device/circuit simulation [88] is offered to embed numerically simulated devices in circuits with compact models. The devices can be connected both electrically and thermally.

The simulator deals with different complex structures and materials, such as Si, Ge, SiGe, GaAs, AlAs, InAs, GaP, InP, their alloys and non-ideal dielectrics. In combination with this comprehensive material database, the state-of-the-art set of physical models enables the user to simulate all kinds of advanced device structures, such as MOS devices of the sub-100 nm technology, silicon-on-insulator devices, and heterostructures. The implemented physical models take all important physical effects such as bandgap narrowing, surface recombination, transient trap recombination, impact ionization, self-heating, and hot electron effects into account.

Besides the basic semiconductor equations [193], several different types of transport equations can be solved. Among these are the energy-transport equations which capture hot-carrier transport [22, 209], a six moments transport model [85], the lattice heat-flow equation to cover thermal effects like self-heating [227]. Furthermore, various interface and boundary conditions are taken care of, which include *Ohmic* and *Schottky* contacts, thermionic field emission over and tunneling through various kinds of barriers.

MINIMOS-NT is equipped with a very powerful control language called *Input-Deck Programming Language* (IPL) [118]. For the specification of a simulation various keywords must be set. These keywords can contain arbitrarily nested expressions and can depend on the current simulation status. With the IPL, the user is able to customize a simulation by creating input-deck files written in plain ASCII text. MINIMOS-NT provides various default input-deck files with standard settings.

MINIMOS-NT is the successor of MINIMOS [195]. Whereas the latter is restricted to simple MOS structures, MINIMOS-NT can be employed for arbitrary device structures with unstructured grids.

Chapter 2

Device Simulation

In order to analyze the electronic properties of semiconductor structures under all kinds of operating conditions, the effects related to the transport of charge carriers under the influence of external fields must be modeled.

Most applied macroscopic models for carrier transport in semiconductors are based on the semiclassical *Boltzmann* transport equation. By applying the method of moments [22] this six-dimensional equation can be transformed into an infinite series of three-dimensional equations. This allows one to derive a hierarchy of increasingly complex transport models:

- The drift-diffusion transport model: Keeping only the zero- and first- order moment equations together with proper closure assumptions [193] yields the current relations of the drift-diffusion model. Together with the *Poisson* equation, they form the basic semiconductor equations as given first by *VanRoosbroeck* [225] (see Section 2.1.1).
- The energy-transport transport model: Considering two additional moments gives the energy-transport model [22, 89, 209], where the carrier temperatures are allowed to be different from the lattice temperature. Since the current densities depend then on the respective carrier temperature, two more quantities, the electron temperature and the hole temperature, are added.
- The six moments transport model [84, 85]: according to the name, six moments are included to describe carrier transport. The two additional quantities are the electron and hole kurtosis.

Whereas in most simulators carrier transport can be treated by the drift-diffusion and the energy-transport transport models, MINIMOS-NT additionally provides the six moments transport model. Recent research indicates, that the six moments model is able to accurately cover the important range of gate lengths from 25 nm to 100 nm [83]. Above this gate length less costly four moments models, for example an energy-transport model, can be used if the more accurate descriptions of the distribution function of the carriers is not required. Despite the fact that the drift-diffusion transport model loses its accuracy already for gate length below 250 nm, it is still frequently and industrially applied due to its efficiency.

In this chapter, the analytical problem based on the three transport models as well as its discretization is discussed. The second part of the chapter deals with the simulation modes of the simulator. The discussion of the steady-state simulation mode covers also the nonlinear solution technique. This is followed by the transient simulation and eventually the derivation of the small-signal system.

2.1 The Analytical Problem

The analytical problem is basically formed by nonlinear partial differential equations. These equations are the *Poisson* equation (2.1) and the current continuity equations for the two carrier types in semiconducting materials, electrons and holes,

$$\nabla \cdot (\varepsilon \nabla \psi) = -\varrho, \quad (2.1)$$

$$\nabla \cdot \mathbf{J}_n = q \left(R + \frac{\partial n}{\partial t} \right), \quad (2.2)$$

$$\nabla \cdot \mathbf{J}_p = -q \left(R + \frac{\partial p}{\partial t} \right). \quad (2.3)$$

The unknown quantities of this equation system are the electrostatic potential ψ , and the electron and hole concentrations n and p , respectively. q is the elementary charge, ϱ denotes the space charge density and R stands for the net recombination rate, which is defined as

$$R = R_n - G_n = R_p - G_p. \quad (2.4)$$

These three equations can be derived from the four *Maxwell* equations [100]:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \quad (2.5)$$

$$\nabla \cdot \mathbf{B} = 0, \quad (2.6)$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t}, \quad (2.7)$$

$$\nabla \cdot \mathbf{D} = \varrho. \quad (2.8)$$

\mathbf{E} and \mathbf{D} are the electric field and displacement vectors, respectively, and \mathbf{H} and \mathbf{B} are the magnetic field and induction vectors, respectively. \mathbf{J} is the conduction current density.

\mathbf{D} is related to the electric field \mathbf{E} by the permittivity tensor $\hat{\varepsilon}$ (assumed to be a scalar ε hereafter):

$$\mathbf{D} = \hat{\varepsilon} \mathbf{E}, \quad (2.9)$$

which is valid for all materials which have a frequency-independent permittivity and do not have piezo-electric or ferroelectric effects. The *Poisson* equation can be derived by introducing a vector potential \mathbf{A} defined by $\mathbf{B} = \nabla \times \mathbf{A}$, which is inserted in (2.5):

$$\nabla \times \mathbf{E} = -\frac{\partial}{\partial t}(\nabla \times \mathbf{A}). \quad (2.10)$$

After interchanging the order of the time derivative and the curl operator and using the associative property of the curl operator, the argument of the curl operator can be substituted by the gradient of a scalar potential, because $\nabla \times \nabla \psi$ yields zero:

$$\mathbf{E} + \frac{\partial \mathbf{A}}{\partial t} = -\nabla \psi. \quad (2.11)$$

Since the wavelength is much larger than the typical semiconductor device dimensions, the quasi-stationary approximation can be assumed. With an operating frequency $f_{\text{op}} = 200$ GHz, c_0 as the speed of light in vacuum and $\varepsilon_r = 11.9$ [217] and $\mu_r = 1$ as the relative permittivity and permeability in silicon, respectively, $\lambda = c_0/(\sqrt{\varepsilon_r \mu_r} f_{\text{op}})$ equals $435 \mu\text{m}$ [91]. The typical gate length of industrially produced devices is already below 100 nm . In the quasi-stationary approximation, the time derivative of the vector potential

can be neglected. By inserting $\mathbf{E} = -\nabla \psi$ into (2.9) and by inserting the result into (2.8), the *Poisson* equation (2.1) is finally obtained. The space charge density ϱ in semiconductors is composed of the charges of electrons, holes, the ionized dopant atoms and other charged defects:

$$\varrho = q(p - n + C). \quad (2.12)$$

The fixed charge C is commonly modeled as

$$C = N_D^+ - N_A^-, \quad (2.13)$$

where N_D^+ is the concentration of the positively charged donor atoms and N_A^- the concentration of the negatively charged acceptor atoms. Dynamic recombination processes such as the *Shockley-Read-Hall* recombination require an additional dynamically changing trap charge N_t . It is important to note that not all dopants are electrically active and that not all electrically active dopants are always ionized. Particularly at low temperatures this fact has to be included in the model. However, in silicon at room temperature $N_D^+ = N_D$ and $N_A^- = N_A$ is assumed.

The *Poisson* equation without considering a magnetic field is finally obtained by

$$\nabla \cdot (\varepsilon \nabla \psi) = q(n - p + N_A^- - N_D^+). \quad (2.14)$$

The continuity equation for the conduction current density is derived by applying the divergence operator to (2.7) and using (2.8):

$$\nabla \cdot \mathbf{J} + \frac{\partial \varrho}{\partial t} = 0. \quad (2.15)$$

The equation can be interpreted, that all sources and sinks of the conduction current density are compensated by the time variation of the space charge density.

Whereas the *Maxwell* equations can be used to derive the *Poisson* equation and the current continuity equations, the current relations cannot be derived from them. The carrier transport equations are therefore discussed in the next sections.

2.1.1 The Drift-Diffusion Transport Model

Many causes of the current flows can be identified, for example gradients of the carrier concentrations, temperatures or material properties or a contribution determined by *Ohm's* law [137,138]. The latter component is called drift current and can be formulated as

$$\mathbf{J}^{\text{drift}} = \sigma \mathbf{E}. \quad (2.16)$$

With $\sigma_n = qn\mu_n$ as the electrical conductivity, μ_n as the mobility of electrons, and $\mathbf{V}_n = -\mu_n \mathbf{E}$ as the mean velocities of electrons, the current density is obtained by

$$\mathbf{J}_n^{\text{drift}} = qn\mu_n \mathbf{E} = -qn\mathbf{V}_n. \quad (2.17)$$

Whereas the electric field accelerates the carriers, the velocity of the carriers is limited by various scattering mechanisms such as lattice vibrations (phonon scattering) [31]. The saturation velocity V_{sat} is generally modeled independently of the doping, whereas below V_{sat} an indirect relation is encountered. The diffusion current is caused by the thermal motion of the carriers described by the gradient of the carrier concentration. Derived from the law of diffusion, the following relation can be given:

$$\mathbf{J}_n^{\text{diffusion}} = qD_n \nabla n. \quad (2.18)$$

D_n is the diffusion coefficient for electrons which is often modeled using the mobility by the *Einstein* relation

$$D_n = \mu_n \frac{k_B T_n}{q}, \quad (2.19)$$

where k_B is the *Boltzmann* constant. This relation is valid for conditions close to thermal equilibrium and for non-degenerate carrier systems (*Boltzmann* statistics). Both current components are added to obtain the isothermal drift-diffusion transport model

$$\mathbf{J}_n = qD_n \nabla n - q\mu_n n \nabla \psi = q(D_n \nabla n - \mu_n n \nabla \psi), \quad (2.20)$$

which is applied in (2.2) to obtain the current continuity equation. Together with the *Poisson* equation (2.1), they form the basic semiconductor equations as given first by *VanRoosbroeck* [225]:

$$\nabla \cdot (\epsilon \nabla \psi) = -\rho, \quad (2.21)$$

$$\nabla \cdot (D_n \nabla n - \mu_n n \nabla \psi) = R + \frac{\partial n}{\partial t}, \quad (2.22)$$

$$\nabla \cdot (D_p \nabla p + \mu_p p \nabla \psi) = R + \frac{\partial p}{\partial t}. \quad (2.23)$$

To this set the lattice heat-flow equation (2.24) is frequently added to account for thermal effects such as self-heating in the device:

$$\nabla \cdot (\kappa_L \nabla T_L) = \rho_L c_L \frac{\partial T_L}{\partial t} - H. \quad (2.24)$$

This equation requires modeling of the thermal conductivity κ_L , the mass density ρ_L , the heat capacity c_L , and the locally generated heat H . As the parameters of equations (2.21) to (2.23) depend also on the lattice temperature and a gradient in the lattice temperature causes a current flow [227], the thermal drift-diffusion carrier continuity equations for electrons and holes are given as [194]

$$\mathbf{J}_n = \nabla \cdot \left(\mu_n \frac{k_B}{q} \nabla n T_L - \mu_n n \nabla \psi \right) = R + \frac{\partial n}{\partial t}, \quad (2.25)$$

$$\mathbf{J}_p = \nabla \cdot \left(\mu_p \frac{k_B}{q} \nabla p T_L - \mu_p p \nabla \psi \right) = R + \frac{\partial p}{\partial t}, \quad (2.26)$$

forming a coupled system with (2.24).

2.1.2 Types of Partial Differential Equations

Nonlinear partial differential equations of second-order can appear in three variants: elliptic, parabolic, and hyperbolic. The *Poisson* equation as well as the steady-state continuity equations form a system of elliptic partial differential equations, whereas the lattice heat-flow equation is parabolic.

This can be shown for the *Poisson* equation posed in a bounded domain $D \in \mathbb{R}^2$ by rewriting the differential operators of (2.21) in (2.27) and for *Cartesian* coordinates in (2.28), which is then compared with a general partial differential equation of second order (2.29):

$$\nabla^2 \psi = -\rho/\epsilon, \quad (2.27)$$

$$\partial_x^2 u + \partial_y^2 u = -\rho/\epsilon, \quad (2.28)$$

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F_u = G. \quad (2.29)$$

The coefficients A to G are piecewise continuous functions of x and y . In analogy to quadratic forms, one can calculate the determinant

$$d = AC - B^2, \quad (2.30)$$

and classify the given equation: in case $d > 0$ the equation is elliptic (for the *Poisson* equation: $A = C = 1, B = 0$), in case $d = 0$ parabolic and in case $d < 0$ hyperbolic.

To completely determine the solution of an elliptic partial differential equation boundary conditions have to be specified. Since parabolic and hyperbolic partial differential equations describe evolutionary processes, time normally is an independent variable and an initial condition is additionally required. As the transient continuity equations are parabolic, they require such an initial condition.

2.1.3 Systematic Derivation of the Macroscopic Transport Models

Macroscopic transport models can be systematically derived from the *Boltzmann* transport equation [134]

$$\frac{\partial f}{\partial t} + \mathbf{u} \cdot \nabla_{\mathbf{r}} f + \mathbf{F} \cdot \nabla_{\mathbf{p}} f = Q[f(\mathbf{k})]. \quad (2.31)$$

This equation is a time-dependent partial integro-differential equation in the six-dimensional phase space (\mathbf{k}, \mathbf{r}) , which assumes that the carrier motion is determined by *Newton's* laws. The solution is the distribution function f , \mathbf{u} denotes the group velocity, \mathbf{F} stands for the external force. The scattering operator $Q[\cdot]$, which is in general nonlinear in f , represents the rate of change of f due to collisions and is modeled via *Fermi's* Golden Rule.

The distribution function f as solution of (2.31) is used to obtain the probability of finding a carrier inside a phase-space volume $d\mathbf{k}d\mathbf{r}$. The equilibrium distribution function is the *Fermi-Dirac* distribution. If the *Pauli* exclusive principle is neglected, the *Maxwell* distribution is obtained. In the drift-diffusion model the cold and in the energy-transport the heated *Maxwell* distribution is used. The six moments model uses an analytical distribution function in the approach pursued here [84].

This can be performed either directly by Monte Carlo simulations [112, 115, 122, 123] or by the methods based on the expansion of the distribution function in momentum space into a series of spherical harmonics [97].

Monte Carlo simulations are particularly useful to obtain a physically accurate picture including various effects with as few approximations as possible. Since the simulations require much computational resources in terms of time and memory, it is rarely used on an engineering level. The extraction of small-signal parameters is often prohibitively costly, because the evaluation of a small-signal perturbation requires a low variance in the results.

Whereas for extremely small devices with gate lengths below 10 nm systems based on the *Wigner-Boltzmann* equation [70, 149–151] have been applied, the ballistic limit has been investigated by using transport models based on the *Schrödinger* [124] equation.

As its direct solution requires considerable computational resources, simpler solutions are obtained by investigating lower moments of the distribution function only. These include the electron concentration n , the electron temperature T_n , and the electron kurtosis β_n .

By multiplying (2.31) with a weight function $\phi(\mathbf{k})$ and integrating the result over the \mathbf{k} space, the macroscopic transport equations are obtained. It is assumed that the *Brillouin* zone extends towards infinity, which is justified due to the exponential decline of the distribution function [240]. This procedure results in the following partial differential equations in (\mathbf{r}, t)

$$\frac{\partial n\langle\phi\rangle}{\partial t} + \nabla_{\mathbf{r}} \cdot n\langle\mathbf{u} \otimes \phi\rangle - n\mathbf{F} \cdot \langle\nabla_{\mathbf{p}} \otimes \phi\rangle = \int \phi Q[f(\mathbf{k})] d^3\mathbf{k}, \quad (2.32)$$

while the coordinates of the \mathbf{k} space are saturated. As macroscopic quantities describing the average behavior of the microscopic quantity $\phi(\mathbf{k})$, moments neither depend on \mathbf{k} nor contain the respective information any more.

For the balance and flux equations, the weight functions \mathcal{E}^i and $\mathbf{p} \mathcal{E}^i$ with the momentum $\mathbf{p} = \hbar \mathbf{k}$ are often used [84]. For the drift-diffusion transport model, the expansion is truncated at $i = 1$, for the energy-transport transport model at $i = 2$, and eventually for the six moments transport model at $i = 3$. The balance and flux relations read as follows [83]:

$$\phi_0 = \mathcal{E}^0 = 1 \quad \Rightarrow \quad \frac{\partial n}{\partial t} + \nabla \cdot (n \mathbf{V}_0) = n q_0 \quad (2.33)$$

$$\phi_1 = \mathbf{p} \mathcal{E}^0 = \mathbf{p} \quad \Rightarrow \quad \frac{\partial n \mathbf{P}_0}{\partial t} + \nabla \cdot (n \hat{\mathbf{U}}_1) - n \mathbf{F} = n \mathbf{Q}_0 \quad (2.34)$$

$$\phi_2 = \mathcal{E}^1 \quad \Rightarrow \quad \frac{\partial n w_1}{\partial t} + \nabla \cdot (n \mathbf{V}_1) - 1 n \mathbf{F} \cdot \mathbf{V}_0 = n q_1 \quad (2.35)$$

$$\phi_3 = \mathbf{p} \mathcal{E}^1 \quad \Rightarrow \quad \frac{\partial n \mathbf{P}_1}{\partial t} + \nabla \cdot (n \hat{\mathbf{U}}_2) - n \mathbf{F} \cdot (w_1 \hat{\mathbf{I}} + \hat{\mathbf{U}}_1) = n \mathbf{Q}_1 \quad (2.36)$$

$$\phi_4 = \mathcal{E}^2 \quad \Rightarrow \quad \frac{\partial n w_2}{\partial t} + \nabla \cdot (n \mathbf{V}_2) - 2 n \mathbf{F} \cdot \mathbf{V}_1 = n q_2 \quad (2.37)$$

$$\phi_5 = \mathbf{p} \mathcal{E}^2 \quad \Rightarrow \quad \frac{\partial n \mathbf{P}_2}{\partial t} + \nabla \cdot (n \hat{\mathbf{U}}_3) - n \mathbf{F} \cdot (w_2 \hat{\mathbf{I}} + 2 \hat{\mathbf{U}}_2) = n \mathbf{Q}_2 \quad (2.38)$$

$$\vdots \quad \quad \quad \vdots$$

with

$$\mathbf{P}_i = \langle \mathbf{p} \mathcal{E}^i \rangle, \quad \mathbf{V}_i = \langle \mathbf{u} \mathcal{E}^i \rangle, \quad w_i = \langle \mathcal{E}^i \rangle, \quad \hat{\mathbf{U}}_i = \langle \mathbf{u} \otimes \mathbf{p} \mathcal{E}^{i-1} \rangle. \quad (2.39)$$

The moments of the scattering integral are

$$n q_i = \int \mathcal{E}^i Q[f(\mathbf{k})] d^3 \mathbf{k}, \quad (2.40)$$

$$n \mathbf{Q}_i = \int \mathbf{p} \mathcal{E}^i Q[f(\mathbf{k})] d^3 \mathbf{k}. \quad (2.41)$$

\mathbf{F} stands for the external force, which is given as

$$\mathbf{F}(\mathbf{r}) = -q \mathbf{E}(\mathbf{r}), \quad (2.42)$$

for homogeneous band structures and neglected magnetic fields. In addition, the system is claimed to be diffusion dominated [174]. As a consequence, the convective terms and the time derivatives in the flux relations are neglected resulting in parabolic partial differential equation systems [89].

The moment n is the carrier concentration, $n \mathbf{V}_0$ the average carrier flux, w_1 the average energy, $n \mathbf{V}_1$ the average energy flux, w_2 the average energy squared, and $n \mathbf{V}_2$ the average kurtosis flux. \mathbf{V}_0 is the average carrier velocity, \mathbf{P}_0 is the average momentum, $\hat{\mathbf{U}}_1$ the energy tensor, $\hat{\mathbf{U}}_2$ the second energy tensor, and $\hat{\mathbf{U}}_3$ is eventually the third energy tensor. Furthermore it can be noted that (2.33) – (2.38) are the continuity equation and current relation, the energy balance and energy-flux relation, and the kurtosis balance and kurtosis-flux relation, respectively.

As this equation system contains more unknowns than equations and each equation is coupled to the next higher one, the so-called closure problem comes into play: additional relations have to be introduced in order to close the equation system, thus making it solvable. There are several approaches to overcome this problem [84], for example the maximum entropy method [129], *Grad's* method [80], the cumulant

closure method [176, 192, 235], or a *Maxwell* closure [85]. The choice of variables which are expressed by other ones is critical. The current density is proportional to the average velocity \mathbf{V}_0 . Furthermore, the average energy w_1 is used to model many physical effects. For that reason, \mathbf{V}_i and w_i are chosen to remain as solution variables and the additional moments are expressed as functions of the solution variables. According to [82], the energy-like tensor $\hat{\mathbf{U}}_i$ are

$$\hat{\mathbf{U}}_i = \langle \mathbf{u} \otimes \mathbf{p} \mathcal{E}^{i-1} \rangle = \frac{2}{3} H_i w_i \hat{\mathbf{I}} + U_i^{(2)}(w_i, \mathbf{V}_i), \quad (2.43)$$

where the anisotropic second term is neglected. By applying the macroscopic relaxation time approximation [82, 87] the equation system in the form of (2.44) and (2.45) is obtained.

$$\frac{\partial n w_i}{\partial t} + \nabla \cdot (n \mathbf{V}_i) - i \mathbf{F} \cdot n \mathbf{V}_{i-1} = n q_i, \quad (2.44)$$

$$\nabla \cdot (n \hat{\mathbf{U}}_{i+1}) - n \mathbf{F} \cdot (w_i \hat{\mathbf{I}} + i \hat{\mathbf{U}}_i) = n \mathbf{Q}_i, \quad (2.45)$$

with

$$q_i = -\frac{w_i - w_{i,\text{eq}}}{\tau_i}, \quad (2.46)$$

$$\mathbf{Q}_i = -\frac{\mathbf{V}_i}{\mu_i}. \quad (2.47)$$

The fluxes can be explicitly written as

$$n \mathbf{V}_i = -\frac{2\mu_i H_{i+1}}{3q} \left(\nabla(n w_{i+1}) - n \mathbf{F} w_i \frac{3 + 2i H_i}{2H_{i+1}} \right). \quad (2.48)$$

The non-parabolicity correction factors H_i equal unity in the case of parabolic bands and are modeled as either energy-dependent using a simple analytical expression [28], by the incorporation of bulk Monte Carlo data [84, 219] or via analytic models for the distribution function [82].

Note that (2.45) still contains the tensors $\hat{\mathbf{U}}_i$ which have to be approximated by the unknowns of the equation system w_i and \mathbf{V}_i . Finally, the following variable transformation according to [86] is introduced:

$$w_1 = \frac{3}{2} k_B T, \quad w_2 = \underbrace{\left(\frac{15}{4} k_B^2 T^2 \right)}_{\text{Maxwell}} \beta, \quad w_3 = \underbrace{\left(\frac{105}{8} k_B^3 T^3 \right)}_{\text{Maxwell}} \gamma. \quad (2.49)$$

β and γ are the contributions of the non-*Maxwell* closure. Thus, in order to obtain the *Maxwell* closure, $\beta = 1$ and $\gamma = 1$ is used. The quantity β is the kurtosis of the distribution function and indicates the deviation from the *Maxwell* shape. The quantity γ depends on the applied closure relation and equals β^c for the generalized *Maxwell* closure, which will be used in the following. Then, the following form of the flux equations can be obtained:

$$n \mathbf{V}_0 = -A_0 \left(\nabla(n k_B T) - n \mathbf{F} h_0 \right), \quad (2.50)$$

$$n \mathbf{V}_1 = -A_1 \left(\nabla(n k_B^2 T^2 \beta) - n \mathbf{F} h_1 k_B T \right), \quad (2.51)$$

$$n \mathbf{V}_2 = -A_2 \left(\nabla(n k_B^3 T^3 \gamma) - n \mathbf{F} h_2 k_B^2 T^2 \beta \right), \quad (2.52)$$

with

$$A_i = \frac{\mu_i H_{i+1}}{2^i q} \frac{1}{3} \prod_{j=0}^{i+1} (1 + 2j) \quad \text{and} \quad h_i = \frac{3 + 2i H_i}{(3 + 2i) H_{i+1}}. \quad (2.53)$$

The balance equations are derived as

$$\frac{\partial n}{\partial t} + \nabla \cdot (n \mathbf{V}_0) = 0, \quad (2.54)$$

$$C_1 \frac{\partial n T}{\partial t} + \nabla \cdot (n \mathbf{V}_1) - \mathbf{F} \cdot n \mathbf{V}_0 = -n C_1 \frac{T - T_{\text{eq}}}{\tau_1}, \quad (2.55)$$

$$C_2 \frac{\partial n T^2 \beta}{\partial t} + \nabla \cdot (n \mathbf{V}_2) - 2\mathbf{F} \cdot n \mathbf{V}_1 = -n C_2 \frac{T^2 \beta - T_{\text{eq}}^2 \beta_{\text{eq}}}{\tau_2}, \quad (2.56)$$

with $C_1 = 3k_B/2$ and $C_2 = 15k_B^2/4$. By using the equilibrium solution of the *Boltzmann* transport equation, the equilibrium values T_{eq} and β_{eq} can be calculated for the *Maxwell* distribution.

The equilibrium carrier temperature defined by (2.49) is different from the lattice temperature, because a non-parabolic band structure is used. One obtains $T_{\text{eq}} \approx 309.452$ K and $\beta_{\text{eq}} \approx 1$ [84]. These are also the values used for the *Dirichlet* boundary conditions for T and β . For the carrier concentration a standard *Ohmic* contact model is used, corresponding to a cold *Maxwell* distribution at the contacts.

In the following sections, the transport models for the parabolic case are used. Thus, $h_i = H_i = 1$, $T_{\text{eq}} = T_L$, and $\beta_{\text{eq}} = 1$, resulting in

$$A_0 = \frac{\mu_0}{q}, \quad A_1 = \frac{5\mu_1}{2q}, \quad A_2 = \frac{35\mu_2}{4q}. \quad (2.57)$$

2.1.4 The Six Moments Transport Model

The closure of the six moments transport model at ϕ_6 was determined as $\gamma = \beta^c$. Thus, the following flux equations can be derived:

$$\mathbf{J}_n = -q(n \mathbf{V}_0) = -qA_0(\nabla(nk_B T) - n\mathbf{F}), \quad (2.58)$$

$$\mathbf{S}_n = (n \mathbf{V}_1) = -A_1(\nabla(nk_B^2 T^2 \beta) - n\mathbf{F}k_B T), \quad (2.59)$$

$$\mathbf{K}_n = (n \mathbf{V}_2) = -A_2(\nabla(nk_B^3 T^3 \beta^c) - n\mathbf{F}k_B^2 T^2 \beta). \quad (2.60)$$

The balance equations are

$$\frac{\partial n}{\partial t} + \nabla \cdot (n \mathbf{V}_0) = 0, \quad (2.61)$$

$$\frac{3k_B}{2} \frac{\partial n T}{\partial t} + \nabla \cdot (n \mathbf{V}_1) - \mathbf{F} \cdot n \mathbf{V}_0 = -n \frac{3k_B}{2} \frac{T - T_L}{\tau_1}, \quad (2.62)$$

$$\frac{15k_B^2}{4} \frac{\partial n T^2 \beta}{\partial t} + \nabla \cdot (n \mathbf{V}_2) - 2\mathbf{F} \cdot n \mathbf{V}_1 = -n \frac{15k_B^2}{4} \frac{T^2 \beta - T_L^2}{\tau_2}. \quad (2.63)$$

With $c = 2.7$ the best match for w_3 was obtained in comparison to Monte Carlo simulations [83]. The boundary conditions are derived by applying a cold *Maxwell* distribution:

$$f_{\text{eq}} = A \exp\left(-\frac{\mathcal{E}}{k_B T_L}\right), \quad (2.64)$$

$$n = \int f g d\mathcal{E}, \quad (2.65)$$

resulting in the boundary conditions for the higher moments

$$w_1 = \mathcal{E}_{\text{eq}} = \frac{1}{n} \int \mathcal{E} f g d\mathcal{E} = \frac{3}{2} k_B T, \quad (2.66)$$

$$w_2 = \frac{1}{n} \int \mathcal{E}^2 f g d\mathcal{E} = \frac{3 \cdot 5}{2 \cdot 2} k_B^2 T^2. \quad (2.67)$$

2.1.5 The Energy-Transport Model

By using the closure $\beta = 1$ at ϕ_4 , the flux equations of the energy-transport transport model can be obtained:

$$\mathbf{J}_n = -q(n\mathbf{V}_0) = -qA_0 \left(\nabla(nk_B T) - n\mathbf{F} \right), \quad (2.68)$$

$$\mathbf{S}_n = (n\mathbf{V}_1) = -A_1 \left(\nabla(nk_B^2 T^2) - n\mathbf{F}k_B T \right). \quad (2.69)$$

The balance equations are (2.61) and (2.62).

2.1.6 The Drift-Diffusion Transport Model

For the drift-diffusion transport model, the closure at ϕ_2 was found to be $T_n = T_L$, yielding the following balance and flux equation:

$$\mathbf{J}_n = -q(n\mathbf{V}_0) = -qA_0 \left(\nabla(nk_B T_L) - n\mathbf{F} \right), \quad (2.70)$$

$$\frac{\partial n}{\partial t} + \nabla \cdot (n\mathbf{V}_0) = \frac{\partial n}{\partial t} - \frac{1}{q} \nabla \cdot \mathbf{J}_n = 0. \quad (2.71)$$

These equations look very familiar, as they were the result of the derivation in Section 2.1.1. In the flux equation (2.70), the external force is substituted by $\mathbf{F} = -q\mathbf{E} = q\nabla\psi$.

Eventually, (2.70) must be inserted into (2.71), which is the balance equation corresponding to (2.2).

$$\nabla \cdot \mathbf{J}_n = -q \frac{\partial n}{\partial t} \quad (2.72)$$

$$\nabla \cdot \left[-q \frac{\mu_0}{q} \left(\nabla(nk_B T_L) - nq\nabla\psi \right) \right] = -q \frac{\partial n}{\partial t} \quad (2.73)$$

When T_L is assumed to be constant, the isothermal drift-diffusion model as already shown in (2.22) can be finally obtained:

$$\nabla \cdot \left(\underbrace{\frac{\mu_0 k_B T_L}{q}}_{D_n} \nabla n - \mu_0 n q \nabla \psi \right) = \frac{\partial n}{\partial t} + R, \quad (2.74)$$

with the net recombination rate $R = R_0$ according to

$$nq_0 = n \frac{\langle 1 \rangle - \langle 1 \rangle_{\text{eq}}}{\tau_0} + R_0 = R_0 \quad (2.75)$$

2.2 The Discretized Problem

In general, the problem as defined in section Section 2.1.3 cannot be solved analytically. Thus, the solution has to be calculated by numerical methods, which normally require a discretization of the partial differential equations. For that reason, the domain \mathcal{V} where the equations are posed has to be partitioned into a finite number of subdomains \mathcal{V}_i , which are usually obtained by a *Voronoi* tessellation [33, 156]. In order to obtain the solution with a desired accuracy, the equation system is approximated in each of these subdomains by algebraic equations. The unknowns of this system are approximations of the continuous solutions at the discrete grid points in the domain [193].

Several approaches for the discretization of the partial differential equations have been proposed. Due to the discretization of the current continuity equations it has been found to be advantageous to apply the finite boxes discretization scheme for semiconductor device simulation [193]. This method considers the equation integral form for each subdomain, which is the so-called control volume \mathcal{V}_i associated with the grid point P_i .

Before the *Gauss* integral theorem can be applied, the fluxes (2.58)–(2.60) are inserted in the balance equations (2.61)–(2.63) (analogously to the drift-diffusion model above):

$$\frac{\partial n}{\partial t} - \frac{1}{q} \nabla \cdot \mathbf{J}_n = 0, \quad (2.76)$$

$$C_1 \frac{\partial nT}{\partial t} + \nabla \cdot \mathbf{S}_n + \frac{1}{q} \mathbf{F} \cdot \mathbf{J}_n + n C_1 \frac{T - T_L}{\tau_1} = 0, \quad (2.77)$$

$$C_2 \frac{\partial nT^2\beta}{\partial t} + \nabla \cdot \mathbf{K}_n - 2\mathbf{F} \cdot \mathbf{S}_n + n C_2 \frac{T^2\beta - T_L^2}{\tau_2} = 0. \quad (2.78)$$

The inner products $\mathbf{F} \cdot \mathbf{J}_n$ and $\mathbf{F} \cdot \mathbf{S}_n$ are discretized as [85]

$$\mathbf{F} \cdot \mathbf{J}_n \Big|_{\psi=\psi_i} = q \nabla \cdot ((\psi - \psi_i) \cdot \mathbf{J}_n), \quad (2.79)$$

which can be handled in a straightforward way by employing the box integration method [193]. Consequently, equations (2.76)–(2.78) as well as the *Poisson* equation (2.21) are integrated resulting in

$$\varepsilon \oint_{\partial \mathcal{V}} \nabla \psi \cdot d\mathbf{A} + \int_{\mathcal{V}} \rho dV = 0, \quad (2.80)$$

$$\int_{\mathcal{V}} \frac{\partial n}{\partial t} dV - \frac{1}{q} \oint_{\partial \mathcal{V}} \mathbf{J}_n \cdot d\mathbf{A} = 0, \quad (2.81)$$

$$C_1 \int_{\mathcal{V}} \frac{\partial nT}{\partial t} dV + \oint_{\partial \mathcal{V}} \mathbf{S}_n \cdot d\mathbf{A} + \oint_{\partial \mathcal{V}} (\psi - \psi_i) \mathbf{J}_n \cdot d\mathbf{A} + C_1 \int_{\mathcal{V}} n \frac{T - T_L}{\tau_1} dV = 0, \quad (2.82)$$

$$C_2 \int_{\mathcal{V}} \frac{\partial nT^2\beta}{\partial t} dV + \oint_{\partial \mathcal{V}} \mathbf{K}_n \cdot d\mathbf{A} - 2q \oint_{\partial \mathcal{V}} (\psi - \psi_i) \mathbf{S}_n \cdot d\mathbf{A} + C_2 \int_{\mathcal{V}} n \frac{T^2\beta - T_L^2}{\tau_2} dV = 0. \quad (2.83)$$

Finally, the discretized equations are written implicitly as control functions:

$$F_{\psi,i} = \varepsilon \sum_j \frac{\psi_j - \psi_i}{d_{ij}} A_{ij} + \rho_i V_i = 0, \quad (2.84)$$

$$F_{n,i} = \frac{\partial n}{\partial t} V_i - \frac{1}{q} \sum_j J_{n,ij} A_{ij} = 0, \quad (2.85)$$

$$F_{T,i} = C_1 \frac{\partial nT}{\partial t} V_i + \sum_j S_{n,ij} A_{ij} + \sum_j (\psi_j - \psi_i) J_{n,ij} A_{ij} + n C_1 \frac{T - T_L}{\tau_1} V_i = 0, \quad (2.86)$$

$$F_{\beta,i} = C_2 \frac{\partial nT^2\beta}{\partial t} V_i + \sum_j K_{n,ij} A_{ij} - 2q \sum_j (\psi_j - \psi_i) S_{n,ij} A_{ij} + n C_2 \frac{T^2\beta - T_L^2}{\tau_2} V_i = 0, \quad (2.87)$$

with d_{ij} as the distance between grid point P_i and P_j , A_{ij} as the interface area between the respective domains, and V_i as the volume of the domain. $J_{n,ij}$, $S_{n,ij}$, and $K_{n,ij}$ are the projections of the fluxes \mathbf{J}_n , \mathbf{S}_n , and \mathbf{K}_n onto the edge \mathbf{e}_{ij} , respectively, and are evaluated at the center point of this edge. For the grid point P_i a general control function for the quantity x is implicitly given as

$$F_{x,i}^S = \sum_j F_{x,ij} + G_i = 0, \quad (2.88)$$

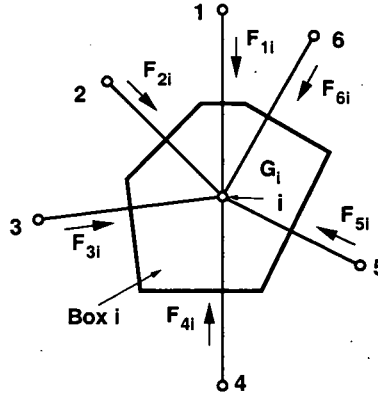


Figure 2.1: Box i with six neighbors

where j runs over all neighboring grid points in the same segment, $F_{x_{ij}}$ is the flux between points i and j , and G_i is the source term (see Figure 2.1).

Grid points on the boundary ∂V are defined as having neighbor grid points in other segments. Thus, (2.88) does not represent the complete control function F_{x_i} , since all contributions of fluxes into the contact or the other segment are missing. For that reason, the information for these boxes has to be completed by taking the boundary conditions into account. Common boundary conditions are the *Dirichlet* condition, which specifies the solution on the boundary ∂V , the *Neumann* condition, which specifies the normal derivative, and the linear combination of these conditions giving an intermediate type:

$$\mathbf{n} \cdot \nabla x + \sigma x = \delta. \quad (2.89)$$

Generally, the form of these conditions depends on the respective boundary models, and the conditions depend in turn on the interior information. For that reason, the equation assembly is often performed in a coupled way, causing complicated modules. For instance, it is absolutely necessary to differentiate between interior and boundary points. Considering a general tetrahedron, there exist many kinds of boundary points (depending on the number of edges involved), which have to be treated separately. This leads to a complicated implementation of the models and can make simplifications necessary. Thus, due to organizational and implementational issues this form of coupling should be avoided.

More complex models with exponential interdependence between the solution variables such as thermionic field emission interface conditions [185, 201] have also been implemented.

A method has been under development to implement segment models calculating the interior fluxes and their derivatives independently from the boundary models. The segment models do not have to differentiate the point type, they do not even have to care about the boundary model used. The assembly system is responsible for combining all relevant contributions by using the information given by the boundary models.

2.2.1 Interface Conditions

To account for complex interface conditions, grid points located at the boundary of the segments (see Figure 2.2a) have three values, one for each segment (see Figure 2.2b) and a third point located directly at the interface which can be used to formulate more complicated interface conditions like for example, interface charges. However, to simplify notation these interface values will be omitted in the discussion and only the two interface points, i and i' , are used.

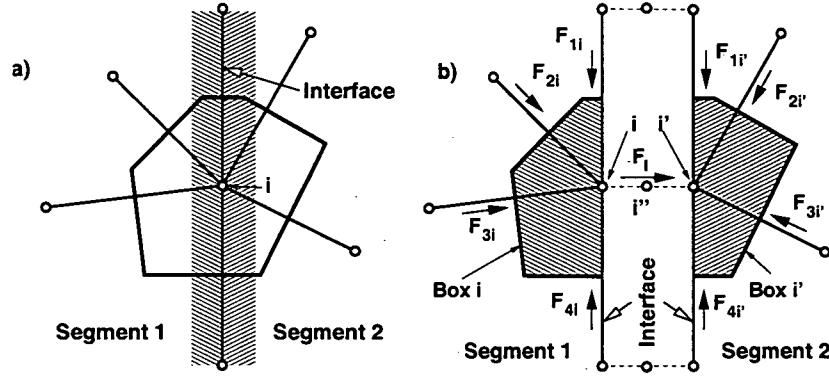


Figure 2.2: Splitting of interface points: Interface points as given in a) are split into three different points having the same geometrical coordinates b)

Basically, the two (incomplete) equations $F_{x_i}^S$ and $F_{x_{i'}}^S$ are completed by adding the missing boundary fluxes $F_{x_{i,i'}}$:

$$F_{x_i} = F_{x_i}^S + F_{x_{i,i'}} = 0, \quad (2.90)$$

$$F_{x_{i'}} = F_{x_{i'}}^S - F_{x_{i,i'}} = 0. \quad (2.91)$$

The intermediate type of interfaces (2.89) and thus also the two other types of interfaces are generally given in linearized form by:

$$\alpha(x_i - \beta x_{i'} + \gamma) = F_{x_{i,i'}}. \quad (2.92)$$

α , β , and γ are linearized coefficients, $F_{x_{i,i'}}$ represents the flux over the interface. The three types of interfaces differ in the magnitude of α .

In the case of an arbitrary splitting of a homogenous region into different segments, the boundary models have to ensure that the simulation results remain unchanged. By adding (2.91) to (2.90), the box of grid point P_i can be completed and the boundary flux is eliminated. The merged box is now valid for both grid points, for that reason the respective equation cannot only be used for grid point P_i , but also for $P_{i'}$.

Whereas the segment models assemble the so-called segment matrix, the interface models are responsible for assembling and configuring the interface system consisting of a boundary and special-purpose transformation matrix. New equations based on (2.92) can be introduced into the boundary matrix without any limitations on α , thus from 0 (*Neumann*) to ∞ (*Dirichlet*). The interface models are also responsible for configuring the transformation matrix to combine the segment and boundary matrix correctly. Depending on the interface type there are two possibilities:

- *Dirichlet* boundaries are characterized by $\alpha \rightarrow \infty$. Thus, the implicit equation $x_i = \beta x_{i'} - \gamma$ can be used as substitute equation. As these equations are normally not diagonally dominant they have a negative impact on the condition number and are configured to be preeliminated (see Section 4.8).
- For the other types (explicit boundary conditions) the boundary flux is simply added to the segment fluxes. In the case of a large α , the transformation matrix could be used to scale the entries by $1/\alpha$ because of the preconditioner used in the solver module (see Section 4.10).

Note, that all interface-dependent information is administrated by the respective interface model only.

As an additional feature, the transformation matrix can be used to calculate several independent boundary quantities by combining the specific boundary value with the segment entries (also in the case of *Dirichlet*

boundaries). For example, the dielectric flux over the interface is calculated as $\sum_i F_{x_i}^S$ and introduced as a solution variable because some interface models require the cross-interface electric field strength to model effects such as tunnel processes. Calculation of the normal electric field is thus trivial. Note that this is not the case when the normal component of the electric field E_n has to be calculated using neighboring points when unstructured two- or three-dimensional grids are used.

See Figure 2.3 for an illustration of these concepts. The transformations are set up to combine the various segment contributions with the boundary system.

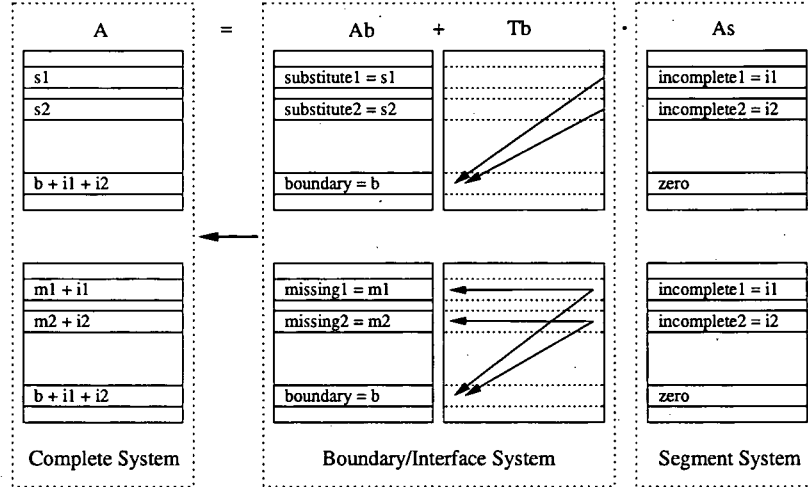


Figure 2.3: The complete equations are a combination of the boundary and the segment system. This combination is controlled by the transformation matrix and depends on the interface type. In the upper figure, the case for *Dirichlet* boundary conditions including substitute equations is shown, in the lower figure for all other cases.

2.2.2 Boundary Conditions

Contacts are handled in a similar way to interfaces. However, in the contact segment there is only one variable available for each solution quantity (x_C). Note that contacts are represented by spatial multi-dimensional segments. Furthermore, all fluxes over the boundary are handled as additional solution variables F_C (for example, contact charge Q_C for the *Poisson* equation, contact electron current I_{nC} for the electron continuity equations, or H_C as the contact heat flow).

For explicit boundary conditions one gets

$$F_{x_i} = F_{x_i}^S + F_{x_i,C} = 0, \quad (2.93)$$

$$F_{F_C} = F_C + \sum_i F_{x_i}^S = 0, \quad (2.94)$$

with i running over all segment grid points.

At *Schottky* contacts explicit boundary conditions apply. The semiconductor contact potential ψ_s is fixed and given as the difference of the metal quasi-*Fermi* level (which is specified by the contact voltage ψ_C) and the metal workfunction difference potential ψ_{wf} .

$$\psi_s = \psi_C - \psi_{wf}, \quad \text{where} \quad \psi_{wf} = -\frac{E_w}{q}. \quad (2.95)$$

The difference between the conduction band energy E_C and the metal workfunction energy gives the workfunction difference energy E_w which is the barrier height of the *Schottky* contact.

For *Dirichlet* boundary conditions one gets

$$F_{x_i} = x_C - h(x_i) = 0, \quad (2.96)$$

$$F_{F_C} = F_C + \sum_i F_{x_i}^S = 0. \quad (2.97)$$

Here, x_C is the boundary value of the quantity, which is a solution variable, whereas (2.97) is used as constitutive relation for the actual flow over the boundary F_C .

For example, at *Ohmic* contacts simple *Dirichlet* boundary conditions apply. The contact potential ψ_s , the carrier contact concentrations n_s and p_s , and in the energy-transport simulation case, the contact carrier temperatures T_n and T_p are fixed. The metal quasi-*Fermi* level (which is specified by the contact voltage ψ_C) is equal to the semiconductor quasi-*Fermi* level. With the constant built-in potential ψ_{bi} (calculated after [65]), the contact potential at the semiconductor boundary reads

$$\psi_s = \psi_C + \psi_{bi}. \quad (2.98)$$

For *Neumann* boundaries the flux over the boundary is zero hence the equation assembled by the segment model is already complete.

2.3 Steady-State and Transient Analysis

This section gives an overview about the steady-state and transient simulation modes including a discussion of the nonlinear solution technique. For the steady-state analysis, the discretized equations (2.21), (2.22), and (2.23) can be symbolically written as:

$$F_\psi(\mathbf{w}) = 0, \quad (2.99)$$

$$F_n(\mathbf{w}) = 0, \quad (2.100)$$

$$F_p(\mathbf{w}) = 0, \quad (2.101)$$

with

$$\mathbf{w} = \begin{pmatrix} \psi \\ n \\ p \end{pmatrix}. \quad (2.102)$$

Note that for the sake of simplification, the vectors of the discretized quantities and equations are not explicitly noted. The resulting discretized problem is then usually solved by a damped *Newton* method which requires the solution of a linear equation system at each step. The result of the steady-state simulation mode is the operating point, which is a prerequisite for any subsequent transient or small-signal simulation.

2.3.1 Solving the Nonlinear System

As the resulting discretized equation system is still nonlinear, the solution \mathbf{w}^* , which is assumed to exist is obtained by applying a linearization technique. The nonlinear problem can be defined as

$$\mathbf{F}(\mathbf{w}) = 0, \quad (2.103)$$

with

$$\mathbf{F}(\mathbf{w}) = \begin{pmatrix} F_\psi(\mathbf{w}) \\ F_n(\mathbf{w}) \\ F_p(\mathbf{w}) \end{pmatrix} \quad (2.104)$$

Most iterative methods are based on a fixpoint equation $\mathbf{w} = \mathbf{M}(\mathbf{w})$, where $\mathbf{M}(\mathbf{w})$ is constructed in such a way that the fixpoint \mathbf{w}^* is a solution of that equation [193]. During the iteration the error between the current solution \mathbf{w}^k of the k -th iteration step and \mathbf{w}^* converges to zero, if specific properties and requirements on the initial guess \mathbf{w}^0 are fulfilled. With a neighborhood $S(\mathbf{w}^*)$, $\mathbf{M}(\mathbf{w}) \in S$, $\mathbf{w} \in S$ and a constant $\alpha \in [0, 1]$, the iteration will converge for any $\mathbf{w}^0 \in S(\mathbf{w}^*)$ to \mathbf{w}^* , if

$$\|\mathbf{M}(\mathbf{w}) - \mathbf{w}^*\| \leq \alpha \|\mathbf{w} - \mathbf{w}^*\|, \forall \mathbf{w} \in S. \quad (2.105)$$

Then, $\mathbf{M}(\mathbf{w})$ is a so-called contractive mapping, and the locally convergent iteration does converge for any $\mathbf{w}^0 \in S$ to \mathbf{w}^* . In order to fulfill (2.105) it is assumed that the *Frechet* derivative $\mathbf{M}'(\mathbf{w})$ exists at the fixpoint \mathbf{w}^* and that its eigenvalues are less than one in modulus [193]. According to the *Ostrowski* theorem [243], $\mathbf{M}(\mathbf{w})$ is contractive if the spectral radius $\rho(\mathbf{M}'(\mathbf{w})) < 1$, which is the maximal modulus of all eigenvalues of $\mathbf{M}(\mathbf{w})$. If $\mathbf{M}'(\mathbf{w})$ exists such that

$$\lim_{h \rightarrow 0} \frac{\mathbf{M}(\mathbf{w}^* + h) - \mathbf{M}(\mathbf{w}) - \mathbf{M}'(\mathbf{w})h}{\|h\|} = 0, \quad (2.106)$$

$\mathbf{M}'(\mathbf{w})$ is the *Frechet* derivative. The most prominent of such linearization techniques is the *Newton* method [136] based on a *Taylor* series expansion. It can be written in the form [193, 201]:

$$-\mathbf{J}(\mathbf{w}) \cdot \begin{pmatrix} \Delta\psi \\ \Delta n \\ \Delta p \end{pmatrix} = \mathbf{f}(\mathbf{w}), \quad (2.107)$$

where $\mathbf{J}(\mathbf{w})$ is the *Jacobian* matrix with the first partial derivatives [136]:

$$\mathbf{J}(\mathbf{w}) = \begin{pmatrix} \frac{\partial F_\psi}{\partial \psi} & \frac{\partial F_\psi}{\partial n} & \frac{\partial F_\psi}{\partial p} \\ \frac{\partial F_n}{\partial \psi} & \frac{\partial F_n}{\partial n} & \frac{\partial F_n}{\partial p} \\ \frac{\partial F_p}{\partial \psi} & \frac{\partial F_p}{\partial n} & \frac{\partial F_p}{\partial p} \end{pmatrix} \quad (2.108)$$

As the iteration can be rewritten in the form

$$\mathbf{w}^{k+1} = \mathbf{M}(\mathbf{w}^k) = \mathbf{w}^k - \mathbf{J}^{-1} \mathbf{F}(\mathbf{w}^k), \quad (2.109)$$

the *Frechet* derivative evaluated at \mathbf{w}^* equals $\mathbf{I} - \mathbf{J}(\mathbf{w}^*)\mathbf{F}'(\mathbf{w}^*)$ resulting in $\rho = 0$. Hence, the *Newton* method converges for all \mathbf{w}^0 sufficiently close to \mathbf{w}^* .

It is important to note that \mathbf{J} must only be an approximation of the *Frechet* derivative, which follows from the derivation of ρ [193]. Furthermore, in order to enlarge the radius of convergence and thus improve the convergence behavior of the *Newton* approximation, the couplings between the equations can be reduced, especially during the first steps of the iteration. Before the update norm, that is the infinity norm of the update vectors of all quantities, has fallen below a specified value, the derivatives as shown in Table 2.1 are normally ignored. Besides the driving force for electrons and holes in the drift-diffusion model, F_n and F_p , and the tunneling current density J_{tun} , all quantities are already known from the previous sections. Note that for the sake of simplification just the symbols are given without vector notations.

	ψ	n	p	T_n	T_p	β_n	β_p	F_n	F_p
J_n				$\frac{\partial J_n}{\partial T_n}$				$\frac{\partial J_n}{\partial F_n}$	
J_p					$\frac{\partial J_p}{\partial T_p}$				$\frac{\partial J_p}{\partial F_p}$
S_n	$\frac{\partial S_n}{\partial \psi}$			$\frac{\partial S_n}{\partial T_n}$		$\frac{\partial S_n}{\partial \beta_n}$			
S_p	$\frac{\partial S_p}{\partial \psi}$				$\frac{\partial S_p}{\partial T_p}$		$\frac{\partial S_p}{\partial \beta_p}$		
R	$\frac{\partial R}{\partial \psi}$	$\frac{\partial R}{\partial n}$	$\frac{\partial R}{\partial p}$						
μ_n								$\frac{\partial \mu_n}{\partial F_n}$	
μ_p									$\frac{\partial \mu_p}{\partial F_p}$
J_{tun}	$\frac{\partial J_{\text{tun}}}{\partial \psi}$	$\frac{\partial J_{\text{tun}}}{\partial n}$	$\frac{\partial J_{\text{tun}}}{\partial p}$	$\frac{\partial J_{\text{tun}}}{\partial T_n}$	$\frac{\partial J_{\text{tun}}}{\partial T_p}$				

Table 2.1: Ignored derivatives during the first steps of the *Newton* iteration.

The linear equation system for the k -th iteration step looks like:

$$-\mathbf{J}^k \mathbf{x}^{k+1} = \mathbf{F}(\mathbf{w}^k). \quad (2.110)$$

The right-hand-side vector $\mathbf{F}(\mathbf{w}^k)$ is the residual and \mathbf{x}^{k+1} is the update and correction vector. This solution vector \mathbf{x} of the linear equation system is used to calculate the next solution \mathbf{w} of the *Newton* approximation:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mathbf{x}^{k+1}. \quad (2.111)$$

To avoid overshoot of the solution and to extend the local convergence of the method several damping schemes suggested by *Deuffhard* [50] or *Bank and Rose* [15] can be used to calculate a damping factor D . The damped update reads

$$\mathbf{w}^{k+1} = \mathbf{w}^k + D \mathbf{x}^{k+1}. \quad (2.112)$$

Investigations have shown that damping based on the potential delivers the most promising results [65]:

$$D = \frac{1 + \delta \cdot \ln \frac{\|\mathbf{x}_\psi\|}{V_{\text{th}}}}{1 + \delta \cdot \left(\frac{\|\mathbf{x}_\psi\|}{V_{\text{th}}} - 1 \right)}, \quad \text{with } 0 \leq \delta, \quad (2.113)$$

where δ is an adjustable parameter of the damping scheme, \mathbf{x}_ψ the update norm of the potential sub-vector, and V_{th} the thermal voltage. Larger δ yields more logarithm-like damping. The potential damping scheme avoids the expensive evaluation of the right-hand-side vector, which is for example required for the scheme of *Bank and Rose* [15].

2.3.2 Transient Simulation

The transient problem arises if the boundary condition for the electrostatic potential or the contact currents becomes time-dependent. Hence, the partial time derivatives of the carrier concentrations in (2.22) and (2.23) have to be taken into account.

There are several approaches for transient analysis [193], among them are the forward and backward *Euler* approaches. Whereas the former shows significant stability problems, the latter is unconditionally stable for arbitrarily large time steps Δt . However, full backward time differencing requires much computational resources for solving the large nonlinear equation system at each time step, but gives good results. The quality of the results can be measured by the truncation error [146]. Equations (2.21), (2.22) and (2.23), discretized in time and symbolically written, read then at the m -th time step when $m+1$ is to be calculated:

$$F_\psi(\mathbf{w}^{m+1}) = 0, \quad (2.114)$$

$$F_n(\mathbf{w}^{m+1}) = \frac{n^{m+1} - n^m}{\Delta t}, \quad (2.115)$$

$$F_p(\mathbf{w}^{m+1}) = \frac{p^{m+1} - p^m}{\Delta t}. \quad (2.116)$$

From a computational point of view it is to note, that in comparison to the steady-state solution the algebraic equations arising from the time discretization are significantly easier to solve [193]. This has mainly two reasons: first, the partial time derivatives help to stabilize the spatial discretization. Second, the solutions can be used as a good initial guess for the next time step. Furthermore, the equation assembly structures can be reused (see Section 4.12).

2.4 Small-Signal Simulation

Small-signal device simulation is used to extract the relationship between small sinusoidal terminal voltages and currents which are superimposed upon an already calculated steady-state operating point. This relationship depends on the DC operating point and on the frequency. The amplitude of the superimposed signal is considered to be *small* as long as harmonics are not generated within the device.

A small-signal simulation mode can be based on several approaches, some of them which will be shortly discussed in this section in accordance with the well-known overview from [127]. Whereas many of these approaches are based in the time domain and can thus use a transient simulation mode, the S³A approach (*Sinusoidal Steady-State Analysis*) is directly applied in the frequency domain.

In Figure 2.4 a comparison between approaches based in the transient and frequency domain is shown. The time derivatives are usually discretized by a backward *Euler* discretization, and thus a high number of steps has to be performed to achieve sufficient accuracy. For that reason the time consumption is usually reduced by extracting an equivalent circuit using the information of only one frequency.

Fourier decomposition techniques were one of the first choices to characterize AC device behavior [127]. The entries of the admittance matrix are obtained after the *Fourier* transformation of transient current and voltage responses. This allows to employ a transient simulation mode followed by transformation algorithm, for example a fast *Fourier* transformation. The technique is rigorous and universally applicable, but requires much computational resources, as a high number of time steps is required in order to achieve sufficient accuracy in the time and frequency domain.

An alternative are the so-called incremental charge partitioning heuristics. An entry of the capacitance matrix is obtained by $C_{ij} = \Delta Q_i / \Delta V_j$. ΔQ_i is the incremental charge at the contact i . Whereas the

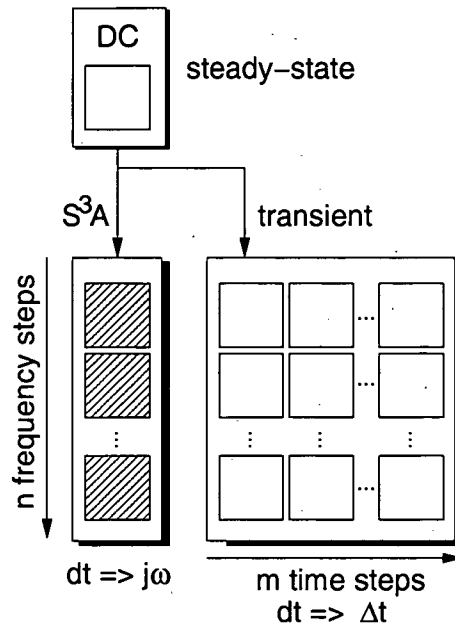


Figure 2.4: Comparison of transient and frequency-domain-based approaches [233]. The dashed rectangles of the S^3A approach symbolize complex-valued equation systems, the other real-valued ones.

results can be accurate and computationally inexpensive, this approach cannot be applied in a general-purpose device simulator. For example the gate capacitances of a MOSFET can be readily computed, since the transient current is solely a displacement current and the integral can be evaluated. The incremental charge is then simply the incremental charge induced at the gate by the voltage perturbation. The charge partitioning technique is heuristic and cannot be generally applied. For specific problems good results can be obtained with small computational resources.

2.4.1 Equivalent Circuits

In order to perform small and large signal simulations, equivalent circuits are frequently extracted and applied. The advantages of these circuits can be summarized as follows: like the compact models used in circuit simulations, they can be evaluated very efficiently. In addition, the values for the circuit elements can be optimized in order to deliver a nearly perfect match with the measurement data used for the calibration. The extraction procedure and the limitation for predefined operating conditions can be regarded as the disadvantages of this approach.

The small-signal modeling of a GaAs heterojunction bipolar transistor is often based on the linear hybrid Π model [74, 187]. The applied model is extended by the separation of inner and outer base resistance and of the base-collector capacitances. An alternative model is the T-model as discussed in [130]. In Figure 2.5 a standard Π -type small-signal equivalent circuit of a HEMT [169] (left) and a T-type eight-element small-signal equivalent circuit of an HBT are shown [160]. Although this approach can be very efficient, inaccurate compact models can endanger the quality of the results.

2.4.2 Sinusoidal Steady-State Analysis

The most rigorous small-signal simulation mode is based on the sinusoidal steady-state analysis (S^3A) approach, which is well-established in the device simulation area [77, 90, 111, 127, 214, 216, 218]. In contrast to the alternative approaches, transient analysis is not employed. The approach is rigorously

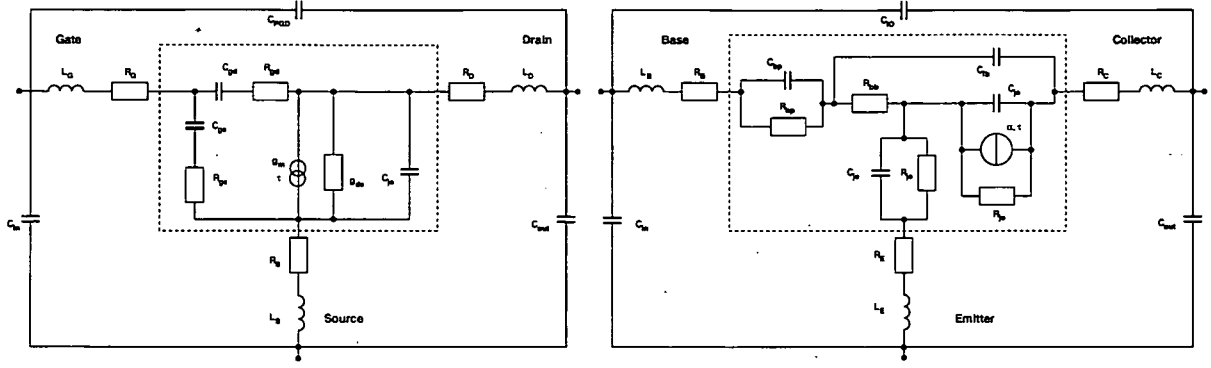


Figure 2.5: These figures [160] show a standard Π -type small-signal equivalent circuit of a HEMT (left) and a T-type eight-element small-signal equivalent circuit of an HBT. The dashed rectangles denote the intrinsic devices, the terminal resistances can be additionally included in the simulation.

correct and can be applied with small computational effort. Its results are very accurate due to the formal linearization of the device. In addition, the analysis is based directly in the frequency domain. Harmonics cannot be considered, because the device is linearized by a *Taylor* series expansion terminated after the linear term. Two properties can be identified why S^3A is both an extremely accurate and very efficient approach:

1. Due to the linearization of the device, no harmonic generation is possible. Hence, no errors related to the perturbation amplitude are possible.
2. Furthermore, this approach is not based on a time discretization and hence no respective discretization error can occur.

2.5 Derivation of the Complex-Valued Small-Signal System

In order to derive the complex-valued small-signal system based on the S^3A approach, the equations (2.21), (2.22), and (2.23) can be symbolically written as [90]:

$$F_\psi(\psi, n, p) = 0, \quad (2.117)$$

$$F_n(\psi, n, p) = \frac{\partial G_n(n(t))}{\partial t}, \quad (2.118)$$

$$F_p(\psi, n, p) = \frac{\partial G_p(p(t))}{\partial t}. \quad (2.119)$$

For the derivation of the linear small-signal system, the following time-dependent functions

$$\psi(t) = \psi_0 + \underline{\psi}e^{j\omega t}, \quad (2.120)$$

$$n(t) = n_0 + \underline{n}e^{j\omega t}, \quad (2.121)$$

$$p(t) = p_0 + \underline{p}e^{j\omega t}, \quad (2.122)$$

are substituted into the equations (2.117), (2.118), and (2.119) to obtain

$$F_\psi(\psi_0 + \underline{\psi}e^{j\omega t}, n_0 + \underline{n}e^{j\omega t}, p_0 + \underline{p}e^{j\omega t}) = 0, \quad (2.123)$$

$$F_n(\psi_0 + \underline{\psi}e^{j\omega t}, n_0 + \underline{n}e^{j\omega t}, p_0 + \underline{p}e^{j\omega t}) - \frac{\partial G_n(n_0 + \underline{n}e^{j\omega t})}{\partial t} = 0, \quad (2.124)$$

$$F_p(\psi_0 + \underline{\psi}e^{j\omega t}, n_0 + \underline{n}e^{j\omega t}, p_0 + \underline{p}e^{j\omega t}) - \frac{\partial G_p(p_0 + \underline{p}e^{j\omega t})}{\partial t} = 0. \quad (2.125)$$

The zero subscript stands for the steady-state operating point of the device. The time derivatives of the G functions in (2.118) and (2.119) are calculated using the chain rule as follows:

$$\frac{\partial G_n(n(t))}{\partial t} = \frac{\partial G_n(n(t))}{\partial n} \frac{\partial n}{\partial t} = \frac{\partial G_n(n_0 + \underline{n}e^{j\omega t})}{\partial n} \underline{n}j\omega e^{j\omega t}, \quad (2.126)$$

$$\frac{\partial G_p(p(t))}{\partial t} = \frac{\partial G_p(p(t))}{\partial p} \frac{\partial p}{\partial t} = \frac{\partial G_p(p_0 + \underline{p}e^{j\omega t})}{\partial p} \underline{p}j\omega e^{j\omega t}. \quad (2.127)$$

To obtain the linearized version of (2.117), (2.118), and (2.119), a *Taylor* series expansion is performed, which is generally defined for a function with several unknowns as follows [29]:

$$F(x + h, y + k, \dots) = F(x, y, \dots) + \sum_{i=1}^N \frac{1}{i!} \left(h \frac{\partial}{\partial x} + k \frac{\partial}{\partial y} + \dots \right)^i F(x, y, \dots) + R_N, \quad (2.128)$$

where R_N is the remainder term of the approximation. This formula can be specialized for the three unknowns ψ , n , and p , and termination after the linear part ($i = 1$) resulting in the small-signal approximation:

$$F(\psi + h, n + k, p + l) = F(\psi, n, p) + \left(h \frac{\partial}{\partial \psi} + k \frac{\partial}{\partial n} + l \frac{\partial}{\partial p} \right) F(\psi, n, p) + R_1. \quad (2.129)$$

By applying (2.129) for the functions F of equations (2.123), (2.124), and (2.125) with the functions (2.120), (2.121), and (2.122), the following approximation is derived:

$$\begin{aligned} F(\psi_0 + \underline{\psi}e^{j\omega t}, n_0 + \underline{n}e^{j\omega t}, p_0 + \underline{p}e^{j\omega t}) &= \\ &= \underbrace{F(\psi_0, n_0, p_0)}_{\text{dc solution}} + \frac{\partial [\underline{\psi}e^{j\omega t} F]}{\partial \psi} + \frac{\partial [\underline{n}e^{j\omega t} F]}{\partial n} + \frac{\partial [\underline{p}e^{j\omega t} F]}{\partial p}. \end{aligned} \quad (2.130)$$

If this approximation is substituted into equations (2.123), (2.124), and (2.125), the resulting equation system reads

$$0 = \underbrace{F_\psi(\psi_0, n_0, p_0)}_{\text{dc solution}} + e^{j\omega t} \left[\underline{\psi} \frac{\partial F_\psi}{\partial \psi} + \underline{n} \frac{\partial F_\psi}{\partial n} + \underline{p} \frac{\partial F_\psi}{\partial p} \right], \quad (2.131)$$

$$0 = \underbrace{F_n(\psi_0, n_0, p_0)}_{\text{dc solution}} + e^{j\omega t} \left[\underline{\psi} \frac{\partial F_n}{\partial \psi} + \underline{n} \left(\frac{\partial F_n}{\partial n} - j\omega \frac{\partial G_n}{\partial n} \right) + \underline{p} \frac{\partial F_n}{\partial p} \right], \quad (2.132)$$

$$0 = \underbrace{F_p(\psi_0, n_0, p_0)}_{\text{dc solution}} + e^{j\omega t} \left[\underline{\psi} \frac{\partial F_p}{\partial \psi} + \underline{n} \frac{\partial F_p}{\partial n} + \underline{p} \left(\frac{\partial F_p}{\partial p} - j\omega \frac{\partial G_p}{\partial p} \right) \right]. \quad (2.133)$$

According to equations (2.99), (2.100), and (2.101), the steady-state solutions are equal to zero. This linear equation system can be written in the following matrix notation [127], where the subscript dc emphasizes that all derivatives are evaluated at the steady-state operating point:

$$\begin{bmatrix} \frac{\partial F_\psi}{\partial \psi} & \frac{\partial F_\psi}{\partial n} & \frac{\partial F_\psi}{\partial p} \\ \frac{\partial F_n}{\partial \psi} & \frac{\partial F_n}{\partial n} - j\omega \frac{\partial G_n}{\partial n} & \frac{\partial F_n}{\partial p} \\ \frac{\partial F_p}{\partial \psi} & \frac{\partial F_p}{\partial n} & \frac{\partial F_p}{\partial p} - j\omega \frac{\partial G_p}{\partial p} \end{bmatrix}_{dc} \begin{bmatrix} \underline{\psi} \\ \underline{n} \\ \underline{p} \end{bmatrix} = 0. \quad (2.134)$$

The real-valued part of the system matrix equals the *Jacobian* matrix as shown in (2.108). For that reason, the assembly of this part of the matrix can be performed in exactly the same way as for steady-state analysis. The complex-valued contributions are then added by the transient models. As a consequence, the real-valued part can be stored during a frequency stepping since only the complex-valued part is modified.

As already discussed in Section 2.3.1, the exact *Jacobian* matrix can be replaced by a simpler matrix. Since the solution of the S³A small-signal system does not involve an iterative process but is based on the linearization of the device, it is absolutely necessary to take all derivatives into account. If the simulator offers the possibility to skip several entries (for example iteration schemes, see Section 3.6.4), it has to be ensured that the complex-valued system matrices contain all necessary entries. The same problem occurs while iteratively solving the complex-valued equation system. The preconditioner has to be configured in such a way that no entries are removed (see Section 5.2.5).

2.5.1 Boundary Conditions and the Complete System

In contrast to the *Newton* procedure, the right-hand-side vector is mostly zero. The small-signal *Neumann* boundary conditions are the same as during steady-state analysis. The frequency-independent boundary conditions for n and p are zero, because the derivatives vanish in the *Taylor* series expansion of the contact control function. Real- or complex-valued *Dirichlet* boundary conditions for ψ can be used to excite the system.

After the complete complex-valued linear system for a small-signal analysis is assembled, the following parts can be identified:

$$-[\mathbf{J} + j\omega \mathbf{C}] \underline{\mathbf{x}} = \underline{\mathbf{b}}, \quad (2.135)$$

where \mathbf{J} is the real-valued *Jacobian* as shown in (2.108) and (2.134), \mathbf{C} contains the contributions of the time-dependent nonlinear G functions, $\underline{\mathbf{x}}$ is the complex-valued solution vector

$$\underline{\mathbf{x}} = \begin{pmatrix} \underline{\psi} \\ \underline{n} \\ \underline{p} \end{pmatrix}, \quad (2.136)$$

and $\underline{\mathbf{b}}$ the complex-valued right-hand-side vector.

2.5.2 Extension for Higher-Order Transport Models

The small-signal system for the energy-transport transport model reads as follows:

$$-\underline{\mathbf{A}}_{HD} \begin{bmatrix} \underline{\psi} & \underline{n} & \underline{p} & \underline{T}_n & \underline{T}_p \end{bmatrix}^T = \underline{\mathbf{B}}_{HD}. \quad (2.137)$$

with

$$\underline{\mathbf{A}}_{\text{HD}} = \begin{bmatrix} \frac{\partial F_\psi}{\partial \psi} & \frac{\partial F_\psi}{\partial n} & \frac{\partial F_\psi}{\partial p} & \frac{\partial F_\psi}{\partial T_n} & \frac{\partial F_\psi}{\partial T_p} \\ \frac{\partial F_n}{\partial \psi} - j\omega \frac{\partial G_n}{\partial n} & \frac{\partial F_n}{\partial n} & \frac{\partial F_n}{\partial p} & \frac{\partial F_n}{\partial T_n} & \frac{\partial F_n}{\partial T_p} \\ \frac{\partial F_p}{\partial \psi} & \frac{\partial F_p}{\partial n} & \frac{\partial F_p}{\partial p} - j\omega \frac{\partial G_p}{\partial p} & \frac{\partial F_p}{\partial T_n} & \frac{\partial F_p}{\partial T_p} \\ \frac{\partial F_{T_n}}{\partial \psi} & \frac{\partial F_{T_n}}{\partial n} & \frac{\partial F_{T_n}}{\partial p} & \frac{\partial F_{T_n}}{\partial T_n} - j\omega \frac{\partial G_{T_n}}{\partial T_n} & \frac{\partial F_{T_n}}{\partial T_p} \\ \frac{\partial F_{T_p}}{\partial \psi} & \frac{\partial F_{T_p}}{\partial n} & \frac{\partial F_{T_p}}{\partial p} & \frac{\partial F_{T_p}}{\partial T_n} & \frac{\partial F_{T_p}}{\partial T_p} - j\omega \frac{\partial G_n}{\partial n} \end{bmatrix} \quad (2.138)$$

With matrix $\underline{\mathbf{A}}_{\text{HD}}$ from (2.138), the small-signal system for the six moments transport models is eventually given by

$$- \begin{bmatrix} \underline{\mathbf{A}}_{\text{HD}} & \underline{\mathbf{S}}_{\text{I}} \\ \underline{\mathbf{S}}_{\text{II}} & \underline{\mathbf{S}}_{\text{III}} \end{bmatrix} \begin{bmatrix} \underline{\psi} & \underline{n} & \underline{p} & \underline{T_n} & \underline{T_p} & \underline{\beta_n} & \underline{\beta_p} \end{bmatrix}^T = \underline{\mathbf{B}}_{\text{SM}}, \quad (2.139)$$

with

$$\underline{\mathbf{S}}_{\text{I}}^T = \begin{bmatrix} \frac{\partial F_\psi}{\partial \beta_n} & \frac{\partial F_n}{\partial \beta_n} & \frac{\partial F_p}{\partial \beta_n} & \frac{\partial F_{T_n}}{\partial \beta_n} & \frac{\partial F_{T_p}}{\partial \beta_n} \\ \frac{\partial F_\psi}{\partial \beta_p} & \frac{\partial F_n}{\partial \beta_p} & \frac{\partial F_p}{\partial \beta_p} & \frac{\partial F_{T_n}}{\partial \beta_p} & \frac{\partial F_{T_p}}{\partial \beta_p} \end{bmatrix}, \quad (2.140)$$

$$\underline{\mathbf{S}}_{\text{II}} = \begin{bmatrix} \frac{\partial F_{\beta_n}}{\partial \psi} & \frac{\partial F_{\beta_n}}{\partial n} & \frac{\partial F_{\beta_n}}{\partial p} & \frac{\partial F_{\beta_n}}{\partial T_n} & \frac{\partial F_{\beta_n}}{\partial T_p} \\ \frac{\partial F_{\beta_p}}{\partial \psi} & \frac{\partial F_{\beta_p}}{\partial n} & \frac{\partial F_{\beta_p}}{\partial p} & \frac{\partial F_{\beta_p}}{\partial T_n} & \frac{\partial F_{\beta_p}}{\partial T_p} \end{bmatrix}, \quad (2.141)$$

$$\underline{\mathbf{S}}_{\text{III}} = \begin{bmatrix} \frac{\partial F_{\beta_n}}{\partial \beta_n} - j\omega \frac{\partial G_{\beta_n}}{\partial \beta_n} & \frac{\partial F_{\beta_p}}{\partial \beta_n} \\ \frac{\partial F_{\beta_n}}{\partial \beta_p} & \frac{\partial F_{\beta_p}}{\partial \beta_p} - j\omega \frac{\partial G_{\beta_p}}{\partial \beta_p} \end{bmatrix} \quad (2.142)$$

2.6 Concluding Remarks

If a device, for example a power RF transistor, has to be characterized under conditions where a harmonic generation takes place, the small-signal approximation has to be replaced by large-signal simulations [241] such as the nonlinear harmonic balance, envelope, or shooting-method simulation [220]. In contrast to small-signal simulation, the device is then not linearized and the harmonic generation as depicted in Figure 2.6 can be taken into account. Rather than ignoring the harmonics and thus getting results which actually violate physical conditions, the additional voltage and current vectors are included in the simulation. Simulators for Harmonic Balance simulations are for example proposed in [6, 8, 206].

An alternative are *Volterra* methods which are similar to the small-signal systems besides the fact that they take more than the first term of the *Taylor* series expansion into account [125].

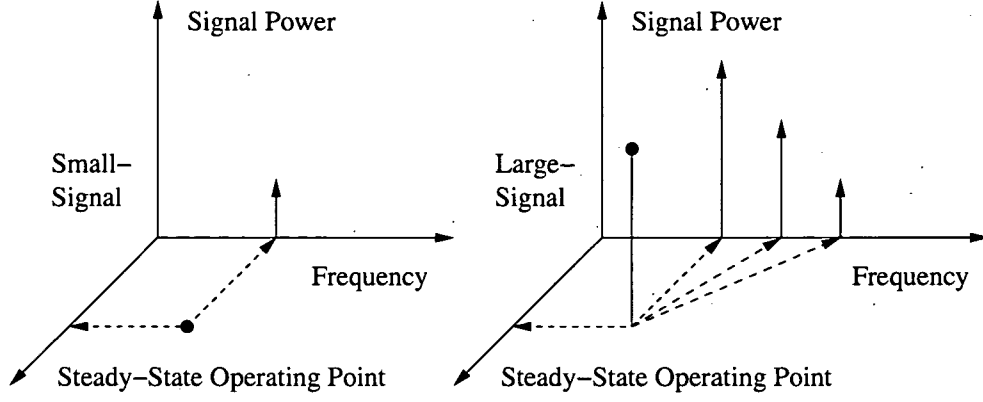


Figure 2.6: Comparison of small-signal and large-signal simulations. In the case of too high RF power, harmonics are generated within the non-linear devices. These additional voltage and current vectors cannot be taken into account by linearized small-signal approximations [45].

2.6.1 Harmonic Balance

Three important parameters for RF and power devices are the power gain and output level, the power added efficiency as the ratio between the difference of the output and input power and the steady-state input power, as well as distortion [241]. These quantities can be efficiently extracted by the Harmonic Balance method.

Harmonic Balance is a highly accurate analysis technique which is also based in the frequency domain [220]. For that reason the large-signal steady-state performance of a semiconductor device can be much faster obtained than using a computationally more expensive transient analysis [177]. Harmonic Balance can be used for simulating analog RF problems which are naturally handled in the frequency domain. Among the results are harmonic distortion, third-order intercept points, gain compression, phase noise, and intermodulation products in non-linear circuits [5]. As the input stimulus is assumed to consist of steady-state sinusoids, the solution of the Harmonic Balance simulation is a sum of steady-state sinusoids including the input frequencies additionally to significant harmonics and mixing terms.

The Harmonic Balance approach consists of an expansion of each device variable, that is for example ϕ , n , and p , as a *Fourier* series and solves for the coefficients X_{n0} , $\underline{X}_{n,h} = X_{n,h}^R + jX_{n,h}^I$ given as [177]

$$x_n(t) = X_{n0} + \sum_{h=1}^H (X_{n,h}^R \cos(\omega_h(t)) - X_{n,h}^I \sin(\omega_h(t))) = \Re \left[\sum_{h=1}^H (\underline{X}_{n,h} \exp(j\omega_h t)) \right], \quad (2.143)$$

with H as the number of harmonics. Thus, the main objective is to calculate the coefficients in such a way that the nonlinear partial differential equations as derived above are satisfied within a given tolerance. Higher-order frequency components are neglected as they are assumed to be irrelevant. This results in the characteristic trade-off between accuracy and performance, because smaller numbers of H can be faster solved while being less accurate. As the *Fourier* coefficients are calculated, the time-domain signal can be obtained from the *Fourier* expansion [177].

The time domain state vector x of length $N = 3K + Q$ is given by [221]

$$\mathbf{x} = [\psi_1, n_1, p_1, \dots, \psi_K, n_K, p_K, v_1, \dots, v_Q]^T, \quad (2.144)$$

with K internal nodes and Q terminals of the linear network. For the three quantities ψ , n , and p , the lengths and the dimension of the equation system is given by $(3K + Q)(2H + 1)$, because $(2H + 1)$ values are required to represent each state variable waveform $x_n(t)$. Thus, the memory consumption can be regarded as the major disadvantage of that approach.

The input stimulus is usually an either one- or two-tone sinusoid (more than two sinusoids are normally already too expensive) with the form [241]:

$$v(t) = A_1 \cos(\Omega_1 t + \phi_1) + A_2 \cos(\Omega_2 t + \phi_2), \quad (2.145)$$

where A_2 is zero for an one-tone simulation. In an intermodulation distortion test, the frequencies Ω_1 and Ω_2 are very closely spaced [221].

The benefits of Harmonic Balance are that this approach is able to extract steady-state performance of a device in the presence of potentially longer time constant phenomena, while it avoids excessive number of time steps in multitone analyses [177]. A particular strength of the Harmonic Balance method is that it can include models for linear components which are directly based in the frequency domain. Such models are required for lossy or dispersive transmission lines [125].

For the solution of the resulting equation systems, special algorithms and numerics, such as the restarted version of GMRES, are required [220]. Furthermore, difficulties are reported in case of strongly nonlinear circuits or circuits containing signals with abrupt transitions [125].

Finally it has to be noted that the linearity is an important prerequisite for advanced systems as higher-order harmonics at the output can be neglected. In a nonlinear device harmonics are generated even for pure sinusoidal inputs. An important figure of merit is the so-called third-order inter-modulation intercept point [171], at which the output amplitude of the third harmonic equals that of the fundamental one [135]. For that reason, the third harmonic intercept voltage parameter is considered to give a good indication of the device linearity. For MOS devices, V_{IP3} can be approximately obtained from steady-state characteristics [239]

$$V_{IP3} = \sqrt{\frac{24g_m}{g_{m3}}}, \quad (2.146)$$

with the transconductance g_m and its third derivative g_{m3} .

2.6.2 Complex-Valued Equation Systems

The S^3A approach obviously requires the ability for solving complex-valued linear equation systems, for which several methods can be applied. One possibility is to reuse a real-valued assembly and solver system, split the real and imaginary part as suggested in [127] and solve both systems separately. If the complex-valued matrix has the form

$$-[\mathbf{J} + j\mathbf{C}] \underline{\mathbf{x}} = \underline{\mathbf{b}}, \quad (2.147)$$

the real-valued system looks as follows:

$$-\begin{bmatrix} \mathbf{J} & -\mathbf{C} \\ \mathbf{C} & \mathbf{J} \end{bmatrix} \begin{bmatrix} \mathbf{x}_R \\ \mathbf{x}_I \end{bmatrix} = \begin{bmatrix} \mathbf{b}_R \\ \mathbf{b}_I \end{bmatrix}, \quad (2.148)$$

where

$$\underline{\mathbf{x}} = [\mathbf{x}_R + j\mathbf{x}_I], \quad (2.149)$$

$$\underline{\mathbf{b}} = [\mathbf{b}_R + j\mathbf{b}_I]. \quad (2.150)$$

In terms of memory consumption this approach has, especially for three-dimensional simulations, severe disadvantages, since the dimension doubles causing a fourfold-sized system matrix. Thus, the computational effort for factorization can be excessive. In [127] iterative methods like block-*Gauss-Seidel* or block-SOR are suggested for reducing this effort. In Chapter 4 and Chapter 5, complex-valued assembly and solver systems are discussed. In addition, the results of an evaluation of these approaches is given.

Chapter 3

Small-Signal AC Analysis

The small-signal analysis mode of MINIMOS-NT is based on the S³A approach presented in Chapter 2 and discussed in Section 2.5. After a conventional DC step at a given operating point the simulator is switched to the simulation mode in the frequency domain, where the device is excited by a complex sinusoidal perturbation of infinitesimal amplitude. In comparison to transient methods performance is better (only one equation system per frequency step has to be solved) and the results are more accurate as time discretization is not involved.

3.1 Introduction

The major objective of small-signal simulations is to extract various figures of merit of the devices or networks. Two-port parameter sets are useful design aids provided by manufactures for high-frequency transistors. In addition, they are used to extract the cut-off frequency or the maximum oscillation frequency, which are further required for characterization of the devices. The following basic definitions are generally used [164]:

Impedance	$\underline{Z} = R + jX,$	$Z = \underline{Z} = \sqrt{R^2 + X^2},$
Resistance	$R = \Re(\underline{Z}),$	$R = \frac{G}{Y^2}$
Reactance	$X = \Im(\underline{Z}),$	$X = -\frac{B}{Y^2}$
Admittance	$\underline{Y} = \frac{1}{\underline{Z}} = G + jB,$	$Y = \underline{Y} = \sqrt{G^2 + B^2},$
Conductance	$G = \Re(\underline{Y}),$	$G = \frac{R}{Z^2}$
Susceptance	$B = \Im(\underline{Y}),$	$B = -\frac{X}{Z^2}$

with the imaginary unit $j = \sqrt{-1}$.

Furthermore, a distinction between intrinsic and extrinsic parameters has to be made. Measured admittance and scattering parameters are normally different from the simulation results. If the errors are systematically introduced by the measurement environment, it is useful to represent the device as embedded in a parasitic equivalent circuit. Hence, the intrinsic parameters represent the de-embedded device. Based on a standard parasitic equivalent circuit, the simulator can take all parasitics into account and can calculate also extrinsic two-port parameters.

In order to clarify the notation of the various parameters hereafter, the following definition shall be used: \underline{Y} refers to a specific complex-valued admittance value, however for the sake of readability Y as in Y -parameters refers to a general admittance quantity. A lower case letter for an intrinsic parameter is sometimes used to distinguish from extrinsic parameters written with an upper case letter. For the sake of clarification, this distinction is not made in this work and the context of the respective parameter is always clearly indicated.

3.1.1 Admittance, Impedance, Hybrid Matrices and Parameters

An N -port device/network can be represented by several matrices or parameter sets. At low frequencies, these are usually Y -, Z -, H -, or A -matrices or parameters, because they can be easily measured with open or short circuits.

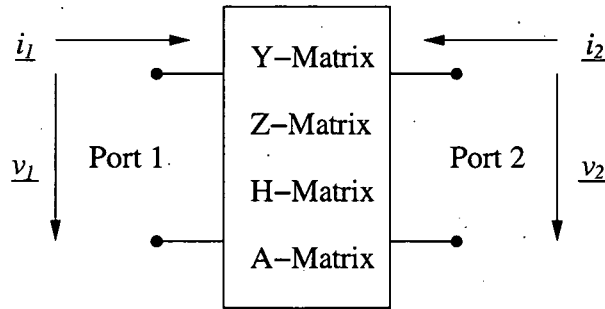


Figure 3.1: Voltages and currents of a two-port device/network.

For a two-port device/network as depicted in Figure 3.1, they are defined as follows:

$$\begin{pmatrix} \underline{i_1} \\ \underline{i_2} \end{pmatrix} = \begin{pmatrix} \underline{Y_{11}} & \underline{Y_{12}} \\ \underline{Y_{21}} & \underline{Y_{22}} \end{pmatrix} \begin{pmatrix} \underline{v_1} \\ \underline{v_2} \end{pmatrix}, \quad (3.1)$$

$$\begin{pmatrix} \underline{v_1} \\ \underline{v_2} \end{pmatrix} = \begin{pmatrix} \underline{Z_{11}} & \underline{Z_{12}} \\ \underline{Z_{21}} & \underline{Z_{22}} \end{pmatrix} \begin{pmatrix} \underline{i_1} \\ \underline{i_2} \end{pmatrix}, \quad (3.2)$$

$$\begin{pmatrix} \underline{v_1} \\ \underline{i_2} \end{pmatrix} = \begin{pmatrix} \underline{H_{11}} & \underline{H_{12}} \\ \underline{H_{21}} & \underline{H_{22}} \end{pmatrix} \begin{pmatrix} \underline{i_1} \\ \underline{v_2} \end{pmatrix}, \quad (3.3)$$

$$\begin{pmatrix} \underline{v_1} \\ \underline{i_2} \end{pmatrix} = \begin{pmatrix} \underline{A_{11}} & \underline{A_{12}} \\ \underline{A_{21}} & \underline{A_{22}} \end{pmatrix} \begin{pmatrix} \underline{v_1} \\ -\underline{i_2} \end{pmatrix}. \quad (3.4)$$

Hybrid (H -) parameters are often used for the description of active devices such as transistors. Like Y -parameters, they are difficult to measure at high frequencies. The absolute value of the $\underline{H_{21}}$ parameter is used to characterize f_T , where the current gain has dropped to unity. The so-called chain or A -parameters, sometimes also referred to as ABCD-parameters, are useful for cascaded circuit topologies, since these parameters allow matrix multiplications of the single elements:

$$\begin{pmatrix} \underline{A} & \underline{B} \\ \underline{C} & \underline{D} \end{pmatrix}_{\text{all}} = \begin{pmatrix} \underline{A} & \underline{B} \\ \underline{C} & \underline{D} \end{pmatrix}_1 \cdot \begin{pmatrix} \underline{A} & \underline{B} \\ \underline{C} & \underline{D} \end{pmatrix}_2 = \begin{pmatrix} (\underline{A_1 A_2} + \underline{B_1 C_2}) & (\underline{A_1 B_2} + \underline{B_1 D_2}) \\ (\underline{C_1 A_2} + \underline{D_1 C_2}) & (\underline{C_1 B_2} + \underline{D_1 D_2}) \end{pmatrix} \quad (3.5)$$

Measurements of Z -parameters require (analogously to Y -parameters) open circuit connections which may act as short circuits at RF frequencies due to stray capacitances.

3.1.2 Scattering Parameters

Advanced devices are operated under their originally intended environment conditions for higher frequencies above 100 MHz. A steady-state bias is applied to all terminals superimposed by an additional RF excitation. The sinusoidal currents and voltages of all terminals with magnitude and phase should be measured. This would normally involve Z-, Y-, H- and A-parameters of the linear two-port theory, which are able to completely describe the electrical properties of the device. Unfortunately, three main problems can be identified [45]:

1. At high frequencies it is problematic to directly measure voltages and currents. Voltmeters and current probes cannot be simply connected due to the probe impedances and problems related to a correct positioning. It is necessary to apply either AC-wise open or short circuits as part of the Z-, Y-, and H-parameter measurement. Especially at high frequencies short and open circuits are difficult to obtain because of lead inductances and capacitances. Such measurements typically require tuning stubs which are adjusted at each measurement frequency. This is not only inconvenient and cumbersome, but a tuning stub shunting the input or output may cause active devices to oscillate or self-destruct and thus prevent measurements or making them invalid [98].
2. The voltages and currents depend on the length of the cable used to connect the device under test to the measurement setup. Hence, the measured values depend on the position along the cable.
3. At high frequencies, true open and short termination of the device is hard to achieve.

To avoid these drawbacks, S-parameters can be used to characterize a two-port network, which are related to the scattering and reflection of traveling waves (power or equivalent voltage waves). Instead of open and short termination, the ports are terminated by a cable of the characteristic impedance Z_0 . The device is so embedded into a transmission line of a certain characteristic impedance Z_0 , usually $50\ \Omega$. This scattering and reflection is comparable to optical lenses which transmit and reflect a certain amount of light. The traveling waves can so be interpreted in terms of normalized voltage and current amplitudes. S-parameters are the complex-valued reflection coefficients at each port and complex-valued transmission coefficients of the equivalent voltage wave between each pair of ports. Hence, an N -port device or circuit with N^2 S-parameters has N reflection coefficients and $N^2 - N$ transmission coefficients. An additional advantage is the fact, that traveling waves do not vary in magnitude at points along a lossless transmission line. In contrast to other parameter measurements, S-parameters can be measured at some distance from the measurement transducers [98].

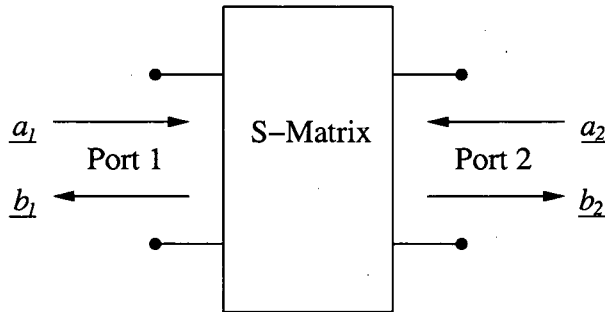


Figure 3.2: Traveling waves at a two-port device/network.

S-parameters provide detailed information on the linear behavior of the two-port. As shown in Figure 3.2, they are basically defined as follows:

$$\begin{pmatrix} \underline{b_1} \\ \underline{b_2} \end{pmatrix} = \begin{pmatrix} \underline{S_{11}} & \underline{S_{12}} \\ \underline{S_{21}} & \underline{S_{22}} \end{pmatrix} \begin{pmatrix} \underline{a_1} \\ \underline{a_2} \end{pmatrix}, \quad (3.6)$$

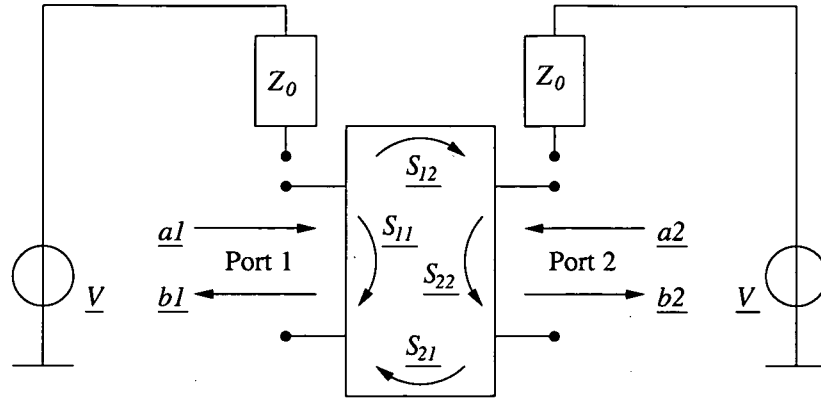


Figure 3.3: Definition of S-parameters.

with \underline{a}_i and \underline{b}_i as traveling waves. As illustrated in Figure 3.3, the S-parameters are defined as:

$$\begin{pmatrix} \underline{b}_1 \\ \underline{b}_2 \end{pmatrix} = \begin{pmatrix} \underline{S}_{11} & \underline{S}_{12} \\ \underline{S}_{21} & \underline{S}_{22} \end{pmatrix} \begin{pmatrix} \underline{a}_1 \\ \underline{a}_2 \end{pmatrix}, \quad (3.7)$$

with

$$\begin{aligned} |\underline{a}_i|^2 & \dots \text{ power wave traveling towards the two-port,} \\ |\underline{b}_i|^2 & \dots \text{ power wave reflected from the two-port,} \\ |\underline{S}_{11}|^2 = \frac{|\underline{b}_1|^2}{|\underline{a}_1|^2} \Big|_{a_2=0} & \dots \text{ power reflected from port 1,} \\ |\underline{S}_{12}|^2 = \frac{|\underline{b}_1|^2}{|\underline{a}_2|^2} \Big|_{a_1=0} & \dots \text{ power transmitted from port 1 to port 2,} \\ |\underline{S}_{21}|^2 = \frac{|\underline{b}_2|^2}{|\underline{a}_1|^2} \Big|_{a_2=0} & \dots \text{ power transmitted from port 2 to port 1,} \\ |\underline{S}_{22}|^2 = \frac{|\underline{b}_2|^2}{|\underline{a}_2|^2} \Big|_{a_1=0} & \dots \text{ power reflected from port 2.} \end{aligned}$$

The parameters \underline{S}_{11} and \underline{S}_{21} are obtained by terminating the port 2 by a perfect Z_0 load ($|\underline{a}_2|^2 = 0$) and measuring the incident, reflected, and transmitted signals. Parameter \underline{S}_{11} is equivalent to the complex-valued input reflection coefficient (impedance) of the device and \underline{S}_{21} is the complex-valued forward transmission coefficient. In turn, while terminating port 1 by a perfect load ($|\underline{a}_1|^2 = 0$), the parameters \underline{S}_{22} and \underline{S}_{12} are measured, which are the complex-valued output reflection coefficient (output impedance) or reverse transmission coefficient, respectively. The accuracy of the measurements strongly depends on the quality of the terminations. If the perfect load cannot be established, the S-parameter definition requirements are not met.

The magnitudes of the reflection parameters \underline{S}_{11} and \underline{S}_{22} , which are always smaller than 1, can be interpreted as follows: In the case of -1, all voltages are inverted and reflected (0Ω), zero means perfect impedance matching and no reflections (50Ω), and at +1 all voltages are reflected ($\infty \Omega$).

In the case of an active amplification, the magnitudes of the transfer parameter \underline{S}_{21} and reverse parameter \underline{S}_{12} can be larger than 1. They can also start at a negative value in the case of a phase inversion. If the magnitude is zero, there is no signal transmission, between 0 and +1 a damping takes place, at +1 there is a unity gain transmission and above +1 an input signal amplification.

S_{ii} on the real-valued axis characterize *Ohmic* resistors. S_{ii} above the real-valued axis characterize inductive impedances. S_{ii} below the real-valued axis characterize capacitive impedances. S_{ii} curves in the *Smith* chart are followed clock-wise to increasing frequencies.

S_{ij} curves in the polar chart are followed clock-wise to increasing frequencies.

3.1.3 Extraction of the Cut-Off and Maximum Oscillation Frequency

The cut-off frequency f_T and maximum oscillation frequency f_{max} are the most important figures of merit for the frequency characteristics of microwave transistors. They are often used to emphasize the superiority of newly developed semiconductors or technologies. For example, as a rule of thumb, the operating frequency of a transistor, sometimes referred as f_{op} should be ten times smaller than f_T [189]. Thus, extraction of these parameters is a commonly performed simulation task usually done by small-signal simulations.

The cut-off frequency f_T is the frequency at which the gain or amplification is unity, thus the absolute value of the short circuit current gain H_{21} equals unity:

$$|H_{21}|_{f=f_T} = 1. \quad (3.8)$$

H_{21} is defined as the ratio of the small-signal output current to the input current of a transistor with short-circuited output. For a bipolar junction transistor, H_{21} basically characterizes the ratio between the small-signal collector current i_C and the small-signal base current i_B . For a MOS transistor, a similar ratio regarding the small-signal drain and gate currents can be specified:

$$\beta_{bipolar} = \frac{|i_C|}{|i_B|}, \quad (3.9)$$

$$\beta_{mos} = \frac{|i_D|}{|i_G|}. \quad (3.10)$$

The cut-off frequency is normally extracted for various operating points. Thus, the peak f_T value is the highest frequency for this range of operating points. See the left side of Figure 3.4 for a typical curve. To extract such a complete curve I_C or I_D (or V_B or V_G respectively) is stepped. Hence, two stepping variables (see Appendix B) are necessary to obtain f_T : one for the steady-state operating point and one for the frequency. Whereas the operating point is a matter of ordinary stepping functions, there are several approaches for frequency stepping:

- The frequency can be simply increased until β is smaller than one. Then, an interpolation algorithm is used to obtain the frequency for which $\beta = 1$.
- This so-called unity current gain frequency can be calculated by extrapolation of $|H_{21}|$, because for a conventional transistor $|H_{21}|$ drops with a slope of -20 dB/dec or -6 dB/oct at higher frequencies. This roll-off slope of a one-pole low-pass is depicted on the right side of Figure 3.4. Thus, the frequency is increased until the -20 dB/dec region of the curve is reached. An extrapolation yields the cut-off frequency. Measurement equipments often use this approach to obtain f_T , since they are normally not able to measure such high frequencies as required for today's cut-off frequencies. However, this method depends on the frequency chosen, that means which frequency assures the validity of the single-pole approximation [172].
- For simulation purposes it is very inconvenient to run a simulator and a post-processing script in the end. For that reason a conditional stepping approach was developed to use a mathematical iteration algorithm to approximate f_T for a given accuracy (see Section 3.3.3).

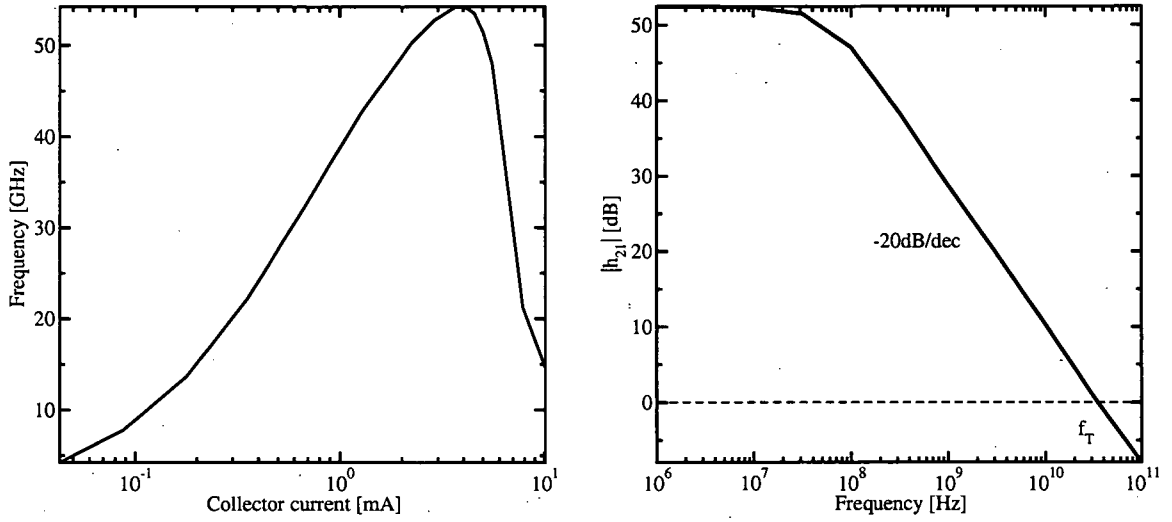


Figure 3.4: Complete f_T curve of a bipolar junction transistor (left). Slope of the absolute value of the short circuit current gain and the cut-off frequency at the unity gain point at $I_C = 0.865$ mA (right).

The second important RF figure of merit is the maximum oscillation frequency f_{\max} , which is related to the frequency at which the device power gain equals unity. The value of f_{\max} can be determined in two ways. The first one is based on the unilateral power gain U as defined by *Mason*

$$U(f) = \frac{\left| \frac{S_{21}}{S_{12}} - 1 \right|^2}{2k \left| \frac{S_{21}}{S_{12}} \right| - 2\Re \left(\frac{S_{21}}{S_{12}} \right)}, \quad (3.11)$$

where k is *Kurokawa's* stability factor [94] defined as

$$k(f) = \frac{1 - |S_{11}|^2 - |S_{22}|^2 + |S_{11}S_{22} - S_{12}S_{21}|^2}{2 |S_{12}| |S_{21}|}. \quad (3.12)$$

Therefore, f_{\max} is the maximum frequency at which the transistor still provides a power gain [189]. An ideal oscillator would still be expected to operate at this frequency, hence the name *maximum oscillation frequency*. Like the short-circuit current gain H_{21} , U drops with a slope of -20 dB/dec.

The second way to determine f_{\max} , which is not entirely correct [189], is based on the maximum available gain (MAG) and the maximum stable gain (MSG). Whereas MAG shows no definite slope, MSG drops with -10 dB/dec.

f_{\max} does not have to be necessarily larger than f_T . Generally, transistors have useful power gains up to f_{\max} , that above they cannot be used as power amplifiers any more. However, the importance of f_T and f_{\max} depends on the specific application. Thus, there is no general answer whether f_{\max} should be prioritized over f_T . Both figures should be as high as possible, and manufactures often strive for $f_T \approx f_{\max}$ in order to enter many different markets for their transistors [189].

3.2 Overview of the Minimos-NT Small-Signal Capabilities

As depicted in the overview in Figure 3.5, the small-signal capabilities which were implemented into MINIMOS-NT can be divided into two branches:

1. Standard small-signal capabilities: general complex-valued amplitudes can be applied to an arbitrary number of terminals of the device. In combination with the frequency setting, the simulator can be used to calculate the respective complex-valued terminal currents or voltages. For example, this feature can be efficiently applied to extract the cut-off frequency f_T of the simulated device.
2. Extended small-signal capabilities: for several simulation tasks such as the S-parameter extraction, it is necessary to calculate the complete admittance matrix of the device, which can be obtained by applying the unity voltage once to each terminal. Since the device is linearized, it does not matter which voltage is applied at the terminal (besides of numerical considerations). For the special case of the unity voltage, there is no subsequent division necessary to obtain the admittance value. By using the standard capabilities, the calculation of the matrix can be a cumbersome task, because appropriate stepping variables have to be defined and the respective post-processing for collecting these values has to be implemented and configured. Furthermore, speed-up features suitable for this kind of simulation task cannot be employed. See for example the discussion of the multiple right-hand-side feature in Section 4.3. For that reason, a feature for automatically calculating the complete admittance matrix is provided. Based on these features, various sets of intrinsic and extrinsic parameters can be extracted.

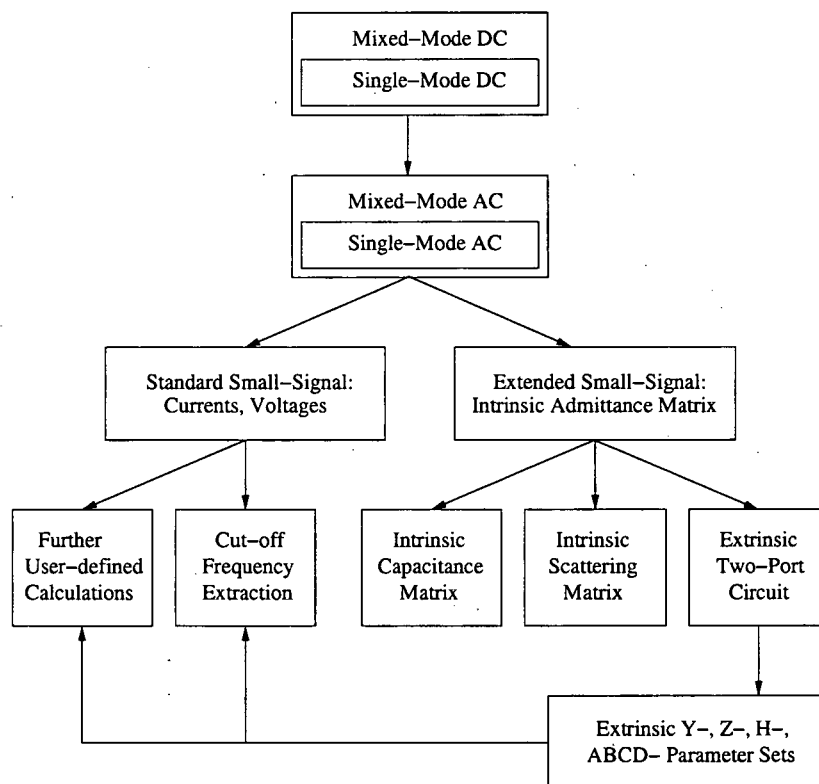


Figure 3.5: Overview of the MINIMOS-NT small-signal capabilities.

The main simulator output of the extended branch is the intrinsic (de-embedded) admittance matrix. Based on this matrix, the intrinsic capacitance and intrinsic scattering matrix can be calculated. As an optional

feature these parameters can be transformed into extrinsic parameters in order to take parasitics introduced by the measurement set-up into account.

Both branches finally provide comfortable features to extract the cut-off frequency f_T or additional figures of merit such as the maximum oscillation frequency f_{max} . In addition, the user is able to inquire all small-signal simulation results (see Appendix A.6) and can employ the input-deck built-in functions to perform further calculations, which can then be written to the simulator output file. Thus, as a matter of improved usability no subsequent post-processing is necessary for the most common simulation tasks.

As already seen in Figure 3.5, all small-signal capabilities are available also for mixed-mode simulations, which are discussed in the second part of this chapter.

3.3 Standard Single-Mode AC Analysis

This section deals with the basic settings and configuration of the small-signal simulation mode. In addition, some information about the internal implementation is given. As transient and small-signal analysis are related simulation modes, their setup and configuration is very similar (see Appendix A.1).

3.3.1 Introduction

The AC steps are based on a fully converged steady-state operating point. Therefore, it has to be ensured, that a DC simulation is started prior to any AC step. Note that this is also implicitly the case for transient simulations. During initialization of a transient or small-signal simulation, the internally used time or frequency representation is updated. Due to the backward *Euler* discretization the transient simulation mode requires the difference between two time points. Thus, in addition to the current time point also the last point has to be stored.

All models obtain the reciprocal time as input, which is used by the transient models to calculate their time-dependent contributions to the equation systems. Depending on the simulation mode, the variable contains either the reciprocal time difference or the angular frequency. Therefore, it is set as follows:

$$r_{\text{transient}} = \frac{1}{t_0 - t_1}, \quad (3.13)$$

$$r_{\text{ac}} = 2\pi f. \quad (3.14)$$

This value is then passed to each model, indicating a DC step in case of zero and a transient or small-signal step in case of non-zero values. All models are subsequently called to add their contributions to the linear equation system. In order to excite the device, the user can specify the complex-valued amplitude of each terminal of the device.

3.3.2 Simulation Example

These standard small-signal capabilities will be demonstrated on a simple diode as shown in Figure 3.6. In the example discussed below, the capacitances over a wide range of frequencies are extracted. The anode voltage is an additional parameter of the simulation. Refer to Appendix A.3 for further details on the simulation setup.

In addition, the diode was also subject to transient simulations in order to demonstrate the generation of harmonics when the device is excited with large signals. Three different amplitudes A were used in the signal applied at the amplitude:

$$V_{\text{Anode}} = V_0 + A \sin(2\pi ft), \quad (3.15)$$

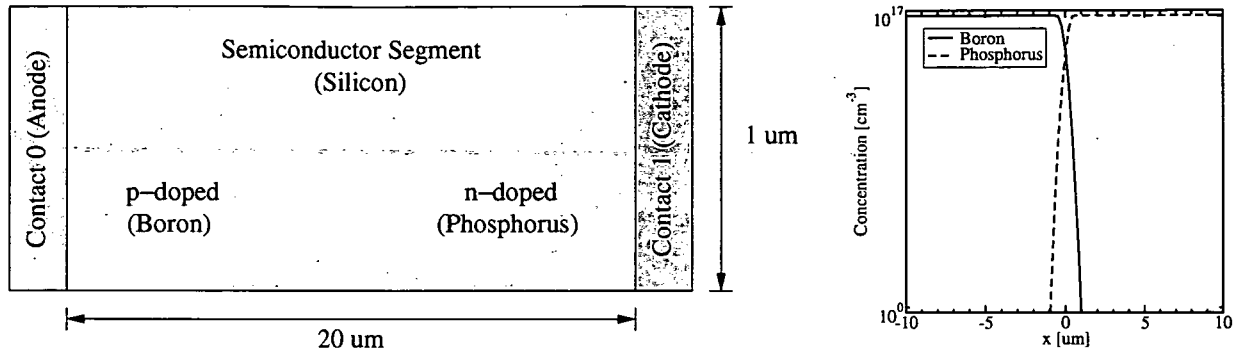


Figure 3.6: Simple diode structure under investigation. The boron concentration in the p-doped half on the left is $1 \times 10^{17} \text{ cm}^{-3}$ equal to the phosphorus concentration in the n-doped part of the diode.

with $V_0 = -0.5 \text{ V}$, $f = 1 \text{ MHz}$, and the time stepped from 0 s to $2 \mu\text{s}$ in 2,048 steps (two periods). The last 1,024 data points of the cathode currents were the input for the *Fast Fourier Transformation* [165]. In the left figure of Figure 3.7, the results of the transient simulations are shown for all four amplitudes $A = 10 \text{ mV}$, $A = 250 \text{ mV}$, $A = 500 \text{ mV}$, and $A = 1 \text{ V}$. Note that the cathode currents are scaled to fit in the same graph. The right figure depicts the results of the *Fast Fourier Transformation*.

The upper left side of Figure 3.8 shows the comparison of the simulation results with those obtained by the commercial simulator DESSIS [111]. The other three figures compare small-signal and transient simulation results of MINIMOS-NT. For an anode voltage of -0.8 V , both modes are used to extract the capacitances and arguments. In the lower right figure the simulation effort and the relative error of the transient results in comparison to those obtained by the small-signal mode are shown. It is important to note that the transient mode requires more than twice the time of the small-signal mode for only ten steps. For 1000 transient steps, the effort is more than 60 times larger than for the small-signal simulation.

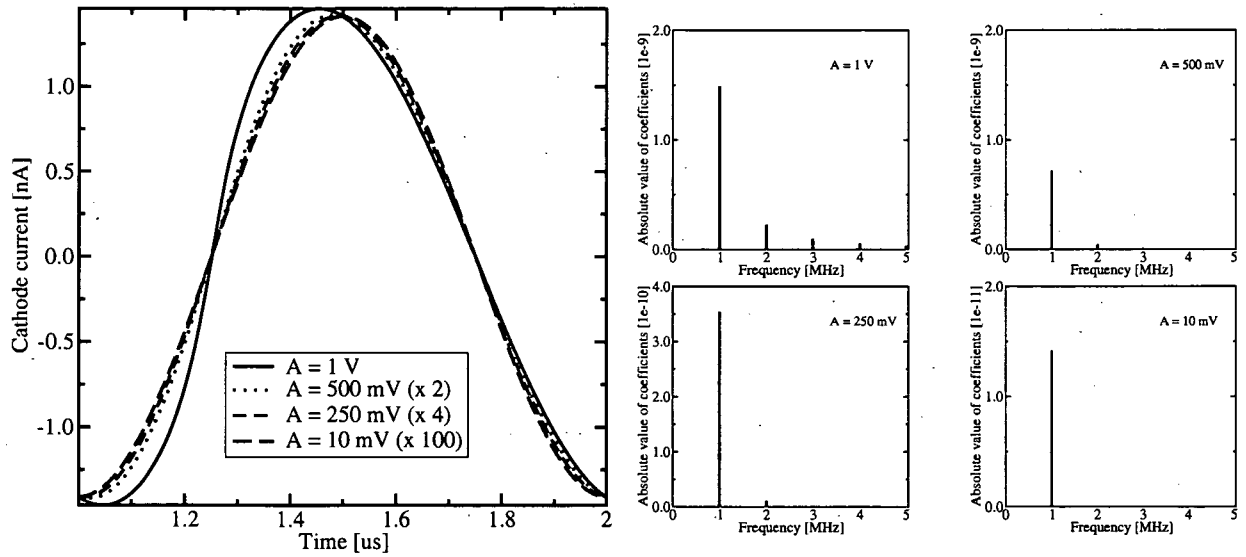


Figure 3.7: The left figure shows the results of the transient simulations for the three amplitudes $A = 10 \text{ mV}$, $A = 250 \text{ mV}$, and $A = 1 \text{ V}$. The small figures depict the results of the *Fast Fourier Transformation* of the respective cathode currents.

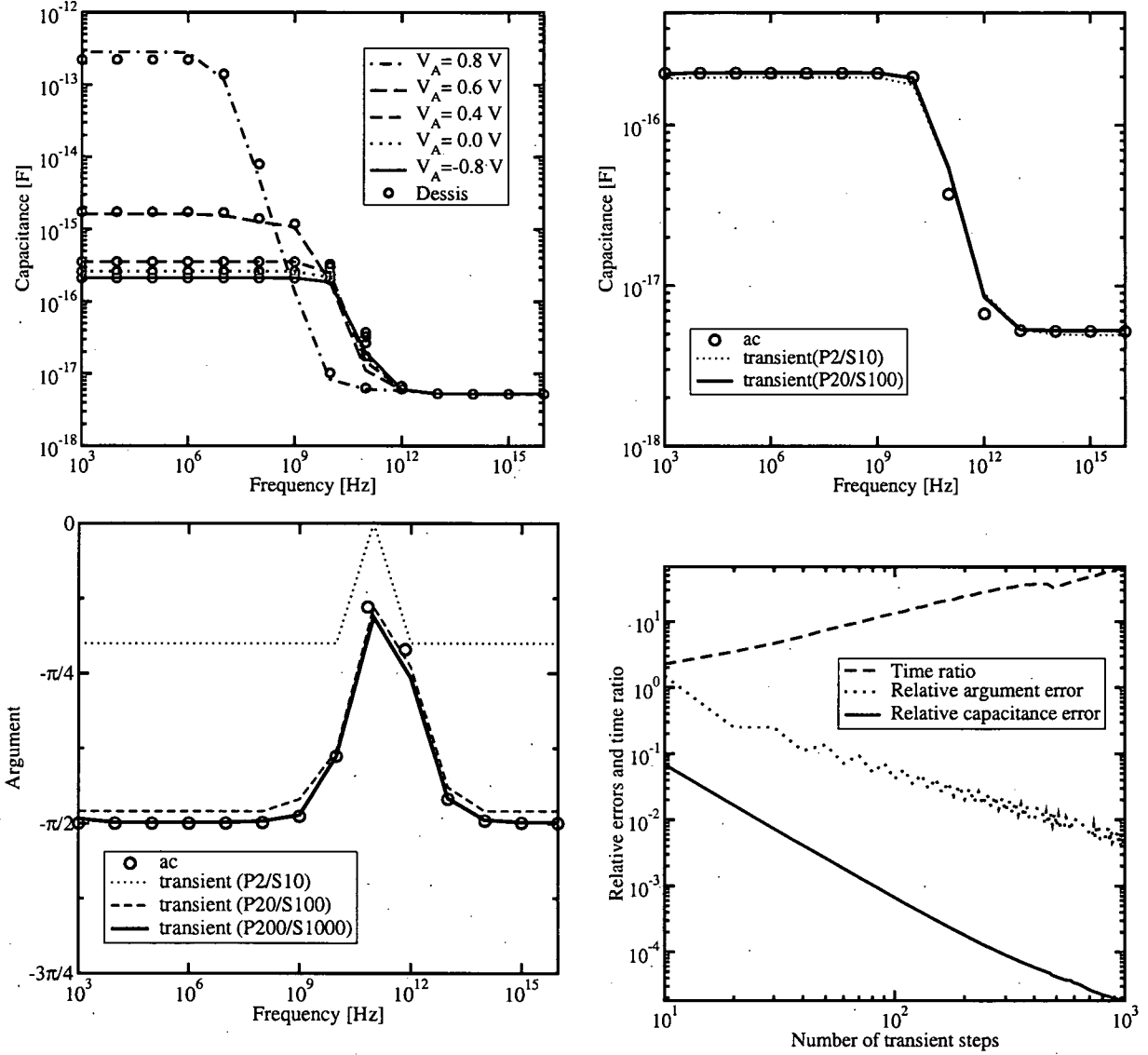


Figure 3.8: The upper left figure compares the small-signal results of MINIMOS-NT and DESSIS. The other three figures compare the small-signal results of MINIMOS-NT with its transient results: In the upper right figure the capacitance versus the frequency is shown. The argument versus the frequency is given in the lower left figure. In both figures, different number of periods (P) and number of steps per period (S) are compared. Finally, the dependence of the relative errors and the time scaled to the small-signal result (time ratio) on the number of transient steps is illustrated in the lower right figure. The number of periods is 2, the frequency is 1 MHz and the transient data is compared with the small-signal results. The trade-off between the reduction of the relative errors and the increasing computational effort can be clearly seen.

3.3.3 Extraction of the Cut-Off Frequency

Frequency stepping can be based on several approaches (see Section 3.1.3). One possibility is to apply a conditional stepping algorithm which is directly stepping to the unity gain point. Such a method is not only faster and more accurate, but additionally makes a post-processing for interpolations or extrapolations obsolete. As for a bipolar transistor β is defined as

$$\beta(f) = \frac{|i_C(f)|}{|i_B(f)|}, \quad (3.16)$$

one can easily obtain a nonlinear function

$$F(\beta) = \beta - 1.0 = 0. \quad (3.17)$$

The objective of the conditional stepping is to find the root of the function (3.17) with an error below a specified value. As such a simulator capability can be applied to extract also other quantities, for example the threshold voltage, it is generally implemented and derived in the following.

Basically, the root of the nonlinear function F is bracketed in the interval $[f_l, f_u]$. The values $F(f_l)$ and $F(f_u)$ have opposite signs and the function is continuous, for that reason one root lies in the interval given. The conditional stepping algorithm is based on the well-known *Regula Falsi* (*False Position*) algorithm. According to [165] the nonlinear function is assumed to be approximately linear in the local region of interest.

False Position converges less rapidly than the related *Secant* method. In contrast to a *Newton* method which requires the derivatives, only the actual function values are used. For that reason, the algorithm can be directly integrated in the respective stepping module (see Appendix B).

The first two iteration steps are used to obtain the initial values $F(f_l)$ and $F(f_u)$ at the specified start boundaries f_l (lower boundary) and f_u . These values must have opposite signs, otherwise the method fails. The next value is found at the intersection of the line connecting these two points. This line is given by L :

$$L(f) = F(f_l) + (f - f_l) \frac{F(f_u) - F(f_l)}{f_u - f_l}. \quad (3.18)$$

Separating the intersection frequency f_I ($L(f) = 0$) yields

$$f_I = f_l - F(f_l) \frac{f_u - f_l}{F(f_u) - F(f_l)}. \quad (3.19)$$

Depending on the sign of $F(f_I)$ either the lower or the upper boundary is replaced by f_I . The loop is terminated if $F(f_I)$ is smaller than the accuracy given or if the maximum iteration count is reached.

For f_T extraction this algorithm seems to be promising as the frequency characteristics are indeed an approximately linear function (in the area of interest) with only one root. However, an important drawback of this approach is the use of fixed boundaries f_l and f_u . First, the frequency range of a f_T extraction can be very wide, especially if more than one operating point is required. Second, the method fails if both values $F(f_l)$ and $F(f_u)$ have the same sign. Third, the method converges slowly if the interval $[f_l, f_u]$ is very wide. To overcome this trade-off between usability and performance, the boundaries f_l and f_u are always reread during the simulation. Furthermore, an automatic adaptation is provided such that too narrow boundaries are automatically extended. This allows on the one hand side the specification of very narrow boundaries, which is important to speed up the convergence, and exempt the user of finding practicable values for f_l and f_u , which can be a cumbersome task.

In order to quantify these considerations, an evaluation of various boundary settings and error values has been performed. The narrowness of the boundaries N in respect of an f_T (which is already known) means, that the lower and upper boundaries equal $f_T \pm N$, respectively ($f_T = 35.8$ GHz). In Figure 3.9 the effort for the extraction of f_T in terms of the required steps is analyzed. The left figure shows the iteration steps for two different sets of boundaries. The right figure compares various boundaries and errors. It can be seen that a significant speed-up can be achieved with narrow boundaries even for very accurate simulations.

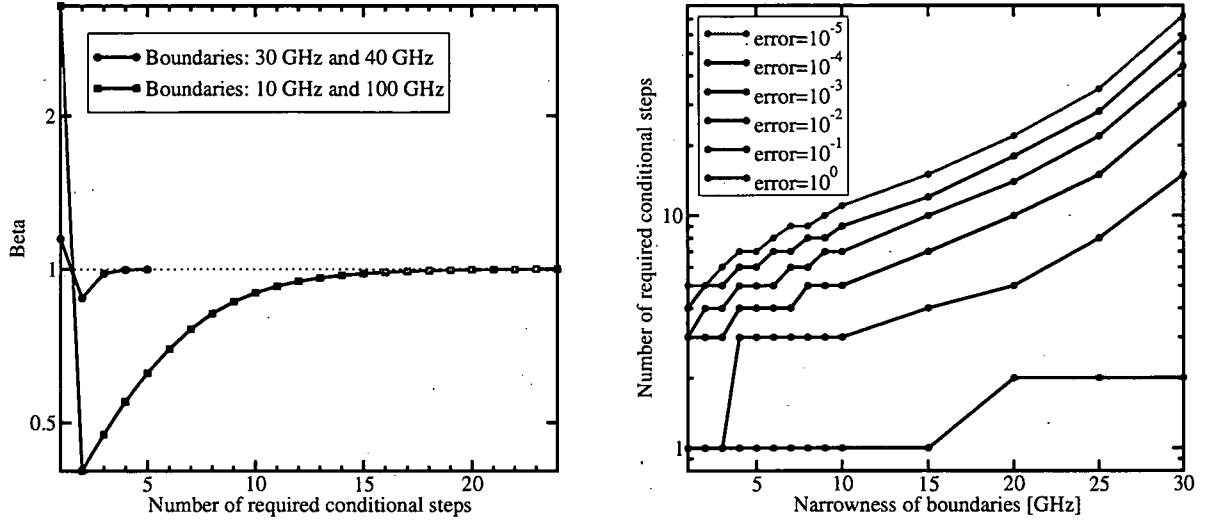


Figure 3.9: These figures analyze the effort for the extraction of f_T by conditional stepping. On the left side, two different sets of lower and upper boundaries are compared for an error value of 10^{-3} . Whereas for the wide boundaries of 10 GHz and 100 GHz 24 steps are required, the narrow boundaries of 30 GHz and 40 GHz reduce this effort down to five steps. On the right side, the number of steps depending on the narrowness of the boundaries for different errors are shown.

In Figure 3.10, extracted cut-off frequencies for MOS transistors with different gate lengths and the three transport models as discussed in Chapter 2 are shown. The results obtained by the small-signal simulation mode of MINIMOS-NT are compared with Monte Carlo data [83] and results based on the quasi-static approximation. The latter are obtained as a result of steady-state simulations. By applying the quasi-static approximation, the following definition can be used for MOS transistors [115, 217]:

$$f_T = \frac{1}{2\pi} \frac{dI_D}{dQ_G} \approx \frac{1}{2\pi} \frac{I_D(V_G + \Delta V) - I_D(V_G - \Delta V)}{Q_G(V_G + \Delta V) - Q_G(V_G - \Delta V)}, \quad (3.20)$$

with the gate voltage V_G varied and all other voltages kept constant. The quasi-static simulation results differ from those of the small-signal mode, which is evident due to the approximation and in consistence with the results reported in [142]. The quasi-static approach, which consists of two steady-state steps, has general performance advantages over the small-signal simulations. If a conditional stepping is applied as discussed before, the number of steps depends on the narrowness of the boundaries. As one can assume that the first steady-state step is a good initial guess for the second one for the quasi-static simulation, the small-signal simulations generally require more computational resources in terms of simulation time than the quasi-static approximation. However, as can be seen in Figure 3.10, the error compared to the exact result can be quite significant.

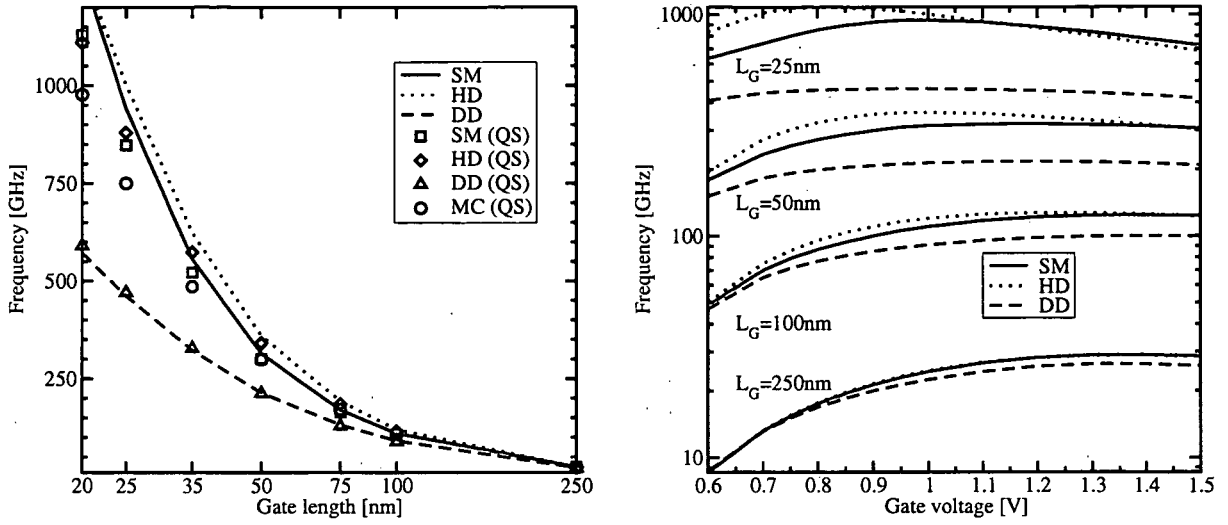


Figure 3.10: In the left figure, the three transport models are compared with quasi-static simulation results of MINIMOS-NT as well as with Monte Carlo data. The right figure shows the cut-off frequency depending on the gate voltage. Whereas for larger devices, the difference is again minimal, the superiority of the six moments transport model for smaller devices can be clearly seen.

3.4 Extended Single-Mode AC Analysis

As already discussed above, by using the standard single-mode AC a general complex-valued amplitude can be applied to an arbitrary number of terminals of the device. However, under these circumstances the calculation and assembly of the complete admittance matrix is a cumbersome task. For that reason the simulator has been extended to provide a feature for the automatic calculation of the complete matrix.

The intrinsic scattering matrix \underline{S} can be calculated by the following analytical formula [244]:

$$\underline{S} = 2(\underline{I} + \underline{Y})^{-1} - \underline{I}, \quad (3.21)$$

where \underline{I} is the identity matrix and \underline{Y} the intrinsic admittance matrix. In order to obtain the usually applied formulae for a two-port network (see Appendix D), the following calculation can be performed:

$$\underline{A} = \underline{I} + \underline{Y} = \begin{pmatrix} 1 + \underline{Y}_{11} & \underline{Y}_{12} \\ \underline{Y}_{21} & 1 + \underline{Y}_{22} \end{pmatrix}, \quad (3.22)$$

$$\underline{A}^{-1} = \frac{1}{D_S} \begin{pmatrix} 1 + \underline{Y}_{22} & -\underline{Y}_{12} \\ -\underline{Y}_{21} & 1 + \underline{Y}_{11} \end{pmatrix}, \quad (3.23)$$

with

$$D_S = (1 + \underline{Y}_{11})(1 + \underline{Y}_{22}) - \underline{Y}_{12}\underline{Y}_{21}. \quad (3.24)$$

After multiplying by two and deducting the identity matrix, the following well-known formulae can be derived:

$$\underline{S}_{11} = ((1 - \underline{Y}_{11})(1 + \underline{Y}_{22}) + \underline{Y}_{12}\underline{Y}_{21})/D_S, \quad (3.25)$$

$$\underline{S}_{12} = (-2\underline{Y}_{12})/D_S, \quad (3.26)$$

$$\underline{S}_{21} = (-2\underline{Y}_{21})/D_S, \quad (3.27)$$

$$\underline{S}_{22} = ((1 + \underline{Y}_{11})(1 - \underline{Y}_{22}) + \underline{Y}_{12}\underline{Y}_{21})/D_S. \quad (3.28)$$

3.4.1 Capacitance Matrix

The capacitances are calculated according to the charge-based capacitance model [9]: the terminal charges are in general a function of the terminal voltages. Each terminal has a capacitance with respect to the remaining terminals. For that reason a four terminal device has 16 capacitances.

All capacitances form the so-called indefinite admittance matrix. Each element C_{ij} of this matrix describes the dependence of the charge at the terminal i with respect to the voltage applied at the terminal j with all other voltages held constant. In general,

$$C_{ij} = \begin{cases} -\frac{\partial Q_i}{\partial V_j}, & i \neq j. \\ \frac{\partial Q_i}{\partial V_j}, & i = j. \end{cases} \quad (3.29)$$

The signs are chosen to keep the capacitances positive for all devices for which the node charge is directly proportional to the voltage at the same node, but indirectly proportional to the voltage of any other node. Thus, for a four terminal MOS transistor the 16 capacitances of the matrix C_{ij} are defined as follows:

$$C_{ij} = \begin{bmatrix} C_{GG} & -C_{GD} & -C_{GS} & -C_{GB} \\ -C_{DG} & C_{DD} & -C_{DS} & -C_{DB} \\ -C_{SG} & -C_{SD} & C_{SS} & -C_{SB} \\ -C_{BG} & -C_{BD} & -C_{BS} & C_{BB} \end{bmatrix}. \quad (3.30)$$

Each row and column sum must be zero in order to fulfill *Kirchhoff's* laws. For that reason, the capacitances are not independent from each other, but one of the four can be calculated with the remaining three. The gate capacitance C_{GG} is therefore:

$$C_{GG} = C_{GS} + C_{GD} + C_{GB}. \quad (3.31)$$

The four capacitances C_{GG} , C_{GG} , C_{GG} , and C_{GG} are the self-capacitances of a MOS transistor, whereas the remaining twelve are the internodal, intrinsic, or trans-capacitances. They are non-reciprocal and the corresponding capacitances differ both in value and physical interpretation [9].

3.4.2 Simulation Example

The extended small-signal features have been evaluated by a comparison with results of the commercial simulator DESSIS [111]. The structure, which has been designed with the program MDRAW, was converted by using ISE2PIF and is shown in Figure 3.11. For $f = 100$ MHz, $V_{GS} = 0.0$ V, and $V_{DS} = 0.0$ V the simulator calculates the admittance matrix shown in Table 3.1.

	Gate		Drain		Source		Bulk		Σ	
	Re	Im	Re	Im	Re	Im	Re	Im	Re	Im
Gate	2.2e-11	2.9e-07	4.2e-11	-9.6e-08	4.2e-11	-9.6e-08	-1.0e-10	-1.0e-07	-1.5e-19	9.2e-18
Drain	4.2e-11	-9.6e-08	2.0e-10	5.5e-07	1.2e-10	-1.0e-10	-3.7e-10	-4.5e-07	-1.2e-14	-1.2e-16
Source	4.2e-11	-9.6e-08	1.2e-10	-1.0e-10	2.0e-10	5.5e-07	-3.7e-10	-4.5e-07	-1.4e-15	-2.8e-15
Bulk	-1.0e-10	-1.0e-07	-3.7e-10	-4.5e-07	-3.7e-10	-4.5e-07	8.5e-10	1.0e-06	-7.0e-17	5.3e-17
Σ	1.8e-17	1.3e-19	-1.2e-14	-7.2e-17	-1.4e-15	-2.8e-15	-7.2e-17	9.5e-18		

Table 3.1: Admittance matrix calculated by MINIMOS-NT for a frequency $f = 100$ MHz and terminal voltages $V_{GS} = 0.0$ V and $V_{DS} = 0.0$ V.

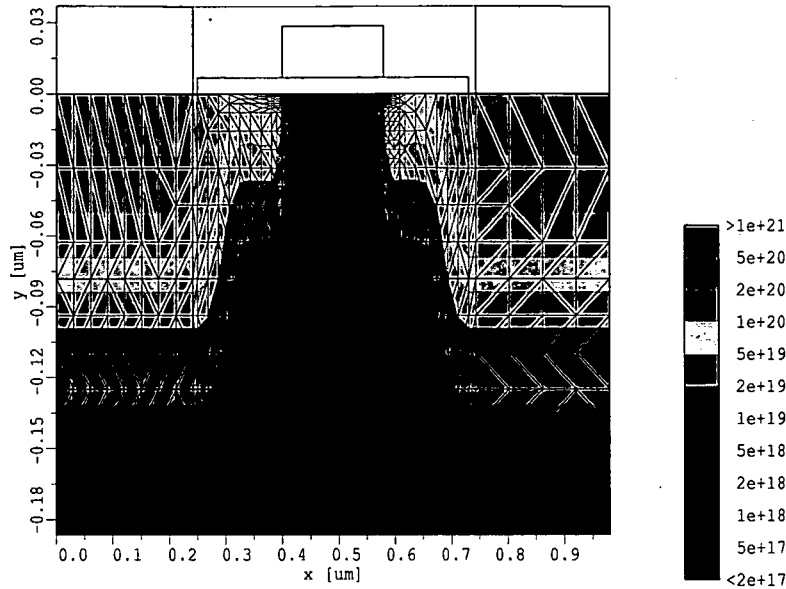


Figure 3.11: Part of the investigated device structure with a depth of $1\ \mu\text{m}$.

The last row of Table 3.1 contains the column sums, and the last column the row sums. Due to numerical reasons, zero can hardly be obtained, but the error is significantly lower than the entries in the matrix. After the calculation of the steady-state operating point, the solution of the complex-valued linear equation system requires 4.2 s on a 2.4 GHz *Intel* Pentium IV with 1 GB memory running under *Suse* Linux 8.2. The dimensions of the complete and inner equation system are 6,610 and 4,805, respectively. In Figure 3.12, the gate drain capacitance C_{GD} as calculated by MINIMOS-NT is compared with results of DESSIS. Note, that the sign of the DESSIS result had to be inverted.

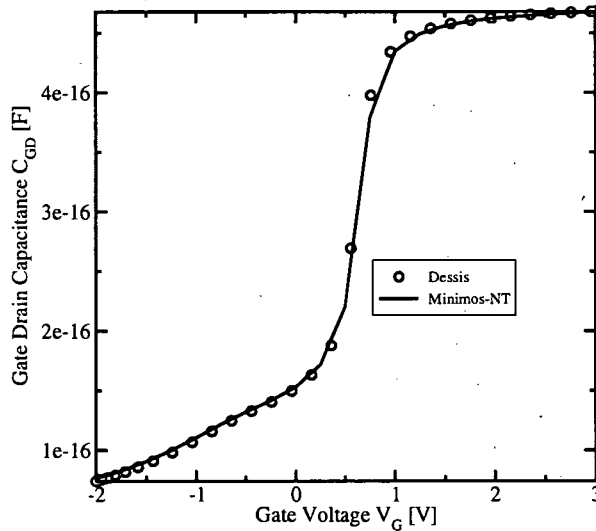


Figure 3.12: Gate drain capacitance versus gate voltage at $V_D = 0\text{ V}$: comparison of simulation results of MINIMOS-NT and DESSIS.

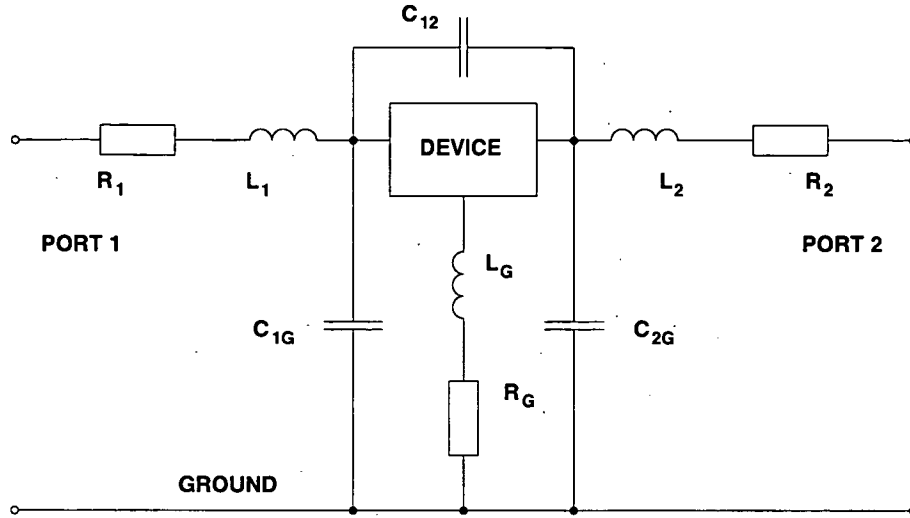


Figure 3.13: Two-Port parasitic equivalent circuit.

3.5 Transformation to Extrinsic Parameters

As already discussed above, the calculation of extrinsic parameters is provided in order to take the parasitics introduced by the measurement environment into account. Based on a standard parasitic equivalent circuit, the simulator can take all parasitics into account and can provide also extrinsic two-port parameters. First of all, it is necessary to configure this equivalent circuit consisting of two ports (contacts) and one ground port, see Appendix A.5. Based on all data given in the contact descriptions, four intrinsic Y-parameters are extended by the parasitics and are also transformed to extrinsic S-parameters as well as to H-, Z-, and ABCD-parameters. The equivalent circuit consists of a single inductance and a single resistance for each port (or contact), and one capacitance between each port and the ground.

Since the intrinsic simulation results are transformed to extrinsic parameters, one can think of embedding the device in an equivalent circuit describing the parasitics. In this section, that embedding strategy is discussed. In turn, de-embedding strategies transforms extrinsic parameters into intrinsic ones, which can be already performed by the measurement equipment. The calculation starts with the intrinsic Y-parameters which are the actual result of the numerical simulation. The embedding strategy consists of the following steps:

1. Convert to Z-parameters and add serial parasitics.
2. Convert to Y-parameters and add parallel parasitics.
3. Calculate other two-port parameter sets.

In order to account for the effects of the serial parasitic elements, a Z-parameter transformation is performed. If the parasitic Z-parameters are known, they can be added to the intrinsic simulation results:

$$\underline{Z} = \begin{pmatrix} \underline{Z}_{11} + (R_1 + R_3) + j\omega(L_1 + L_3) & \underline{Z}_{12} + R_3 + j\omega L_3 \\ \underline{Z}_{21} + R_3 + j\omega L_3 & \underline{Z}_{22} + (R_2 + R_3) + j\omega(L_2 + L_3) \end{pmatrix}. \quad (3.32)$$

Including the transformation to Z-parameters, these equations read like this:

$$\underline{Z}_{11} = \underline{Y}_{11}/|\underline{Y}| + (R_1 + R_3) + j\omega(L_1 + L_3), \quad (3.33)$$

$$\underline{Z}_{12} = -\underline{Y}_{12}/|\underline{Y}| + R_3 + j\omega L_3, \quad (3.34)$$

$$\underline{Z}_{21} = -\underline{Y}_{21}/|\underline{Y}| + R_3 + j\omega L_3, \quad (3.35)$$

$$\underline{Z}_{22} = \underline{Y}_{22}/|\underline{Y}| + (R_2 + R_3) + j\omega(L_2 + L_3). \quad (3.36)$$

The determinants of the 2×2 matrices are calculated after the standard formula:

$$|\underline{\mathbf{Y}}| = \underline{Y}_{11}\underline{Y}_{22} - \underline{Y}_{12}\underline{Y}_{21}, \quad (3.37)$$

$$|\underline{\mathbf{Z}}| = \underline{Z}_{11}\underline{Z}_{22} - \underline{Z}_{12}\underline{Z}_{21}. \quad (3.38)$$

The purpose of the second step is to take the parallel capacitances into account:

$$\underline{\mathbf{Y}} = \begin{pmatrix} \underline{Y}_{11} + j\omega(C_{13} + C_{12}) & \underline{Y}_{12} + j\omega C_{12} \\ \underline{Y}_{21} + j\omega C_{12} & \underline{Y}_{22} + j\omega(C_{23} + C_{12}) \end{pmatrix}. \quad (3.39)$$

Before the parallel parasitics are added, the Z-parameters have to be re-transformed to Y-parameters.

$$\underline{Y}_{11} = \underline{Z}_{11} / |\underline{\mathbf{Z}}|, \quad (3.40)$$

$$\underline{Y}_{12} = \underline{Z}_{12} / |\underline{\mathbf{Z}}|, \quad (3.41)$$

$$\underline{Y}_{21} = \underline{Z}_{21} / |\underline{\mathbf{Z}}|, \quad (3.42)$$

$$\underline{Y}_{22} = \underline{Z}_{22} / |\underline{\mathbf{Z}}|. \quad (3.43)$$

Finally, Y-parameters are multiplied by the characteristic impedance of the respective port:

$$\underline{Y}_{11} = \underline{Y}_{11} Z_1, \quad (3.44)$$

$$\underline{Y}_{12} = \underline{Y}_{12} Z_1, \quad (3.45)$$

$$\underline{Y}_{21} = \underline{Y}_{21} Z_2, \quad (3.46)$$

$$\underline{Y}_{22} = \underline{Y}_{22} Z_2. \quad (3.47)$$

The embedding process consists of nine free parameters: three resistors, three capacitors, and three inductors. So, an optimization can be performed to minimize the error between measured data and simulation results. Whereas for example the *Simplex* Method can be employed, also optimizations based on the methods mentioned in Appendix C.3.9 are possible. After finding an appropriate parameter setting, the extrinsic parameters are eventually calculated according to the respective formulae in Appendix D.

3.6 Small-Signal Capabilities for Mixed-Mode Device/Circuit Simulations

All presented small-signal features are not only provided for single-mode simulations, but also for the mixed-mode of MINIMOS-NT. After a short introduction, the mixed-mode AC capabilities of MINIMOS-NT are discussed.

3.6.1 Mixed-Mode Simulation

Traditional device simulation has considered the behavior of isolated device structures under artificial boundary conditions (single-mode). To gain additional insight into the performance of devices under realistic dynamic boundary conditions imposed by a circuit, mixed-mode simulations have proven to be invaluable [81]. The main advantages of mixed-mode simulations are [231]:

- A calibrated device simulator can be directly employed for circuit simulations: No subsequent and often expensive parameter/model extraction is necessary. Thus, in time-to-market considerations results of many different devices are available at significantly earlier times.
- It is common practice to create optimization loops consisting of process and device simulators. Controlled by various kinds of optimizers, device figures of merit (for example, cut-off frequency f_t) trigger process variations in order to be improved. By switching the device simulator into the mixed-mode, also circuit figures of merit can be optimization targets.

The major drawback in comparison to compact model approaches is the significant performance difference, since much larger equation systems have to be assembled and solved. However, the compact models can only be applied after the cumbersome extraction of the various parameters of the respective models. For example, the BSIM model [48] for short-channel MOS transistors provides over 300 parameters for calibration purposes, the VBIC95 model [143] for bipolar junction transistors offers about 30.

A physical circuit consists of an interconnection of circuit elements. Two well-known different aspects have to be considered when developing a mathematical model for a circuit. First, the circuit equations must satisfy *Kirchhoff's* topological laws:

- *Kirchhoff's* current law: the algebraic sum of currents leaving a circuit node must be zero at every instant of time (provided there is no charge on the nodes).
- *Kirchhoff's* voltage law: the algebraic sum of voltages around a circuit loop must be zero at every instant of time.

Second, each circuit element has to satisfy its branch relation which will be called a constitutive relation in the following. There are current-defined branches where the branch current is given in terms of circuit and device parameters, and voltage-defined branches where the branch voltage is given in terms of circuit and device parameters. Devices with N terminals can be described using $N \cdot (N - 1)/2$ branch relations. It is not necessary to include all branch currents and voltages into the vector of unknowns. It is possible to also include charges and fluxes. The wide choice of possible unknown quantities leads to a variety of equation formulations that are available. From the number of published methods, the nodal approach and the tableau approach [46] are the most important. Whereas the latter is the most general approach allowing also simulation of many idealized theoretical circuit elements, it has several inherent disadvantages (for example, ill-conditioned system matrices). Since one main objective is to solve realistic devices, the nodal approach perfectly suits the needs.

3.6.2 The Nodal Approach and Modified Nodal Approach

The independent variables of the nodal approach are the node voltages of each circuit node to a reference node which can be chosen arbitrarily. *Kirchhoff's* current law is applied to each node other except the reference node in the circuit such that the summation of the currents leaving the node is zero. Thus, in matrix representation, the admittance matrix of the circuit is assembled, which consists of $N - 1$ independent equations for a circuit of N nodes.

The admittance matrix can be assembled by taking all contributions of each element into account. The various admittance matrices of the circuit elements can simply be superpositioned to yield the complete circuit admittance matrix. Current sources contribute to the current source vector on the right-hand-side of the equation system [81]. All contributions are commonly referred to as stamps as they can be directly stamped into the equation system without considering the rest of the circuit.

For circuits containing conductances and current sources only, the condition of the resulting system matrix is very good, because the nodal approach produces diagonally dominant matrices which are well suited for iterative solution procedures. Two additional devices can be modeled, namely a voltage controlled current source and the gyrator [204]. However, these devices destroy the diagonal dominance of the circuit admittance matrix.

One disadvantage of the nodal approach is the inadequate treatment of voltage sources. Ideal voltage sources and current controlled elements cannot be modeled with this approach. However, a very large class of integrated circuits can be accommodated by adding a provision for grounded sources. The modified nodal approach [99] overcomes these shortcomings by introducing branch currents as independent variables, which are available to formulate the device constitutive relations.

Since it is not difficult to implement, the modified nodal approach enjoys large popularity. However, the numerically well-behaved system matrix obtained by the nodal approach is distorted by those additional equations, and some additional measures (see Chapter 4) have to be taken. Furthermore, the additional equations can even produce zero diagonal entries which are avoided by exchanging the rows of the admittance matrix [147].

3.6.3 Two-Level Newton and Full Newton Methods

Several efforts dealing with circuit simulation using distributed devices have been introduced [144, 175]. Most publications deal with the coupling of device simulators to SPICE [147, 168]. This results in a two-level *Newton* algorithm since the device and circuit equations are handled subsequently. Each solution of the circuit equations gives new operating conditions for the distributed devices. After creating a new input-deck the device simulator is then invoked to calculate the resulting currents and the derivatives of these currents with respect to the contact voltages [81].

The alternative approach is called full *Newton* algorithm as it combines the device and circuit equations in one single equation system. This equation system is then solved applying a *Newton* method. In contrast to the two-level *Newton* algorithm where the device and circuit unknowns are solved in a decoupled manner, here the complete set of unknowns is solved simultaneously. In MINIMOS-NT the capability to solve circuit equations was added to the simulator kernel. This allows to assemble the circuit and the device equations into one system matrix which results in a real full *Newton* method. Since the contact currents are solution variables, derivatives of the contact currents in respect to the contact voltages need not be calculated explicitly.

Although the full *Newton* algorithm seems to be more effective as the complete set of capabilities is part of the simulator, the two-level *Newton* algorithm has particular parallelization advantages. Whereas the relatively quick circuit simulation can be done on one host, the simulations of all devices can be distributed over a network in a straightforward way. In contrast, the full *Newton* algorithm is restricted to parallelization strategies regarding the solution of the large linear equation system (see Chapter 5).

3.6.4 Iteration Schemes

The need for iteration schemes arises from the fact that when solving very complex coupled equation systems, the solution can often not be obtained from the available initial-guess as the region of attraction for the *Newton* scheme would be too small. Since the equations are split into their terms, the flexible equation assembly can simply neglect some of the contributions. Thus, it is possible to apply iteration schemes as described in [58, 59], where for example some of the derivatives are neglected (see also Section 2.3.1).

Hence, the problem can be split into different levels of complexity with each of them using the previous level as an initial-guess to further refine the solution by applying more complicated models. This procedure is called iteration scheme. MINIMOS-NT provides an interface so that iteration schemes can be arbitrarily programmed with several additional options making use of the features provided by the input-deck [81]. An iteration scheme consists of arbitrarily nested iteration blocks. Each block can have subblocks which will be evaluated recursively.

For mixed-mode simulations an iteration scheme consisting of two blocks has been created. In the first block, specified node voltages are kept constant in order to obtain a converged solution for the distributed devices. This block is similar to single-mode device simulation. In the second block, the fixed voltages are set free in order to start the full circuit simulation. This procedure can be further improved by providing a previously obtained solution in an initial file.

3.6.5 The Mixed-Mode AC Capabilities

The AC features are activated the very same way than for the single-mode. However, circuits have to be extended by complex-valued sources. The sources are basically responsible for the setup of the mixed-mode AC mode. The mixed-mode of MINIMOS-NT can be used as pure circuit simulator with compact models only. In order to demonstrate the complex-valued sources, a parallel resonant circuit and a band rejection filter are simulated. For the simulation setup, see Appendix A.8. The output curves of both examples are shown in Figure 3.14.

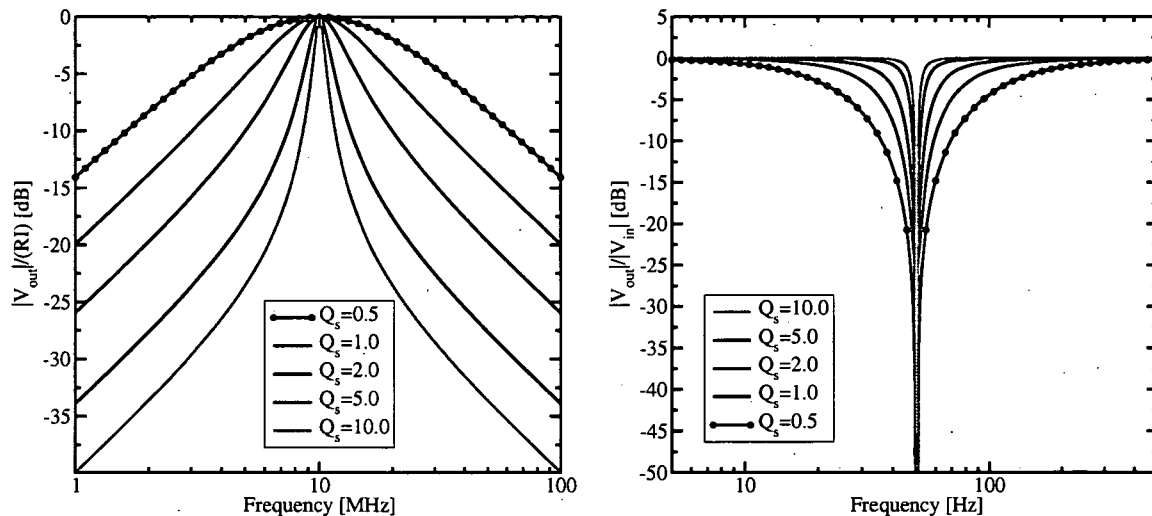


Figure 3.14: Results of mixed-mode AC simulations with compact models only: the resonant circuit on the left side and the band rejection filter on the right side.

MINIMOS-NT provides a feature to calculate the complete admittance, scattering, and capacitance matrix of a circuit. In contrast to the single-mode, the user has to specify the output nodes which should act as small-signal terminals of the circuit. This is performed by including voltage sources. In Appendix A.10, an example setup for the simulation of a heterojunction bipolar transistor is shown. The H-, Z-, and ABCD-parameter sets provided for the single-mode are bound to the parasitic circuit, so they are not provided for the mixed-mode, because the parasitic elements can be included in the fully simulated circuit.

3.7 Concluding Remarks

MINIMOS-NT has been equipped with a set of capabilities which allows the user to efficiently extract various small-signal quantities. The implemented small-signal simulation mode can be applied to

- perturb the device with an arbitrary complex-valued sinusoidal input at any terminal,
- extract important figures of merit such as the cut-off frequency f_T ,
- obtain the complete intrinsic admittance and scattering matrix of the device,
- calculate other commonly used two-port parameter sets,
- transform the intrinsic matrices to extrinsic ones,
- simulate circuits using the small-signal mixed-mode capabilities,
- make further calculations by using the convenient and powerful output and input-deck functions.

The implemented simulator features have been successfully applied for various simulations as discussed in detail in Chapter 6. Since the method requires capabilities for assembling and solving complex-valued linear equation systems, these problems will be discussed in the following two chapters.

Chapter 4

The Assembly Module

Many numerical simulations require the solution of a nonlinear system of partial differential equations. Generally, such a system cannot be solved analytically, and the solution must be calculated by numerical methods. This approach normally consists of three tasks [193]:

1. The domain is partitioned into a finite number of subdomains, in which the solution can be approximated with a desired accuracy.
2. The system of partial differential equations is approximated in each of the subdomains by algebraic equations. The unknowns of the algebraic equations are approximations of the continuous solutions at discrete grid points in the domain. Thus, generally a large system of nonlinear, algebraic equations is obtained with unknowns comprised of approximations of the unknown functions at discrete points.
3. The third task is to derive a solution of the unknowns of the nonlinear algebraic system. In the best case an exact solution of this system can be obtained, which represents a good approximation of the solution of the analytically formulated problem (which cannot be solved exactly). The quality of the approximation depends on the fineness of the partitioning into subdomains as well as on the suitability of the approximating functions for the dependent variables.

This nonlinear problem is usually solved by a damped *Newton* algorithm (see Section 2.3.1) demanding the solution of a sparse non-symmetric linear equation system at each step. As many simulators, for example MINIMOS-NT, are based on this approach, specific capabilities are required to assemble and solve equation systems. Due to their independence from the other parts of the simulators, these capabilities are frequently incorporated in separate modules.

In this chapter, the assembly module is going to be discussed, subject of the next one is the solver module.

4.1 Key Demands on the Assembly Module

From the perspective of a simulator, which can be an arbitrary code requiring assembling and solving of linear equation systems, the key demands on the assembly module can be summarized as follows:

1. Application Programming Interface (API) providing methods for:
 - Adding values to the equation system.
 - Deleting equations.

- Invoking the solving process.
 - Retrieving the solution.
2. Usability.
 3. Performance.
 4. Conditioning of the equation system for solving.
 5. Handling of real-valued and complex-valued equation systems.

Hence, the assembly module employed by the simulator shall be responsible for storing the contributions of the various physical models of the simulator. In addition, for the sake of consistence and simplicity the API shall embrace also the interface to the solver module. This allows also the conditioning of the assembled equation system in order to improve its solvability.

By providing these abstracted features the simulator can be designed and implemented in a very efficient way. This has the following reasons:

- The model developer is able to focus on the model implementation only, because the respective contributions can be conveniently added without taking any aspects regarding the implementation (allocation, access, deallocation etc.) of the assembly module into account.
- The further development of the simulator is not limited by the assembly module in case this module is generally designed and does not impose restrictions for its application.

4.2 Approaches to Meet these Demands

Meeting these requirements can be managed by pursuing several approaches:

- The design of the simulator can be based on a specific computational numerics package and adhere to the requirements given for this module. These requirements include the field of application, type of the equations, their discretization etc. Many of these packages also include a non-linear solver and tools regarding the discretization of the equation system. Some examples of the specific computational numerics packages are given in Section 4.2.1.
- A new module for the specific simulator is implemented which implicitly meets all requirements. Such a module can contain third party components as well, for example by coupling this module to packages like LAPACK or Blitz++ (see Section 5.1).
- A new generally applicable module is implemented which exactly offers the requested capabilities without restricting the simulator in the usage of them.

4.2.1 Third Party Modules and Packages

The general-purpose finite element analysis package ANSYS [7] is a product of ANSYS, Inc., Canonsburg, PA. There are seven generic steps to solving any problem in ANSYS [114]:

1. Building Geometry: After a preliminary design, a two- or three-dimensional representation of the object is constructed by using the work plane coordinate system within ANSYS.

2. Definition of Material Properties: The materials the object is composed of are selected in order to account for the thermal and mechanical properties.
3. Mesh generation.
4. Applying loads: After the complete design, the system is completed by specifying loads or constraints, for example thermal boundary conditions.
5. Calculating the solution: depending on the simulation mode the solution is calculated.
6. Results: the results obtained by the solution steps can be graphically illustrated and used for further applications or improving the simulation setup.

ANSYS is able to perform structural analysis which is one of the most common applications of the finite element method. Examples are simulation of buildings, ships, aircraft, or mechanical components. Linear and nonlinear static analysis is used to determine stresses under static loading conditions, whereas transient analysis takes dynamic effects under time-varying loads into account. In addition, buckling analysis is used to calculate solutions for buckling loads. The effects of thermal loads, for example convection or heat fluxes, can be simulated by means of the steady-state thermal analysis. Specific packages allow to analyze magnetics, fluid-flows, or acoustics. The package also provides features for fracture mechanics, composites, fatigue, p-Method, and beam analysis. In the recent version a distributed package called DANSYS [7] is also available which allows to parallelize several costly tasks, such as calculating the stiffness matrix, solving the equations, and performing the results calculations.

FEMLAB [40], originally a product of Sweden and today marketed by *Comsol, Inc.*, Burlington, MA, provides an interactive modeling environment for scientific and engineering applications, which are based on partial differential equations. FEMLAB is based on the finite element method and claims to provide unprecedented speed and accuracy through its high-performance solvers [40]. The term *multi-physics* is frequently used to stress that multiple physical effects can be combined and solved in a coupled manner (in the ANSYS packages, this is called coupled-field analysis).

ABAQUS software, a product of ABAQUS, Inc., Providence, RI, provides a suite of interoperable applications for finite element analysis [1]. ABAQUS can be used to solve nonlinear finite element problems, for example arising from mechanical, structural, civil, biomedical, and related engineering applications.

LS-DYNA of *Livermore Software Technology Corporation*, Livermore, CA, can be used to analyze the nonlinear dynamic response of three-dimensional structures. The general purpose, explicit finite element program provides a comprehensive selection of material models, element formulations, and contact algorithms. These capabilities have been already used to solve many complex automotive safety, metal forming, structural, and failure analysis problems [133].

MAFIA, a product of the *Gesellschaft für Computer Simulationstechnik* in Darmstadt, Germany, offers a collection of programs for solving electrical engineering problems. The software discretizes the *Maxwell* equations with the *Finite Integration Theory* method, which is a special finite-volume method, and uses the same mesh for different problems. For example, eigenmodes analysis and magnetic fields calculations can be performed for the same structure. The product claims that almost all problems related to the *Maxwell* equations and some outside the area of electrodynamics, for example acoustics, can be treated [73].

PDE2D is a successor of TWODEPEP [196] and provides an interactive, general-purpose solver for partial differential equations [197]. The program can be used to solve partial differential equations in two-dimensional polygonal regions and applies a *Galerkin* finite element method and triangular elements. In addition, it solves problems in general two-dimensional regions with arbitrary curved boundaries [196]. Furthermore, it has been extended to handle also three-dimensional systems.

DIFFPACK is provided by *inuTech GmbH*, Nürnberg, Germany and offers an object-oriented problem-solving environment for the numerical solution of partial differential equations [108]. The package is a

collection of C++ libraries with classes, functions, and utilities [126]. The kernel and add-on toolboxes provide a substantial collection of data structures and numerical algorithms. The list of functionality includes vectors, matrices, general multi-index arrays, representation of linear systems, also large sparse systems, iterative methods for sparse linear systems, solvers for non-linear systems, finite element and finite difference meshes, finite element algorithms, high level finite difference support, real- and complex-valued arithmetic, adaptive meshes, multigrid methods, domain decomposition methods, generalized (mixed) finite element methods, and parallel computing on linear algebra and domain decomposition level. Therefore, DIFFPACK can be employed to develop specialized finite element solver programs. It also provides interfaces to other products such as ANSYS [7], ABAQUS [1], MATLAB [141], or the VTK [186], which can be particularly used as pre- and post-processing tools. Flexibility is the strength of DIFFPACK as it can be employed for non-standard problems such as multi-physics systems of partial differential equations. DIFFPACK provides most of the solution process so that the simulator has to be developed for data input and decisions only. Object-oriented approaches are used for administrative functions, whereas the computations are performed in highly-optimized lower level code [76].

SCILAB by INRIA, Rocquencourt, France, is a scientific software package with a high-level language for numerical computations. The software features data structures such as polynomial, rational and string matrices, lists, multivariable linear systems, sophisticated interpreter and programming language with MATLAB-like syntax, various built-in mathematical functions, two- and three-dimensional graphics as well as animations, interfaces to FORTRAN, C, and MAPLE, and various built-in libraries for linear algebra, control systems, signal, processing, simulation, optimization, and network analysis [106].

4.2.2 A New Generally Applicable Assembly Module

Although a large variety of already existing packages is available, the decision was made to design and implement a new assembly module, because of the following two reasons:

1. This module shall be applied for assembling linear equation systems and preparing of these systems for the solving process. Thus, the nonlinear solving system and all problems related to the discretization of the underlying physical models shall be solved within and by the simulator. This offers the possibility for providing a generally applicable module which can be effectively used for different kind of numerical simulators.
2. Third party modules are frequently bound to license agreements, which restrict their application especially for commercial application. As the institute provides its codes to industrial partners and binary release versions to the general public, third party license requirements would make such distributions more complicated. An in-house assembly module avoids such complications and allows the institute to freely distribute complete versions.

One additional reason for that decision is also the history of one important simulator, namely MINIMOS-NT. This simulator has been under development for a long time (see Section 1.3.2) and has always employed an assembly and solver module. All experiences with these old modules were taken into account while the new modules were designed [65]. However, it is a crucial question why two core modules, which were tested and properly working for a long time, are replaced at all. The answer to that question consists of the following three reasons:

1. Providing a more convenient application interface: due to the significant internal improvements it is always possible to randomly access all parts of the linear equation system. In contrast to the former module, which required a specific four-phases assembly sequence [228], the model implementations can now be implemented in a much easier way.

2. Handling of complex-valued equation systems: For the steady-state analysis and transient simulation, MINIMOS-NT assembles real-valued linear equation systems. In context of an efficient small-signal simulation mode, complex-valued contributions have to be handled as already discussed in Section 2.6.2. The redesigned module is able to assemble and solve both real-valued and complex-valued equation systems while featuring an advanced C++ design and implementation.
3. Working in a deterministic way: as discussed in Section 5.2.5, multiple starts of a solver all result in exactly the same solution. This is a very important feature while the simulator is extended and during model evaluation and debugging.

4.3 Refined Key Demands on the Assembly Module

After the decision to design and implement new modules has been made, the key demands on the assembly module can be refined. This also brings a row transformation as discussed in Section 2.2.1 into play. Due to its advantages, the assembly module shall provide the required capabilities. First of all, the following definitions are given in order to clarify some terms:

- Assemble: *assemble* means *to bring together*. Thus, assembling regards the adding of values to the linear equation system. Assembling is completed when all values have been added.
- Assembly: in this context, the *assembly* or the *assembly module* specifically provides all necessary features which allow a simulator to assemble those values. However, the *assembly module* is generally responsible for meeting all key demands given above.
- Compile: *compile* means *to compose out of materials from other sources, to collect and edit into a volume, to run through a compiler, and to build up gradually*. Since four matrices (A_b , A_s , T_b , and T_v) and two sets of right-hand-side vectors (b_b and b_s) are assembled, the first task is to *compile* all of these parts to one linear equation system. Refer to equations (4.12) and (4.13) to see how the parts are compiled.
- Compilation result: the *compilation result* is one linear equation system $Ax = b$, the so-called *complete* linear equation system.
- Pre-elimination: after the complete linear equation system is compiled, all equations marked with an elimination flag are *pre-eliminated*. Hence, the assembly module also provides a solving capability based on *Gaussian* elimination (see Section 4.8).
- Inner linear equation system: one result of the pre-elimination is the so-called *inner* linear equation system. It consists of all equations which were not pre-eliminated.

The refined requirements can be summarized as follows:

1. Application Programming Interface (API) providing methods for
 - Adding values to the boundary system: A_b and b_b .
 - Adding values to the segment system: A_s and b_s .
 - Adding values to the transformation matrix T_b .
 - Deleting equations.
 - Administration of priority information required for the correct handling of grid-points which are part of several segments.

- Setting elimination flags for pre-elimination: mark equations which are eliminated from the linear equation system before it is passed to the solver module.
- Invoking the solving process.
- Returning the solution: After reverting all transformations and back-substituting the pre-eliminated equations, the output of the assembly module is the complete solution vector (or vectors in the case of more than one right-hand-side vector). In addition, the right-hand-side vector(s) are returned which can be used for various norm calculations frequently required for the damping and update of the *Newton* method.

2. Usability

- Random access during assembling: all parts of the linear equation system should be always accessible.
- Centralized administration of control parameters.

3. Preparation of the linear equation system for solving:

- Row transformation: linear combination of rows to extinguish large entries.
- Variable transformation: reduce the coupling of the equations.
- Pre-elimination: eliminate problematic equations by *Gaussian* elimination to improve the condition of the inner system matrix.
- Scaling: Since some preconditioners (for example the Incomplete-LU factorization, see Section 5.2.5) use a threshold to decide whether to keep or skip an entry, the entries of the system matrix are normalized.
- Sorting: Reduction of the bandwidth of a matrix in order to reduce the factorization fill-in and thus the memory consumption.

4. Handling of real-valued and complex-valued linear equation systems: In the context of an efficient small-signal mode, handling of complex-valued systems is particularly important.

5. Performance-related features:

- Handling of multiple right-hand-side vectors: For some simulations, for example calculating the complex-valued admittance matrix (see Section 3.4), several linear equation systems differ only in the right-hand-side vector. Thus, the effort for assembling, compiling, pre-eliminating, sorting, scaling, and factorizing of the system matrix actually has to be done only once. This factored matrix can then be used for all right-hand-side vectors. For that reason the module is able to simultaneously assemble several right-hand-side vectors.
- Reassembling of the imaginary part only: during a frequency stepping, only the imaginary part is changed. In order to speed up the simulation, the real-valued part should remain unmodified.

6. Efficient handling of the sparse linear equation systems: storing large linear equation systems in dense format requires a huge amount of memory. Since the system matrices contain relatively few non-zero entries (see Section 5.5.1), far less memory has to be allocated if sparse matrix formats, for example MCSR (see Section 4.6 and Appendix E), are used.

A plug-in concept has been implemented for scaling, sorting and solving the inner linear equation system, making it possible to adapt or replace these modules easily. The sorting and scaling modules obtain the system matrix on input and return the sorting and scaling (diagonal) matrices which are then applied by the assembly module. The solver module obtains the system matrix and all right-hand-side vectors on input and returns the solution vectors of all inner linear equation systems.

4.4 Condition of the Linear System

The use of the discretized and linearized semiconductor equations yields large linear equation systems of the form $\mathbf{Ax} = \mathbf{b}$. Such a system has to be solved with a given accuracy.

As described in [65], the results of both direct and iterative solvers depend on the accuracy of digitally stored numbers and the condition of the system matrix \mathbf{A} . The internal storage representation of numbers is responsible for their accuracy. The assembly module currently provides the C data type `double` according to its IEEE norm 754-1985 standard for binary floating-point arithmetic. This standard defines four binary formats for 32-bit single, (normally 43-bit) single-extended, 64-bit double, and (normally 80-bit) double-extended precision numbers. They are composed of three parts. A double precision number has a sign bit being either zero or one, an eleven bit exponent ranging from $E_{\min} = -1022$ to $E_{\max} = 1023$, and a 52 bits fraction [60]. The standard also defines how zero, infinity, and so-called NaNs (Not a Number) are to be encoded. NaNs represent undefined or invalid results, for example the square root of a negative number. For the sake of completeness it is stated that a complex-valued number stores both the real and imaginary part in the double precision format. Furthermore, due to the limited representation results of mathematical formulae generally vary although they are correctly changed according to mathematical laws such as the commutative law.

The condition of a matrix can be used to estimate the worst possible error of the solution vector \mathbf{x} of $\mathbf{Ax} = \mathbf{b}$ obtained by *Gaussian* elimination using a restricted number representation. This is measured by the condition number of that matrix. Well-conditioned matrices have a small condition number [51]:

$$\kappa_{\infty} = \|\mathbf{A}\|_{\infty} \|\mathbf{A}^{-1}\|_{\infty}, \quad (4.1)$$

with the infinity norm for the system matrix

$$\|\mathbf{A}\|_{\infty} = \max_i \sum_{k=1}^n |a_{i,k}|. \quad (4.2)$$

For iterative solvers the spectral condition norm (spectrum of the system matrix) is more characteristic [78]. It is defined as ratio of the largest eigenvalue to the smallest one:

$$\kappa_s = \frac{\lambda_{\max}}{\lambda_{\min}}. \quad (4.3)$$

The larger the value of κ_s the poorer is the condition of the system matrix. Iterative solvers are particularly sensitive to bad condition numbers which can then cause

- a large number of necessary iterations;
- the failure of the solver to converge at all, or
- convergence to a wrong solution.

An important way to handle ill-conditioned matrices ($\kappa(\mathbf{A}) \gg 1$) is to precondition the matrix \mathbf{A} . Hence, iterative methods usually determine a second matrix that transforms the system matrix into one with a better condition. This second matrix is called a *preconditioner* and improves the convergence of the iterative solver (see Section 5.2.5).

Beside the purely numerical concept of preconditioning, a good approach to improve the condition during the assembly process is to aim for diagonal dominance of the equations, since it is a necessary (but not sufficient) condition in the proof of convergence for a range of iterative solver schemes. A matrix is

diagonally dominant, if in all equations the absolute value of the diagonal element is larger than the sum of the absolute values of all off-diagonal elements in all equations, and equal in at least one row [52]:

$$\sum_{j=1}^n |A_{i,j}| \leq |A_{i,i}|, \quad \text{for } i \neq j. \quad (4.4)$$

A matrix is strictly diagonally dominant if in all equations the absolute value of the diagonal element is larger than the sum of the absolute values of all off-diagonal elements [52]:

$$\sum_{j=1}^n |A_{i,j}| < |A_{i,i}|, \quad \text{for } i \neq j. \quad (4.5)$$

Besides of this advantage it is important to note that in case of diagonal dominance direct solution techniques can apply a diagonal strategy and avoid alternative and costly pivoting steps [188].

4.5 The Parameter Administration

The parameter administration is a crucial aspect regarding the usability of the system. As the assembly and solver module can be controlled by more than hundred parameters, it would be inconvenient if all of them had to be specified as arguments. For that reason, all parameters are centralized in one structure, which has a threefold purpose:

1. It provides a method of passing data to the solver function that influence the behavior of the function. This includes the choice of a proper solver, preconditioner, sorter, and scaler, the choice of maximum limits for memory allocation and time consumption, and other parameters.
2. It supports the persistence of data that has to be conserved between multiple runs of the solver function on the same type of equation system, for example the repeated solving during the *Newton* iteration. In such cases, information about the previous solving step is used to improve the solution speed on the next call to the solver function, for example by choosing proper preconditioning parameters.
3. It provides a method of returning statistical data about the solution process to the simulator.

4.6 Assembling the Complete Linear Equation System

The assembly module can be used to assemble arbitrary linear equation systems $\mathbf{Ax} = \mathbf{b}$ independently of the concept the simulator is based on. This fulfills the major key demand of general applicability. However, the row transformation feature necessitates to continue the discussion with a more specific field of application. Although not mandatory, this feature can be well applied for the finite volume (or box integration) discretization method. For that reason it shall be used as example during the following discussion.

A semiconductor device which is going to be simulated is normally divided into several segments that are geometrical regions employing a distinct set of models. The implementation of each model is completely independent from other models and each model is basically allowed to enter its contributions to the linear equation system. All boundary and interface issues are completely separated from the general segment models represented by assembly structures for the boundary system which are independent from the segment ones.

Thus, the system matrix \mathbf{A} will be assembled from two parts, namely the direct part \mathbf{A}_b (boundary models) and the transformed part \mathbf{A}_s (segment models). The latter is multiplied by the row transformation matrix \mathbf{T}_b from the left before contributing to the system matrix \mathbf{A} . The right-hand-side vector \mathbf{b} is treated the same way:

$$\mathbf{A} = \mathbf{A}_b + \mathbf{T}_b \mathbf{A}_s, \quad (4.6)$$

$$\mathbf{b} = \mathbf{b}_b + \mathbf{T}_b \mathbf{b}_s, \quad (4.7)$$

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \quad (4.8)$$

Although in principle every model is allowed to add entries to all components, the assembly module checks two prerequisites before actually entering the value: first, the quantity the value belongs to must be marked to be solved (the user may request only a subset of all provided models), and second the priority of the model has to be high enough to modify the row transformation properties. As stated before, the row transformation is used to complete missing fluxes in boundary boxes. Since a grid point can be part of more than two segments, a ranking using a priority has been introduced. For example contact models have usually the highest priority and thus their contributions are always used for completion.

All three matrices \mathbf{A}_b , \mathbf{A}_s , and \mathbf{T}_b and the two vectors \mathbf{b}_b and \mathbf{b}_s may be assembled simultaneously, so no assembly sequence must be adhered to. In addition, a fourth matrix \mathbf{T}_v is assembled which contains information for an additional variable transformation (see Section 4.7.2).

The assembly module offers an additional feature for quantity administration. The simulator is able to use this feature to store and obtain the information about quantity indices and other properties from the assembly module. This has the specific advantage that multi-level solvers can directly be provided with the required connectivity information between the equations (see Section 5.3.3).

During the assembling process, all contributions are added to values stored in a flexible sparse matrix structure based on a balanced binary tree. The advantage of this structure is that new entries can be easily added at any place and any time. The purpose of this matrix format is for data entry only, so no mathematical operations are defined for this structure. Since diagonal entries are always required to be assembled (zero diagonals are not allowed), they are stored in an array allowing very fast access. So the dimension of the linear equation system must be known in advance before the structure can be allocated. In the format sorted by row indices, all off-diagonal entries are stored in a balanced binary tree for each row. This allows one to delete complete rows very efficiently. If complete columns have to be deleted, for example required for the transformation matrix, it is faster to store the off-diagonal entries sorted by columns, which can be specified on construction.

After the assembling has been finished, that is after the flexible sparse matrix structure is completely constructed, these structures are converted to the sparse matrix format MCSR, which stands for *Modified Compressed Sparse Row* [178]. The analogous, column-oriented *Modified Compressed Sparse Column* format MCSC is used to speed up column deleting. See Appendix E.1 for a detailed description of these formats. The advantages of using compressed structures can be summarized as follows [54]: if \mathbf{A} is considered as a dense matrix with a dimension of n , it requires $O(n^2)$ storage. The so-called big- O notation $O(n)$ [158] is a theoretical measure usually for the time or memory required by an algorithm, given for the problem size n , which is normally the number of items processed. Thus, a matrix with $n = 100,000$ stored with double precision numbers requires $100,000^2 \times 8$ bytes. These roughly 75 GB are a huge chunk of memory even for today's computers. Since typically sparse systems are solved with similar or even higher dimensions, dense formats and algorithms require prohibitively high amounts of memory and time. Thus, the objective of sparse matrix formats and algorithms is solving linear equation systems with time and space proportional to $O(n) + O(n_{\text{non-zero}})$, with $n_{\text{non-zero}}$ as the number of non-zeros.

During the *Newton* iterations the structural configuration of these matrices is not modified very often. A structural reconfiguration can be triggered by a change of iteration schemes, for example for enabling more derivatives in the *Jacobian* (see Section 3.6.4). The assembly module is designed to take such considerations into account. If the structure remains unchanged, the balanced binary trees can be skipped and the variables may be entered directly in the already existing MCSR structures. Hence, the effort for deleting, tree assembling, reallocating and converting can be saved which drastically speeds up the assembly process. The so-called *Newton* adjustment addresses not only the assembly matrices, but also the resulting structures of the compilation and pre-elimination process. The performance impact of these features has been analyzed and is discussed in Section 5.5.2. See Section 4.12 for more details on the *Newton* adjustment levels.

After the four compressed sparse matrix structures have been completely constructed, the following steps discussed in the respective sections are performed:

- Compiling of the complete linear equation system: see Section 4.7.
- Pre-elimination to obtain the inner linear equation system: see Section 4.8.
- Sorting the inner linear equation system: see Section 4.9.
- Scaling the inner linear equation system: see Section 4.10.
- Solving the inner linear equation system: see Chapter 5.
- Back-substitution and retransformation: see Section 4.11.

4.7 Compiling the Complete Linear Equation System

After the assembly process has been finished, all four matrices and two vectors have to be compiled to obtain the complete linear equation system. The first step is to compile the segment and the boundary system in the following way:

$$\mathbf{b} = \mathbf{b}_b + \mathbf{T}_b \mathbf{b}_s, \quad (4.9)$$

$$\mathbf{A} = \mathbf{A}_b + \mathbf{T}_b \mathbf{A}_s, \quad (4.10)$$

$$\mathbf{A} \mathbf{x} = \mathbf{b}, \quad (4.11)$$

$$(\mathbf{A}_b + \mathbf{T}_b \mathbf{A}_s) \mathbf{x} = \mathbf{b}_b + \mathbf{T}_b \mathbf{b}_s. \quad (4.12)$$

The second compilation step regards the variable transformation matrix \mathbf{T}_v :

$$((\mathbf{A}_b + \mathbf{T}_b \mathbf{A}_s) \mathbf{T}_v) (\mathbf{T}_v^{-1} \mathbf{x}) = (\mathbf{b}_b + \mathbf{T}_b \mathbf{b}_s). \quad (4.13)$$

The left side of Figure 4.2 shows the completely compiled system matrix arising from the discretization of a two-dimensional MOS transistor structure. Since the linear equation systems has been assembled by the drift-diffusion models of MINIMOS-NT, it consists of three major quantities. For the semiconductor segment, the values of and the couplings between the potential (row or column number 45–955), electron (956–1879) and hole concentrations (1880–2803) can be clearly seen. After the discussions of the pre-elimination, the sorting, and the scaling, the respective graphical representations of the same system matrix will be shown.

In the next sections the row and variable transformations are going to be discussed in a more detailed way.

4.7.1 Row Transformation

As already discussed in Section 4.6, semiconductor device simulation based on the finite volume method is used as an example to discuss the row transformation. The complete linear equation system is built from a segment system, which is the segment system matrix A_s and the segment right-hand-side vector b_s , both of them representing cumulated fluxes and their derivatives to the system variables. Basically, the fluxes are calculated from segment models describing the interior of discretized regions. The matrix is a linear superposition of very small matrices, one for each flux, with few non-zero elements only. Consequently, the same superposition applies for the vector b_s .

All fluxes are assigned to boxes, a box is in turn assigned to each variable. As the control function for a box is defined by the user, for example being the sum of all fluxes leaving the box, the fluxes leaving the boxes are entered into the vector b_s in the places appropriate for the variables that are assigned to the boxes. In context of the *Newton* method, A_s is part of the *Jacobian* matrix and contains the negative derivatives of the values in b_s to the system variables. The right-hand-side vector depends on the current solution of the *Newton* iteration.

The boundary conditions will enforce some special physical conditions at the boundaries. The control functions of boxes along the boundaries will usually be completed by the boundary conditions. For example, a *Dirichlet* boundary condition will use the dielectric flux cumulated in the boundary box to calculate the surface charge on the surface of the adjacent material. The equation used to calculate the value of the boundary variable, however, will not always make use of the fluxes accumulated in the segment system.

The boundary conditions are therefore implemented by two elements: a boundary system (A_b and b_b) and a transformation matrix T_b . The purpose of the matrix T_b is the forwarding of the fluxes of the main system to their final destinations or their resetting if they are not required. The system of A_b and b_b represents additional or, in case of *Dirichlet* boundary conditions, substitutional parts of the final equation for the variables at the boundaries. Again, the entries in the matrix A_s are the negative derivatives of the right-hand-side vector b_b to the variable vector v .

4.7.2 Variable Transformation

Especially in the case of mixed quantities in the solution vector, a variable transformation is sometimes helpful to improve the condition of the linear system. The representation chosen here allows to specify fairly arbitrary variable transformations to be applied to the system. Basically, a matrix T_v is assembled and multiplied with the system matrix from the right.

For example, to reduce the coupling of the semiconductor equations and thus improve the condition of the system matrix, a transformation of the stationary drift-diffusion model is suggested in [10]. The system matrix can be diagonalized to leading terms by substituting $d\phi$, dn , and dp by

$$\begin{pmatrix} d\phi \\ dn \\ dp \end{pmatrix} = T_v \begin{pmatrix} d\tilde{\phi} \\ d\tilde{n} \\ d\tilde{p} \end{pmatrix}, \quad (4.14)$$

$$\tilde{A} = AT_v, \quad (4.15)$$

$$T_v = \begin{pmatrix} 1 & 0 & 0 \\ \frac{nq}{k_B T_L} & 1 & 0 \\ -\frac{pq}{k_B T_L} & 0 & 1 \end{pmatrix}. \quad (4.16)$$

This transformation is the *Gummel-Ascher transformation*, and was extended for the differential equations

of the energy-transport model in [201]:

$$\mathbf{T}_v = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{nq}{k_B T_n} & 2 & 0 & -\frac{2n}{T_n} & 0 \\ -\frac{pq}{k_B T_p} & 0 & 2 & 0 & \frac{2p}{T_p} \\ 0 & \frac{T_n}{n} & 0 & 2 & 0 \\ 0 & 0 & \frac{T_p}{p} & 0 & 2 \end{pmatrix} \quad (4.17)$$

The *Gummel-Ascher transformation* meets all requirements for such variable transformations: first the transformation is expressed by a matrix \mathbf{T}_v which has an inverse \mathbf{T}_v^{-1} . Second, it does not destroy the diagonal dominance. In fact there is no qualitative difference between \mathbf{A} and $\tilde{\mathbf{A}}$ in terms of this condition property because the original system is substituted by a related one (see (4.14)). Third, the transformation matrix decomposes into small submatrices with a limited number of variables involved in a single transformation. The variable transformation is restricted to variables on the specific grid points, thus, it is a local transformation.

For compactness, the following substitutions will be used hereafter for the complete linear equation system:

$$\tilde{\mathbf{A}} = ((\mathbf{A}_b + \mathbf{T}_b \mathbf{A}_s) \mathbf{T}_v), \quad (4.18)$$

$$\tilde{\mathbf{x}} = (\mathbf{T}_v^{-1} \mathbf{x}), \quad (4.19)$$

$$\tilde{\mathbf{b}} = (\mathbf{b}_b + \mathbf{T}_b \mathbf{b}_s). \quad (4.20)$$

4.8 The Pre-Elimination

The main matrix \mathbf{A}_s consists of fluxes that will (if the control functions are correctly assigned to the variables) satisfy the criterion of diagonal dominance which is necessary to make the linear equation system solvable with an iterative solver (see Section 4.4).

The transformations and additional terms imposed by the boundary conditions may heavily disrupt this feature both in structural and numerical aspects. Some of the boundary or interface conditions can make the full system matrix so ill-conditioned thereby simply preventing iterative linear solvers to converge.

This problem can be simply passed to the solver module which is likely to employ a direct solver to solve such heavily ill-conditioned problems. Alternatively, an elimination concept as designed and presented in [65] can be pursued which applies a *Gaussian* elimination to some parts of the linear equation system only. It is important to note that this solving capability is part of the assembly module. It is possible to disable this feature and pass the complete linear equation system to the solver module (after sorting and scaling, if enabled).

The elimination concept is based on the idea to apply *Gaussian* elimination to the problematic or critical equations before the system is passed on to the linear solver. After the iterative solver has converged, the eliminated variables are calculated by back-substitution into the eliminated equations. This process is thus a partial *Gaussian* factorization of the matrix, which is called pre-elimination in the context of the assembly module.

Before these equations can be eliminated, they are sorted to the end of the matrix, together with their assigned variables. This is done by applying a permutation matrix \mathbf{P} to the linear equation system. The permutation matrix is calculated automatically on solving the system. All equations causing a possible ill condition have to be marked for pre-elimination by the simulator.

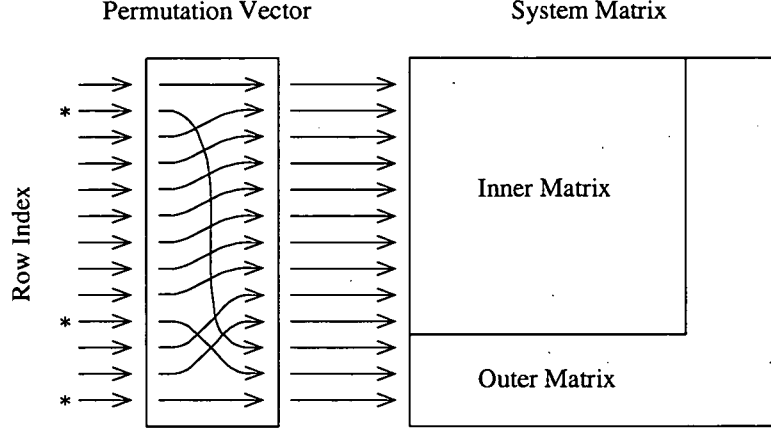


Figure 4.1: All equations marked for pre-elimination (*) are moved to the outer system matrix, the others remain in the inner one [65, 228].

The outer system is removed from the linear equation system and later solved by *Gaussian* elimination, the inner system with an improved condition suitable for iterative solvers is passed on to the solver module. See Figure 4.1 [65, 228] for an illustration of this concept. The resulting system is given by

$$(\mathbf{E}\tilde{\mathbf{P}}\mathbf{A}\mathbf{P}^T)(\mathbf{P}\tilde{\mathbf{x}}) = (\mathbf{E}\tilde{\mathbf{P}}\mathbf{b}), \quad (4.21)$$

where \mathbf{P} is the permutation matrix with its inverse equal to its transposed matrix \mathbf{P}^T . As the factorization starts with the last equation and proceeds upwards, the usual terms for direct *Gaussian* LU factorizations have a different meaning. The upper triangular matrix \mathbf{E} stores the elimination coefficients obtained as the lower matrix \mathbf{L} of a *Gaussian* elimination. It contains non-zero off-diagonals in the outer parts only, the inner matrix domain is a strict unity matrix. The unity diagonal of \mathbf{E} is not stored. The result of the multiplication of \mathbf{E} with \mathbf{A} is a matrix with some (but not all) entries in the domain right of the diagonal removed, and some newly created entries. This matrix is split into two parts: the core matrix containing all equations of the inner matrix and the outer matrix.

In Figure 4.2 the effect of the pre-elimination for the system matrix shown. The so-called inner system matrix does not contain the problematic equations any more, for example the equations in rows 1-44 of the complete system matrix. It is obvious that the majority of equations remains in the inner system matrix. Thus, the effort of the *Gaussian* elimination is kept small whereas the solver module is expected to bear the main solving effort.

4.9 Sorting the Inner Linear Equation System

Matrices arising from the discretization of differential operators are sparse, because only neighbor points are considered. For that reason, only the non-zero elements are stored in order to reduce the memory consumption (see the MCSR format in Appendix E.1). However, during the factorization of the system matrix \mathbf{A} into an upper and lower triangular matrix $\mathbf{A} = \mathbf{L}\mathbf{U}$, additional matrix elements termed fill-in [193] become non-zero. The profile $p(\mathbf{A})$ is a measure for this fill-in

$$p(\mathbf{A}) = \sum_{i=1}^n m_i, \quad (4.22)$$

$$m_i = i - \min_{a_{i,j} \neq 0} (j), \quad (4.23)$$

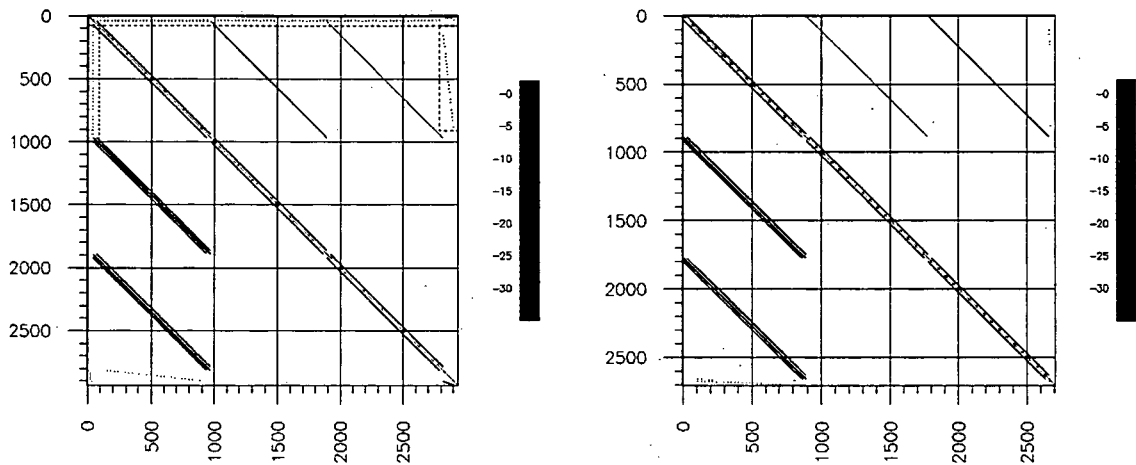


Figure 4.2: On the left the completely compiled system matrix of a discretized two-dimensional MOS transistor structure assembled by MINIMOS-NT is shown. The magnitude of the entries are encoded by the colors according to the legend in the right. Some regions with problematic equations are indicated by red dashed rectangles. After the pre-elimination, the inner system matrix (right) does not contain the problematic equations any more. The dimension is not significantly reduced since the majority of equations is not affected.

and the bandwidth of the matrix is $\max_i(m_i)$. Thus, the bandwidth of the system matrix is the maximum distance between a diagonal and an off-diagonal entry of the same row. Since storing of $p(\mathbf{A})$ requires additional memory, a transformation is applied to reduce the bandwidth and the profile. Thus, sorting algorithms sort the rows and columns of the system matrix in such a way that the elimination performed by a *Gaussian* type factorization yields a small number of fill-ins. The term *reordering* can be used instead of *sorting*.

The standard module provided by default obtains the sorting matrix \mathbf{R}_s (similar to \mathbf{P}) by a reverse *Cuthill-McKee* algorithm [43, 193]. It contains a single unity entry in each row and is applied in such a way that rows and columns are equally swapped in order to keep the diagonal dominance. In Figure 4.3 the reordered inner system matrix with a significantly reduced bandwidth is shown. With $\mathbf{A}_i \mathbf{x}_i = \mathbf{b}_i$ as the inner system and \mathbf{R}_s^T equal \mathbf{R}_s^{-1} , sorting can be written as follows:

$$(\mathbf{R}_s^T \mathbf{A}_i \mathbf{R}_s)(\mathbf{R}_s^T \mathbf{x}_i) = (\mathbf{R}_s^T \mathbf{b}_i). \quad (4.24)$$

The effort of the sorting algorithms as well as the effort for evaluating the required storage (cf. symbolic phase) is $O(n_{\text{non-zero}})$ each, with $n_{\text{non-zero}}$ as the numbers of non-zeros. By defining an average bandwidth b_{avg} as the average line length in the \mathbf{L} part or column depth in the \mathbf{U} part of the matrix, one can roughly estimate the space consumption as $O(nb_{\text{avg}})$ and the time consumption as $O(n/2b_{\text{avg}}^2)$ [65].

As alternative to the in-house implementation of the reordering algorithm, external packages can be employed. For example, the Boost++ [27] (see Section 5.1.5) libraries provide a graph package with respective algorithms. In [193], the *Gibbs-Poole-Stockmeyer* algorithm [75] is suggested as an efficient alternative to the *Cuthill-McKee*-based algorithms. Further alternatives are the *minimum degree* [72] or *nested dissection* [71] algorithms.

In [44], a column approximate minimum degree ordering algorithm is presented. Basically, sparse *Gaussian* elimination with partial pivoting computes the factorization $\mathbf{PAQ} = \mathbf{LU}$. While the row ordering \mathbf{P} is generated during factorization, the column ordering \mathbf{Q} is used to limit the fill-in by considering the non-zero pattern in \mathbf{A} . A conventional minimum degree ordering requires the sparsity structure of \mathbf{AP}^T .

Since the computation can be expensive, alternative approaches are based on the sparsity structure of A instead.

In [183], an introduction is given to find unsymmetric permutations which try to maximize the elements on the diagonal of the matrix. Since matrices with zero diagonal entries cause problems for both direct and iterative solving techniques, the rows and columns are permuted in order that only non-zero elements remain on the diagonal only. Due to performance considerations and the applicable diagonal strategy during factorization (see Section 4.4), the assembly module does not provide such a feature. As a consequence, the simulator is obliged to avoid zero diagonal entries, which can be done in all cases of interest.

4.10 Scaling the Inner Linear Equation System

Scaling is the final step before the inner linear equation system is passed to the solver module in order to obtain its solution. Since preconditioners like the Incomplete-LU factorization compare the entries per row, a normalized representation of the matrix has to be provided. Such a normalization is not required when external modules include their own capabilities or when different kind of preconditioners are used (see Section 5.2.5). In those cases, the scaling should be switched off in order to save the computational effort.

The standard algorithm used by default works with a two-stage strategy [66]: In the first stage, the matrix is scaled such that the diagonal elements equal unity. The second stage attempts to suppress the off-diagonals while keeping the diagonals at unity. The resulting scaling matrices S_r and S_c are diagonal matrices. With $A_i x_i = b_i$ as the inner system, the effect of sorting and scaling is given as:

$$(S_r(R_s^T A_i R_s) S_c)(S_c^{-1}(R_s^T x_i)) = S_r(R_s^T b_i). \quad (4.25)$$

In Figure 4.3 a cut-out of the scaled inner system matrix is shown. Since the values are modified while keeping the structure constant, only the colors are changed. Note the red color of the diagonal entries indicating the unity entries.

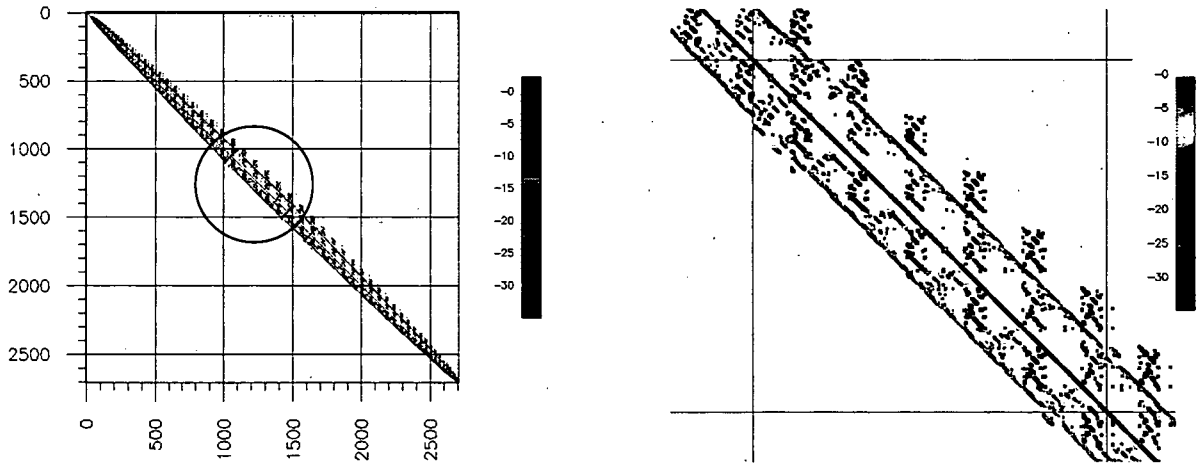


Figure 4.3: In comparison to the pre-eliminated structure, the reordering algorithm significantly reduces the bandwidth from 2,867 to 102 in order to reduce the factorization fill-in (left). The circle indicates the range of the cut-out of the scaled matrix shown in the right figure. The scaled inner system matrix has diagonal entries equal unity, which is demonstrated by the red color. Since only the values are changed, no structural difference can be seen in comparison to the sorted matrix.

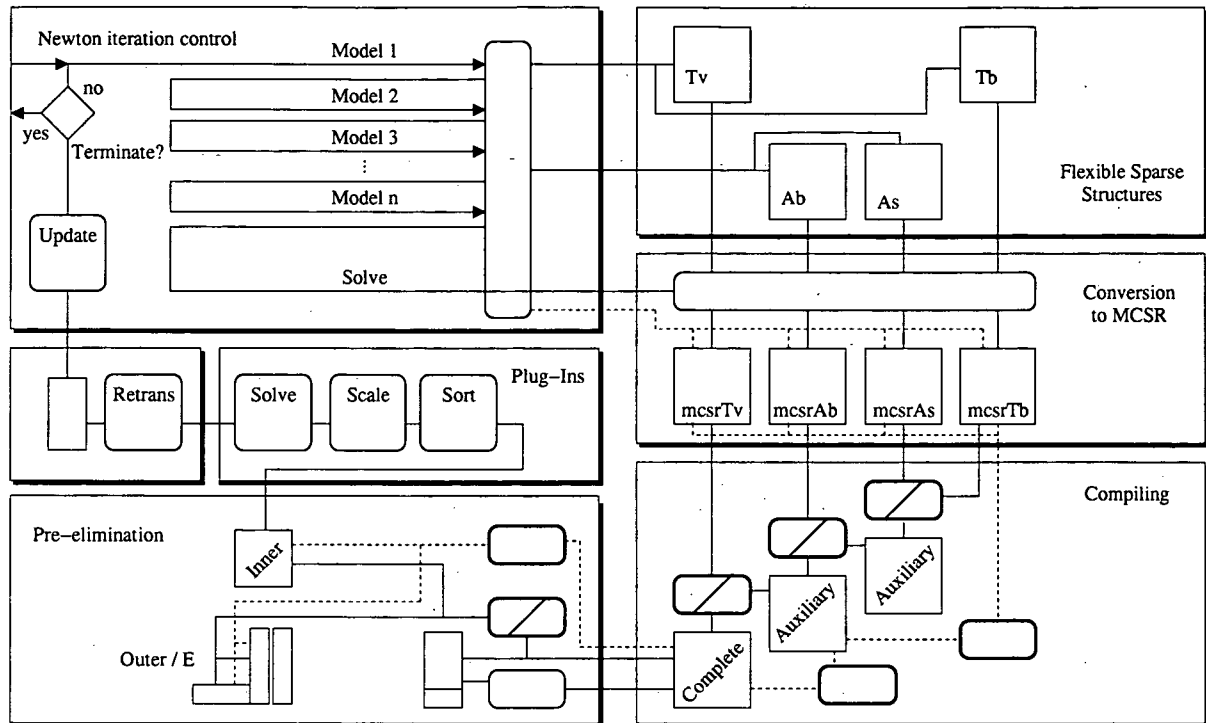


Figure 4.4: Schematic assembly overview.

4.11 Solving and Back-Substitution

The solver module is responsible for calculating the solution vector of the inner linear equation system. Basically, it is a separate module, but is invoked by the assembly module. As discussed in the next chapter, the solver module provides three in-house solvers as well as an interface to external solver modules. To summarize the various steps between assembling the linear equation system and returning its complete solution vector to the simulator, a schematic overview is given in Figure 4.4.

In the upper left corner the *Newton* iteration control of the simulator is represented, which uses an interface class to access the assembly. The inputs of the assembly module are the contributions of the various segment and boundary models implemented in the simulator, which are subsequently called. Following the black solid lines beginning at the interface, the four matrices T_b , A_s , A_b , and T_v are assembled by using the flexible sparse storage class based on balanced binary trees.

These structures are then converted to the MCSR format and compiled resulting in the complete linear system, which is pre-eliminated to obtain the inner and outer system. The inner one is sorted, scaled, and finally passed to the solver module. After the solver module has returned the solution vector (or solution vectors in case of multiple right-hand-side vectors), the assembly module has to back-substitute the pre-eliminated equations as well as revert scaling, sorting, and the variable transformation to obtain the complete solution vector(s).

The *Newton* adjustment levels (red dashed lines) reuse already existing MCSR structures to reduce the assembling effort: the flexible sparse structures may be skipped completely, and during compilation and pre-elimination much simpler functions (red bold boxes) can be used than in the conventional mode (black bold boxes with slash). The latter requires symbolic phases in order to calculate the result storage requirements, which are known for the already existing structures. In the next section, the *Newton* adjustment is described in more detail.

As the assembly module also provides an interface to a system for calculating eigenvalues, the system matrices shown in Figures 4.2 and 4.3 have been analyzed. The spectral condition for a non-symmetric matrix [52] is given by

$$\kappa_s(\mathbf{A}) = \sqrt{\frac{\lambda_{\max}(\mathbf{A}\mathbf{A}^T)}{\lambda_{\min}(\mathbf{A}\mathbf{A}^T)}}. \quad (4.26)$$

The results summarized in Table 4.1 have been obtained by the ARPACK package [128], which stands for *Arnoldi Package* and provides a collection of FORTRAN 77 subroutines for the solution of large scale eigenvalue problems. The package can be applied for *Hermitian*, non-*Hermitian*, standard or generalized eigenvalue problems. It is designed to compute a user-specified number of eigenvalues only, for example the largest or smallest eigenvalue. The ARPACK package links against the LAPACK and BLAS libraries (see Section 5.1).

System matrix after	$\lambda_{\max}(\mathbf{A}\mathbf{A}^T)$	$\lambda_{\min}(\mathbf{A}\mathbf{A}^T)$	κ_s
compiling	230.323	2.53722×10^{-35}	3.01294×10^{18}
pre-elimination	574.806	4.26668×10^{-17}	3.67042×10^{09}
sorting	574.806	4.32143×10^{-17}	3.64709×10^{09}
scaling	4.28363	1.32148×10^{-12}	1.80043×10^{06}

Table 4.1: Eigenvalues of the compiled, pre-eliminated, sorted, and scaled system matrix.

A spectral condition in the order of 10^c and the precision of the numeric representation of 10^{-p} approximately results in an error of 10^{c-p} [52]. Thus, a small error follows from a small residual only for a small condition number [188]. As it is demonstrated by the results in Table 4.1, the condition number is significantly improved by twelve orders of magnitude due to the applied measures.

4.12 Newton Adjustment

One possibility to speed up the assembly process is to reuse already allocated MCSR structures for the next *Newton* iteration step. For some applications such as semiconductor device simulation, this should be possible since the structure of the equation system hardly ever changes: on the one side, the mesh used in the simulation remains normally unmodified, on the other side changes in the *Newton* iteration regarding models, quantities, or iteration schemes occur infrequently. Furthermore, the efficiency of the *Newton* adjustment can be increased if separate equation systems are created for each activated simulation mode and iteration scheme. Consequently, the assembly module provides two assembly modes: first, the conventional assembly mode based on flexible sparse structures (see Section 4.6), and second the *Newton* adjustment mode presented here.

The benefit of the *Newton* adjustment does not only originate from fewer memory allocations and deallocations but also from the possibility to skip symbolic phases of all involved mathematical operations. For sparse matrices, a mathematical operation must first count the required number of entries in the result, which is done in the symbolic phase. Then, the result structure is allocated and the actual mathematical operation is performed.

However, if one entry in the structure intended to be reused is missing, measures have to be taken to correct this so-called *Newton* adjustment error. One possibility might be to add the missing entry to the structure, which is a cumbersome and complicated task (see also Section 4.13.3). However, it is advantageous to simply restart the complete assembly in the conventional assembly mode.

The reasons for this are:

- Restarting is a simpler solution since no additional code is required.
- The circumstances when *Newton* adjustment errors occur are known in advance. Therefore, the simulator should take care of the correct settings and avoid *Newton* adjustment errors in advance as well.
- The effort of the enlargement algorithms must not be neglected since several requirements have to be met (see also Section 4.13.3).
- For the same reasons, there is no code provided which copies the already assembled entries to flexible sparse structures and silently continues in the conventional mode.

4.12.1 The Administration Scheme

In the first implementation of the assembly module the main class was supposed to be bound to one linear equation system or one *Newton* iteration step, respectively. The class provided two administration methods: one method was responsible for allocating all required structures, the other one for their deallocation. In other words, the new modules were designed for solving one linear equation system during the life cycle between construction and destruction as illustrated in the left of Figure 4.5. Some information the next step could benefit from is stored in the separate parameter class, for example the fill-in control parameter.

This administrative scheme of the main assembly class was changed in the refined version of the module: the class itself should be used for all iterations of a complete *Newton* approximation. Only some of its members should be allocated for a single iteration only and deleted afterwards. Since the structure was divided into two parts with different life times, a division of the allocation and deallocation process as depicted in the center of Figure 4.5 was necessary.

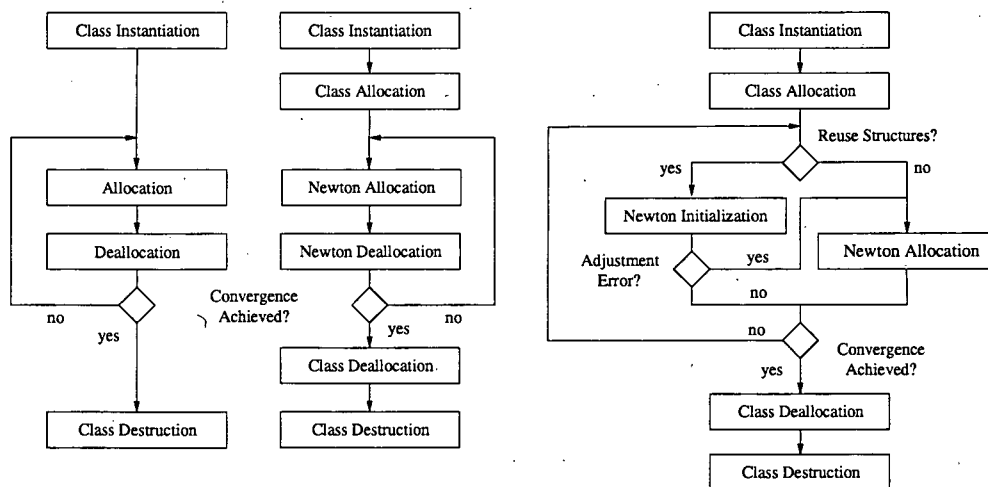


Figure 4.5: Former (left), refined (center), and final (right) administrative scheme.

The step from the former to the refined scheme does not result in a *Newton* adjustment, but lays the foundations for actually reusing already allocated structures. The final administrative scheme is shown in the right figure of Figure 4.5. If *Newton* adjustment is activated, the already existing assembly structures are not deallocated but kept in the memory after reinitialization. In the case of a *Newton* adjustment error, the conventional assembly mode is used instead. Note that in this figure the *Newton* deallocation is not separately shown any more, because it is implicitly called by the allocation and the class deallocation.

4.12.2 Newton Adjustment Levels

The *Newton* adjustments were introduced in three levels. Level two and three depend on the success of the preceding one.

1. The first level reuses the structure of all four matrices of the assembly process: A_b , A_s , T_b , and T_v . If a *Newton* adjustment error occurs, the simulator must correct the situation, preferably by restarting the complete assembly in the conventional assembly mode.
2. The second level reuses all resulting structures created during the compilation of the complete linear equation system. In contrast to the first level the assembly process is already finished at execution time. Thus, in the case of a *Newton* adjustment error the simulator is not affected at all.
3. The third level reuses all matrices resulting from the pre-elimination process. As for the second step, *Newton* adjustment errors are corrected internally.

An overview about the *Newton* adjustment is given in Table 4.2.

Level	Status	Target	Errors
1	assembly	A_b , A_s , T_b , and T_v	handled by simulator
2	compilation	compilation results	handled internally
3	pre-elimination	pre-elimination results	handled internally

Table 4.2: *Newton* adjustment levels.

The *Newton* adjustment levels do not change the sequence of mathematical operations, so both assembly modes yield exactly the same linear equation systems and exactly the same solution is obtained. Thus, the *Newton* adjustment levels are a performance improvement feature only – and do not affect the accuracy or any values at all.

4.12.3 Improved Sorting Feature

In the implementation of the sorting permutation of the inner linear equation system an additional potential for speed-up can be identified. This possibility is related to the *Newton* adjustment, because a conventional mode is involved as well. In this mode, a sorting vector is calculated and applied. However, once the sorting vector is known, it can be already used during pre-elimination. This saves not only the recalculation of the sorting vector, but makes it also possible (and necessary) to skip the sorting permutation.

The lower part of the pre-elimination vector is basically initialized by ascending indices. If both sorting and pre-elimination are enabled, the already existing sorting vector can directly become the lower part of the pre-elimination vector. If the inner system matrix is already sorted, the subsequent sorting must be skipped. As already discussed for the *Newton* adjustment levels, the simulator is responsible for setting the appropriate parameters to have the sorting vector recalculated if respective modifications make this necessary.

In contrast to the *Newton* adjustment levels, the improved sorting feature causes a changed sequence of mathematical operations and thus different results are observed.

4.13 The Transferred-Transformation Problem

By comparing saved MCSR structures of the same system matrix assembled by the formerly used and the new module, misordering problems have been detected. As a consequence, the new modules were presumed to deliver wrong results and an investigation of the problem was started which finally resulted in an elegant solution for the problem which is directly related to the row transformation.

The row transformation as discussed in Section 4.7.1 is used to supply equations (or rows of the system matrix) with equations having a higher priority. This is necessary, because several segments (with different priorities) could contain the same grid point. Each of the control functions determines a result for this grid point, but only the one with the highest priority should be used.

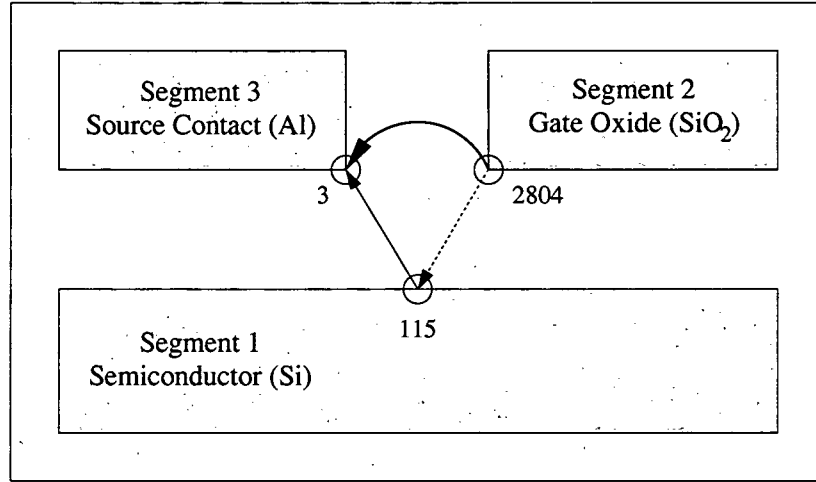


Figure 4.6: Transformations involved if three segments share one physical grid point.

See Figure 4.6 for an illustration of the physical background as it occurs for a MOS structure. The semiconductor segment (segment 1), the gate oxide (segment 2), and the source contact (segment 3) share one physical grid point. Thus, three equations are assembled to obtain the solution for the potential at this point.

Since the contact segment has the highest priority, its solution has to be used by the other two segments. However, during the assembly of the three interface equations, the interface models (which are responsible for setting up the transformation matrix), do not have the full information about all three segments involved. They only take two segments into account, namely the segments the interface is in-between. As a consequence, the following transformation entries can be found in T_b^T :

1. Equation 3 belongs to the boundary charge of the source contact (segment 3) and has to be completed by contributions from the neighboring segments.
2. Equation 115 belongs to the potential quantity of the semiconductor segment and is transferred to equation 3 in order to calculate the boundary charge. Its diagonal entry is zero (a substitute equation is used instead) and the single off-diagonal entry in column 3 equals one. Note, that in T_b^T the column index is the target of the transformation. This transformation is represented by the black/solid arrow in Figure 4.6.
3. Equation 2804 belongs to the potential quantity of the gate oxide segment. Since semiconductor segments have a higher priority than oxide segments, a substitute equation is used: $-\psi_{115} + \psi_{2804} = 0$. While the diagonal entry of row 2804 of T_b^T is zero, the only off-diagonal entry being non-zero is that of column 115 and equals one. This transformation is represented by the red/dashed arrow in Figure 4.6.

These transformations result in the following equations of the complete linear equation system. Depending whether a correction is activated or not, equation 3 is shown in (4.27) and (4.28), respectively. C_x stand for other contributions equal for both cases and thus not of interest here. The coefficients D_x denote entries wrongly transferred to (4.30) instead of (4.27). With $\psi_{C,3}$ as the contact potential of segment 3 (source contact), the correct and wrong equation 115 can be seen in (4.29) and (4.30), respectively:

$$Q_3 + C_{\psi/s1} + C_{n/s1} + C_{p/s1} + C_{\psi/s2} + D_1 \psi_{2804} + D_2 \psi_{2805} + D_3 \psi_{2807} = C_{\text{rhs}} + D_{\text{rhs}} \quad (4.27)$$

$$Q_3 + C_{\psi/s1} + C_{n/s1} + C_{p/s1} + C_{\psi/s2} = C_{\text{rhs}} \quad (4.28)$$

$$-\psi_{C,3} + \psi_{115} = 0 \quad (4.29)$$

$$-\psi_{C,3} + \psi_{115} + D_1 \psi_{2804} + D_2 \psi_{2805} + D_3 \psi_{2807} = D_{\text{rhs}} \quad (4.30)$$

With			
Source	$V = 1.0 \text{ V}$	$I = -6.9374202644 \times 10^{-17} \text{ A}$	$Q = -5.3344762188 \times 10^{-19} \text{ C}$
Gate	$V = 1.2 \text{ V}$	$I = 0.0000000000 \times 10^0 \text{ A}$	$Q = 1.1515548905 \times 10^{-15} \text{ C}$
Drain	$V = 1.2 \text{ V}$	$I = 1.9820343061 \times 10^{-16} \text{ A}$	$Q = -5.3344761939 \times 10^{-19} \text{ C}$
Bulk	$V = 0.0 \text{ V}$	$I = -1.2882922796 \times 10^{-16} \text{ A}$	$Q = -4.0243642506 \times 10^{-20} \text{ C}$
Without			
Source	$V = 1.0 \text{ V}$	$I = -6.9354539375 \times 10^{-17} \text{ A}$	$Q = -2.6653626903 \times 10^{-16} \text{ C}$
Gate	$V = 1.2 \text{ V}$	$I = 0.0000000000 \times 10^0 \text{ A}$	$Q = 1.1515496006 \times 10^{-15} \text{ C}$
Drain	$V = 1.2 \text{ V}$	$I = 1.9818396270 \times 10^{-16} \text{ A}$	$Q = -2.6653626903 \times 10^{-16} \text{ C}$
Bulk	$V = 0.0 \text{ V}$	$I = -1.2882936666 \times 10^{-16} \text{ A}$	$Q = -4.0243767580 \times 10^{-20} \text{ C}$

Table 4.3: Comparison of terminal quantities with and without the correction. Note that the boundary charges are significantly different.

Since the segment equation should also be transferred to the equation of the boundary charge, the red/dashed transformation is wrong and should be replaced by the green/thick one. As a consequence, the gate oxide is transferring its incomplete equation to the semiconductor segment. Such situations could be prevented if the respective models are provided with the complete information. Since this is not the case for the main simulator MINIMOS-NT and for unstructured meshes an arbitrary number of segments would have to be considered, the assembly module was equipped with an algorithm which corrects the transformation matrix before it is applied. This algorithm has the full information available, but remains deactivated for all simulators which do not require this correction. See Table 4.3 for a comparison of the terminal quantities showing significant differences particularly for the boundary charge.

The reason for the differences between the formerly used and new module can be explained as follows: one important difference between the modules is the assembly sequence. The new assembly module assembles four matrices and compiles them afterwards to the complete linear system (see Section 4.14). This compiling step uses matrix additions and multiplications which operate on the completely assembled structures.

In contrast, the formerly used module calculates all transformations during a symbolic assembly phase in advance and is therefore able to assembly only one system matrix during the actual assembly phase (cf. *Newton* adjustment). So the matrix additions and multiplications are performed immediately while an entry is added. As analyzed, these transformations do not perform a multiplication in the strict mathematical form, but are already adapted for the requirements described above. This behavior should be demonstrated on a simple mathematical example.

4.13.1 Mathematics

For the demonstration of the matrix transformation, consider a simple A_s matrix multiplied by T_b from the left:

$$A = T_b A_s, \quad (4.31)$$

$$A_s = \begin{pmatrix} 1.0 & & & \\ & 2.0 & & \\ & & 3.0 & \\ & & & 4.0 \end{pmatrix} \quad (4.32)$$

By using three different transformation matrices, the actual problem should be made evident. The first transformation matrix and the multiplication result are:

$$T_b = \begin{pmatrix} 0.0 & & 1.0 & \\ & 1.0 & & \\ & & 1.0 & \\ & & & 1.0 \end{pmatrix}, \quad A = T_b A_s = \begin{pmatrix} & & & 4.0 \\ & 2.0 & & \\ & & 3.0 & \\ & & & 4.0 \end{pmatrix} \quad (4.33)$$

The entries of T_b can be interpreted as follows: the column index is interpreted as source, the row index as target of a transformation. The respective equations of A_s are scaled by the value of these positions of T_b . In the example all values in T_b are either one or zero. The entry in row zero and column three is taken as an example: equation three of A_s (the source) is completely (factor one) added to the equation zero of A . Since in the first column of T_b all entries are zero, the first equation of A_s is never used.

The second transformation matrix and the multiplication result are:

$$T_b = \begin{pmatrix} 0.0 & & 1.0 & 1.0 \\ & 1.0 & & \\ & & 0.0 & \\ & & & 1.0 \end{pmatrix}, \quad A = T_b A_s = \begin{pmatrix} 0.0 & & 3.0 & 4.0 \\ & 2.0 & & \\ & & 0.0 & \\ & & & 4.0 \end{pmatrix} \quad (4.34)$$

The interpretation is straightforward: Now equation zero is a linear combination of the equations two and three. Row two of A is zero because there are no non-zero entries in row two of T_b .

The third transformation matrix and the multiplication result are:

$$T_b = \begin{pmatrix} 0.0 & & 1.0 & \\ & 1.0 & & \\ & & 0.0 & 1.0 \\ & & & 1.0 \end{pmatrix}, \quad A = T_b A_s = \begin{pmatrix} 0.0 & & 3.0 & \\ & 2.0 & & \\ & & 0.0 & 4.0 \\ & & & 4.0 \end{pmatrix} \quad (4.35)$$

Now equation three is transferred to equation two, which in turn is transferred to equation zero. The result of this operation is mathematically correct, but does not meet the requirement of the assembly process. The result should be the same as for the second transformation matrix. If one equation is transferred to an equation which is transferred itself to another target, it must be further transferred to that target. Hence, the problem was called *transferred-transformation problem*.

4.13.2 The Basic Correction Algorithm

As discussed above, the formerly used module takes such situations into account while calculating all contributions of a matrix entry. However, the new assembly module requires a correction algorithm before the mathematically correct matrix multiplication is processed during the compiling process. For the following reasons, the development of a correction algorithm is a crucial part for the success of the new modules:

- Waiving a particular assembly sequence is a very important benefit of the new assembly module. However, this benefit fundamentally depends on the matrix multiplication.
- The correction code of the particular matrix can be turned on and off by keeping the original compilation code unmodified. This is advantageous if the (mathematically correct) multiplication algorithm is once replaced by a more optimized version (cf. LAPACK).
- A correction code can deliver the required result since the complete information is available.
- A correction code can deliver the required result in an efficient way because only few entries are involved and the effort is linear.
- A correction code fully fits in the *Newton* adjustment concept of the new module (see below).
- Instead of a correction algorithm, the matrix multiplication itself could be modified in the required manner. This would actually result in two implementations: one mathematically correct and one “incorrect”. In this case the doubled code in a generally applicable module would have to be justified by a special-purpose requirement.

For these reasons, it was decided to develop a correction algorithm which is presented here. Based on the third example in the last section, the problem is described verbally once again: equation three is transformed to equation two, that is transformed itself to equation zero: thus, a transformation source is also a transformation target. A correction to replace the (three to two to zero) problem by a straightforward (three to zero) transformation is required.

The basic correction algorithm can use all information provided by the transposed MCSR data structure. In T_b^T , the column indices represent the targets of the transformations.

$$T_b^T = \begin{pmatrix} 0.0 & & & & & & \\ & 1.0 & & & & & \\ 1.0 & & 0.0 & & & & \\ & & & 1.0 & 1.0 & & \end{pmatrix} \quad (4.36)$$

Note that the transposed MCSR structure is actually stored in the MCSC format. For simplification, the discussion here still uses the MCSR format. The index and value arrays of the T_b^T matrix read like this (*u* stands for unused, see Appendix E.1):

pos	0	1	2	3	4	5	6
val	0.0	1.0	0.0	1.0	<i>u</i>	1.0	1.0
idx	5	5	5	6	7	0	2

The algorithm has to focus on the off-diagonal entries in the upper part of the arrays. The column index stands for the target of the transformation. Hence it must be checked if the target row is a source of a transformation which is indicated by off-diagonal entries.

$$\text{idx}[\text{idx}[\text{row}]] \neq \text{idx}[\text{idx}[\text{row}] + 1] \quad (4.37)$$

The correction algorithm loops over all off-diagonal entries, starting at position five in the example. For the first entry the test is negative since

$$\text{idx}[\underbrace{\text{idx}[5]}_0] = \text{idx}[\underbrace{\text{idx}[5] + 1}_1] = 1 \quad (4.38)$$

However, for the second entry the test is positive:

$$\underbrace{\text{idx}[\text{idx}[6]]}_2 \neq \underbrace{\text{idx}[\text{idx}[6] + 1]}_3 . \quad (4.39)$$

Row two does have off-diagonals, so a transferred transformation is detected. The basic algorithm is able to correct such situations if there is one entry only which is checked by the following condition:

$$\text{idx}[\text{idx}[\text{row}]] + 1 == \text{idx}[\text{idx}[\text{row}] + 1] . \quad (4.40)$$

More than one entry (as required in cases where more than three segments share one grid point) would require a restructuring of the complete MCSR array or alternative approaches as discussed in the next section. If there is only one entry, the correction is just a simple assignment:

$$\text{idx}[\text{row}] = \text{idx}[\text{idx}[\text{idx}[\text{row}]]] , \quad (4.41)$$

$$\text{idx}[6] = \underbrace{\text{idx}[\text{idx}[\text{idx}[6]]]}_5 = 0 . \quad (4.42)$$

After the correction is completed, \mathbf{T}_b^T has to be transposed back resulting in the same transformation matrix as used in (4.34):

$$\mathbf{T}_b = \begin{pmatrix} 0.0 & & 1.0 & 1.0 \\ & 1.0 & & \\ & & 0.0 & \\ & & & 1.0 \end{pmatrix} . \quad (4.43)$$

4.13.3 The Advanced Algorithm

The basic algorithm explained above could not be used for more complicated structures, for example the ones which contain grid points calculated by four control functions.

In addition, duplicate entries in the resulting MCSR structure (as they could occur using the basic algorithm) should be avoided. In contrast to the full format, a sparse format stores only entries which are intended to be non-zero. Besides the actual value, also the row and column number have to be stored. Thus, it is possible to store one matrix entry more than one time, which is a duplicate entry of the same position. Most of the mathematical operations defined for MCSR take those implicitly into account, since they simply process all entries in the structure. However, due to the fact that all multiple entries should be actually summed up to one entry, numerical inaccuracies may occur. So duplicate entries are best avoided at all.

The main objective of the advanced algorithm should provide a generally applicable correction of the transformation matrix while avoiding duplicate entries. A new example should demonstrate the actual problem. In a transposed transformation matrix \mathbf{T}_b^T the entries (row:34, col:3) and (47, 34) are set to one. The basic algorithm corrects the latter entry to (47, 3). Since there is only one entry, this correction is successful (case one).

To extend the example, an additional entry in (78, 34) is supposed to be one. That means, that equation 34 is not only transferred to equation 3, but also to equation 78. Note that in \mathbf{T}_b^T the column index stands for the target of a transformation. In that case, the basic algorithm fails, because there is not enough space to add new entries (case two). Both cases are graphically represented in Figure 4.7.

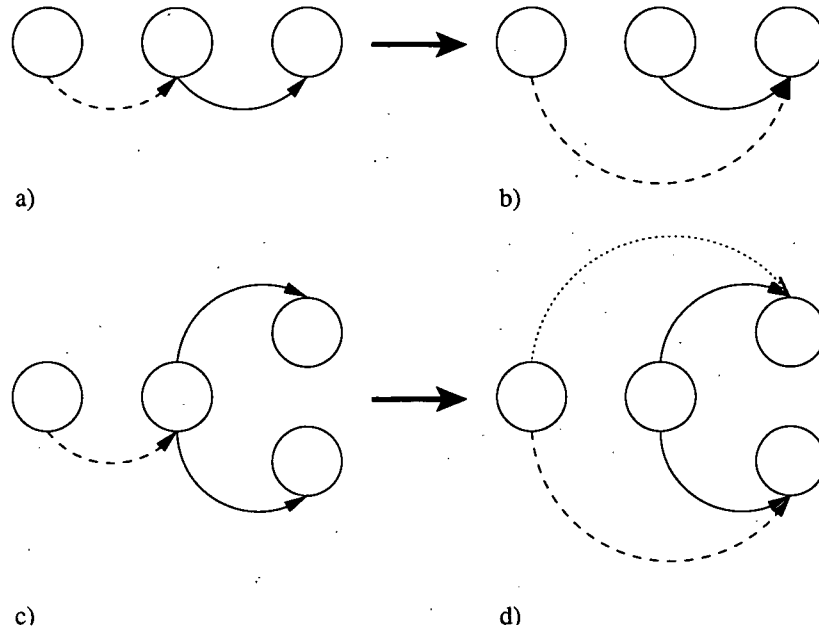


Figure 4.7: Graphical representation of the multiple transfer problem.

The first case is shown in Figure 4.7a and Figure 4.7b. In Figure 4.7a, two arrows represent two transformations, the blue/dashed arrow has to be redirected to the position the black/solid arrow points to. In Figure 4.7b, the green/dashed arrow represents the redirection. After the correction the number of arrows stays the same.

The second case is shown in Figure 4.7c and Figure 4.7d. Since the center position is transferred to multiple (here two, generally n) right positions, there are also n arrows needed to point from the left position to all right positions. The number of arrows increases by $n - 1$. In Figure 4.7d, the red/dotted arrow has to be created additionally. In the existing MCSR structure, there is no space for this entry. Hence, no thorough correction of the second case could be made.

If equation 47 of A_s contains the non-zero column entries 49, 51, and 55, the following misorderings are the result of the omitted correction:

- Equation 3: entries in columns 49, 51, and 55 are missing.
- Equation 34: entries in columns 49, 51, and 55 should not be there.
- Equation 78: entries in columns 49, 51, and 55 are missing.

For that reason new entries have to be added in order to completely correct the transformation matrix. There are two solutions for this problem:

1. The first approach requires an enlargement and reordering algorithm for the existing arrays. This algorithm must take the requirement into account, that duplicate entries should be avoided. Thus, the resulting algorithm must count all additional needed entries actually required, allocate a new structure, copy all entries and eventually dismiss the old structure. However, all of these parts are already implemented, which motivates the second approach.
2. The flexible sparse structure of the transformation matrix provides already many of these required features: it can be used to add new entries easily and it avoids duplicate entries. So the second approach yields an algorithm that reuses and iteratively improves the still existing flexible sparse structure of the transformation matrix.

By pursuing the second approach, an algorithm was developed which benefits from already existing and applied implementations. The system can now take all possible situations into account since it does not limit the number of commonly used grid points (independent of the number of spatial dimensions) any more. Table 4.4 summarizes the initial situation and the correction result for both the transposed and untransposed transformation matrix.

Before:	T_b	3	34	47	78
	3	1.0	1.0		
	34			1.0	
	47				
	T_b^T	3	34	47	78
	3	1.0			
	34	1.0			1.0
	47		1.0		
After:	T_b	3	34	47	78
	3	1.0	1.0	1.0	
	34				
	47				
	T_b^T	3	34	47	78
	3	1.0			
	34	1.0			1.0
	47	1.0		1.0	
	T_b	3	34	47	78
	3	1.0	1.0	1.0	
	34				
	47				
	T_b^T	3	34	47	78
	3	1.0			
	34	1.0			1.0
	47	1.0		1.0	

Table 4.4: The transposed and untransposed transformation matrix before and after the correction. The column index stands for the source or the target of the transformation, respectively.

Regarding the *Newton* adjustment it is important to note, that the advanced algorithm is fully employable due to the following considerations. After a successful first *Newton* adjustment level, no flexible sparse structure exists which could be used to add additional entries. However, the already existing T_b matrix does already contain all required entries. This assumption holds since all deterministic algorithms always yield the same result for the same inputs. Therefore, no *Newton* adjustment errors will occur.

4.14 Concluding Remarks

At the end of this chapter concluding remarks shall be given in order to summarize the benefits of the in-house assembly module.

- Shared library concept: the simulator can be directly linked against the assembly module.
- Comprehensive Application Programming Interface: this API enables random access to all structures and is therefore an important contribution to simplify and improve model development. See Figure 4.8 for a comparison of the four-phases and one-phase approach.
- Rigorous implementation in C/C++: 30,500 lines of code.
- C++-Templates (type parameterization): efficient implementation based on templates [212].
- Handling of real-valued and complex-valued equation systems.
- Ability to provide multi-level solvers with the respective connectivity information: the simulator can use the assembly module to administer its quantities.
- Centralized administration of control parameters.
- Comprehensive input-/output system: all structures can be written to files and read from the files to be further processed. These features can be used to check the assembly and to efficiently evaluate

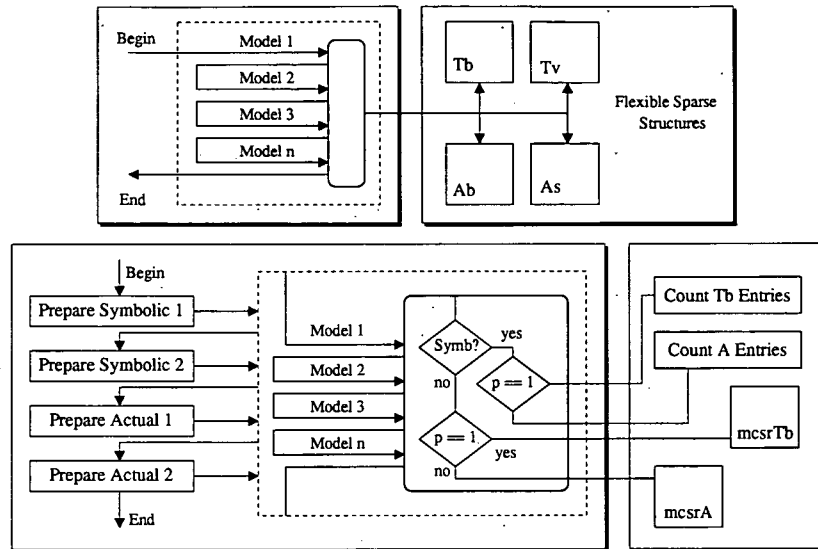


Figure 4.8: Comparison of the one-phase (upper) and four-phases (lower) approach. In the latter case the implementation of the simulator is much more complicated as the assembly module requires a specific assembling sequence. Whereas in the one-phase approach the loop over all models is processed only once, it has to be processed four times in the four-phases approach. In addition it is necessary to call specific preparation functions and each model implementation has to take the current phase into account, which leads to complicated codes.

different parameter settings for the solution of the same linear equation system. In addition these files can be used for debugging and quality assessment of the simulator as they include the full quantity information for all equations.

- Stand-alone version: several test programs are available to see how the library is applied.

The ability to be generally applicable has been proved as the assembly module is used for another simulator at the institute, namely the *Finite Element Diffusion and Oxidation Simulator* FEDOS [32].-

Due to the rigorous implementation in C++ and the application of inline methods, the performance difference to the former version, which was implemented in C, is minimal. Especially in combination with the *Newton* adjustment, the new system is not systematically slower than the old one.

Chapter 5

The Solver Module

The assembly and solution of sparse linear equation systems is a fundamental task in numerical simulators which discretize nonlinear partial differential equations on a mesh. As already discussed in Section 2.3.1, the *Newton* method [136] is commonly used as a linearization technique, which requires the solution of one linear equation system per iteration step.

As solving linear equation systems is a common and well-known computational task, an overview of third party solutions is given in Section 5.1. However, there are four main reasons for providing, maintaining, and extending an in-house solver module:

1. Whereas the calculation of the model contributions represents the physical modeling and thus the main purpose of employing TCAD tools in general, the largest share of the run-time of the numerical simulators is spent in the linear modules, that is for assembling and solving linear equation systems. In order to quantify this statement, a respective evaluation was performed, which is discussed in Section 5.5.2. A subsequent evaluation was performed to analyze the performance of various solver systems for different kinds of simulation tasks. So the in-house solver module does not provide only one linear solver, but an interface to various in-house and external solver systems.
2. The solvability of a linear equation system depends on specific properties of the system itself, for example the condition of the system matrix. Although several measures have already been taken to improve these properties (see Chapter 4), some kinds of solver techniques may still fail during the calculation of the solution. For that reason, again a choice of several different solver systems can increase the probability for finding a useful solution for the complete simulation task. However, insufficient convergence for example may also point to inappropriate simulation setups such as inadequate meshes or inaccurate physical modeling. Hence, the behavior of the solver modules and respective feed-back information can be used to assess and improve the complete simulation. It is therefore advantageous to benefit from a direct access to the solver module.
3. External modules are often bound to license agreements, which frequently contain restrictions especially for commercial application. As the institute provides its codes to industrial partners and binary release versions to the general public, third party license requirements would restrict such distributions. So an in-house solver module enables the institute to independently release complete versions which are directly applicable also from a legal point of view.
4. The quality assessment approach of MINIMOS-NT (see Section C.4) requires a deterministic behavior of the solver system both in the short and long run. By using the in-house solvers, which are intended to remain basically unmodified, this behavior can be assumed to be guaranteed.

5.1 Third Party Modules and Libraries

Since solving linear equation systems is a very common computational task, a short overview of the available third party solutions shall be given.

The *Basic Linear Algebra Subprograms* BLAS provide high quality building block routines for performing basic vector and matrix operations: Level 1 BLAS for vector-vector operations, Level 2 BLAS for matrix-vector operations, and Level 3 BLAS for matrix-matrix operations [18]. Since BLAS libraries are efficient, portable, and widely available for many different platforms, many mathematical developments are based on them.

The original objective of the *Linear Algebra PACKage* LAPACK, which is based on the BLAS library, was to provide a version of the widely used EISPACK [202] and LINPACK [53] libraries that runs efficiently on shared memory vector and parallel processors. The bottle necks of these modules are the memory access patterns disregarding the multi-layered memory hierarchies of the machines. Thus, too much time is spent for data manipulation rather than for doing useful floating-point operations. LAPACK addresses this problem by making use of block matrix operations, such as matrix multiplication, in the innermost loops. Since these block operations can be optimized for various architectures to account for the memory hierarchy, a way to achieve high efficiency on diverse modern machines was found [13]. The Scalable LAPACK (SCALAPACK) library provides a subset of LAPACK routines for distributed memory parallel computers [21]. Whereas SCALAPACK can be applied for dense systems, a library for parallel linear algebra computation on sparse matrices is presented in [64]. That PSBLAS package addresses the parallel implementation of iterative solvers and is designed for distributed memory computers.

5.1.1 Commercial Libraries

The NAG libraries of the *Numerical Algorithms Group* contain a wide range of robust numerical and statistical routines, for example for linear algebra, eigenvalue analysis, and differential equations [155]. The libraries are offered in FORTRAN 77 and 90, C, as well as an SMP Library for shared memory and a Parallel Library for distributed memory parallel computing.

The IMSL libraries of *Visual Numerics* provide accurate and reliable FORTRAN algorithms with full coverage of mathematics and statistics. According to [226], the libraries are claimed to be a cornerstone of high-performance and deep computing as well as predictive analytics applications in science, technical and business environments for well over three decades. IMSL stands for *International Mathematical and Statistical Library*.

The HSL Software Library, formerly known as the *Harwell Subroutine Library* and commercially distributed by *Hyprotech UK Ltd*, offers a collection of portable, fully documented FORTRAN packages. Besides the commercial distribution, there are special arrangements for licensing to academic users. The library can be particularly applied for sparse matrix computations and large-scale optimization. HSL routines can be found in advanced software applications such as for chemical engineering and finite element modeling.

5.1.2 Libraries of Hardware Vendors

Intel offers the *Math Kernel Library* MKL to provide highly optimized thread-safe mathematical routines for High-Performance Computing (HPC), science, engineering and financial applications, which are able to take advantage of the maximum performance on *Intel* processors [107]. The key features of the MKL are its optimization for recent *Intel* platforms, an automatic run-time detection of the CPU actually used, scaling on multi-processor environments, thread-safety, and royalty-free distribution of the run-time li-

brary. The Linux non-commercial development license is also free. The functionality embraces not only linear algebra (BLAS and LAPACK), but also discrete *Fourier* transforms, vector mathematics and a vector statistical library with random number generators.

The AMD *Core Math Library* ACML for Linux and Windows offers BLAS, LAPACK and FFT routines, which can be applied by a wide range of software developers to obtain excellent performance on AMD platforms. The highly optimized library provides numerical functions for mathematical, engineering, scientific and financial applications. ACML is available both as a 32-bit and 64-bit library which is able to fully exploit the large memory space and improved performance offered by new AMD 64-bit architectures [3].

Applications running on IBM pSeries computers can take advantage of optimized ESSL (*Engineering and Scientific Subroutine Library*) routines. This allows both to reduce programming requirements and performance improvements. The library provides optimized versions of dense matrix kernels contained in the BLAS and LAPACK libraries. Another approach to improve the performance of applications is the IBM *Mathematical Acceleration SubSystem Library* MASS. It provides high-performance versions of a subset of intrinsic functions while sacrificing a small amount of accuracy [2].

5.1.3 Recent Developments

In [190], an overview of parallel frontal solvers for large sparse linear systems can be found. These codes were implemented in the HSL package. As a variant of *Gaussian* elimination, the frontal method was originally developed [109] to solve large linear equation systems. Because the computational work is limited to a relatively small frontal matrix, large problems could be solved even on computers with small memories. The assembly and elimination procedure were combined, and the additional data was written to secondary storage systems. Today the motivation for frontal solvers arise from the fact, that the frontal matrices can be stored in full format [191]. This allows one to employ high-level BLAS routines. Furthermore, multiple front methods were developed, which can simultaneously process subproblems, and allow to parallelize the factorization.

A new parallel direct solver for large sparse linear systems is presented in [56], which is also incorporated in the HSL package as the HSL_MP48 algorithm. This algorithm is designed to solve highly unsymmetric systems by employing several processors, typically up to 16. It is based on the serial code MA48, which implements a sparse variant of *Gaussian* elimination with sparse data structures and threshold pivoting. Since MA48 stores the matrix and its factorization in main memory, HSL_MP48 can parallelize the problem in order to reduce the memory demand as well as to speed up the factorization process. The parallel approach is based on partitioning the system matrix into a small number of loosely connected submatrices (singly bordered block diagonal form [56]). A modified version of MA48 is then used to factorize all matrices which can be done in parallel. The interface problem can be solved by any sparse solver. Three advantages in comparison to a general parallelization of a sparse solver are given: all processors can be preassigned all data in advance, the factorizations of the submatrices can be done in parallel, and interprocessor communication as required for the interface problem is limited and structured. For the majority of test cases arising from chemical process engineering problems HSL_MP48 outperforms MA48 if two processors are employed. For some problems and eight processors speed-ups in excess of four are achieved. Since the code does not require a single file system and is based on MPI [159], it can be applied both on shared memory and distributed memory machines (see Section 5.3.1).

The comparison in [93] involves different strategies mainly regarding the numerical factorization and the target architecture. However, it is concluded that improvements in all phases of solving sparse linear equation systems have been achieved, which are the main reasons for the significant speed-up already observed.

The following solvers are compared [93]:

- UMPACK, the *Unsymmetric Multi-Frontal Package*, is a set of serial routines for solving unsymmetric sparse linear systems using the unsymmetric multi-frontal method. Whereas versions 2.2 and 3.1 are compared in [93], [47] provides version 4.4 now.
- SuperLU_{MT} and SuperLU_{dist} [223]: whereas the former is designed for *Shared Memory Parallel Processors* (SMPs), the latter is designed for *Distributed Memory Parallel Processors* (see Section 5.3.1). The factorization is based on the super-nodal left-looking or right-looking approach, respectively.
- SPOOLES stands for *SParse Object Oriented Linear Equations Solver* [140] and is a library for solving sparse real- and complex-valued linear equation systems. Numerical factorization is done by a super-nodal *Crout* [131] approach. The target architectures include serial, shared-memory parallel, and distributed-memory parallel.
- MUMPS stands for *MUlti-frontal Massively Parallel sparse direct Solver* and provides capabilities for solving linear equation systems with symmetric positive definite matrices, general symmetric matrices, general unsymmetric matrices, complex or real arithmetic matrices. Furthermore, it offers parallel factorization as well as iterative refinement and backward error analysis [163]. The implementation is written in FORTRAN 90 and based on MPI. Partial pivoting and dynamic distributed scheduling is used to accommodate both numerical fill-in and multi-user environment. The BLAS, LAPACK, and SCALAPACK packages are employed. The numerical factorization is based on symmetric-pattern multi-frontal and the target architecture is distributed-memory parallel.
- WSMP: The *Watson Sparse Matrix Package* (WSMP) is a high-performance software package for solving large sparse linear equation systems by a direct method on IBM pSeries Systems, serial or multi-processor Linux workstations and Linux clusters [2]. WSMP can be used as a serial package, in a shared-memory multiprocessor environment, or as a scalable parallel solver in a message-passing environment, where each node can either be a uniprocessor or a shared-memory multiprocessor. The numerical factorization is based on unsymmetric-pattern multi-frontal and the target architecture is shared-memory parallel.

Two additional modules are PARDISO [182] and SAMG [37]. The former is also provided by DESSIS [111] and the MKL [107]. The latter is industrially employed for computational fluid dynamics as used in the car industry, oil reservoir simulations, electromagnetics, groundwater flows, semiconductor physics, and structural mechanics [213]. Both packages are discussed later in this chapter in a more detailed way, see Section 5.3.2 and Section 5.3.3, respectively.

5.1.4 Solver Frameworks

The *Boeing Math Group* commercially provides the *Intelligent Iterative Solver Service* IISS, which is based on the performance considerations regarding the advantage of iterative solvers for large sparse linear equation systems. The optimal selection of an iterative solver varies and these methods are also known to suffer from irregular performance in different applications. Furthermore, it can be difficult for users to choose the correct parameters required for performance. For that reason the *Boeing Math Group* provides a collection of state-of-the-art iterative solvers [24].

This collection contains BCSLIB-EXT [23], a preconditioned iterative solver, which has been also used for semiconductor device simulation. Hypre [63], a scalable software for solving large, sparse linear equation systems on massively parallel computers, and AZTEC [222], a massively parallel iterative solver library with advanced partitioning techniques and dense matrix algorithms, are additionally available [24].

Trilinos is a project aiming to provide parallel solver algorithms and libraries for the solution of large-scale, complex multi-physics engineering and scientific applications. Since an object-oriented software framework is developed, *Trilinos* is based on packages [180]. Each package is focused on important, state-of-the-art algorithms in its problem regime, developed by a small expert group, self-contained, as well as configurable, buildable and documented on its own [61]. For example, *Trilinos* supports the object-oriented version of AZTEC, SuperLU, Hypre, HSL and many more. In addition, three specific package collections are provided: one for linear algebra classes, one for abstract solver classes, and one for basic tools like BLAS or MPI interfaces.

5.1.5 Special-Purpose Libraries

The *Automatically Tuned Linear Algebra Software* ATLAS provides portable linear algebra software, including a complete C or FORTRAN 77 BLAS API and a very small subset of the LAPACK API. For all supported operations, ATLAS achieves performance comparable with machine-specific tuned libraries [224, 238]. Thus, ATLAS can be used as an alternative to the BLAS library.

Blitz++ is a C++ class library for scientific computing showing a comparable performance with FORTRAN 77/90 [157]. The library incorporates dense arrays and vectors, random number generators and small vectors and matrices. It is available under an open source license. The basic idea is to use the advanced features of the C++ programming language while avoiding the performance drawback. The speed-up is not achieved by better compiler optimization and related measures, but by using templates, which allow to perform optimizations such as loop fusion, unrolling, tiling, and algorithm specialization automatically at compile time. Thus, instead of compiler optimizations, the Blitz++ library parses and analyzes array expressions at compile time and performs respective loop transformations in order to speed-up the code. In addition, Blitz++ extends the conventional dense array model to incorporate new and useful features, such as flexible storage formats, tensor notation and index placeholders [157].

Finally, there is Boost++, a set of free portable C++ source libraries [27] which fully comply to the C++ standard and work well with the C++ standard library. It is based on C++-templates and the system is said to be included into the next C++ standard. One template class library regards linear algebra: UBLAS provides BLAS functionalities for dense, packed, and sparse matrices.

The design and implementation is based on operator overloading and efficient code generation via templates. Classes are provided for dense, unit, and sparse vectors, dense, identity, triangular, banded, symmetric, hermitian, and sparse matrices. Ranges, slices, and adaptor classes can be defined to view into vectors and matrices. The library provides reductions like different norms, addition and subtraction of vectors and matrices and multiplication with a scalar, inner and outer products of vectors, matrix-vector and matrix-matrix products and triangular solver. The implementation is mostly STL conform including the iterator interface [26]. The known limitations are that the implementation assumes a linear memory address model and that dense matrices have been the tuning focus.

5.1.6 Discussion

As can be easily seen, there is a large variety of solver systems and packages available and many of them are still developed and improved. As they are highly optimized, particularly for specific platforms, the two-strategy approach for the in-house solver module presented here seems to be promising. On the one hand side own implementations are provided, on the other hand side the application can benefit from external codes which are directly coupled to the solver module.

However, the integration of external modules is restricted by purpose and does not embrace the full spectrum of available systems presented above. By now, only two external modules with specific properties have been made available. The choice of the external modules depends on the following considerations:

1. Performance should be the main priority for providing different solvers. As evaluated, both integrated external solver systems show a higher performance for specific kinds of simulations than the in-house solver systems. So the choice was motivated by performance advantages rather than offering alternative solvers to try getting non-converging simulations run. Whereas some solver systems are known for being advantageous in context of more ill-conditioned problems (see Section 5.5.1), a trial-and-error convergence strategy frequently hides problems in the underlying simulation setup. These short-comings can cause questionable overall simulation results although one linear solver is maybe able to deliver a result with the desired accuracy at each step of the *Newton* approximation.
2. Many users of the simulators regard the linear solver modules as black boxes which are supposed to deliver the “correct” solution vector. There are plenty of settings both for the assembly and solver module, but in most cases the default settings are used. Therefore, several solvers are integrated to automatic solver hierarchies (see Section 5.4), which lose efficiency with an increased number of solvers. Furthermore, to avoid contradictions with the first item regarding the convergence problem, appropriate feedback has to be given. It could be useful to obtain a questionable solution in combination with a respective feedback in order to specifically improve the simulation setup.
3. Typical simulators require a specific set of capabilities which should be available. Independently of the particular implementation and target architecture, the key demands on the solver module can be summarized as follows: solving of sparse real-valued non-symmetric and complex-valued non-hermitian linear equation systems, speed-up features like *Newton* adjustment and multiple right-hand-side vectors, shared library concept, as well as a control concept based on parameters. If complex-valued linear equation systems are not directly supported, the interface can transform them to double-sized real-valued ones (see Section 2.6.2). Since the linear equation systems are sparse, the solver systems designed for sparse matrices are preferred, because the memory consumption of compressed storage formats is far less than that of full matrices (see Section 4.6). However, due to linear memory addressing, algorithms working on full or band matrices can be far better optimized.
4. Since the simulators are provided for several platforms, for example Linux and AIX, portability of the libraries is a critical issue. Platform-dependent libraries normally utilize all capabilities of the respective hardware. However, if the same functionality and/or API used in the interface is not available or efficient on alternative platforms, the module should not be chosen.
5. As already stated in the introduction of this chapter, economic and legal considerations must not be neglected. First of all, the various license agreements have to be taken into account. Besides the technical evaluation, the decision whether to employ a module bound to commercial run-time or development licenses, must take the market, financial return, costs, and the economic situation of the vendor into account.

5.2 Overview of the Solver Module

As outlined in the introduction of this chapter, an in-house module is provided which consists of an extensible interface to three in-house solvers and to two external (third party) solver modules. In combination with the assembly module (see Chapter 4), this module is currently employed by MINIMOS-NT and FE-DOS. In Figure 5.1 an overview is given how these modules are integrated in the simulation flow of these simulators.

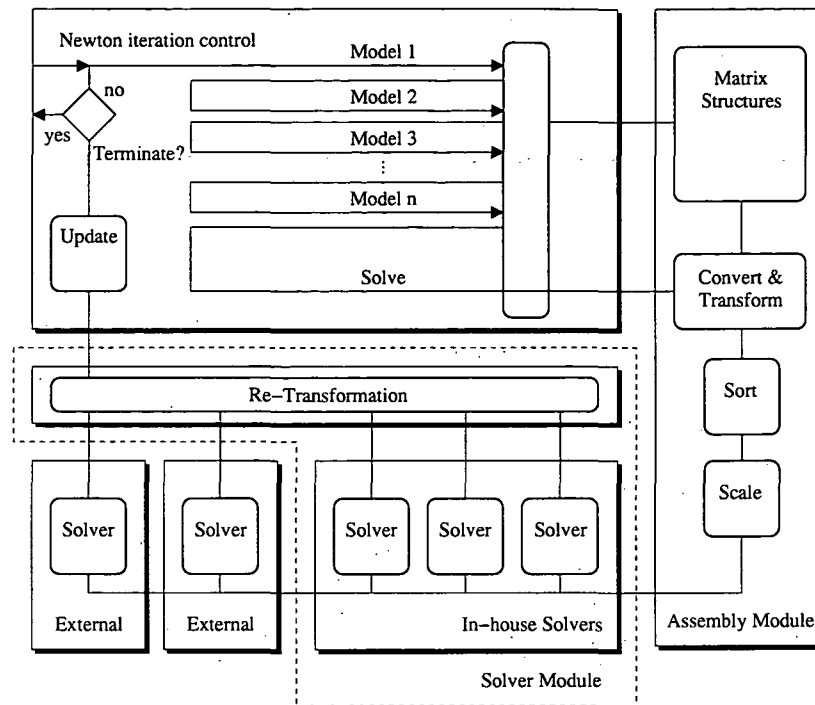


Figure 5.1: Schematic overview of the linear modules [230].

In the upper left corner the *Newton* iteration control of the simulator is shown. Depending on the kind of simulation, various model implementation functions are selected and subsequently called. They add their contributions to the matrix structures in the assembly module [229].

After assembling has been completed, the simulator requests the solution of the equation system by starting the solving process which is preceded by the compilation, pre-elimination, sorting, and scaling as discussed in Chapter 4. Eventually, a solver module is called which actually calculates the solution vector. After the chosen solver module has returned the solution vector, all transformations are reverted. MINIMOS-NT uses the solution to calculate the update for *Newton* approximation or terminates the iteration if a specific criterion is fulfilled.

There are different approaches available to calculate the solution vector. The in-house solver module provides one direct *Gaussian* solver as well as two iterative solvers, namely BICGSTAB [49] and GMRES(M) [179] in combination with an ILU-preconditioner. In addition, an interface is provided to employ recently developed modules, which provide highly-optimized mathematical code and allow a significant speed-up of the solving process.

5.2.1 Solver Selection

The choice of the solver basically depends on the following considerations:

- In contrast to direct methods, the convergence rate of iterative methods highly depends on spectral properties of the system matrix (see Section 5.2.5). Several measures are taken in order to improve the solvability. For example, iterative methods involve a second matrix, the so-called preconditioner, which transforms the system matrix into one with better properties (see Section 5.2.5). However, for some kinds of simulations still direct methods have performance and solving advantages (see Section 5.5.1).

- Although direct methods show solving advantages in general, their memory consumption is the main limiting factor. Since iterative methods require less computational resources, they are normally employed as standard solvers for well-conditioned linear equation systems.

As can be seen from the MINIMOS-NT examples given below, the iterative solvers have also performance advantages for problems with smaller dimensions. In the literature, often the opposite is found [182]. The reason for this fundamental difference are the various transformations done in the assembly module resulting in well-conditioned inner equation systems. Of course, the overall effort could be less if all transformations are skipped and a direct solver employed instead.

5.2.2 In-House Direct Solvers

Direct solution techniques refer to *Gaussian* elimination using a factorization of the system matrix A in a lower and upper triangular matrix as $A = LU$. The linear equation system $Ax = b$ can thus be written as $L(Ux) = b$. Therefore, the following three steps can be specified, that determine the *Gaussian* algorithm:

1. $A = LU$: *Gaussian* elimination (factorization: computing L and U)
2. $Ly = b$: forward-substitution (compute y)
3. $y = Ux$: back-substitution (compute x)

If the linear equation system has to be solved for multiple right-hand-side vectors b , only steps two and three have to be repeated. So the factorization has to be done only once. The effort of the factorization is $O(n^3)$ in the general case, resulting in a prohibitively high computational effort for problems with higher dimensions, for example three-dimensional device simulations. The time and memory consumption of the in-house LU factorization is indeed very high. The effort for determining space requirements is $O(n_{\text{non-zero}})$, that for determining time requirements $O(\text{space})$, and that for factorizing is $O(\text{time})$. For the sake of completeness it is to note that the *Cholesky* method [52] has not been implemented since the assembled matrices are normally not symmetric and positive definite.

5.2.3 In-House Iterative Solvers

Iterative methods refer to techniques which use successive approximations to obtain more accurate solutions at each step [17]. The iteration is stopped if the error is sufficiently reduced or a maximum number of iterations has been reached. There are two types of iterative methods:

- Stationary methods: they are older and simpler in both understanding and implementing, but generally not as effective. The four main stationary methods are the *Jacobi* method, the *Gauss-Seidel*, the Successive Over-Relaxation method (SOR), and the Symmetric Successive Over-Relaxation method (SSOR). Stationary methods can be generally expressed by $x^{(k)} = Bx^{(k-1)} + c$, where B and c are independent from the iteration counter k .
- Non-stationary methods: in contrast to stationary methods, they use information that changes at each iteration. Most of the methods are based on the idea of sequences of orthogonal vectors. The most prominent examples are the *Krylow* subspace methods GMRES(M), CG, CGS, BICG, and the BICGSTAB.

The *Biconjugate Gradient Stabilized* [49] (BICGSTAB) is applicable for non-symmetric matrices. This method avoids the irregular convergence patterns but shows almost the same convergence speed as the alternative Conjugate Gradient Squared method CGS [17]. The Conjugate Gradient method (CG) calculates

a sequence of conjugate vectors, which are the residuals of the iterates. The minimization of these residuals is equivalent to solving the linear equation system [17]. The BICG method calculates two sequences of vectors: one of the system matrix and one of the transposed system matrix. The CGS and BICGSTAB are variants of the BICG with modifications regarding the updating operations of the sequences. The BICGSTAB uses different updates for the sequence of the transposed matrix and therefore obtains smoother convergence than CGS.

5.2.4 The Generalized Minimal Residual Method

The *Generalized Minimal Residual* method (GMRES) is also applicable for non-symmetric matrices, leads to the smallest residual for a fixed number of iteration steps, although these steps require increasingly more computational resources. Thus, GMRES is actually not an iterative solver, since the exact solution of an equation system with rank n can be obtained in at most n steps (if an exact arithmetic is assumed). The solution procedure is based on orthogonal basis vectors, which are combined by a least-squares solve and update.

The dimension of the orthogonal vectors increases with each step. Since this increase in memory consumption is the drawback of this method, a restarted version of GMRES, normally referred by GMRES(M), can be used instead. The iteration will be terminated and the solution will be used as initial guess for the next iteration.

The choice of an optimum restart factor m is not trivial and “requires skill and experience” [17]. In [96], $m < 10$ is suggested for device simulation, but a significant higher value seems to be necessary for more ill-conditioned problems. In [183], m was set to 20 to avoid too high memory consumption. In view of the system memory of the average computer this parameter can be set to higher values at least in the area of classical device simulation. However, a default value for m had to be found and for that reason an empirical investigation was performed (see Section 5.5.3).

In order to provide an alternative solver system, a restarted version of the Generalized Minimal Residual method [179] was implemented for the internal solver module. This was done based on templates provided by the *Netlib repository* [17, 152]. During the implementation, it was absolutely crucial to retain the existing structure of the solver module and to apply all already implemented capabilities.

5.2.5 Preconditioner

The convergence rate of iterative methods depends on spectral properties of the system matrix A [17]. For that reason preconditioning is used to transform the linear equation system into a similar one, which has the same solution but better spectral properties. Thus, by using a preconditioner M the original system $Ax = b$ is transformed to

$$M^{-1}Ax = M^{-1}b, \quad (5.1)$$

where $M^{-1}A$ has better spectral properties than A .

There are many approaches to derive the preconditioner M . One class of them is the Incomplete-LU factorization (ILU), which approximates the matrix A . Basically, a factorization LU is incomplete if not all necessary fill elements are added to L or U . The respective preconditioner has the form

$$M = LU \approx A. \quad (5.2)$$

Adding a preconditioner to an iterative solver causes extra cost, so the resulting trade-off between construction/application and gain in convergence speed must be considered. As outlined in [65], a hierarchical concept is used to minimize the necessary computational time of this system. This time is mainly

influenced by the parameters *fill-in* and *threshold (tolerance)* of the Incomplete-LU factorization process. These parameters are stored by the parameter administration and automatically adapted after each solver call. See Figure 5.2 for an illustration of the hierarchical concept. This form of the Incomplete-LU factorization is sometimes referred as $ILUT(\epsilon, q)$ [183] with ϵ standing for the threshold and q standing for the fill-in parameter specifying the q largest elements which are kept per row.

Alternatives are the $ILU(0)$ and $ILU(p)$ methods. The former has less complexity and simply keeps the elements whose corresponding values in the system matrix are non-zero, that is $\epsilon = 0$. The method called $ILU(p)$ drops elements but takes only the structure of the linear equation system into account. However, the Incomplete-LU factorization faces the same problems as the *Gaussian* elimination. For example, due to zero or negative pivots, incomplete factorizations may break down or result in indefinite matrices, respectively, even if the full factorization exists and yields a positive definite matrix [17].

An alternative to these methods are those which approximate the inverse of A , the so-called *approximate inverses* methods. The major advantage of these approaches is that the application of the resulting preconditioner requires matrix-vector products only and not a solving step [183]. Examples are the SPAI [92] or the AINV methods [19].

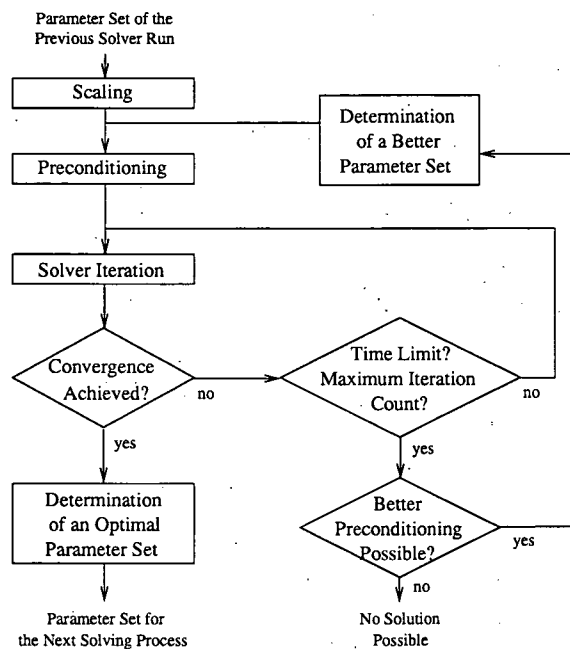


Figure 5.2: The hierarchical concept [65, 228].

For the synchronization between preconditioner and solver the concept of counting mathematical operations is used. In contrast to the system time which was used in the former version this count remains constant for the same simulations, and thus subsequent passes of the optimization loop are deterministic even in a multitasking environment.

The dual drop-off strategy in the incomplete factorization strategy employed in the internal solver module works as follows: Elements in L and U whose size is less than the threshold (relative to the average element size of the current row in U) are dropped. By setting a zero threshold, all elements will be kept. Furthermore, only a specific number of the remaining elements are kept. This number is determined by the fill-in parameter and the selection is done by size. Thus, the largest elements remain in the matrix. One can use a zero fill-in parameter to obtain a strategy based on keeping the largest elements in each row of L and U . Or, by choosing an appropriate threshold but setting the fill-in parameter to n , the usual threshold strategy will be applied, but the number of fill-ins is then unpredictable.

5.3 External Solvers

Since the remarkably increased performance of today's average computers inspires to even more costly simulations (for example optimizations and mixed-mode device/circuit simulations), a further speed-up of the simulators is highly appreciated. Thus, the development of highly-optimized mathematical code must not be neglected. With regard to such new developments, an interface is provided to employ external solver modules.

At the moment, two external modules can be employed: First, the *Parallel Sparse Direct Linear Solver* PARDISO [117, 181, 182], which provides a multi-threaded direct solver as well as an LU-CGS iterative solver implementation. Second, the *Algebraic Multigrid Methods for Systems* SAMG [36, 37], which not only provides multi-level algorithms, but almost the same iterative solvers as those of the in-house module. Both external packages are written in FORTRAN. The only overhead of the interface to these modules is the conversion of matrix storage formats (see Section E), which is regarded as negligible due to its linear implementation effort (see Section 5.5.2 for quantitative information).

Since PARDISO provides a multi-threaded solver, a discussion of several parallelization issues is given in the next section.

5.3.1 Parallelization Issues

Parallelization techniques become important in case a simulation is regarded to take a "long" time. Thus, the main objective is the reduction of the simulation time, especially if time-to-market figures are considered also in the TCAD context. An additional utility of parallelization can be seen in the efficient employment of all available computational resources.

Whereas during one kind of long simulations much time is spent for particular tasks such as solving large linear equation systems, the other kind consists of a large number of simulation runs with similar setups. In the latter case one simulator run might be short, but thousands of them take sometimes a prohibitively long time. Obviously, a combination of both makes parallelization even more necessary. Typical examples for these kinds of long simulations are three-dimensional device, process, and mixed-mode device/circuit simulations or expensive optimizations and multi-parameter steppings, respectively.

Basically, the main objective of parallelization is the distribution of the simulation load over several CPUs and/or workstations in order to reduce the execution time. In the case of the first kind of simulations, the program execution itself is parallelized. Libraries, such as OPENMP [145] or MPI [159], can be used to split the simulator into several threads, as far as the code is suited for parallelization. Due to the parallel execution, the total computational effort increases, but the real simulation time can be reduced.

For the second kind of simulations, another approach can be identified: The repeated execution of the simulator with similar setups can be distributed over several CPUs and/or a network. Since the single runs are independent from each other, the simulation time can be significantly reduced. Still, the computational effort is increased if simulator-internal speed-ups or synergies are not fully used. From the engineer's perspective a convenient administration of the various simulation setups and results must be provided, otherwise the overall results are likely inconsistent and thus as contra-productive as useless. Normally, the input/output system causes a prominent restriction for this kind of load distribution.

Distribution of Repeated Executions

The load distribution of repeated executions is extremely important during optimizations. For example, the calculation of gradients offers the possibility to simultaneously start independent runs of the same simulator with slightly different setups. Another example in this field are genetic optimizers [234].

The optimization framework SIESTA [232] consists of an internal host management in order to employ various hosts for such calculations. Due to the distribution, SIESTA is able to significantly reduce the optimization time.

Besides of optimizations, another field of application are fixed sets of similar simulations yielding combined results. A simple example is the calculation of an IV curve. A single-threaded simulator such as MINIMOS-NT calculates each operating point subsequently, using simulator-internal speed-ups such as good initial guesses for the next step. In addition, the pre- and post-processing has to be done only once. However, if several CPUs are available, each branch of the IV curve can be calculated independently, multiplying the pre-processing effort, but reducing the overall real simulation time. See Figure 5.3 for a comparison of the two approaches.

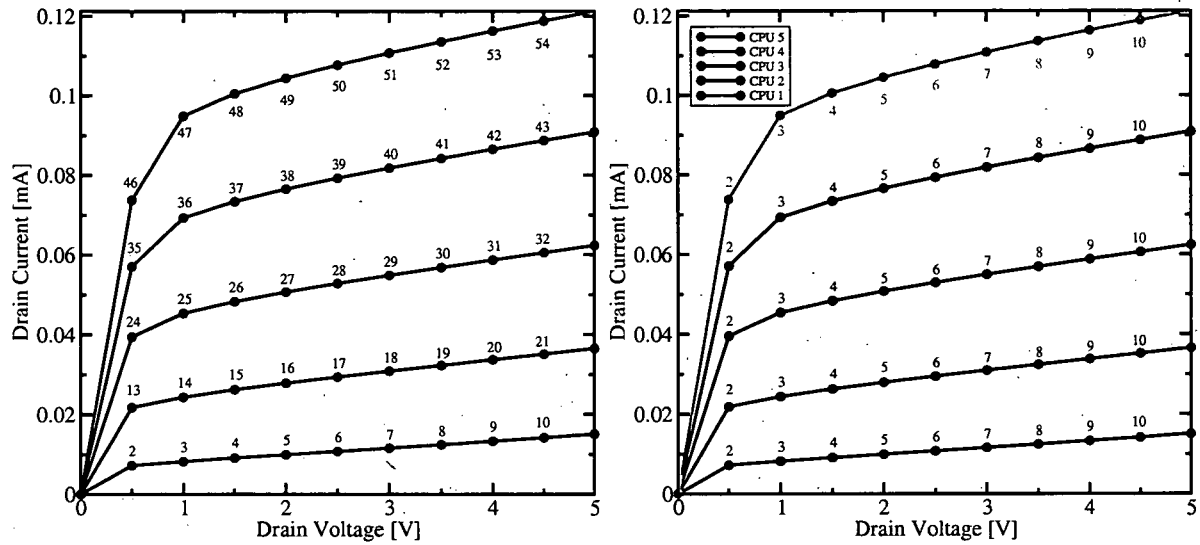


Figure 5.3: Concept of single- (left) and multi-threaded (right) stepping algorithm. Whereas the former calculates all 55 steps subsequently, the latter uses five processes to calculate eleven steps each.

As stated above, the administration of such a parallelization approach might be difficult, cumbersome, and error-prone. Since the simulator is not parallelized itself, additional functionality is required in a higher level of application. As already used for optimizations, so-called meta- or parameterized input-decks are systematically used to create actual simulator setups.

Here, a counter-part to the SIESTA system is required to exempt the user from the administration of these setups and their results as well as from a fault-tolerant host management, simulator calling, and consistency checking. All of these tasks are performed by the new library SEILIB, which is discussed in Appendix C.3. As an application example, the test of the MINIMOS-NT simulator is presented in Chapter C.4. This test takes more than one day on a single-CPU Linux machine, but can be run in approximately one hour on an IBM cluster with 32 CPUs. As additional benefits, this library can also be used to combine several programs to complete simulation flows and to perform optimizations.

Parallelization of the Program Execution

Whereas load distribution of repeated executions does not require modifications of the simulators, this section gives a short introduction in parallelization of the codes itself. This task frequently raises severe design and implementational problems, which might be hard to resolve for already existing single-threaded codes. However, particular tasks which are sufficiently modularized can be targets for simulator-internal parallelizations.

In context of semiconductor device and process simulations, a significant part of the computation time is spent on solving linear equation systems. As can be seen in Table 5.2, the range starts at about 40% for small two-dimensional device simulations up to 90% for large mixed-mode simulations. However, well-known bottle-necks prevent a significant performance impact. Direct algorithms are a promising parallelization target, as shown in several publications, for example about the OPENMP-based PARDISO [181] or MPI-based HSL codes [56].

The implementation of PARDISO is based on OPENMP [145], which has become a standard for parallelization applications. In comparison to single-threaded program developments, additional design and programming complexity have to be taken into account. Algorithms are not only defined in terms what work they are supposed to perform, but also how this work can be distributed to more than one processor. Therefore, OPENMP is an implementation model to support the implementation of parallel algorithms. Based on the standard programming languages FORTRAN and C++, OPENMP provides a set of compiler directives and a small function library. In summary, the main objective of OPENMP is to provide a standard and portable Application Programming Interface (API) for implementing shared memory parallel programs [34].

For that reason, OPENMP is primarily designed for shared memory multiprocessors, where all available processors can directly access the complete memory of the machine. The alternative is distributed memory, where memory is physically associated with each processor. Since each processor can access only the local memory, the programmer is responsible for mapping the program data accordingly. Messages have to be passed to access memory associated with other processors. For that reason, so-called *Message Passing Interfaces* (MPI [159]) and *Parallel Virtual Machines* (PVM) are employed. There are also high-level language approaches for automatic generation of low-level messages, for example *High Performance Fortran* (HPF [119]). Distributed memory systems are particularly employed if the number of processors exceeds the limit for shared memory systems. A schematic comparison of shared and distributed memory architecture is depicted in Figure 5.4 [34].

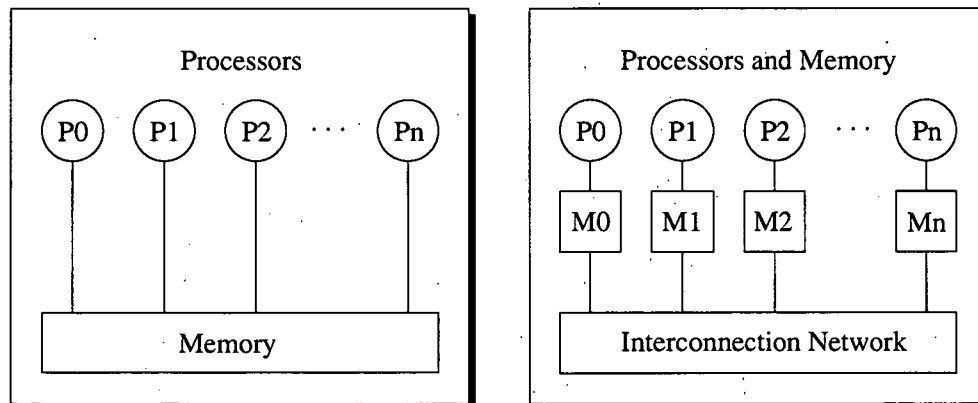


Figure 5.4: Comparison of a canonical shared memory (left) and message passing non-shared memory (right) architectures [34].

5.3.2 Pardiso

PARDISO is a high-performance and memory efficient package for solving large symmetric and unsymmetric linear equation systems on shared memory multi-processors [39]. The implementation is based on OPENMP [145] and a combination of left- and right-looking Level-3 BLAS super-node techniques are used [182]. To improve the performance of sequential and parallel factorizations, a Level-3 BLAS update is used.

In terms of the provided features, PARDISO can be perfectly used as an alternative for the internal solver modules. Basically, it solves real- and complex-valued linear equation systems $\mathbf{Ax} = \mathbf{b}$, where multiple right-hand-side vectors \mathbf{b} may be specified. Actually, PARDISO supports even a wider range of sparse matrix types, which are not all required by MINIMOS-NT: real- and complex-valued, symmetric, structurally symmetric and unsymmetric, positive definite, indefinite, and hermitian sparse linear equation systems.

PARDISO can also reuse already existing structures in the memory (cf. *Newton* Adjustment). The required phases among the four existing phases can be specified [39]:

1. Phase 1: Fill-reduction analysis and symbolic factorization.
2. Phase 2: Numerical factorization.
3. Phase 3: Forward and backward solve including iterative refinements.
4. Phase < 0: Termination and deallocation phase.

For the first call, phases one to three are performed, whereas for subsequent steps only phase two and three have to be repeated. On exit, the destructing phase is invoked. Note that due to the already established *Newton* adjustment levels (see Section 4.12), the solver interface has always the complete information whether to repeat the symbolic phase or not.

The user of the module has the choice between two different solvers: either the direct LU-factorization or a combination of the direct and iterative method LU-CGS. See Appendix A.12 on how to select PARDISO in the input-deck.

5.3.3 Algebraic Multigrid Methods for Systems

The efficiency of solving large linear equation systems arising from discretized partial differential equations can be increased by applying hierarchical algorithms [213]. Over the last decades, the multigrid principle has turned out to be a promising approach and therefore the interest in algebraic methods has been increasing. Such methods automatically derive the hierarchy based on algebraic information which is contained in the discretization matrix. Geometric multigrid methods on the other side use a hierarchy of grids which are defined by coarsening based on grid information.

Two advantages of the algebraic multigrid methods can be given:

1. Increased geometrical complexity which restricts the use of geometric multigrid approaches.
2. Demand for solver modules which are plugged into an existing environment without modifying the already existing interface. Thus, algebraic multigrid approaches can be easily used as an alternative for one-level solver modules without modifying the complete code.

Algebraic multigrid or AMG was the first hierarchical, matrix-based multigrid method. The ideas of geometric multigrid, that is the coarsening and smoothing of the grids, have been extended to specific classes of linear equation systems. The automatically derived hierarchy consists of linear equation systems whose dimension is repeatedly decreasing. Matrix entries are used to derive the operators to transfer information between two levels. Due to this automatic procedure, the robustness, adaptability, and flexibility is remarkably increased.

The SAMG package (*Algebraic Multigrid Methods for Systems*) [37] has been fully coupled to the internal solver module. Like PARDISO, it can be used as an alternative for the in-house solver module. See Appendix A.12 how to select these solvers.

As SAMG is a package of several solver systems, the user can choose between four configurations of preconditioners and solvers (accelerators): AMG-BICGSTAB, AMG-GMRES(M), ILU-BICGSTAB, and ILU-GMRES(M). The SAMG interface is implemented in a separate function. Since SAMG can handle real-valued equation systems only, the approach of splitting a complex-valued one into a real-valued system with double dimension as discussed in Section 2.6.2 is used. As seen for the internal and PARDISO solvers, also SAMG provides specific features for solving multiple right-hand-side vectors and to reuse already allocated structures during repeated calls of SAMG.

5.4 Solver Hierarchy

Since several linear solvers are available, a choice has to be made regarding whether to use one of the two in-house iterative solvers, the in-house direct solver or an external module. This choice can be based on a-priori available information. For example the performance evaluation presented in Section 5.5 resulted in some conclusions about which solvers are best-suited for different kinds of simulations.

However, although a solver is selected due to general a-priori considerations regarding the kind and dimension of the simulation, the solver still might fail whereas an alternative type may succeed in solving the equation system. Although care has to be taken while evaluating the results of such problematic simulations, a successful result may trigger particular improvements in the complete simulation setup. Keeping this goal carefully in mind, the user will normally select an alternative solver if one fails.

The manual selection of alternative solvers has one major drawback: if the convergence problem occurs after a long simulation time, the complete simulation has to be rerun with the alternative solver although this would not be necessary. An automatic solver hierarchy as implemented in the iteration control module of MINIMOS-NT resolves this problem. The simulation is started according to the user settings in the input-deck. Note that the user setting can also enable an automatic solver selection which is based on conclusions drawn from a performance evaluation.

If a problem occurs, the solver parameters are reset in order to adjust them for the new situation. If this does not help, the alternative iterative in-house solver is selected. This means that GMRES(M) replaces BICGSTAB and vice versa. If the problem persists, the direct in-house solver is selected instead of the iterative one. If the direct solver is not able to solve the system, the simulation will be terminated. Otherwise, the user settings are reestablished in order to use the original solver for the next step. In this hierarchy, the external modules can be integrated. Since the in-house direct solver will quickly exceed its limitations for large mixed-mode and three-dimensional simulations, the PARDISO direct solver would be a very good alternative.

An alternative possibility for selecting another solver is to use an interactive mode of the simulator. As shown for MINIMOS-NT (see Appendix C.2), the user can interrupt the simulation process and directly modify the settings in the input-deck database. These modifications include the selection of solver modules and systems.

Note that there exists a second solver hierarchy directly in the solver module. If no selection of a particular solver type is made (default), first the iterative solver is tried, followed by the direct one in case problems occur.

5.5 Practical Evaluation of the Solvers

In the course of this work the performance of these solvers has been evaluated. Rather than using a set of single matrices, the approach is based on complete simulations with consistent settings, as typically encountered during daily work.

In mathematical publications, different solver systems are normally compared by applying them for single linear equation systems. For example, the *matrix market* [25, 139] provides such a data repository for use in comparative studies of linear algebra algorithms. The repository does not only provide matrices, but also matrix generation software in a wide variety of scientific and engineering disciplines. For each matrix and matrix set details regarding the matrix properties and a visualization of the matrix structure is provided. The matrices can be downloaded in several different text file formats. A matrix generator is either a static software for download, a Java applet, or a form-based web service to process requests for generating matrices. In 2000, 482 individual matrices and 25 matrix generators were available. The database includes the *Harwell-Boeing Sparse Matrix Collection* (Release I) [55], the SPARSKIT collection [178], and the *Non-Hermitian Eigenvalue Problem* collection [14].

Such an approach is particularly useful for the evaluation of new mathematical algorithms. The conclusions drawn from the results are used to improve the respective implementations. However, from the perspective of the implementation of a general-purpose simulator, the objective is quite different. Here, a set of solvers are available and the one which fits best for the respective problem shall be chosen. In contrast to the solver development, the research on this topic is restricted to practical evaluations regarding the respective simulators.

5.5.1 Test Examples and Processing

The 16 examples summarized in Table 5.1 were taken from current scientific projects at the institute. They consist of field effect, bipolar, and silicon-on-insulator transistors. Structures such as FINFETs are analyzed by means of three-dimensional simulations.

All examples were simulated on:

- An IBM cluster with four nodes with eight CPUs each based on the Power4+ architecture. Two nodes have 64 GB memory each and the other two nodes have 32 GB each. The operating system is AIX 5.2. For compiling and linking the 32-bit xlc/xlC/xlf compilers with optimization level O5 were used. For PARDISO the solver module was linked against the Engineering and Scientific Subroutine Library ESSL (see Section 5.1.2).
- A 2.4 GHz single *Intel* Pentium IV computer with 1 GB memory running under *Suse* Linux 8.2. For compiling and linking the *Intel* 7.1 compilers for the IA32 architecture with optimization level O3 were used. For PARDISO the solver module was linked against the LAPACK and BLAS libraries (see Section 5.1).

The extensible benchmark is processed by a single program, which has been implemented as a SEILIB application (see Section C.3). The real (wall-clock) and user time is measured by the GNU *time* command, and the fastest of three consecutive runs is taken.

5.5.2 General Quantitative Comparisons

Table 5.2 contains information on how the simulation time is actually spent. The simulator was started on the IBM cluster and the in-house ILU-BICGSTAB was selected. The first columns show the number of the example, the dimensions of the inner and complete system, and the ratio $dim_{inner}/dim_{complete}$. A lower ratio indicates that more equations are pre-eliminated and thus more time is spent for pre-elimination. The CPU column shows the absolute user time required for the respective example. The remaining columns show the shares of the simulation time spent in selected modules or for selected tasks: for initialization, pre-processing, assembling the linear equation systems (including the calculation of the model contribu-

tions), the share of the linear modules including all steps shown in Table 5.3, the update of the quantities, and eventually the postprocessing.

The most important data in the context of that chapter is how much time is spent for the linear system after the assembly has been completed (the *Lin* column). For the smaller two-dimensional examples that share is below 50%, that means more time is spent for calculating and assembling the model contributions as well as for pre- and post-processing. However, for all other simulations including the larger two-dimensional ones, that share rises significantly even up to more than 90%. Thus, for increasing the efficiency of advanced device simulations and optimization, it is essential to reduce the relative effort spent on preparing and solving the linear equation systems.

In addition, the effort of the conversion to alternative sparse matrix formats was evaluated. Since the time required for the conversion is too short, no significant information can be given by comparison of complete run times. However, in order to give quantitative information on the negligibility of the matrix conversions, the input and output system of the assembly module was used.

The assembly module has been equipped with a comprehensive input and output system which enables the user to have the assembled complete and inner linear equation systems read from or written to files. These features can be used for debugging and quality assessment purposes. Furthermore, different sets of parameters or alternative solvers can be efficiently tested. Whereas the simulator is normally used to calculate the entries of the linear equation system, a test program is provided which reads the files and starts the solver.

In order to quantify the effort of the matrix conversions, the first linear equation system assembled for the three-input nand gate (dimension 219,920, 1,925,553 non-zeros) was written to a file. Afterwards, it was read by a modified version of this test program. The modification regards only the activation of one type of matrix conversion. The conversion time is measured by the *gettimeofday* function, the minimum time of three consecutive runs is taken. One conversion to CSR variant 1 takes 38.212 ms, 266 conversions take 9,184.06 ms. For the CSR variant 2, the times are 38.955 ms and 8,798.3 ms, respectively. Since the complete simulation takes 8,468.19 s, the share of all conversions is 0.108% or 0.104%, respectively. These very small shares allow to conclude that the effort of the conversion can be neglected.

The complex-valued variant of that test program was also modified to evaluate another interesting issue. As outlined in Section 2.6.2, one can split a complex-valued equation system into a double-sized real-valued one in order to employ a real-valued assembly and solver system also for complex-valued linear equation systems. Regarding the memory consumption, this approach has disadvantages, which shall be quantified in the following discussion. Three examples were taken and the small-signal systems for 1 GHz were written to files. These files were read by the test program – once for the original version employing the complex-valued solver system and once for the modified version. Thus, this evaluation approach emulates also the different assembling process of the respective matrices. The results can be summarized as follows:

- The small-signal simulation of a two-dimensional MOS transistor structure (example 1) results in an inner system matrix of dimension 2,704 with 25,432 non-zeros, which require 406,912 bytes. If the equation system is split, the number of nonzeros and thus the memory consumption increases to 101,728 and 813,824 bytes, respectively. However, the solving time is reduced from 1.43 to 0.69 s.
- For the three-dimensional simulation of a FINFET structure (example 13 with a dimension of 81,037), the following results can be given: the number of non-zeros and the memory consumption is increased from 841,678 to 3,366,712 and from 13,466,848 to 26,933,696 bytes, respectively. In accordance with the two-dimensional results, the simulation time is decreased by roughly 59 % from 152,66 s to 87,70 s.

- The mixed-mode simulation of the three-input nand gate (example 11) shows similar figures regarding the memory consumption, which rises from 33,587,712 to 67,175,424 bytes (number of nonzeros: 2,099,232 and the four-fold value of 8,396,928). However, the simulation time is increased from 437.96 s to 1,117.88 s.

These results motivates the conclusion that with the current implementation it is advantageous to solve the complex-valued equation systems with the complex-valued solver implementations. The main reason for that is the memory consumption, which is always doubled, whereas different results for the simulation time are obtained.

5.5.3 Simulation Results of the GMRES(m) Evaluation

Several test runs of the in-house GMRES(M) solver were performed and the cumulated number of mathematical operations as a function of m were recorded.

Since values of m smaller than 16 cause errors for some three-dimensional and mixed-mode simulations, the commonly used m has to be set to a value higher than 16. In Figure 5.5, all values are scaled to those of $m = 16$. The following ratios are shown: minimum, maximum, and average of the number of mathematical operations, the user time, and the memory consumption. By analyzing the solid black line with symbols for the average user time, a default value shall be selected which can be commonly set for all kinds of simulations. The first part of that curve is decreasing between 16 and 70 down to a level of approximate 55%. Afterwards, it slightly increases again. One reason for this might be the higher memory consumption. The curve for the mixed-mode device/circuit simulations has a similar shape, however the increasing part starts a little bit earlier. Finally, the curve for the two-dimensional simulations shows a contraproductive effect of an increased m between 70 and 75.

So one can conclude that a value between 50 and 70 seems to be advantageous for all kinds of simulations if enough memory is available. For that reason, the default value was set to $m = 65$. As this value can be easily controlled by the user, different requirements depending on the simulation and the memory resources can be met. In addition it is to note that the simulator itself can automatically adjust this value if the respective information is available.

#	Simulation	Type	Dimension	Entries ([%])	DC it	Remark
1	MOSFET	2D device	2,704	23,662 (3.24)	17	test structure with $L_G = 1 \mu\text{m}$
2	Flash cell	2D device	5,967	47,956 (1.35)	24	tunneling effects
3	Pin-diode	2D device	6,335	56,127 (1.40)	13	optical generation model
4	Bjt transistor	2D device	6,389	33,869 (0.83)	23	ET transport model
5	SA-LIGBT	2D device	16,774	158,223 (0.56)	27	two V_D steps: 0 V, 5 V
6	SiGe HBT	2D device	19,313	210,915 (0.57)	32	self-heating
7	Colpitts oscillator	circuit (1)	3,928	35,002 (2.27)	41	transient (400 steps)
8	Amplifier	circuit (1)	6,391	35,291 (0.86)	30	ET transport model
9	Ring oscillator	circuit (10)	25,246	226,931 (0.36)	29	transient (100 steps)
10	2-input nand gates	circuit (8)	146,614	1,347,138 (0.06)	23	transient (50 steps)
11	3-input nand gates	circuit (12)	219,920	2,020,706 (0.04)	23	transient (50 steps)
12	MagFET	3D device	85,308	921,860 (0.13)	36	magnetic field
13	FinFET	3D device	81,037	807,150 (0.12)	13	thin SOI finger
14	SOI	3D device	87,777	997,296 (0.13)	23	two V_D steps: 0 V, 0.1 V
15	HBT	3D device	175,983	1,833,138 (0.06)	35	self-heating
16	LD-MOSFET	3D device	167,197	1,925,553 (0.07)	25	power device; two V_D steps: 0 V, 1.0 V

Table 5.1: Six two-dimensional, five mixed-mode device/circuit (the number of devices is given in parenthesis), and five three-dimensional simulations were used for evaluating the solver performance. The dimension of the linear equation system, the number of non-zero entries, and the typical number of DC *Newton* iterations are given.

#	Inner	Complete	Ratio [%]	CPU [s]	Init [%]	Pre [%]	Asm [%]	Lin [%]	Upd [%]	Post [%]
1	2,704	2,920	92.60	1.34	14.93	6.72	31.34	34.33	3.73	2.99
2	5,967	13,915	42.88	7.02	3.56	5.98	46.15	40.17	1.71	1.14
3	6,335	6,451	98.20	5.06	16.01	7.51	17.19	55.53	0.59	2.17
4	6,389	6,754	94.60	4.84	9.09	5.99	28.10	51.24	1.45	1.65
5	16,774	17,705	94.74	39.64	12.66	4.69	37.24	42.73	0.33	1.54
6	19,313	19,836	97.36	28.29	39.46	3.45	10.35	44.46	0.24	1.24
7	3,928	5,475	71.74	32.60	1.53	6.20	17.61	64.39	1.20	7.15
8	6,391	8,056	79.33	63.06	0.78	4.60	30.13	59.94	0.76	3.00
9	25,246	37,007	68.22	226.97	0.81	4.88	13.69	76.16	2.07	1.74
10	146,614	202,168	72.52	3,641.37	0.06	2.88	8.34	87.13	0.33	1.10
11	219,920	303,250	72.52	7,671.19	0.04	2.40	6.78	89.60	0.26	0.80
12	85,308	90,203	94.57	593.55	0.39	2.52	7.24	89.03	0.10	0.48
13	81,037	98,870	81.96	99.13	4.68	4.07	9.69	79.72	0.22	1.12
14	87,777	93,572	93.81	190.19	1.45	5.41	23.19	67.32	0.19	1.90
15	175,983	226,687	77.63	988.29	0.65	2.54	6.97	88.42	0.14	1.04
16	167,197	179,111	93.35	495.99	0.74	3.61	9.98	83.93	0.15	1.20

Table 5.2: This table provides general quantitative information about the simulations using the in-house ILU-BICGSTAB on the IBM cluster. After the index of the simulation, the dimensions of the inner and complete equation system is given. The next column shows the share of the inner in respect to the complete system. In order to analyze how the simulation time is spent, the remaining columns contain the shares of the initialization, pre-processing, assembling, solving (includes all steps shown in Table 5.3), quantity update, and post-processing, respectively.

#	Time [s]	MCSR [%]	Compile [%]	Pre [%]	Sort [%]	Scale [%]	ILU [%]	BICGSTAB [%]	Back [%]
1	0.49	0.65	10.93	6.11	2.25	17.02	32.93	29.52	0.38
2	2.74	0.67	6.73	5.07	1.20	8.98	35.63	41.16	0.51
3	2.69	0.27	4.20	2.26	1.05	8.82	42.71	40.54	0.12
4	2.40	0.45	6.69	4.21	1.64	10.70	37.67	38.38	0.19
5	27.67	0.16	4.58	3.08	0.99	8.53	34.37	48.16	0.12
6	13.11	1.25	5.01	3.02	1.10	6.93	36.06	46.51	0.11
7	35.17	0.26	4.75	3.87	0.73	5.97	42.32	41.77	0.31
8	45.53	0.19	5.65	4.31	1.10	8.04	40.00	40.48	0.20
9	224.52	0.26	3.28	2.66	0.66	4.04	30.85	58.04	0.19
10	3139.28	0.13	1.83	1.79	0.40	2.51	29.26	63.93	0.14
11	6855.87	0.12	1.38	1.48	0.31	1.97	27.53	67.11	0.11
12	504.36	3.99	2.32	1.14	0.50	2.39	16.67	72.95	0.04
13	94.43	9.82	4.95	2.75	0.88	3.84	23.16	54.47	0.12
14	146.11	1.08	3.48	2.48	1.01	5.17	26.68	59.97	0.13
15	921.04	37.04	3.51	1.69	0.58	2.71	11.33	43.08	0.07
16	447.12	1.47	2.45	1.82	0.68	3.12	28.21	62.16	0.09

Table 5.3: This table provides general quantitative information about the solving process which includes the MCSR conversion, compiling of the complete linear system, pre-elimination, sorting, scaling, preconditioning (ILU), and solving with the BICGSTAB. The simulations were started on the IBM cluster.

#	Time [s]	MCSR [%]	Compile [%]	Pre [%]	Sort [%]	Scale [%]	ILU [%]	BICGSTAB [%]	Back [%]
1	4.35	0.77	1.88	2.07	0.50	13.25	50.99	30.06	0.15
2	2.86	3.43	5.63	5.36	1.19	8.76	34.20	40.87	0.52
3	2.77	1.98	3.76	2.69	1.02	8.62	41.52	40.27	0.11
4	2.51	2.37	5.67	4.65	1.69	10.60	36.52	38.27	0.18
5	16.22	2.61	3.98	3.41	1.10	8.31	34.40	46.04	0.14
6	17.00	1.32	2.41	1.82	0.66	4.64	45.33	43.74	0.07
7	21.44	1.74	4.17	4.95	0.97	5.92	40.25	41.70	0.24
8	38.72	2.51	5.35	5.70	1.42	8.67	39.08	37.01	0.22
9	180.63	2.19	2.97	3.50	0.67	3.99	29.93	56.55	0.20
10	3388.38	1.45	1.64	2.23	0.41	2.41	27.52	64.22	0.14
11	7618.11	1.42	1.27	1.86	0.34	1.99	25.31	67.71	0.11
12	634.29	21.19	1.21	1.11	0.44	1.96	13.37	60.68	0.04
13	92.97	33.06	2.03	2.05	0.55	2.79	17.92	41.49	0.11
14	129.94	7.13	2.97	2.86	0.94	4.46	25.17	56.32	0.13
15	2499.04	79.80	0.58	0.70	0.18	0.89	3.83	14.00	0.02
16	462.25	9.60	1.79	1.64	0.62	2.83	24.81	58.62	0.08

Table 5.4: This table shows the same information as Table 5.3, but without the *Newton* adjustment.

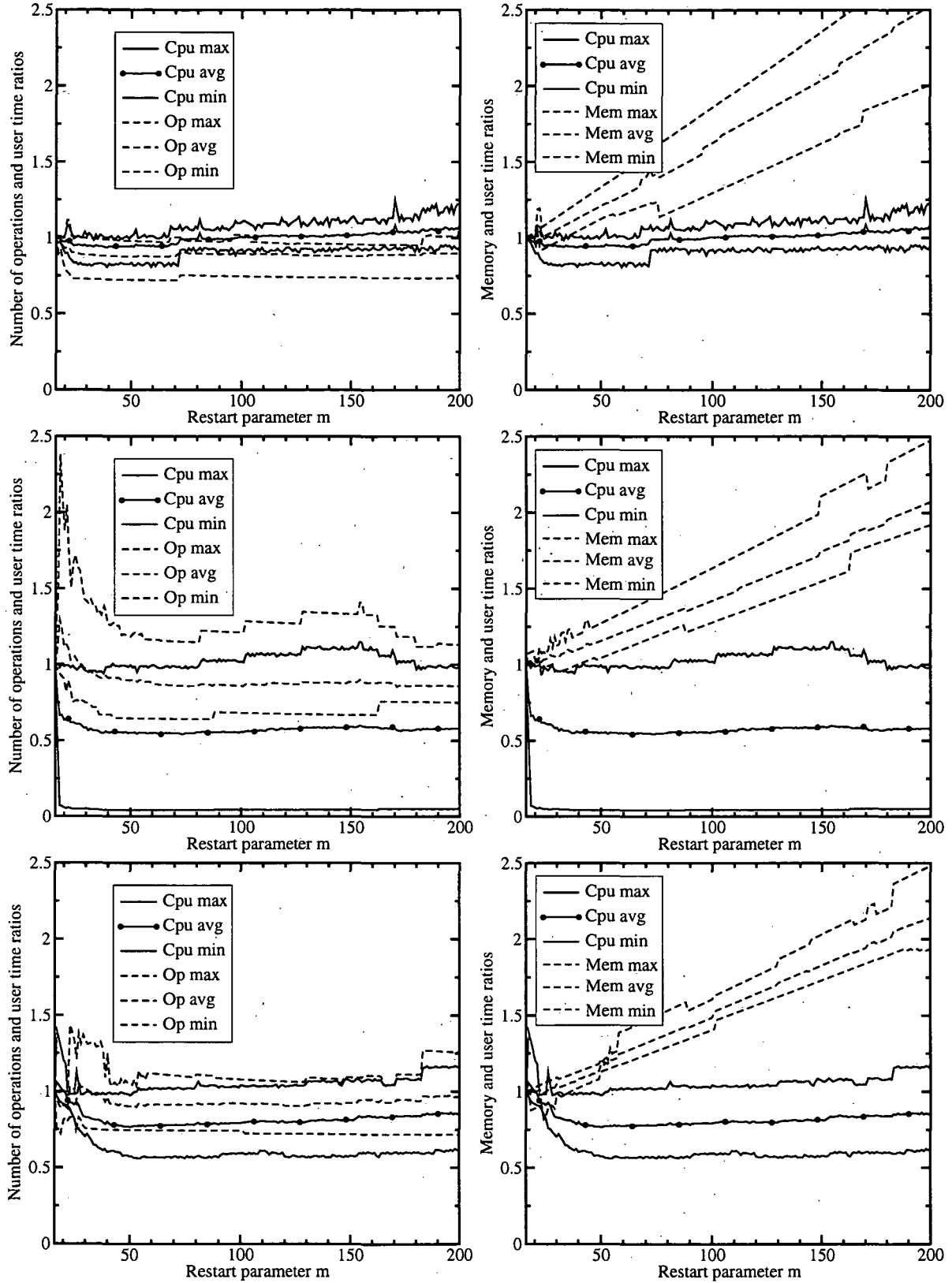


Figure 5.5: User time, number of operations (left), and memory consumption ratios (right) depending on the GM-RES(M) restart factor m for the two-dimensional (above), three-dimensional (center), and mixed-mode simulations (below).

5.5.4 Simulation Results of the Performance Evaluation

The objective of the performance evaluation is to compare the simulation times required by MINIMOS-NT depending on the solver systems selected. Note that the solver hierarchy (see Section 5.4) was deactivated. All three in-house solvers as well as the two PARDISO and four SAMG solvers are evaluated, although not all examples could be solved by all solver systems.

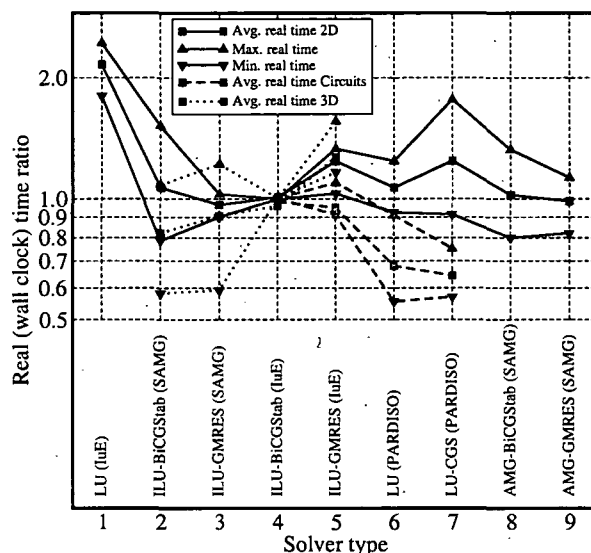


Figure 5.6: Solving times (average, minimum, maximum) on the *Intel* computer. All times are scaled to the in-house ILU-BICGSTAB in the center.

In Figure 5.6 a comparison of different solvers on the *Intel* computer is given. Due to the large simulation time differences, all times are scaled to the in-house ILU-BICGSTAB in the center of the graph. Interesting results are the superiority of the advanced implementations of LU factorization and iterative solvers for circuits and three-dimensional devices, respectively. The in-house GMRES(M) solver has advantages for circuits also, whereas the direct solver on the left hand side can in fact only be used for quality assessment of two-dimensional simulations. Whereas the iterative SAMG solvers show single significant performance advantages up to 35%, the multi-level algorithms require still some effort to be generally applicable.

To show the relative impact of multi-threading, the PARDISO-LU solving ratios (referring to the single-threaded version) against the number of processors/threads are shown in Figure 5.7. For the three-dimensional examples, also the PARDISO-LU-CGS ratios are given. The real (wall clock) time required for solving the example, the cumulated user (CPU) times are shown in Figure 5.7, which increase due to the parallelization overhead. Whereas for two-dimensional device and circuit simulations too many processors can be even contra-productive, the marginal additional utility for three-dimensional simulations is drastically diminishing. Thus, for the average simulation four processors should be sufficient. Especially under scarce conditions, for example during optimizations, assigning two tasks per node of eight processors appears to minimize the real time effort.

The iterative methods still show a significant performance advantage over the direct solvers. However, the 1983 quotation “In 3D sparse direct solution methods can safely be labeled a disaster” [16] describes the experiences (in regard to both time and memory consumption) with the in-house LU solver, but does not embrace the recent developments. Especially for mixed-mode device/circuit simulations the advanced direct methods show a significant performance advantage, even up to the highest dimensions.

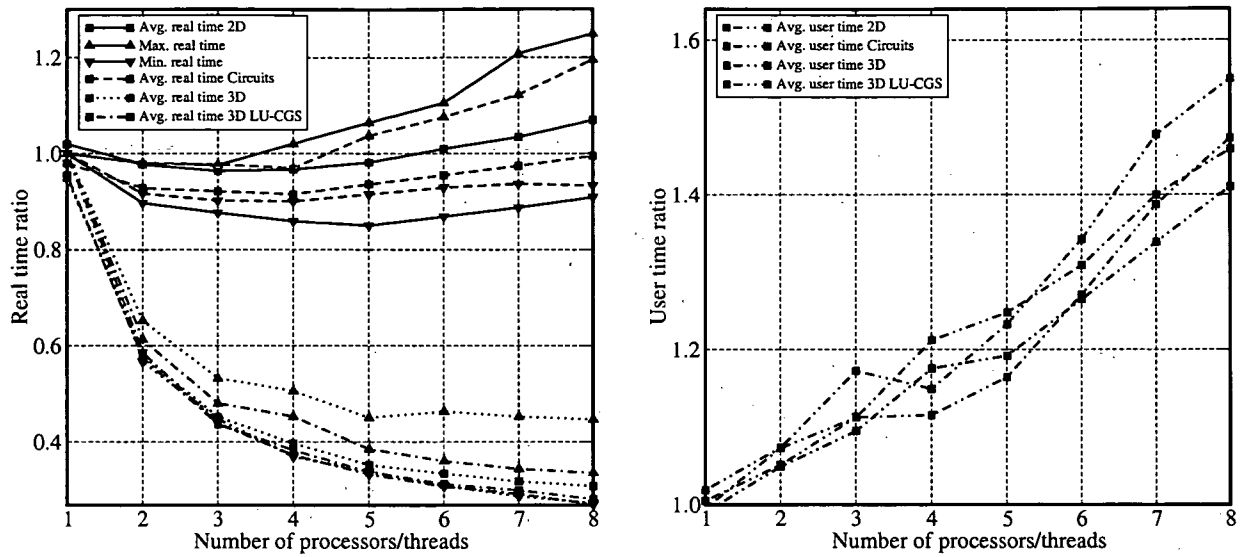


Figure 5.7: In the left figure, the relative PARDISO real/wall clock times (average, minimum, maximum) versus the number of processors/threads on the IBM cluster. The user time in the right figure increases due to the parallelization overhead.

5.5.5 Evaluation of Solvers for Higher-Order Transport Models

The same concept as discussed in the last section is used to evaluate solvers for higher-order transport models. However, instead of the devices presented above, double-gate MOSFETs with different gate lengths were used, see Table 5.5.5.

#	Simulation	SM	ET	DD	DD CPU [s]	DD Linear [%]
1	MOSFET with $L_g = 25$ nm	4642	3522	2402	65.67	56.94
2	MOSFET with $L_g = 35$ nm	3480	2640	1800	41.56	50.57
3	MOSFET with $L_g = 50$ nm	2733	2073	1413	29.54	46.94

Table 5.5: Three two-dimensional device structures with the given gate lengths were simulated. The dimensions of the linear equation systems are given depending on the transport model used. For the drift-diffusion simulation and BICGSTAB, the CPU time as well as the share of the simulation time spent for compiling, pre-eliminating, sorting, scaling, and solving are specified.

The transport models as derived in Section 2.1.3 were used and compared on the *Intel* computer: the six moments transport model (SM), the energy-transport transport model (ET), and the drift-diffusion transport model (DD). Three different simulation tasks were performed:

1. Extraction of an IV-curve: For this simulation example, V_{DS} is stepped from 0.0 V to 1.0 V by 0.025 V with $V_{GS} = 1.0$ V.
2. Quasi-static extraction of f_T : In order to apply the formula given in (3.20) two V_{GS} steps of 1.0 V ± 5 mV are necessary while $V_{DS} = 1.0$ V.
3. Small-signal extraction of f_T : A small-signal simulation based on a conditional frequency stepping was performed in order to extract f_T .

As concluded in the evaluation of the last section, several solvers are better employable for more ill-conditioned problems. In contrast to the drift-diffusion transport models, simulations based on the energy-transport and six moments models tend to be more ill-conditioned. The underlying simulation setup is

more sensitive to the mesh. As adaptive meshes are required, ongoing research outside the scope of this thesis is performed to improve this situation. However, to some extent, solver systems better equipped for ill-conditioned problems can significantly reduce the solving effort. The objective of this evaluation is to confirm the respective conclusions of the last section and to give a quantitative information on this.

The results for the two in-house iterative solvers BICGSTAB and GMRES(M) as well as those of the PARDISO solver are presented in Figure 5.8. Due to the more ill-conditioned problem and the small dimensions of the linear equation system, the direct solver significantly outperforms the two in-house solvers for the higher-order transport models. For the same reasons also the GMRES(M) solver shows advantages over the BICGSTAB. However, for the steady-state drift-diffusion simulations, the advantages of alternative solvers shrinks, whereas for small-signal simulations PARDISO is still up to 50% faster than the in-house ILU-BICGSTAB.

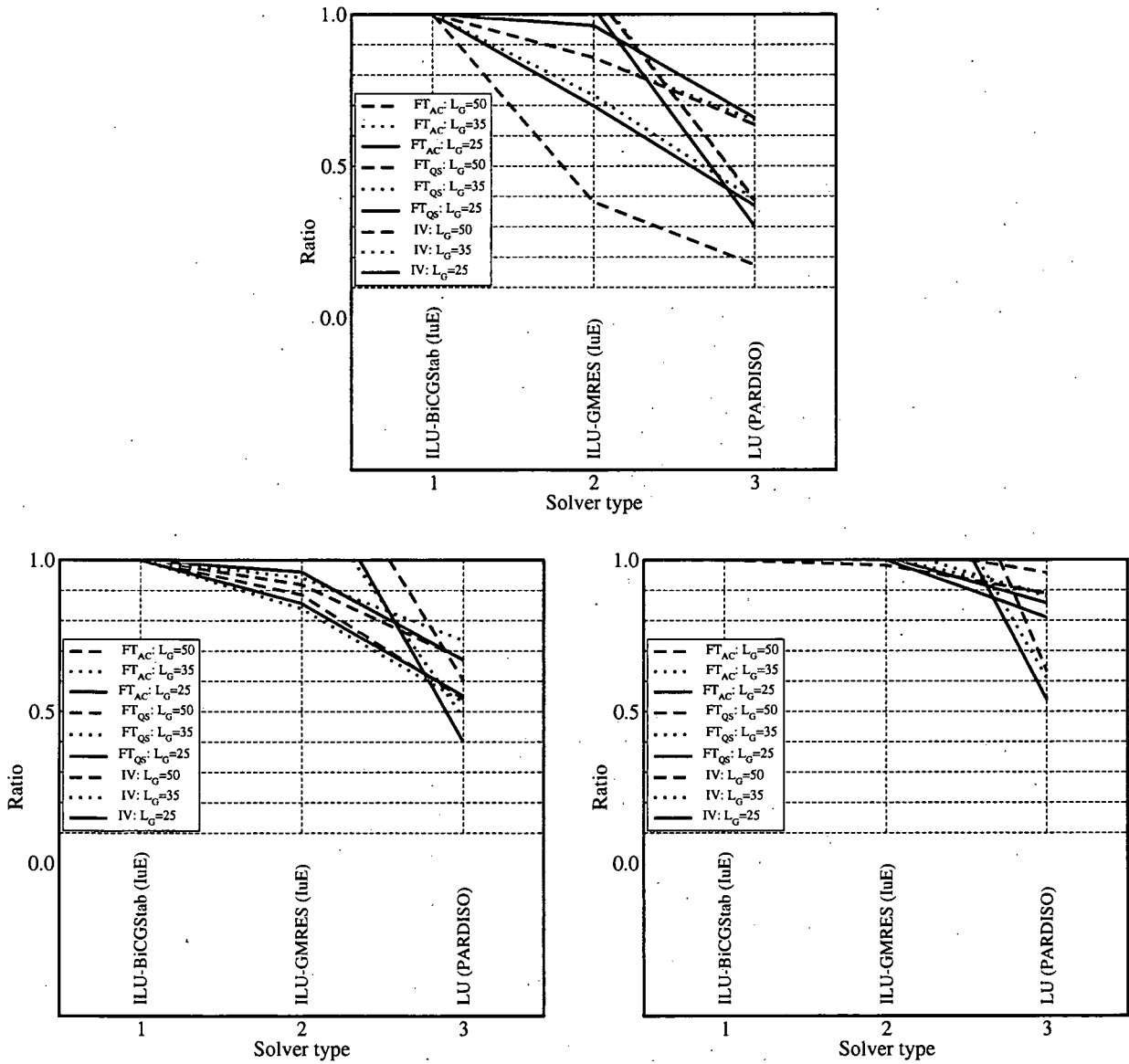


Figure 5.8: Scaled results for the six moments transport models (upper figure). The lower figures show scaled results for the energy-transport (left) and the drift-diffusion (right) transport models.

5.6 Concluding Remarks

The performance evaluation resulted in conclusions which solver system has to be activated for which kind of simulation. As already stated above, the solver system as a core module of a simulator is frequently regarded as a black box obliged to deliver the “correct” results in a “short” time. Hence, the conclusions can be used to implement an automatic solver selection in the simulator. Depending on the simulation mode, for example mixed-mode, the best-suited solver system is chosen, which results in an overall speed-up of the simulator without any user interaction.

Whereas the concept and objective of the performance evaluation has been proven to be worthwhile and promising, the executive part can be improved in order to obtain more specific and validated data:

- The extension of the simulation examples set: the current set can be extended by examples of the same type and by new simulation modes, for example by the six moments transport model. Since the benchmark program is extensible and fully automated, there is virtually no limit besides a reasonable run-time of the complete benchmark. Whereas the original set of examples has pursued the idea of orthogonal examples, which means that they are widely independent from each other, more examples of the same type would yield more significant averages.
- A more differentiating grouping of the results: the existing three groups can be split into more subgroups. For example the three-dimensional examples can be grouped in various dimension ranges, which would allow to assess the choice of iterative and direct solvers more accurately. In addition, the transport models shall form separate groups. Keeping the objective of an automatic solver selection in mind, the grouping can be based on any parameter known in advance, for example simulation mode, dimension, transport model etc.
- Taking the memory consumption into account: As discussed also in [5] for harmonic balance simulations, the selection can be additionally based on the respective amount of required memory compared with the memory available. If costly simulations are considered, the required memory can become an interesting criterion. Since a simulator is able to detect the host type and the amount of available memory, this criterion can also be part of an automatic solver selection.
- The data conditioning and visualization of all results: in addition to the grouping, averaging, and scaling of the results, a more sophisticated profiling of the various solvers can be given.

Basically, the results are obtained by running a solver on a set of n examples and measuring interesting data, for example the simulation time. In [79], a performance profile is used to evaluate and compare the performance of various solvers. This profile is defined as follows: for an example j the solver i yields the data $d_{i,j}$. Since for all examples the performance of the solver i shall be compared with that of the best solver, $d_{j,\min}$ is defined as the minimum data of all solvers for the example j . Depending on an $\alpha \geq 1$ the performance profile of solver i is given by $p_i(\alpha)$:

$$p_i(\alpha) = \frac{\sum_j k(d_{i,j}, d_{j,\min}, \alpha)}{n}, \quad \text{with} \quad (5.3)$$

$$k(d_{i,j}, d_{j,\min}, \alpha) = \begin{cases} 1 & \text{for } d_{i,j} \leq \alpha d_{j,\min} \\ 0 & \text{for } d_{i,j} > \alpha d_{j,\min} \end{cases} \quad (5.4)$$

The performance profile gives the fraction of examples for which solver i is within a factor of α of the best. Thus, $p_i(1)$ is the fraction for which solver i gave the best results. $p_i(2)$ is the fraction for which solver i is within a factor of 2 of the best. Finally, $p_i(\infty)$ is the fraction for which solver i could be successfully employed at all. The last value is particularly interesting, since it is inevitable that the benchmark takes failures explicitly into account.

Chapter 6

Examples

This chapter presents different kinds of examples which were simulated with the small-signal capabilities of MINIMOS-NT. First, results of an InGaP/GaAs HBT are presented, followed by a SiGe-HBT. The features have also been employed for investigating a wide-bandgap SiC MESFET. In order to apply the new mixed-mode AC capabilities, three mixed-mode examples have been simulated. The successful transient simulation of an amplifier is followed by a small-signal simulation of the amplifier extended by a resonant circuit. By feeding back the output of the resonant circuit to the input of the amplifier a *Colpitts* oscillator is set up, which is then again subject to a transient simulation. Finally, results of the simulation of double-gate MOSFETs are compared regarding the drift-diffusion and higher-order transport models. All simulation results are compared either with measurements or with reference results of other simulators.

6.1 Simulation of an InGaP/GaAs Heterojunction Bipolar Transistor

By means of the small-signal simulation mode of MINIMOS-NT, various high-frequency data for a one-finger InGaP/GaAs HBT with an emitter area of $3\text{ }\mu\text{m} \times 30\text{ }\mu\text{m}$ were extracted. This high-power device has been used for power amplifier circuits for mobile communication. Figure 6.1 shows the simulated device structure and the pad parasitics (capacitances and inductances) of the measurement environment used for the S-parameter measurement in the two-port pad parasitic equivalent circuit.

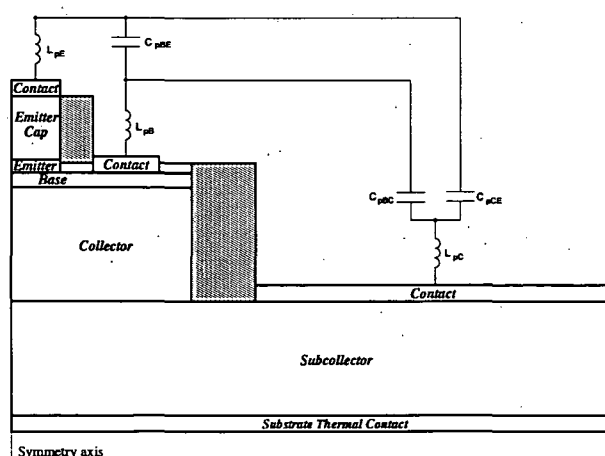


Figure 6.1: Simulated device structure together with pad parasitics used for S-parameter calculation [161].

The pad capacitances of the equivalent circuit are $C_{pBE} = 150$ fF, $C_{pCE} = 75$ fF, and $C_{pBC} = 24$ fF. The parasitic inductance values are $L_{pE} = 1$ pH, $L_{pB} = 75$ pH, and finally $L_{pC} = 50$ pH. The resistive parasitics are neglected, since a rather small device and therefore only low currents are considered.

Figure 6.2 shows a comparison between measured and simulated collector currents i_C and an almost perfect match of the curves in the small-signal area of the figure. A further increase of the input power causes harmonics in the device, which cannot be obtained by the linear small-signal mode (see Section 2.6).

The combined *Smith*/polar charts with a radius of one in Figure 6.3 show a comparison of simulated and measured S-parameters at $V_{CE} = 3$ V, with current densities $J_C = 2$ kA/cm², $J_C = 8$ kA/cm², and $J_C = 15$ kA/cm², respectively, for the frequency range between 50 MHz and 10 GHz. For the same device the high-frequency figures of merits current gain g_m and the squared absolute value of the current ratio parameter H_{21} were extracted. The cut-off frequency f_T and maximum oscillation frequency f_{max} are found at the intersection of these curves with the 0 dB line. The lower right side of Figure 6.3 shows a comparison of the simulated and measured g_m and the absolute value of H_{21} . The measurement data ends at 10 GHz, whereas the simulation could be continued to 20 GHz showing another important advantage of simulators to measurement equipments. In addition, a mixed-mode circuit was set up to compare large signal measurement data in the small-signal range.

The AC-simulation takes about 200 s CPU-time on a 2.4 GHz *Intel* Pentium IV with 1 GB memory running under *Suse Linux* 8.2 for a S-parameters computation with 20 frequency steps. A number of 20 steps is more than sufficient to produce the graphs. For comparison, the conventional small-signal equivalent-circuit approach takes about 590 s CPU-time at the same machine for 200 time steps at only one given frequency. As stated in the introduction, many time steps have to be performed to ensure appropriate accuracy in the time-domain to obtain sufficient accuracy for one frequency. To avoid this number of time steps for all frequencies required, only one frequency is used to extract an equivalent circuit valid in a specific frequency range. The time for such a post-processing of the transient simulation results to obtain the S-parameters at all frequencies is not included. Thus, the more accurate approach can speed up the frequency-domain simulation by about 98% (taking one frequency into account).

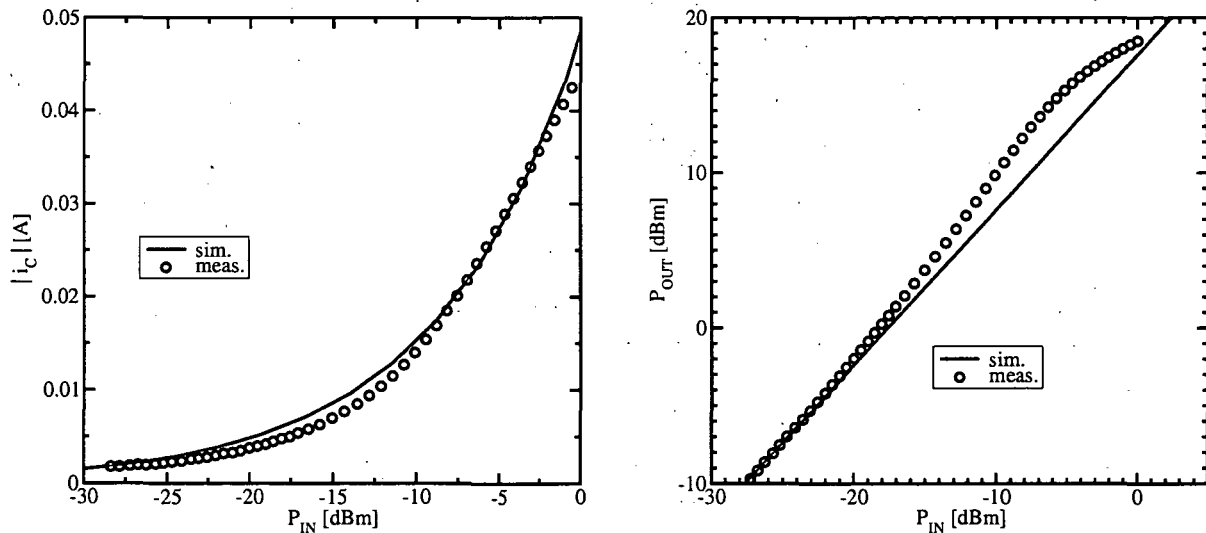


Figure 6.2: Comparison of simulated and measured AC collector current i_C over AC input power P_{IN} (left). Comparison of simulated and measured AC output power P_{OUT} over AC input power P_{IN} (right).

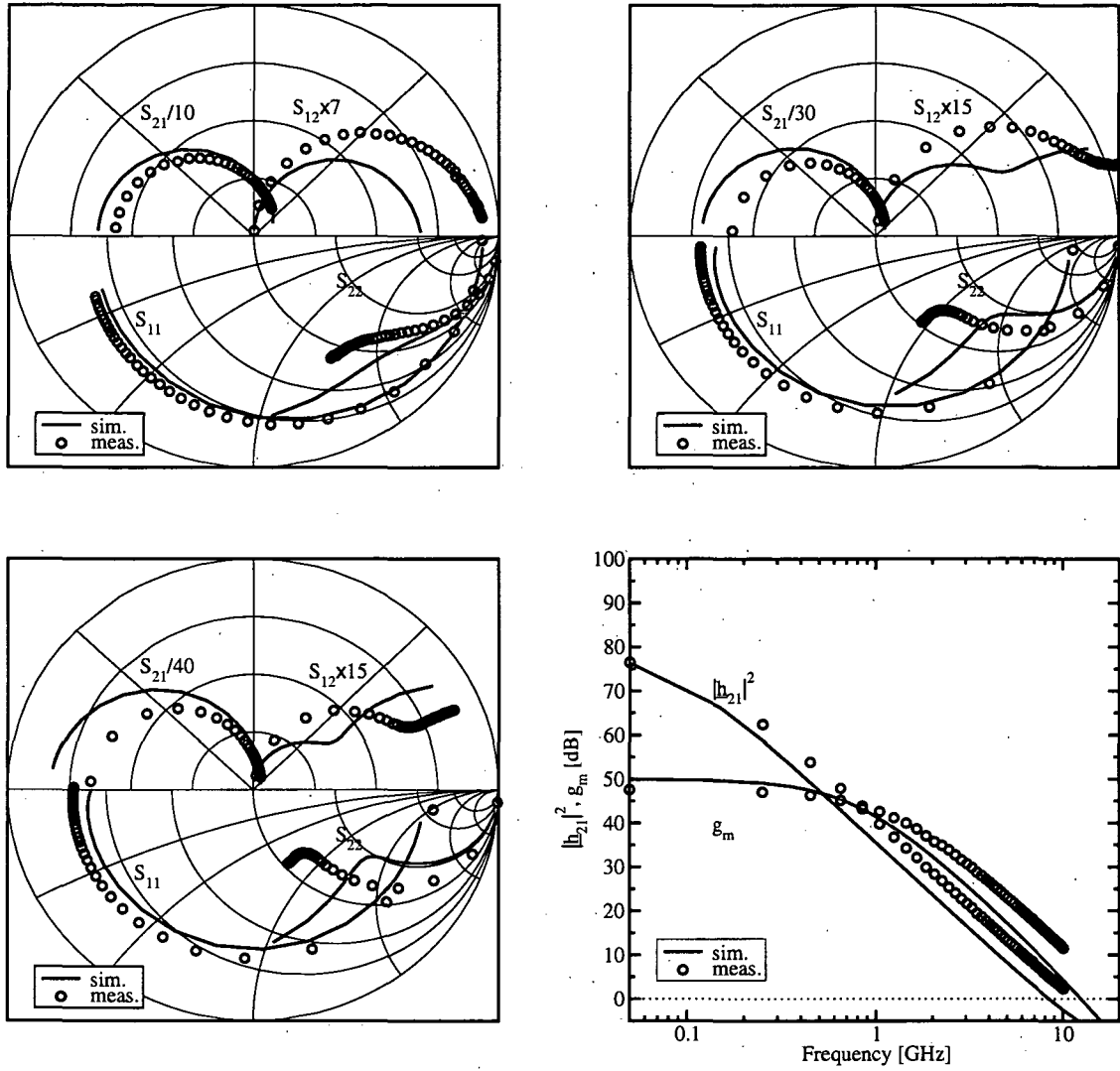


Figure 6.3: S-parameters in a combined *Smith/polar* chart with a radius of one from 50 MHz to 10 GHz at $V_{CE} = 3$ V, $J_C = 2$ kA/cm² (upper left), $J_C = 8$ kA/cm² (upper right), and $J_C = 15$ kA/cm² (lower left). In the lower right figure the short-circuit current gain and matched gain versus frequency at $J_C = 15$ kA/cm² is shown.

6.2 Simulation of a SiGe Heterojunction Bipolar Transistor

The investigated $0.4 \times 12 \mu\text{m}^2$ SiGe-HBT device structure is obtained by process simulation [110], see Figure 6.4. For DC simulations usually only the active part (base and emitter area, collector contact moved to the bottom) of the device is required. For that reason the collector area was cut to speed-up the simulations. Only half of the real structure was simulated because of symmetry. The upper figure in Figure 6.7 shows a comparison of simulated and measured forward *Gummel* plots at $V_{CE} = 1$ V.

For AC simulations, however, it is absolutely necessary to take the complete device structure into account. Otherwise, the simulation of the reduced device structure cannot reproduce the important capacitances between collector and substrate C_{CS} as well as between base and collector C_{BC} . In addition, the correct base and collector resistances are missing. There are two possibilities to overcome this problem. Either the missing parts are approximated by introducing linear elements in a post-processing step or a larger or even complete structure is used for AC simulations. The first option allows faster simulations but

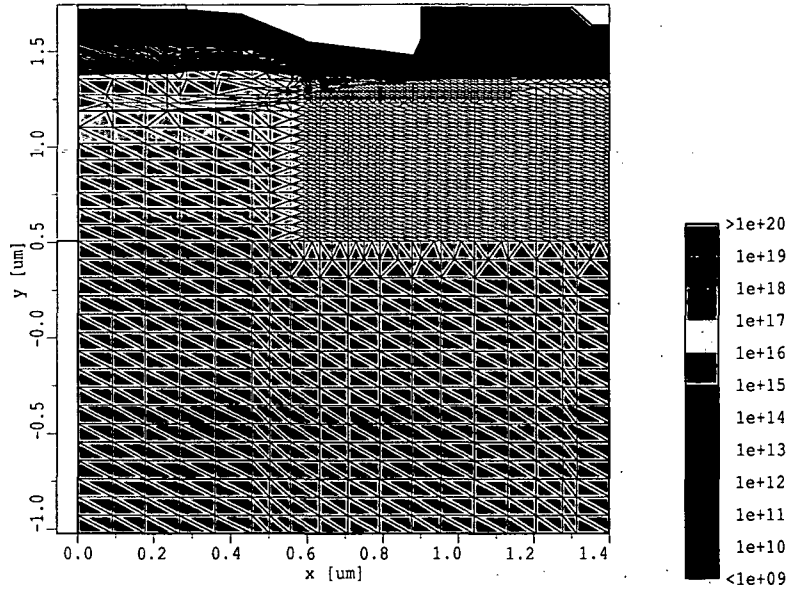


Figure 6.4: Active antimony concentration of the investigated SiGe Heterojunction Bipolar Transistor (large device).

gives approximated results. The second one produces more accurate results and does not require a post-processing step, but takes much more time: in the example the computational effort of device simulation is 2.5 times higher.

In Figure 6.6 both options are compared: in the frequency range between 50 MHz and 31 GHz measured and simulated S-parameters at $V_{CE} = 1$ V and current densities $J_C = 28 \text{ kA/cm}^2$ and $J_C = 76 \text{ kA/cm}^2$ are shown in the frequency range between 50 MHz and 31 GHz. For the first option the device structure is embedded in a circuit containing the following elements: $C_{CS} = 50 \text{ fF}$, $C_{BC} = 20 \text{ fF}$, $R_B = 15 \Omega$ and $R_C = 27 \Omega$. Their values were experimentally estimated. The results of the second option are the intrinsic parameters only.

For the same device the matched gain g_m and the short-circuit current gain H_{21} is calculated in order to extract the figures of merit cut-off frequency f_T and the maximum oscillation frequency f_{max} found at the intersection with the 0 dB line (unity gain point). 6.7 shows the comparison of the simulation results and the corresponding measurement data. While the measurement data ends at 31 GHz the simulation could be extended to frequencies beyond this intersection. Note that the peak f_T in the left figure of Figure 6.7 corresponds exactly to the frequency at the respective intersection in the right figure.

Figure 6.7 shows also the effect of the introduction of an anisotropic electron mobility [160]. In addition, results obtained by the commercial device simulator DESSIS [111] using default models and parameters are included for comparison. The agreement in order of the typical curve characteristics with measured and transformed data proves the efficiency of the approach. In addition, the performance speed-up in comparison to alternatives is an important advantage. However, a general approach to match simulated results and measured data perfectly has to comprise a proper physical modeling of the complete device since there are no extrinsic fitting parameters available.

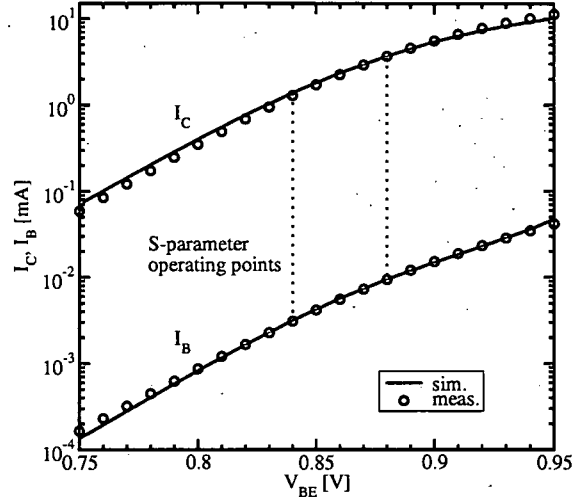


Figure 6.5: Comparison of simulated and measured forward *Gummel* plots at $V_{CE} = 1$ V.

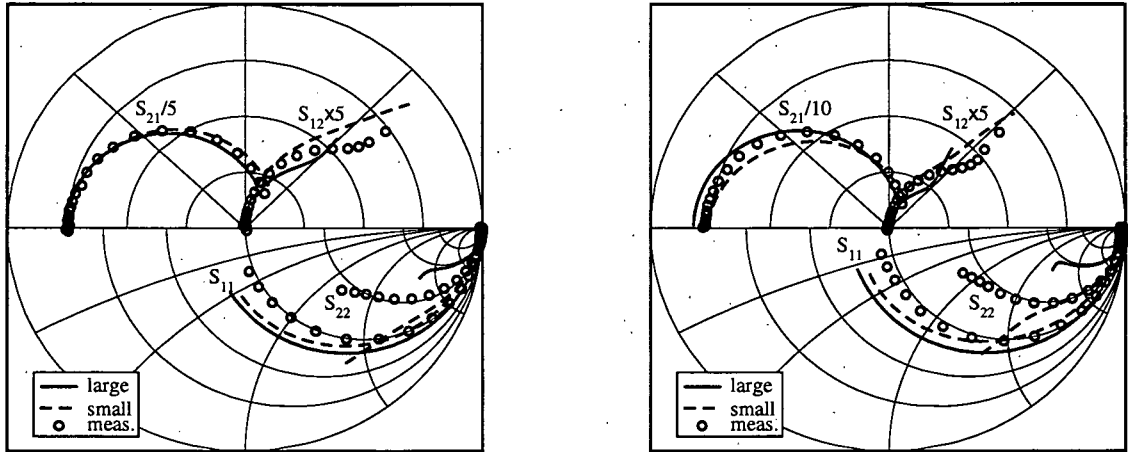


Figure 6.6: The figures compare S-parameters in a combined *Smith/polar* chart with a radius of one from 50 MHz to 31 GHz at $V_{CE} = 1$ V for $J_C = 28$ kA/cm² (left) and $J_C = 76$ kA/cm² (right) for a large device structure and a small one embedded in a circuit.

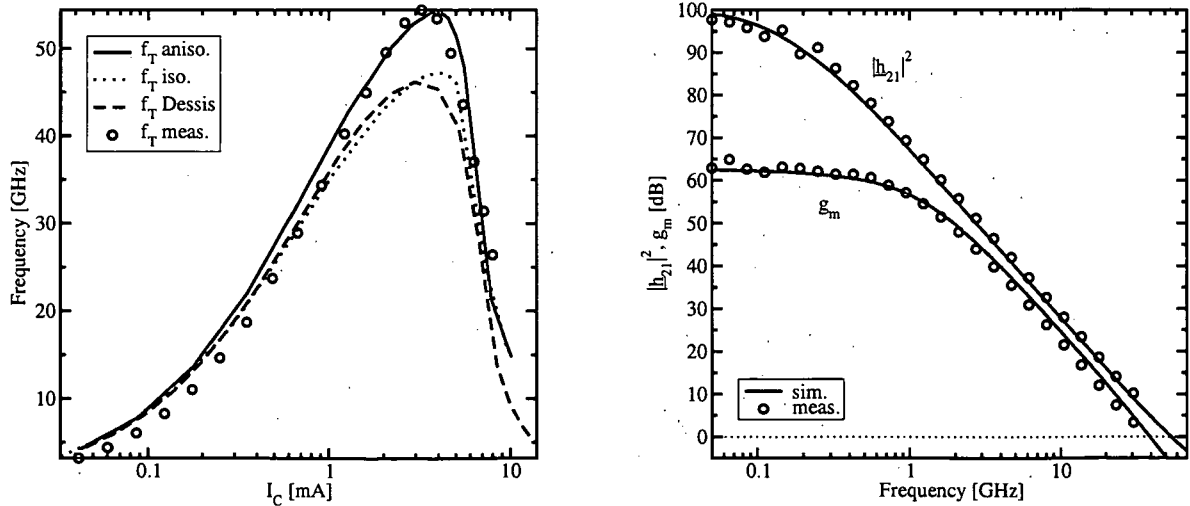


Figure 6.7: The cut-off frequency f_T versus collector current I_C at $V_{CE} = 1$ V (left) and the short-circuit current gain versus frequency (right) is depicted [233].

6.3 Simulation of a 4H-SiC MESFET

An advanced RF SiC MESFET was investigated by means of numerical simulation with MINIMOS-NT. This device is fabricated by using epitaxial layers on semi-insulating substrates. The simulator has been extended by physically based models that permit the operation of the device to be examined and optimum device structures to be determined [12]. Good agreement between simulation results and measurements can be shown.

SiC-based devices have specific properties which allow them to be used in high temperature, high frequency, high power, and radiation hard applications. Due to the progress in SiC-related process steps the new devices have been developed in recent years [35, 154]. Devices such as SiC MESFETs can be employed for microwave power amplifier and oscillator applications due to their excellent DC and RF performance [242]. In particular, the investigated 4H-SiC MESFETs structures are used as base station transmitters for cellular telephone systems and power modules for phased-array radars [166]. The devices are also attractive for higher operation temperatures.

In Figure 6.8, the cross-section of a SiC MESFET is depicted. The typical device parameters are the gate length L_g , the gate width W_g , and the thickness of the epitaxial layer a . Due to the higher electron mobility, MESFETs in SiC are made of an n-type material. Furthermore, the mobility of 4H-SiC is twice than that of 6H-SiC [11]. In order to minimize parasitic capacitances, a MESFET is fabricated using epitaxial layers on semi-insulating substrates. The device has three metal-semiconductor contacts: two *Ohmic* contacts at the source and drain and a gate *Schottky* barrier.

Devices for microwave- or millimeter-wave applications typically have gate lengths in the range of $L_g = 0.1 - 1 \mu\text{m}$. The channel thickness a is typically $1/3$ to $1/5$ of the gate length L_g . The spacing between the electrodes is up to four times L_g . W_g and thus the cross-sectional area is directly related to the current handling capability.

For operation the drain contact is biased at a positive potential while the source contact is grounded. The current flow through the channel is controlled by negative DC and superimposed RF gate voltages. The RF signal modulates the channel current and provides an RF gain.

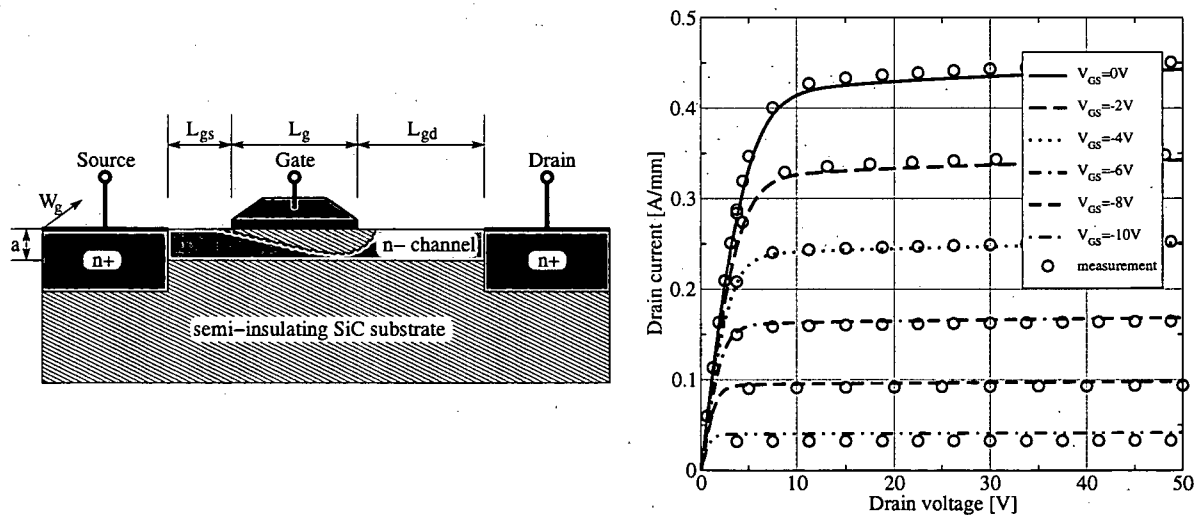


Figure 6.8: On the left, the cross section of a MESFET in SiC is shown and on the right a comparison of measured and simulated DC IV characteristics [12].

The 4H-SiC MESFET as shown in Figure 6.8 is analyzed by means of numerical device simulation for both DC and high frequency characteristics. For the calibration the specifications obtained from *Cree's* CRF-24010 4H-SiC MESFET [41] are used to define the simulated device. The charge carrier transport characteristics, the device dimensions, and the doping details are adjusted until good agreement between the simulated and measured IV characteristics is obtained.

The good agreement between simulated and measured steady-state IV characteristics is shown on the right of Figure 6.8. This MESFET produces a maximum channel current of about 450 mA/mm. The ability of the gate bias to turn the device off and on is good, as indicated by the channel current with zero and high reverse gate bias applied. Good turn-off characteristics are observed for reverse bias slightly greater than -10 V. The ability of the device to modulate current is given by the device transconductance which for this device is about $g_m = 160$ mS/mm. The zero gate voltage drain current at $V_{DS} = 10$ V is 0.42 A/mm. This device simultaneously shows a high breakdown voltage of 110 V and a low leakage current of only 100 μ A/mm at 500 K.

After the successful calibration of the simulator, the small-signal simulation mode was used to obtain additional figures of merit. The frequency range under consideration was 100 MHz to 40 GHz. After adapting the input and output impedances, the agreement between simulated and measured data as shown in the left Figure 6.9 was obtained. It is important to note, that the RF results presented here were obtained at a high drain-to-source bias voltage of 40 V and at the gate quiescent voltage of $V_{GS} = -9$ V. In the right figure of Figure 6.9, the small-signal current and power gain are depicted, showing an $f_T = 5.62$ GHz and $f_{max} = 37.18$ GHz at 0 dB.

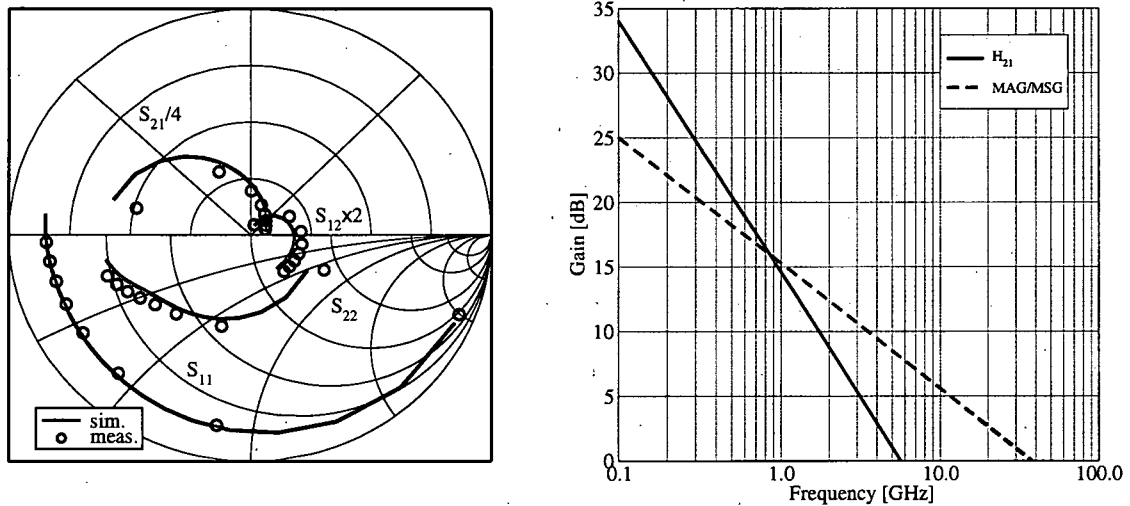


Figure 6.9: Comparison of measured and simulated S-parameters in a combined *Smith*/polar chart with a radius of one (left). Small-signal current and power gain (right) [12].

6.4 Oscillator

This example demonstrates the implemented mixed-mode AC features on the simulation of a *Colpitts* oscillator and two intermediate circuits. The following three circuits are simulated by means of transient and small-signal device/circuit simulation:

1. An amplifier with one active device.
2. A resonant circuit is coupled to the output of the amplifier.
3. The oscillator is eventually constructed by feeding back the output of the resonant circuit to the input of the amplifier.

Since all three circuits consist of equal subcircuits (see Figure 6.10), the input-deck inheritance feature can be perfectly used, see Appendix A.13 for more details on this.

6.4.1 Amplifier

The amplifier circuit is a combination of the core and the AC source subcircuits as well as of load elements. The transistor used in the core subcircuit is a $0.4 \times 12 \mu\text{m}^2$ SiGe-HBT device structure obtained by process simulation [110]. The structure was thoroughly investigated by steady-state and small-signal AC simulations [233] as presented in Section 6.2.

CircuitAmplifier

```
{
  Vsrc : ~SubCircuits.Vsrc { in = "pin1"; }
  Core : ~SubCircuits.Core { in = "pin1";
                           out = "pin2"; }

  CL : ~Devices.C { N1 = "pin2"; N2 = "pin3"; C = 1 nF; }
  RL : ~Devices.R { N1 = "pin3"; N2 = "gnd"; R = 1e3; }
}
```

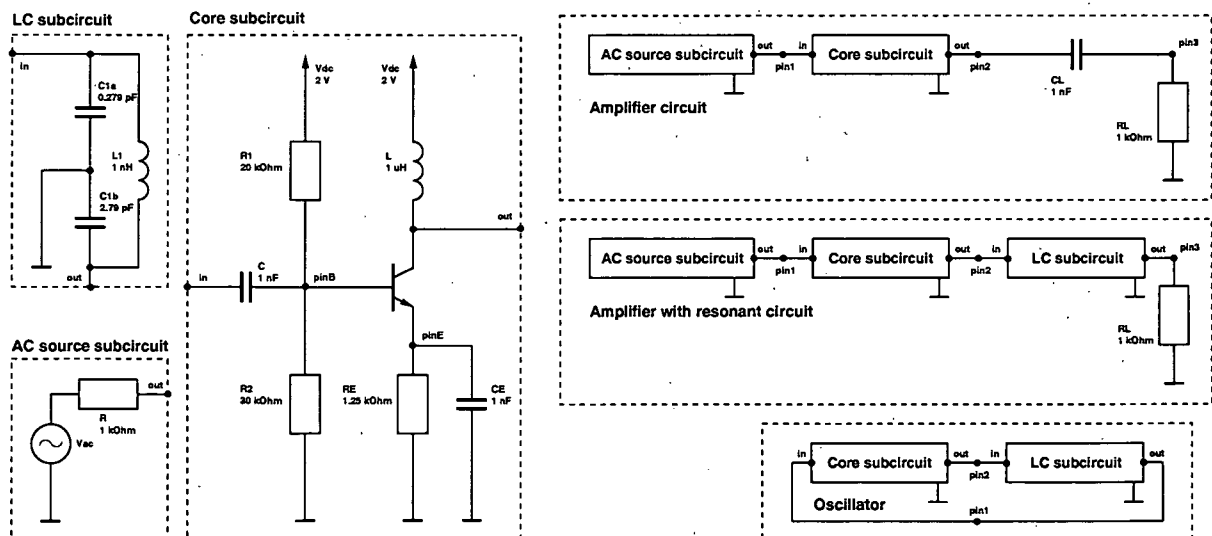


Figure 6.10: The three subcircuits which are used in the example circuits are shown on the left side. They are parts of the three circuits depicted on the right [231].

All simulations use the mixed-mode iteration scheme (see Section 3.6.4). In the first block the fixed node voltages apply static boundary conditions at the transistor terminals in order to improve convergence to an initial solution useful for the subsequent circuit simulations. In this case, the three fixed node voltages ($V_{\text{pin2}} = 2.0$ V, $V_{\text{pinB}} = 1.2$ V, and $V_{\text{pinE}} = 0.4$ V) represent the dimensioning of the circuit in respect to the chosen operating point. Transient simulation results are shown in Figure 6.11. The linear equation system has a dimension of 11,601 and the simulator requires between 1.0 and 2.9 s per time step on a 2.4 GHz Intel Pentium IV with 1 GB memory running under Suse Linux 8.2.

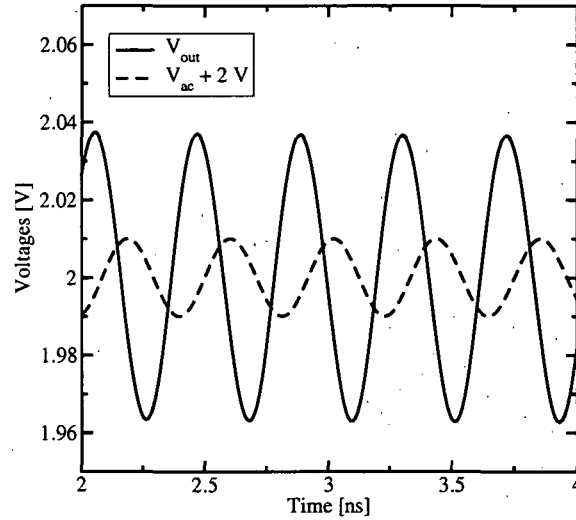


Figure 6.11: Result of transient simulation of the amplifier circuit with $V_{\text{ac}} = 10$ mV and $f = 2.4$ GHz [231].

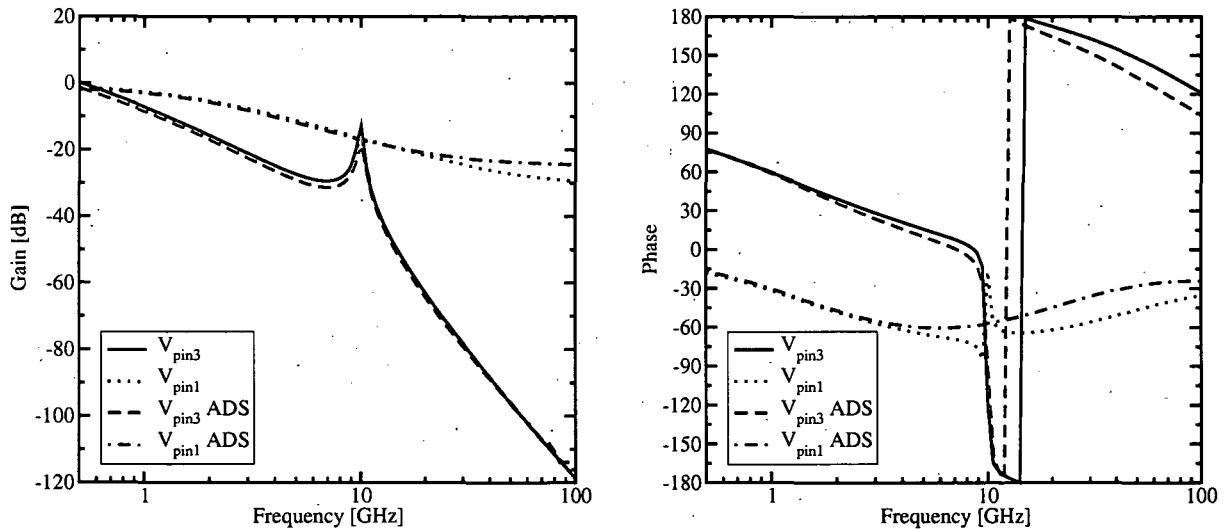


Figure 6.12: Results of small-signal simulations of the resonant circuit: absolute value (left) and argument (right). The results are compared with ADS simulations using a VBIC95 model of a similar transistor [231].

6.4.2 Amplifier with Resonant Circuit

The second example circuit consists of all three subcircuits, since the resonant circuit is now coupled to the output of the amplifier. The resonant circuit is configured for an oscillation frequency of 10 GHz. This can be confirmed by results of a small-signal simulation as shown in Figure 6.12 ($V_{\text{ac}} = 1$ mV). In average,

MINIMOS-NT requires 8.5 s per frequency step. With a VBIC95 compact model of a similar transistor, the circuit simulator ADS [4] was used to obtain data from the same circuit.

CircuitResonant

```
{
  Core : ~BaseCircuits.Core { in = "pin8"; out = "pin4"; }
  Vsrc : ~BaseCircuits.Vsrc { in = "pin8"; }
  LC : ~BaseCircuits.LC { in = "pin4"; out = "pin9"; }

  R5 : ~Devices.R { N1 = "pin9"; N2 = "gnd"; R = 1e3; }
}
```

6.4.3 Colpitts Oscillator Circuit

Finally, a *Colpitts* oscillator circuit is built by feeding back the output of the resonant circuits to the input of the core circuit (amplifier).

CircuitOscillator

```
{
  Core : ~SubCircuits.Core { in = "pin1"; out = "pin2"; }
  LC : ~SubCircuits.LC { in = "pin2"; out = "pin1"; }
}
```

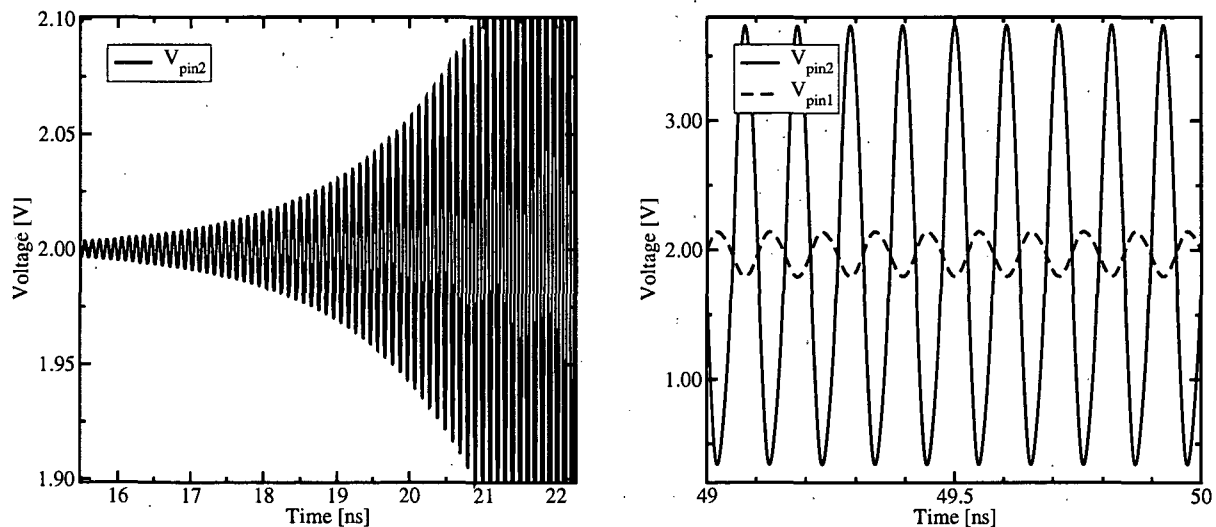


Figure 6.13: Result of the transient simulation of the oscillator: output V_{pin2} in the initial phase (left) and in the state of equilibrium (right) [231].

At turn on, random noise is generated within the active device, which is here the SiGe bipolar junction transistor, and then amplified. This noise is positively fed back through the frequency selective circuit (resonant circuit consisting of an inductor and two capacitors) to the input, where it is amplified again. After the initial phase, a state of equilibrium is reached, where the losses are compensated by the power supply. The amount of feedback to sustain oscillation is basically determined by the C_{1a}/C_{1b} ratio.

Transient simulation results are shown in Figure 6.13. In the simulator, the random noise of the active device is replaced by a numerical noise caused by the restricted representation of floating point numbers. The simulator requires 0.4 s in the initial phase and between 1.9 s and 2.9 s in the state of equilibrium per time step.

6.5 Simulations with Higher-Order Transport Models

The final section in this chapter presents results obtained from simulations of advanced MOSFET devices with different transport models. As derived and discussed in Section 2.1.3, the following transport models are compared in this work:

1. The six moments transport model, see Section 2.1.4.
2. The energy-transport model, see Section 2.1.5.
3. The drift-diffusion transport model, see Section 2.1.6.

For the example a series of double-gate MOSFETs were used. They have gate lengths from $L_g = 250$ nm down to $L_g = 25$ nm. The main objective of this example is to show that the impact of the higher-order transport models significantly increases with smaller gate lengths and that their application is inevitable for $L_g < 100$ nm. This is both demonstrated for IV curves and cut-off frequency extractions. The simulation results are compared with full-band Monte Carlo results [115]. In Figure 6.14, the basic structure of the simulated devices as well as the doping profile including a neutral channel doping are shown depending on the gate length L_g [83].

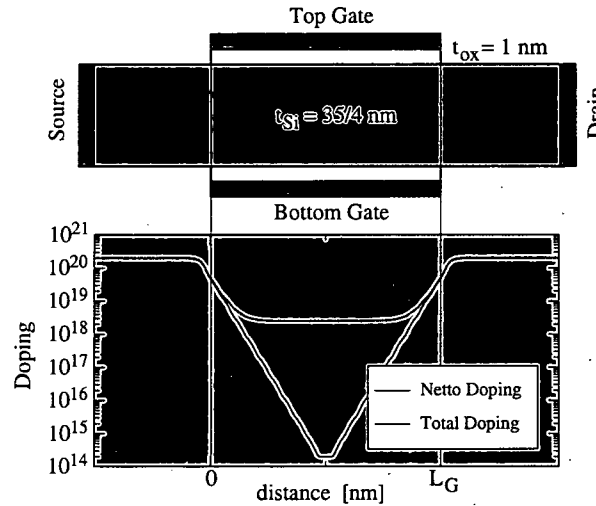


Figure 6.14: Structure of the simulated double-gate MOSFET devices. The gate length is varied from 250 nm down to 25 nm [83].

6.5.1 Simulation Results

Figure 6.15 depicts the results of the steady-state simulation of four double-gate MOSFETs with gate lengths of $L_g = 250$ nm, $L_g = 100$ nm, $L_g = 50$ nm, and $L_g = 25$ nm. Whereas for the largest device the employment of higher-order transport models does not seem to be necessary, this situation significantly changes for smaller devices. The drift-diffusion model delivers a clear underestimation of the drain current, while the energy-transport model starts to overestimate the current.

For the same devices, small-signal simulations have been performed and the results are presented in Figure 6.16. In contrast to the drain current, the error of the drift-diffusion model regarding the cut-off frequency f_T is already significant for the device with $L_g = 250$ nm. The underestimation continues with smaller gate-lengths, resulting in an error of 50% for $L_g = 25$ nm. The energy-transport model delivers the same results as the six moments model for $L_g = 250$ nm. However, for smaller gate lengths the energy-transport model systematically overestimates f_T . Note that in Section 3.3.3, a comparison of the cut-off frequency results with quasi-static simulations are shown.

6.5.2 Conclusions

As derived in Section 2.1.3, the drift-diffusion model is characterized by a very rough closure of $T_n = T_L$. Whereas the calculated terminal currents are not severely wrong, the error becomes worse if distributed quantities such as the carrier concentrations or other important quantities such as the cut-off frequency f_T are considered. In fact, the development and underestimating character of the terminal quantity error is used to justify the industrial application of the drift-diffusion model for such devices, which should have already been subject to simulation with higher-order transport models. The main reasons why the drift-diffusion model is still widely applied are its robust convergence behavior and performance.

The energy-transport models do not show an comparable numerical robustness than the drift-diffusion models any more. Due to the additional temperature quantities, the convergence behavior and the performance are generally worse. The simulation setup is more sensitive to the mesh and the heat-flux reduction degrades the condition of the system matrix [83]. However, the benefit of these models are that instead of the cold the heated *Maxwell* distribution can be used, which allows to take hot-carrier effects into account.

The six moments transport model as applied in the simulations above uses an empirical closure relation calibrated to bulk Monte Carlo data. The six moments models are even more sensitive to the mesh and the condition is more degraded. On an engineering level one can conclude that if the application of energy-transport models has been restrained due to these properties, this will be even more the case for the six moments models. However, they give the best results overall as more details of the distribution function are available. For example, whereas the energy-transport models overestimates the velocity, the six moments models stay closest to the Monte Carlo data.

Furthermore the development of the error of the higher-order transport models with decreasing gate lengths must not be neglected. As already said, the error of the terminal quantities calculated by the drift-diffusion model is not significantly decreasing with smaller gate lengths. In contrast, higher-order transport models indicate that the error is disproportionally increasing with smaller gate lengths. This allows one to conclude that the six moments models should be preferred over the energy-transport models. Although the numerical properties of the assembled equation systems become worse, one can partly counteract on the numerical solver level. The solver evaluation in Section 5.5.5 clearly indicates that some solvers such as the GMRES(M) shows significant advantages over the BICGSTAB in terms of convergence and performance. In addition, as the higher-order transport models are more sensitive to the mesh, advanced generation of adaptive meshes would enable a more convenient and industrial application of that models.

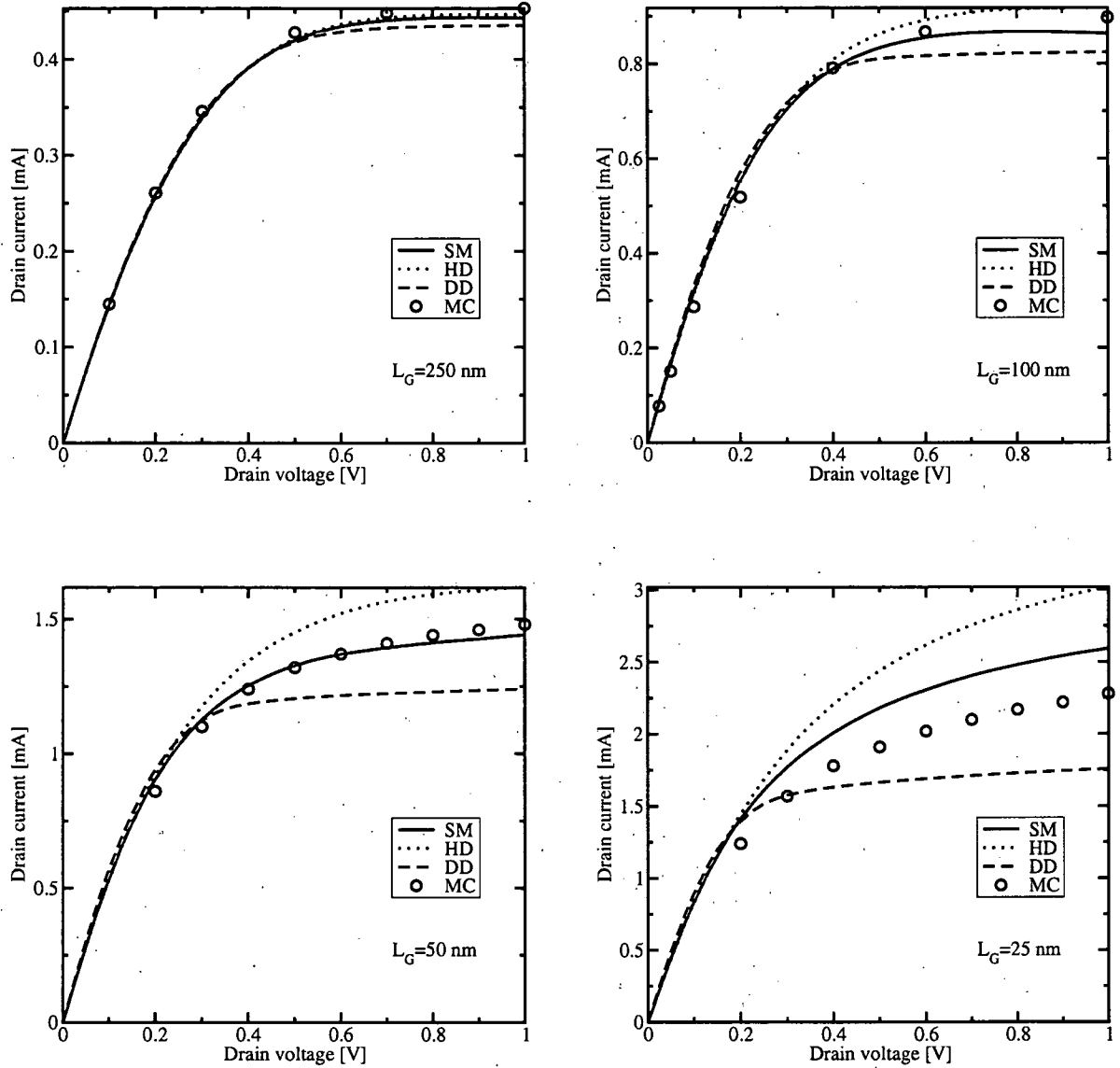


Figure 6.15: These four figures show the increasing errors of the macroscopic transport models with decreasing gate lengths. Whereas at $L_g = 250$ nm in the upper left figure the difference of the drain currents is minimal, it can be clearly seen that for $L_g = 50$ nm the six moments transport model delivers the best results. However, for extremely small gate length, it loses its advantages and even more moments would be necessary. Note that the drift-diffusion model results in terminal quantities which underestimates the Monte Carlo results.

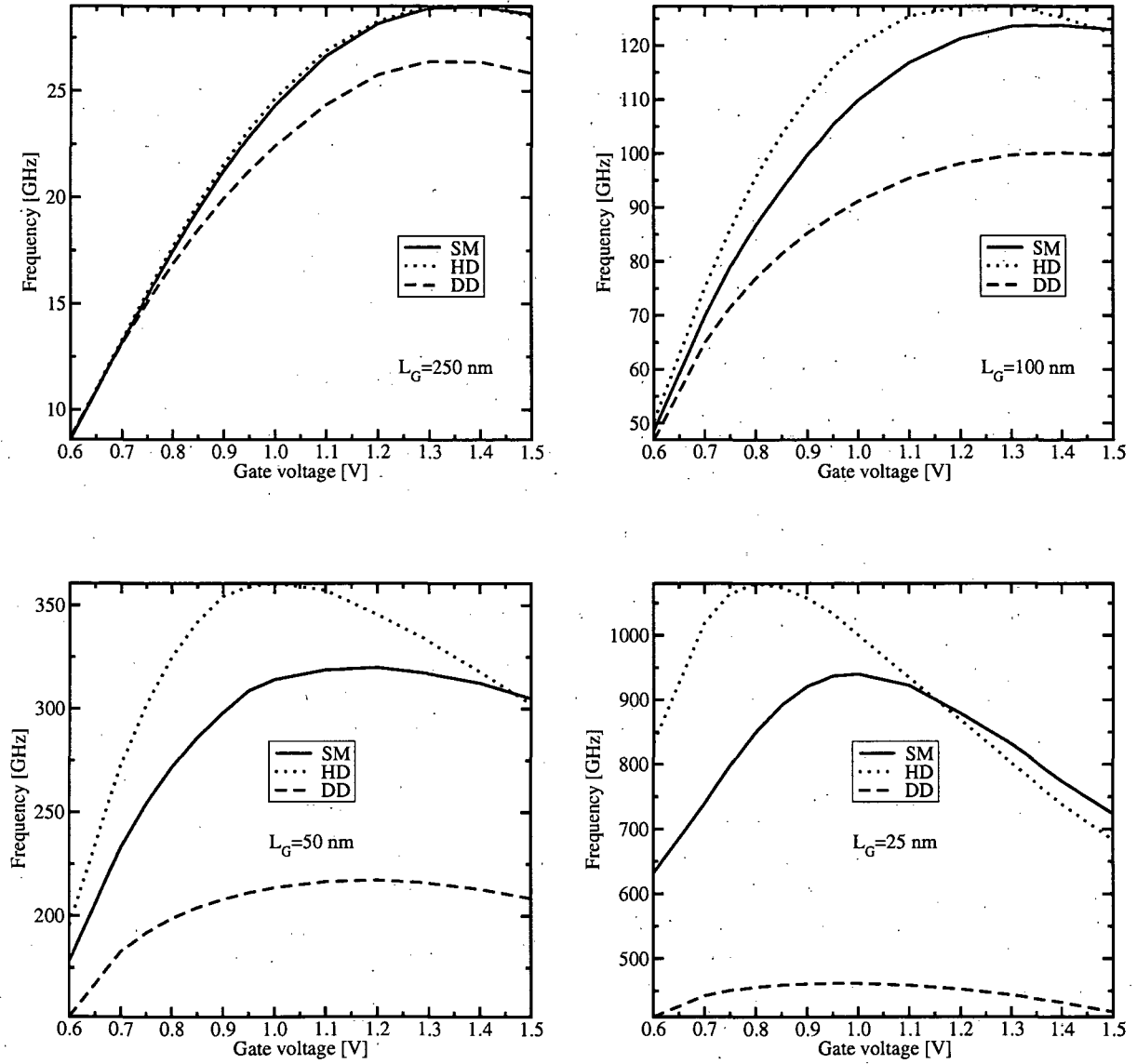


Figure 6.16: These four figures show the cut-off frequency versus the drain current and the much higher sensitivity of that small-signal figure of merit is demonstrated.

Chapter 7

Summary and Outlook

In the course of this work the requirements for a both effective and efficient small-signal simulation mode have been identified. The respective features have been implemented in the general-purpose device/circuit simulator MINIMOS-NT, which provides now various small-signal capabilities for two- and three-dimensional device/circuit simulations. Among them are not only basic features like complex-valued excitations of the devices, but also extended ones which can be used to extract various figures of merit. Besides the correct and efficient implementation, the usability of the new features was an important issue.

Since the small-signal mode is based on the S^3A approach, one complex-valued linear equation system has to be assembled and solved for each frequency step. The respective modules have been equipped with all necessary features to handle both real-valued and complex-valued systems. They have been extended to be generally applicable for all kind of simulations. The solving of linear equation systems accounts for a large share of the simulation time. For that reason, alternative solver systems with external modules among them, have been integrated or coupled to the solver module. These modules, which take all identified numerical requirements of semiconductor simulations into account, are employed also by other simulators. In order to efficiently use new available hardware architectures, parallelization strategies have been discussed. Future developments in this area should focus on further parallelization of the solver algorithms itself, but also of suitable parts of the simulator code.

Basically, the implemented small-signal capabilities can be employed for a wide set of semiconductor device structures. Thus, the simulation of advanced RF CMOS transistors or devices with compound semiconductors is now possible in a straightforward way. In addition, higher-order transport models are directly applicable for small-signal simulations. The new simulation mode was used to describe the properties of different device structures and circuits. In the course of this work, two different heterojunction bipolar transistors based on the InGaP/GaAs and SiGe material system were analyzed. In addition, two amplification circuits and a *Colpitts* oscillator have been simulated by means of mixed-mode device/circuit simulation. Furthermore, an advanced RF silicon carbide MESFET was investigated and higher-order transport models were compared for several double-gate MOSFETs.

From the engineer's perspective, additional features of the simulator might be useful, for example the extraction of noise and linearity parameters. In addition, the large-signal simulation capabilities, which are now possible by means of transient simulations, could be extended by introducing a harmonic balance simulation mode. With respect to performance the on-going parallelization efforts of the various solvers are particularly interesting. In addition, platform-specific optimizations can be utilized. Due to the availability of drastically increased memory resources, also modules based on full matrix storages become more and more interesting. For that reason, the integration of the external LAPACK and BLAS routines might be useful. In order to make the solver module even more attractive for alternative simulators, the integration of solvers, including in-house codes, for symmetric matrices might be interesting.

Appendix A

Input-Deck Interface to the New Simulator Features

While implementing new simulator capabilities it is very important to keep the usability of the new features in mind. MINIMOS-NT employs a powerful input-deck [118], enabling the user to customize the simulation settings in many details. All new features are directly available via respective input-deck keywords. In addition, it was tried to extend already existing and thus familiar functions in case the small-signal or numerical context is related to them. This chapter contains all input-deck definitions the first chapters are referring to.

A.1 Activation of the Transient and Small-Signal Mode

Transient and small-signal analysis are related simulation modes. In respect to the usability of the simulator, their setup and configuration is very similar. In MINIMOS-NT, this setup is done in the `Solve` section of the input-deck. The complete specification of the time or frequency domain is done by stepping functions, see Appendix B.

If either the transient or small-signal mode is activated, the simulator switches to the respective mode after the calculation of the steady-state operating point. It is important to note that the transient and frequency stepping functions can fully interoperate with other stepped parameters, such as contact voltages. In order to ensure efficiency, the time and frequency steppings always have the highest priority within one slot. In case one of them should be used as a lower-prioritized parameter, the simulation results must be reordered in a post-processing step (see Appendix C.1).

While the transient simulation mode is enabled via the `transient` keyword, the small-signal simulation is activated by the keyword `ac`. The time and frequency domains are defined by stepping functions assigned to keywords `time` and `frequency`, respectively. If both modes are simultaneously activated, an error message is issued. The example shows the activation of the transient mode, whereas the small-signal mode is deactivated.

```
Solve
{
  transient = yes;
  time      = step(0 s, 1e-6 s, 1e-8 s); // start, end, delta

  ac        = no;
  frequency = step(1 MHz, 100 GHz, 10, log=yes);
}
```

Whereas for the transient simulations at least two steps are always required, a single frequency small-signal simulation is a useful feature. Since the definition by a stepping function is mandatory, a new function `stepSingle` was introduced. A single stepping might look weird at the first glance, but the extended features, such as slots, justify this concept (see Appendix B).

A.2 Activation of the Admittance Matrix Calculation Feature

The so-called intrinsic admittance matrix can be obtained by activating the admittance matrix calculation feature in section `Solve` of the input-deck, see Section A.2. The activation of this feature has the following consequences:

- The AC boundary conditions as specified in the `Device` section are overruled by the unity voltage.
- The unity voltage is applied once at each terminal while the others are zero. This affects only the right-hand-side vector and the system matrix remains constant.
- For that reason, the multiple-right-hand-side feature of the linear modules is applied. Since the multiple assembly of the system matrix can be skipped, the simulation is sped up significantly.
- Unless `suppressAdmittanceOutput` is set, the output of the simulator changes from the terminal quantities to the admittance matrix.
- Several additional features and output functions can be used.

The keyword `complexPolar` in the `Log` section is provided for printing complex-valued numbers in polar-coordinates.

The example shows the setup for calculating the admittance matrix at the frequency of 1 GHz.

```
Solve
{
    ac = yes;
    frequency = stepSingle(1 GHz);

    calcAdmittanceMatrix = yes;
}
```

A.3 Simulation Setup of the Diode Example

This section contains the simulation setup for the small-signal simulation of the diode.

```
Device
{
    +Contact1 = 0.0 V;
    +Contact0 = step(-0.8 V, 0.8 V, 0.1 V);

    +acContact1 = 0.0 V;
    +acContact0 = 1.0 V;
}
```

```

Solve
{
    ac = yes;
    frequency = step(1 Hz, 1e20 Hz, 10, log=yes);
}

```

In the device section (not completely shown here) the terminal conditions are set once for the DC operating points and once for the AC systems. Note the prefix `ac` of the name of the terminals. The DC anode voltage is stepped by 0.1 V from -0.8 V to +0.8 V.

As seen in the previous section, the frequency domain is defined in the `Solve` section of the input-deck. The simulation consists of a logarithmic frequency stepping from 1 Hz to 10^{20} Hz with a multiplication factor of 10.

A.4 Inquiring Capacitances

This section shows how the gate and the gate-drain-capacitances are inquired by using the function `outputParam` (see Section A.6):

```

Curve
{
    file = "ac.crv";

    Response
    {
        Cg = outputParam("Device", "CMatrix", "Gate", "Gate");
        Cgd = outputParam("Device", "CMatrix", "Gate", "Drain");
    }
}

```

A.5 Configuration of the Two-Port Features

This section describes the configuration of the equivalent circuit as discussed in Section 3.5. The following keywords in the `Contact` section are available:

Keyword	Type	Default	Description
<code>Z0</code>	Real	50 Ω	characteristic impedance
<code>parL</code>	Real	0.0 H	parasitic inductance for the two-port calculation.
<code>parC</code>	Real	0.0 F	parasitic capacitance for the two-port calculation.
<code>parR</code>	Real	0.0 Ω	parasitic resistance for the two-port calculation.

Table A.1: Keywords provided in the `Contact` section.

Thus, all capacitances, inductances, and resistances can be defined in the same way as the keyword `Z0`:

```

Device : DeviceDefaults
{
    Phys
    {

```



```

+Collector{ Contact{ Ohmic{ parL = 50 pH; parC = 75 fF; }}}
+Emitter { Contact{ Ohmic{ parL = 1 pH; parC = 24 fF; }}}
+Base     { Contact{ Ohmic{ parL = 75 pH; parC = 150 fF; }}}
}

Solve
{
    calcTwoPortParameters = yes;

    port1 = "Base";
    port2 = "Collector";
    ground = "Emitter";
}

```

In the Solve section the configuration of the two-port circuit can be found. The two ports are connected to the base and collector terminal, respectively, and the ground is connected to the emitter terminal. By setting the calcTwoPortParameters to yes, the calculation is enabled.

A.6 Output Functions

The standard output function of MINIMOS-NT can also be used to request complex-valued quantities. The return value of the output function can be passed to any built-in input-deck function, for example arg to get the argument of the complex value.

For output of the various matrix parameters the function outputParam is used. See the following examples for details, which are based on the specifications given in the Solve section. The outputParam functions can be used to request the quantities YMatrix, SMatrix, ZMatrix, HMatrix, AMatrix, and CMatrix. Whereas the first five matrices contains the common complex-valued two-port Y-, S-, Z-, H-, and ABCD-parameters, the last matrix is the real-valued capacitance matrix of the device.

Note that the Z-, H-, and ABCD-parameters are available in combination with the parasitic circuit only (see calcTwoPortParameters in Section 3.5)).

```

Curve
{
    file = "ac.crv";
    Response
    {
        +IB = output("Device", "I", "BaseContact");
        +IC = output("Device", "I", "CollectorContact");

        +ICac = output("Device", "I", "CollectorContact", ac=yes);

        // intrinsic Y11 and S11:
        +Y11i = outputParam("Device", "YMatrix", "BaseContact", "BaseContact");
        +S11i = outputParam("Device", "SMatrix", "BaseContact", "BaseContact");

        // extrinsic Y11 and S11:
        +Y11 = outputParam("Device", "YMatrix", "BaseContact", "BaseContact",
                           parasitic=yes);
        +S11 = outputParam("Device", "SMatrix", "BaseContact", "BaseContact",
                           parasitic=yes);
    }
}

```

```

// absolute value and argument of S21:
+S21abs = abs(outputParam("Device", "SMatrix", "CollectorContact",
                           "BaseContact", parasitic=yes));
+S21arg = arg(outputParam("Device", "SMatrix", "CollectorContact",
                           "BaseContact", parasitic=yes));
}
}

```

A.7 Setup for the Cut-Off Frequency Extraction

The simulation concept is divided into two steps:

1. Simulation configuration and test
2. Simulation

Since the curve shape is known in advance the configuration is usually performed very quickly. After setting up a standard input-deck consisting of all parts necessary for the device simulation, the definition of the stepping variables V_B and frequency remain. The base voltage is stepped to vary the collector current in a range between 10^{-6} and 10^{-1} . The frequency variable is combined with the new conditional stepping function. This function requires two boundaries that should be updated for each base voltage step. Thus, an index stepping is used to step through three different arrays (tables): one array contains the base voltages (vbtbl), one the lower and one the upper boundaries. Usually these arrays are defined in the global section of the input-deck (see below).

After the basic part of the configuration is finished, the boundaries have to be tested since their condition values must have different signs. Note the optional argument maxCount of the frequency stepping function. It is now set to 3, which forces the simulator to give additional information on the stepping and which restricts each voltage step to three frequency steps being the minimal conditional simulation.

Thus, the boundaries as defined in tables lower and higher of each voltage step can be tested. After each step the simulator gives detailed information on the steps, for example

```

Conditional Stepping Information: Interval [ 1.500e+08, 2.000e+08]
                                   Values = [ 3.257e-01, -4.488e-03]

```

and in case of invalid boundaries a similar error message, for example

```

#### mmnt: Invalid range for step condition (count=1)
#### mmnt: Interval [ 1.000e+06, 4.000e+06 ]
#### mmnt: Values = [ 4.150e+00, 2.893e-01]

```

Obviously, the only remedy for these errors is adjusting the boundaries. In case of the f_T extraction as presented in this work the first condition value must be positive (lower frequency means higher β), the second one must be negative. If both conditional values are positive, the frequencies should be increased, in case of the opposite, they should be decreased. Note, performance is directly proportional to the narrowness of the boundaries.

A useful feature to speed up the configuration process is to adapt the start index value that should always be the index of the (latest) erroneous index. For example, indices 0 and 1 are processed successfully, but index 2 fails, the start index should be set to 2:

```
index = step(2, <Number of Steps n>, 1);
```

If the complete simulation is done, the start index should be reset to 0 and the maxCount argument should be set to a higher value, for example 150. Then, the simulation can be restarted again. Due to the narrow and tested boundaries it will be both fast, successful and as accurate as in the fourth argument of the stepping function given.

Conditional stepping has been extended by an automatic adaptation for the boundaries. Since it can be generally employed for several applications, for example threshold voltage extraction, the automatic adaptation is implemented as an input-deck-based feature (see Section B.4).

The example demonstrates the setup for the cut-off frequency extraction by conditional stepping (see also Appendix B.3).

```
index = step(0, 10, 1);
vbtbl = [0.74, 0.76, 0.78, 0.80, 0.82, 0.84, 0.86, 0.87, 0.88, 0.90, 0.92];
lower = [1, 32, 15, 12, 3, 5, 20, 29, 29, 0.5, 0.5];
upper = [2, 35, 45, 13, 4, 9, 40, 30, 30, 10, 10];
Vb = ~vbtbl[~index];
```

In order to step these three arrays simultaneously, it is necessary to introduce an index variable which is actually stepped. By using this index, the base voltage V_B is then obtained by subscripting the vbtbl array. The Solve section of the input-deck should look like this:

```
Solve
{
    ac = yes;

    +lb = ~lower[~index] * 1 Hz; // lower boundary
    +ub = ~upper[~index] * 1 Hz; // upper boundary

    frequency = stepCond(lb,
                          ub,
                          ~Curve.Response.beta - 1.0,
                          1e-2,
                          maxCount = 3);
}
```

The Curve section of the input-deck contains the calculation of variable beta:

```
Curve
{
    file = "sim_out_ft.crv";
    Response
    {
        +ic = abs(output("Device", "I", "C", ac = yes));
        +ib = abs(output("Device", "I", "B", ac = yes));
        +beta = ic/(ib + 1e-300A);
    }
}
```

It is important to note that the currents are zero while the simulator is initialized. To prevent a division by zero, a very small and thus negligible current is added to the numerator in the beta assignment.

A.8 Setup of the Resonant Circuit and the Band Rejection Filter

This section contains the setup of the parallel resonant circuit and the band rejection filter:

```
ResonantCircuit
{
    aux vR = 10 kOhm;
    aux f0 = 10 MHz;
    aux w0 = 2 * pi * f0;
    aux Q = stepTbl([0.5, 1.0, 2.0, 5.0, 10.0]);

    aux vL = vR / (w0 * Q);
    aux vC = Q / (w0 * vR);

    Iin : ~Devices.I { P = "out"; M = "gnd"; IO = 1 A; acIO = 1 A; }

    R : ~Devices.R { N1 = "out"; N2 = "gnd"; R = ^vR; }
    L : ~Devices.L { N1 = "out"; N2 = "gnd"; L = ^vL; }
    C : ~Devices.C { N1 = "out"; N2 = "gnd"; C = ^vC; }
}

BandRejection
{
    aux f0 = 50 Hz;
    aux vC = 100 uF;
    aux w0 = 2 * pi * f0;
    aux Q = stepTbl([0.5, 1.0, 2.0, 5.0, 10.0]);

    aux vL = 1 / (w0 * w0 * vC);
    aux vR = 1 / (Q * w0 * vC);

    Vdd : ~Devices.V { P = "Vin"; M = "gnd"; V0 = 1 V; acV0 = 1 V; }

    R : ~Devices.R { N1 = "Vin"; N2 = "out"; R = ^vR; }
    L : ~Devices.L { N1 = "out"; N2 = "ctr"; L = ^vL; }
    C : ~Devices.C { N1 = "ctr"; N2 = "gnd"; C = ^vC; }
}
```

Since both simulations use the same Solve and Curve section, the conditional inheritance is a good choice to improve the setup. This feature can be especially employed for including several similar simulations in one input-deck. Depending on the variable resonant (which could be, for example, bound to an environment variable), either section ResonantCircuit or section BandRejection is inherited by the Circuit section.

```
aux resonant = yes;
```

```
Circuit : ResonantCircuit ? ~resonant, BandRejection ? !~resonant;
```

Note, that the sources provide the following keywords:

- Parameter acIO for complex-valued current sources.
- Parameter acV0 for complex-valued voltage sources.
- Parameter acPort for port names of voltage sources required for the admittance matrix simulations.

A.9 Inquiring Complex-Valued Node Voltages and Terminal Quantities

This section shows how several mixed-mode AC quantities are inquired.

```
Curve
{
    file = if (~resonant, "res_out", "band_out");
    Response
    {
        +mhz = ~Solve.frequency / 1 MHz;    // will become dimensionless

        +outAc      = V("out", ac = yes);
        +outAbs     = abs(V("out", ac = yes));
        +outArg     = arg(V("out", ac = yes));

        +outAbsdB   = 20 * log10(value(outAbs));

        +acVc       = output("C", "acV");
        +acIr       = output("R", "acI");
    }
}
```

The V function has been extended to handle also complex-valued node voltages. In that case, the user has to set the optional parameter `ac` to `yes`. Several built-in functions of the input-deck are used for further calculations finally resulting in the logarithmic value. While the output function remained basically unchanged, it can be used to inquire complex-valued quantities from devices in a straightforward way.

The built-in input-deck functions `abs`, `arg`, `realpart`, `imagpart`, `log10`, `value` (to get the dimensionless value) etc. can be used to transform the actual simulator output to common output formats, such as dB.

A.10 Setup for the Extended Mixed-Mode AC Simulation

The example shows the `Circuit` section for the simulation of a heterojunction bipolar transistor:

```
Circuit
{
    Vbe : ~Devices.V { P = "Vbe"; M = "gnd"; V0 = ~B; acV0 = 1.0 V;
                      acPort = 1; }
    Vce : ~Devices.V { P = "Vce"; M = "gnd"; V0 = ~C; acV0 = 0.0 V;
                      acPort = 2; }
    HBT : ~MyDevices.HBT { Base      = "Vbe";
                           Emitter   = "gnd";
                           Collector = "Vce"; }
}
```

The active device is the heterojunction bipolar transistor. Its setup is found in the `MyDevices.HBT` section not shown here. The three terminals are connected with the `Vbe`, ground, and `Vce` node of the circuit, respectively.

Two voltage sources are defined between `Vbe` and ground as well as between `Vce` and ground. While the steady-state voltages refer to global input-deck variables `B` and `C`, respectively, the small-signal voltages

of 1.0 V and 0.0 V are directly set. According to the acPort configuration, both voltage sources are part of the admittance matrix calculation feature with the base voltage in the first and collector voltage in the second row.

A.11 Inquiring Circuit Quantities

This section contains the statements which inquire the circuit parameter quantities. Note the special-purpose device name `Circuit` as the first parameter of the function `outputParam`. It refers to a quantity of the complete circuit, and can therefore not be used as a device name any more.

Curve

```
{
    Response
    {
        aux B = "Vbe";
        aux C = "Vce";

        +Y11re = realpart(outputParam("Circuit", "YMatrix", B, B));
        +Y11im = imagpart(outputParam("Circuit", "YMatrix", B, B));
        +S21re = realpart(outputParam("Circuit", "SMatrix", C, B));
        +S21im = imagpart(outputParam("Circuit", "SMatrix", C, B));
        +C12   =          outputParam("Circuit", "CMatrix", B, C);
    }
}
```

The built-in functions `realpart` and `imagpart` are used to obtain the real or imaginary part of a complex-valued variable, respectively. The auxiliary variables `B` and `C` are used to shorten the parameter list of `outputParam`.

A.12 Selection of External Solvers

This section demonstrates the selection of external solver modules and their respective solvers. For example, the parallelized direct solver of the PARDISO module can be selected as follows:

```
Num
{
    externalSolver = yes;
    externalModule = "PARDISO";
    externalType   = "PARDISO_LU";
}
```

Analogously, the algebraic multi-level preconditioner in combination with BICGSTAB acceleration is activated by the following input-deck parameters:

```
Num
{
    externalSolver = yes;
    externalModule = "SAMG";
    externalType   = "AMG_BICGSTAB";
}
```

To employ an external solver module, the keyword `externalSolver` has to be set to `yes`. In this case, the settings of `directSolver` and `iterativeSolver` are ignored. Two strings are available to select the module and the solver type, respectively. The case-insensitive selection is checked by the simulator and rejected in case of invalid settings. In Table A.2, the valid selections are summarized.

<code>externalModule</code>	<code>externalSolver</code>
<code>Pardiso</code>	<code>Pardiso_LU</code>
	<code>Pardiso_CGS</code>
<code>Samg</code>	<code>AMG_BICGStab</code>
	<code>AMG_GMRES</code>
	<code>ILU_BICGStab</code>
	<code>ILU_GMRES</code>

Table A.2: Valid selections for external solver modules.

A.13 Netlist Definition of the Oscillator Example

This section contains all netlist definitions found in the input-decks for the oscillator example, see Section 6.4 (page 111).

The input-deck section `SubCircuits` contains the definition of all three subcircuits. The settings for the distributed devices are specified like in the single-mode [105]. In addition, MINIMOS-NT directly provides compact models of the commonly used circuit elements like capacitors and inductors.

Therefore, the respective `Devices` sections have to be simply inherited and their public members accordingly overwritten. In the definition of the resonant circuit both terminals are connected to the input, output, and ground nodes of the subcircuit.

```
SubCircuits
{
  Core
  {
    in  = "";
    out = "";

    V0 : ~Devices.V { P = "pin1"; M = "gnd"; V0 = 2.0 V; }

    T1 : ~NPN { C = ^out; B = "pin3"; E = "pin5"; }

    L : ~Devices.L { N1 = "pin1"; N2 = ^out; L = 1 uH; }
    R1 : ~Devices.R { N1 = "pin1"; N2 = "pin3"; R = 20e3; }
    R2 : ~Devices.R { N1 = "pin3"; N2 = "gnd"; R = 30e3; }

    Re : ~Devices.R { N1 = "pin5"; N2 = "gnd"; R = 1.25e3; }
    Ce : ~Devices.C { N1 = "pin5"; N2 = "gnd"; C = 1.00 nF; }

    C4 : ~Devices.C { N1 = ^in; N2 = "pin3"; C = 1.0 nF; }
  }
}
```

```

LC
{
    in = ""; // is assigned in circuit
    out = "";
    L1 : ~Devices.L { N1 = ^in; N2 = ^out; L = 1.000 nH; }
    C1a : ~Devices.C { N1 = ^in; N2 = "gnd"; C = 0.279 pF; }
    C1b : ~Devices.C { N1 = "gnd"; N2 = ^out; C = 2.790 pF; }
}
}

```

For the AC source subcircuit, the conditional inheritance feature of the input-deck is used. Since the transient and small-signal modes are subsequently employed, the respective voltage source has to be defined for a transient or small-signal result, respectively. This is controlled by the `Solve.transient` keyword:

```

SubCircuits
{
    Vtransient
    {
        VS : ~Devices.V { P = "pin10"; M = "gnd";
            V0 = 1 mV * sin(~Solve.time * 2*pi * 1e9 Hz); }
    }

    Vac
    {
        VS : ~Devices.V { P = "pin10"; M = "gnd";
            V0 = 0 mV; acV0 = 1 mV; }
    }

    Vsrc : Vac ? ~Solve.ac, Vtransient ? ~Solve.transient // conditional
    {
        +in = "";
        +R4 : ~Devices.R { N1 = ^in; N2 = "pin10"; R = 1e3; }
    }
}

```


Appendix B

The Stepping Module of Minimos-NT

Stepping is a fundamental infrastructure feature of any simulation tool, since it is important for both usability and performance considerations. For example during steady-state analysis several operating points are normally of interest to extract an I-V curve. On the one hand side the user can specify these points by variables in one input-deck rather than starting the simulator again and again. On the other hand side the solution of one operating point is normally a very good initial guess for the next one, resulting in a significant speed-up of the complete simulation.

This application of stepping variables is well-known and for reasons of consistency the definition of the time or frequency domain has been based on these concepts. This allows to employ the various powerful stepping functions also for transient and small-signal analysis.

Stepping is controlled via the `step` function family. The main purpose of the `step` functions is to introduce parameters for the simulation. There are several `step` functions available:

Function	Functional Parameters
<code>step</code>	start and stop value, delta, log, deltaMin, deltaMax
<code>stepN</code>	start and stop value, number of steps, log, deltaMin, deltaMax
<code>stepTbl</code>	table of values
<code>stepCrv</code>	curvefile, parameter, response, state
<code>stepSingle</code>	single value
<code>stepCond</code>	lower boundary, upper boundary, formula, error, maxCount

Table B.1: Stepping functions provided by MINIMOS-NT.

Common to all different types of stepping functions are the administration parameters `name`, `pri`, `slot`, `prev`, and `post`:

- `pri` denotes the priority of the parameter. For each combination of values of the stepped parameters a single simulation is run. The parameter with the highest priority runs fastest. Default priority is -1. Overruling of the priority of time and frequency is prohibited, because these variables always get the highest priority during stepping initialization. Furthermore, it is not allowed to combine their stepping functions with other input-deck functions.
- For consecutive stepping of different parameters, the optional arguments `slot`, `prev`, and `post` can be used. Only parameters with the same value of `slot` are stepped together, while other parameters are given the value defined in `prev` or `post`, depending on whether the current slot

comes *before* or *after* the value given in slot. Parameters without a value in slot are stepped in slot -1 as default. If prev and post are not specified, the first or last value of the stepping parameter is used, depending on the slot position.

Stepping control (see Appendix B.4) adjusts the standard delta within a range specified by deltaMin and deltaMax. If these optional arguments are not given, both variables are set to delta. In the case of user specified values they should fulfill $\text{deltaMin} < \text{delta} < \text{deltaMax}$. But this is not enforced by the simulator.

B.1 The Basic Stepping Functions

For simple additive or multiplicative (logarithmic) stepping the normal step function can be used. Here, start is the start value of the parameter, stop its final value, and delta the additive or multiplicative increment. The first three arguments must be of the same type or must be convertible to the same type. The type of the return value is determined by this conversion operation.

```
a = step(0, 2, 1); // returns an integer
b = step(0, 2, 1.0); // returns a real
c = step(0 V, 2 V, 1.0 V); // returns a quantity with unit V
d = step(0 V, 2, 1.0); // returns a quantity with unit V
```

In the case of quantity arguments the first three arguments must be of convertible units. For missing units the default unit (of the first parameter) is used, but it is recommended to provide units to all parameters. Basically, the resulting unit is always the equivalent SI-unit.

```
a = step(0 W, 2 "J/s", 1 eV); // quite a long loop, but OK
                                // returns "W" as unit
b = step(0 V, 2 A, 1.0); // mismatching units, error
```

The optional parameter log determines whether additive or multiplicative stepping is used with additive being the default.

```
Ni = step(10e3, 1e20, 1.2, log = yes);
```

The stepN function is similar to the step function, the only difference being the third argument. Here, it specifies the number of step points to be used.

```
Ni = stepN(10e3, 1e20, 100, log = yes); // use 100 points
```

The number of steps is used to calculate a stepping delta. Consequently, the stepN function is treated like a step function. This is particularly important as both functions can be attached to stepping control algorithm to modify the stepping delta during the simulation (see Appendix B.4).

B.2 The Special-Purpose Stepping Functions

The stepTbl function can be used to step through the values of an array. All values contained in the array will be subsequently assigned to the parameter. As for the additive stepping the type of the values given in the array must be of the same type, or must be convertible to the same type. The type of the return value is determined by this conversion operation.

```
B = stepTbl([ 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55, 0.6, 0.65, 0.7 ]);
```

To get stepping information from a curve file the `stepCrv` function is available.

```
G = stepCrv("init.crv", "G", "Vg", pri=4, slot=2, prev = 8 V, post = 5 V);
```

The simple stepping function `stepSingle` can be used to introduce a single value into the stepping configuration. Such a function is essentially required by a single frequency AC simulation, but can also be used by other variables in combination with useful values of arguments `slot`, `prev`, and `post`. This function must not be used for transient simulations (variable time).

```
frequency = stepSingle(100 MHz, slot=0); // single frequency ac simulation
```

B.3 Conditional Stepping

Function `stepCond` provides a conditional stepping based on a simple zero point approximation within two boundaries given by the first two parameters:

```
// Threshold Voltage Extraction
A = stepCond(0.7V, 0.8V, // boundaries
            output("Device", "I", "A") - 1e-7A, // comparison value
            1e-4, // error value
            pri = 2, maxCount = 20); // optional
```

The stepping is coupled to an internal control mechanism which determines the new stepping value depending on the evaluation of the condition. This implementation of this mechanism is based on the *False Position* or *Regula Falsi* method (see Section 3.3.3).

The condition is the combination of the third and fourth argument. Here, the comparison value in the third argument is coupled to an output current and has to be smaller than the error value of $1e-4$. If the condition cannot be fulfilled within `maxCount` steps (iterations), the approximation is terminated. If `maxCount` is not given, a default value of 1000 is used.

Sometimes it is more applicable to determine the threshold voltage by logarithmic stepping. For this purpose, the stepping function can be defined as follows:

```
// Threshold Voltage Extraction
A = stepCond(0.7V, 0.8V, // boundaries
            log(value(~Curve.Response.Id)) - log(1e-7), // comparison value
            1e-4, // error value
            pri = 2, maxCount = 20); // optional
```

Conditional stepping is particularly useful to determine the cut-off frequency f_T . The small-signal simulation mode (see Appendix A.1) requires the setup of the variable frequency. This variable can be subject of a conditional stepping:

```
// Cut-off Frequency Extraction
frequency = stepCond(10 GHz, 100 GHz, ~Curve.Response.beta - 1.0, 1e-05);
```

As shown in the following example, the `Curve.Response` section contains the calculation of `beta`, which equals 1.0 at the cut-off frequency. For that reason, the condition `beta - 1.0` should be fulfilled within an error of $1e-5$.

Curve

```
{
  +Response
  {
    ic  = abs(output("Device", "I", "CollectorContact", ac=yes));
    ib  = abs(output("Device", "I", "BaseContact",      ac=yes));
    beta = ic / if (ib == 0.0 A, 1e-20 A, ib);
  }
}
```

Conditional stepping obviously fails if the zero point cannot be found within the boundaries given as the first two arguments. By setting up a proper stepping control (see Appendix B.4), the boundaries can be automatically varied during stepping. The boundary adaptation allows to resolve the trade-off between usability and narrow boundaries. Narrow boundaries speed up the iterative stepping process significantly (see Section 3.3.3).

B.4 Stepping Control

Stepping control is a powerful feature for influencing the step increments/decrements while the simulation is already running:

- The stepping delta can be modified for the stepping functions `step` and `stepN`. This is useful to decrease the delta in numerically involved simulation phases, but to speed up simulation by increasing the delta in phases with small changes. Breakdown voltage and transient circuit simulations are prominent application examples for this kind of stepping control.
- The iteration boundaries for the conditional stepping function `stepCond` can be adapted. On the one hand, narrow boundaries significantly speed up the iteration, on the other hand they reduce the flexibility of conditional stepping. If the zero point cannot be found within the boundaries, the simulator fails after the first two steps. For that reason, stepping control can be employed to adapt the boundaries during simulation.

Basically, all stepping variables are subject to default stepping control schemes. These default schemes do literally nothing, so the deltas and boundaries remain constant. If one wants to apply non-trivial algorithms, the `SteppingControl` section has to be redefined. In the subsection `Scheme`, a section with the same name as the stepping variable must be added by inheriting the stepping algorithm:

```
SteppingControl
{
  Scheme
  {
    +time : SimpleTimestep;
  }
}
```

In this example, the stepping variable `time` (obviously of a transient simulation) is assigned to the `SimpleTimestep` stepping control algorithm.

The default files not only provide the default algorithms (ConstantStep and StepCondDefault), but also the following non-trivial ones:

- SimpleStep: A simple algorithm for decreasing and increasing the stepping delta.
- StepCondDirect: Adaptation of the iteration boundaries for directly proportional variables, for example the threshold voltage.
- StepCondIndirect: Adaptation of the iteration boundaries for indirectly proportional variables, for example the cut-off frequency.

If the default algorithms do not serve the purpose, one is encouraged to define individual schemes. For delta steppings, the respective section consists of two independent subsections with predefined names: Failure and Converged. The former is addressed if the simulation of the current stepping configuration failed, the latter is addressed if the solution could be successfully calculated.

The author of the scheme is almost completely free in drawing the right conclusion for the respective simulation. The only constraint is that the characteristic of the stepping function must not be changed. Due to the unmodifiable start and stop boundaries, an increasing stepping function must always be increasing, a decreasing function must always be decreasing.

In order to calculate a new stepping delta, MINIMOS-NT provides the following input to the Failure and Converged section:

Keyword	Type	Description
delta	Real	the current stepping delta
deltaOrig	Real	the original stepping delta as specified
deltaMin	Real	the minimum stepping delta as specified
deltaMax	Real	the maximum stepping delta as specified
deltaLog	Boolean	true if logarithmic stepping is activated
start	Real	the start value of the stepping
stop	Real	the stop value of the stepping
value	Real	the current value of the stepping

Table B.2: Information provided to delta stepping control algorithms.

MINIMOS-NT reads/expects two variables from the respective stepping control section:

Keyword	Type	Description
deltaNew	Real	the new stepping delta
print	String	information for the user

Table B.3: Information expected from delta stepping control algorithms.

The termination of the stepping is indicated by specific values for deltaNew: one for the logarithmic and zero for the linear case. The author of the stepping algorithm is responsible to adhere to the boundaries deltaMin and deltaMax of the respective stepping function. These boundaries are not enforced by MINIMOS-NT.

A good starting point for writing a new stepping algorithm is to create a new section which is inherited from `SchemeDefaults.ConstantStep`. This section will look something like this:

```
NewStepping : SchemeDefaults.ConstantStep
{
  Failure // => reduce the delta
  {
    aux d      = delta / 1.9;
    deltaNew = if(abs(d) < abs(deltaMin), if(deltaLog, 1.0, 0.0), d);

    aux printText = "Decrementing delta: " + delta + " -> " + deltaNew;
    print         = if(deltaNew == delta, "", printText);
  }

  Converged // => increase the delta
  {
    aux d = if (value == start, delta, delta * 2);
    deltaNew = if(abs(d) > abs(deltaMax), deltaMax, d);

    aux printText = "Increasing delta: " + delta + " -> " + deltaNew;
    print         = if(deltaNew == delta, "", printText);
  }
}
```

For the control of conditional steppings, a different kind of algorithm can be applied. MINIMOS-NT provides values for the following variables:

Keyword	Type	Description
valueLow	Real	the current lower stepping boundary
valueHigh	Real	the current higher stepping boundary
condLow	Real	the function value at the lower boundary
condHigh	Real	the function value at the higher boundary

Table B.4: Information provided to conditional stepping control algorithms.

MINIMOS-NT reads/expects three variables from the respective stepping control section:

valueLowNew	Real	the new lower stepping boundary
valueHighNew	Real	the new higher stepping boundary
print	String	information for the user

Table B.5: Information expected from conditional stepping control algorithms.

The stepping algorithm is always addressed in case `valueLow` and `valueHigh` have the same sign since then no zero point can be calculated in-between. This has different consequences for directly and indirectly proportional variables.

Thus, a typical algorithm looks like this:

```
StepCondDirect : StepCondDefault
{
  +checkSign = 1;
  valueLowNew = if (sign(condLow) == checkSign, valueLow / 1.5, valueHigh);
  valueHighNew = if (sign(condLow) == checkSign, valueLow, valueHigh * 1.5);
  print      = if (sign(condLow) == checkSign,
                  "Adaptation: bounds are too high => must be lower",
                  "Adaptation: bounds are too low => must be higher");
}
```

This algorithm is derived from the default one, inheriting all definitions of the input variables above. Depending on checkSign, the boundaries are either shifted up or down. Due to multiplication and division in this algorithm the sign of the boundaries is not changed. By using inheritance again, the indirect variant is very simple:

```
StepCondIndirect : StepCondDirect
{
  checkSign = -1;
}
```

B.5 Additional Stepping Control Keywords

In contrast to the function-level discussed above, stepping can also be controlled at the overall level. In the Solve section, two keywords are provided:

Keyword	Type	Description
breakSteppingLoop	Boolean	break the innermost stepping loop
evalSteppingLoop	Boolean	evaluate the stepping only, do not simulate

Table B.6: General stepping control keywords.

The keyword breakSteppingLoop contains a condition for completely terminating the innermost stepping loop. For example, the stepping can be terminated if the drain current exceeds 500 mA or if the cut-off frequency is already exceeded:

```
breakSteppingLoop = output("Device", "I", "DrainContact") > 5e-1 A;

frequency          = step(1 MHz, 6 GHz, 100 MHz);
breakSteppingLoop = !~Extern.firstStep && ~Curve.Response.beta < 1;
```

In order to evaluate the stepping functions only, the keyword evalSteppingLoop can be set. This flag allows a quick view on how the stepping functions work without starting the actual simulation. As there is no simulator information available, interactive features such as the conditional stepping cannot be tested.

Appendix C

Miscellaneous Projects

The following projects will be presented and discussed in the next sections:

- The MINIMOS-NT Post-Processing System: Appendix C.1
- The MINIMOS-NT Interactive Mode: Appendix C.2
- The Simulation Environment Interaction Library: see Appendix C.3
- The MINIMOS-NT Test Appendix C.4

C.1 The Minimos-NT Post-Processing System

Despite the fact that the contents of both output files of MINIMOS-NT can be configured by the user in a very flexible way, it is still necessary to transform or process these outputs after the actual simulation has been finished. In order to simplify the interface between MINIMOS-NT and such post-processing steps, a new post-processing system directly administered by MINIMOS-NT has been introduced.

Most of the post-processing is normally done in separate scripts, which read and parse the output files of MINIMOS-NT and start some additional calculations. For example, the purpose of this can be data conditioning for visualization purposes or the generation of input files for the next simulation tool in a simulation flow. To simplify the calling procedures and to make an additional wrap-around script obsolete, MINIMOS-NT is now able to execute arbitrary shell commands and to call external programs.

But MINIMOS-NT can also provide additional post-processing steps based on its own output data. A simple example is the `lastLine` post-processing steps, which stores the last line of each curve file block in a new curve file. This feature is especially useful for conditional steppings (see Appendix B.3), for example after the extraction of the f_T curve. Although the respective behavior could also be controlled directly in the curve output module depending on an input-deck keyword, it was decided to use this feature as a demonstration on how to implement MINIMOS-NT post-processing steps.

However, MINIMOS-NT was also equipped with a more sophisticated post-processing step, namely a *Fast Fourier Transformation* functionality. The implementation of the transformation itself is based on the code given in [165]. In addition, a general input-deck-based interface is provided for the user.

The user can specify an arbitrary number of post-processing steps, which are processed in a consecutive order. The respective input-deck section has the name `PostProcessing`.

It is basically structured as follows:

```
PostProcessing
{
    postProcOnly = no;      // Perform only post-processing steps
    printSteps   = no;      // Print available post-processing steps
    enter        = yes;     // Enable post-processing at all

    Steps
    {
        FirstStep
        {
            function = "";
        }
        SecondStep
        {
            FirstSubStep
            {
                function = "";
            }
            ...
        } ...
    }
}
```

The simulator iterates over all subsections found in section `PostProcessing.Steps`. The kind of post-processing step to execute is specified in the function string. Each block can be deactivated by setting `enter` to `no`. Since post-processing uses per definition already calculated data, the preceding simulation can be completely skipped in case the output files are already present (`postProcOnly`). In addition, the user can request a short help (`printSteps`) to see which post-processing functions are available at all.

As stated above, the `lastLine` step is a very simple one. Therefore it is used to sketch the usage and implementation idea. All available post-processing steps can be derived from default steps. In case of `lastLine` this predefined step looks as follows:

```
StepDefaults
{
    GeneralStep
    {
        enter    = yes;      // Perform function in this step?
        function = "";       // Function, such as "LastLine" or "FFT"
    }
    <GeneralIOStep> : GeneralStep
    {
        input    = "";       // Input file
        output   = "";       // Output file
    }
    LastLineStep : GeneralIOStep // Write last line of each block
    {
        function = "LastLine"; // to new curve file
    }
}
```

The necessary inputs are already derived from other sections, so a typical instance can be defined very quickly:

```
PostProcessing
{
    Steps
    {
        +LastLine : ^StepDefaults.LastLineStep.
        {
            input    = ~Curve.file;
            output    = "ft_" + ~Curve.file;
        }
    }
}
```

In this example, the simulation task is to start a conditional stepping in order to extract the f_T curve. The output of each step is written to the file specified in the Curve section. This file is the input file for the post-processing step, whereas the output file name is simply the same name with a preceding `ft_`.

C.2 The Minimos-NT Interactive Mode

MINIMOS-NT has been equipped with an interactive mode. Whereas the conventional operation mode is batch-oriented, the user is now able to influence the simulator settings while the simulation is still running. The simulator is started as usual, so an input-deck is read which contains the complete set of parameters the simulator requires for all kinds of simulations. Actually, there would be no additional user input necessary.

The basic motivation for an interactive mode is related to the various output and logging functions of MINIMOS-NT, which are normally annoying and thus deactivated. Unfortunately, these functions would be of great interest in case problems like non-converging *Newton* iteration steps occur. In former versions of MINIMOS-NT, it was necessary to restart the complete simulation process in order to have the respective logging activated, for example that of the update norms. This could be even more annoying in case of long steppings or very time consuming three-dimensional or mixed-mode simulations. So the idea of an interactive mode came into mind, which would allow to change the input-deck database during simulation.

A typical scenario is then as follows: MINIMOS-NT is started with an input-deck which specifies a three-dimensional simulation. The first steps, which take several hours, run very well, but then convergence problems are encountered. The user, who was glad about a concise simulator output up to now, normally wants then to see the update norms of all quantities, since this would help to identify the problematic region and/or carrier in the device. The user knows about the interactive mode, presses *CTRL-C* and after a while a prompt is offered.

Received signal, waiting to enter interactive mode ...

Interactive Mode (h for help)

~ >

~ > h

h	[command]	... print help message
c		... continue
q		... quit without saving anything
cs	[dir]	... change current section in input-deck
cd	[dir]	... change current directory in input-deck

```

ls      [dir] ... list given section, else current section
alias   command ... create an alias (alias x=y;)
saveAlias ... save all aliases in .mmi_alias
show    [item] ... show information on a variable/section
showComment [item] ... show information on a variable/section
                        with comment
parse   command ... parse ipd/ipl command

```

By issuing "h", the list as shown above is offered. The user can now go to the Log section of the input-deck and can set updateInfo to yes.

```

~ > cd Log
~Log > show

```

```

=====
Item:      ~Log
Type:      section
Name:      Log
Fullname:  ~Log
File:Line: /iuehome/wagner/vproject/mmnt/defaults/root.ipd: 251
Parent:    ~ (base section)
Ancestor:  LogDefaults
=====

```

Type	Name	Vartype	Value	Unit
variable	allTerminalQuantities	Boolean	true	
...				
variable	updateInfo	Boolean	false	
variable	writeAuxiliary	Boolean	false	
variable	writeComplex	Boolean	false	

```

=====

```

```

~Log >
~Log > parse updateInfo = yes;
~Log > show

```

```

=====
Item:      ~Log
...

```

variable	updateInfo	Boolean	true	
variable	writeAuxiliary	Boolean	false	
variable	writeComplex	Boolean	false	

```

=====

```

```

~Log >
~Log > c
Continuing ...

```

This version of the interactive mode offers that specific show capability which also takes all inheritances within the input-deck database into account. In addition, the parsing of complete IPL commands is provided. While these capabilities are deactivated by default, they are automatically enabled when a given number of steps has been exceeded.

The user is able to influence the complete parameter database. This does not make sense for the majority of parameters (for example model parameters), but it might be useful to inquire them on a read-only bases. Furthermore, the interface between the simulator and the input-deck database has to be improved that more keywords are inquired during the simulation process. A prominent example is the choice of the solver for linear equation systems.

C.3 SEILIB - The Simulation Environment Interaction Library

SEILIB – the Simulation Environment Interaction Library — is designed for handling parameterized input-decks and commands. In addition, several tools can be integrated to complete simulation flows based on these parameterized input files and optimization setups can be configured.

Values in text input-decks can be replaced by so-called arguments, which are generally encoded with preceding and succeeding brackets: `<{SEIARG}>`. The parameterized input-decks are regarded as templates or meta input-decks. They are used to create the actual input files for the simulators. The library provides several additional features, such as process administration, multi-threading, and host management.

C.3.1 Motivation

The library is dedicated to all users of an arbitrary number of arbitrary programs, scripts, or functions. Many similar simulations depending on an arbitrary number of input parameters shall be processed and (partly) repeated. It is therefore essential to keep the inputs and outputs consistent.

Furthermore, all available computational resources should be employed to reduce the execution time. The setup is extensible and several tools can be integrated to complete tool flows. In context of the GMRES(M) evaluation discussed in Section 5.5.3, the library successfully processed 23,000 processes on four IBM cluster nodes. It is important to note, that the use of arguments is not a must. So the host management can also be used for simulations based on conventional (non-parameterized) input-decks.

The library is written in PYTHON [167]. The implementation of the library itself can be found in the file `seilib.py`, which has to be accessible for the interpreter.

C.3.2 Getting Started

This section addresses the most important steps how to set up a new SEILIB application. Note that the optimization system is not covered here, see Appendix C.3.9 for this information.

A new PYTHON script normally starts with an execution directive for the operating system. As PYTHON is an interpreted language, the following line tells the loader which interpreter to use:

```
#!/usr/bin/env python
```

After importing the SEILIB library file, a new instance of the main class can be created, the name for it chosen here is `p`:

```
import seilib
p = seilib.seiclass()
```

At this point, the proper processing can be already tested. After setting execute permissions for the script (`chmod u+x name_of_script.py`), it can be directly started like other programs by typing on the command line: `./name_of_script.py`. Alternatively, the name of the script can be passed to the interpreter: `python name_of_script.py`.

Note that the `seilib.py` file must be accessible for the interpreter. If the library is stored in another directory, the search path for libraries can be extended in order to avoid import errors:

```
import sys
sys.path.append('`name_of_directory`')
```

If the script is properly executed, only the welcome message of the SEILIB is printed. Of course, the newly created instance can now be set up for the actual SEILIB application. The setup can be divided in two steps:

1. Define the tool or the tool flow.
2. Define the arguments and prepare the templates.

C.3.3 Basic Nomenclature and Definitions

In order to understand the functionality of the script, the definitions of *tool*, *command*, *host*, *argument*, *template*, and *auxiliary* have to be given:

Tool: A tool represents either an arbitrary program or PYTHON function. The command line for a call and its template input-decks must be specified. In order to be executed, a specific *command* is assigned. Since one program might be represented by several tools, each tool should be given a unique name, which is called *prefix*. It can be used to derive unique file names for all auxiliaries and output files. However, the uniqueness is neither checked nor forced.

Command: A command defines the way how a *tool* is executed. Depending on which of five methods is chosen, the program is either executed on the local host or on a remote host. PYTHON functions are always executed on the local host. In addition, a command can be restricted to a distinct set of *hosts*. This might be necessary for taking the memory consumption of the simulator or license restrictions of commercial products into account.

Argument: An argument is the general expression for either a *variable* or *constant*. The name of the argument, preceded and succeeded by brackets $\langle \{NAME\} \rangle$, can be arbitrarily used in *templates* and all kind of command line definitions. During processing, this name is replaced by the current value of the argument. Thus, an argument stands for a set of values, which are subsequently processed. These values are defined either in arrays or by PYTHON functions. In addition to user-defined arguments, the library provides so-called special-purpose arguments (see Table C.1).

Template: A template looks almost like the input-deck of the employed simulator. However, it contains some arguments instead of some values. For example, if the base voltage is normally defined by $VB = 1.0 \text{ V}$; , it is defined as $VB = \langle \{VB\} \rangle \text{ V}$; . The templates are used to generate actual input-decks for the simulators according to the values of its arguments.

Auxiliary: An auxiliary is an actual input-deck which was created based on a template. Its number and name can be addressed by special-purpose arguments provided by the library.

Host: A host generally represents a computer. The user specifies, how many processes may be simultaneously running and which *nice* level is set. An additional parameter controls the number of subsequently started processes on that host. A host can have an arbitrary number of *host-specific arguments*.

The following examples give a first glance on how the script is configured:

```
p = seilib.seiclass()
p.hostman.newHost("tcad01", 0, 8, 1)
ecmd = seilib.seicmdexec("<{CMD}> > <{LOG}>")

p.newTool("mmnt", # name of the tool
         1, 1, # mode and number of sub-modes
         ecmd, # command
         "short", # prefix
         "<{AUX1}>", # command line options
         ["seimos.ipd" ]) # array with template files

p.newTool(merge, 12, 1, 0, "test", "", [ ])
```

The instance of the main class `seiclass` is `p`. A new host, *tcad01*, is registered at the host management system of `p`. The *nice* level for all processes on that host will be zero, eight processes may run simultaneously, and the *once* parameter is set to one (see below).

Next, a command is created, which is here an instance of the simple execution class `seicmdexec`. One purpose of this class is to compose the command line of the actual tool execution depending on the execution method. The user is absolutely free in defining these commands. In the example, the command line consists of `<{CMD}>` and the pipe of standard output to `<{LOG}>`. The special-purpose argument `<{CMD}>` contains the name of the tool and its command line options as specified in the parameter of the `newTool` call.

Finally, the tools are specified which is here a call of MINIMOS-NT (mode one, one sub-mode, prefix "*short*") with one input-deck only. In the command line, this input file is represented by the special-purpose argument `<{AUX1}>`. The template is stored in array, that could hold an arbitrary number of templates which are consecutively numbered. Note that all automatically generated numbers start with one, whereas PYTHON counting starts with zero.

More than one template is particularly useful if the include statement of the input-deck is used in order to construct a hierarchy of input-decks (as an alternative to conditional inheritance). For example, the physical definition is stored in a base input-deck, which is then included in various application input-decks like for simulating *Gummel* plots and extracting S-parameters. As all of these input-decks may contain SEILIB arguments, the library therefore supports an arbitrary number of templates. Note that template names may be arguments themselves.

The second tool is a PYTHON function (not shown here) and does therefore not require a command. It is supposed to combine all MINIMOS-NT output curve files to one. It belongs to mode twelve.

There are five methods for executing processes:

- Normal execution (`seicmdexec`): A process is simply started on the local host and the script waits until it returns. This method should be used for very short programs or scripts.
- SSH process spawning (`seicmdssh`): The secure shell (SSH) is used to access a remote host. In addition, a new process is spawned and the script does not wait for its return.
- SSH process spawning and waiting (`seicmdsshwait`): Same as the method before, but the script waits for all processes after the last process of the current mode has been spawned. This is necessary for synchronization purposes. For example, a merge function must wait for all output curve files of preceding simulations.

- local process spawning (`seicmdspawn`): Same as `seicmdssh`, but the local host is used. If the name of the local host is found in the list of hosts, the settings regarding the number of processes and *nice* level are considered.
- local process spawning and waiting (`seicmdspawnwait`): Same as the method before including the waiting as described for `seicmdsshwait`.

The library always waits for all spawned processes before it terminates.

C.3.4 The Argument System

What happens to the template and its arguments? This brings the argument system into play which is going to be discussed in this section. The template input-deck `seimos.ipd` contains the following four arguments:

```
aux infile    = "<{CWDIR}>/." + if(getenv("HOSTTYPE") == "i386", "mos",
                                "mosibm");
aux outfile   = "<{DEVOUT}>.";
aux crvfile   = "<{CRV}>.";
aux numSteps  = <{SN}>;
```

As stated above, arguments are addressed by their names extended by preceding and succeeding brackets. The constant `<{CWDIR}>` is always provided by the library and stands for the current working directory. The names of the input device is added to this directory, but depends on the host type due to the binary ordering. The name of the device output and curve files (`<{DEVOUT}>` and `<{CRV}>`) depend on the actual state of the script processing, otherwise they would be overwritten all the time. The only input parameter is the number of steps `<{SN}>`, which is referenced by a stepping function. This example is for demonstration purposes only – the number of scattering events in a Monte Carlo simulator might be a better one.

Note that in the parameterized input-deck the double quotes must still be used for all represented strings such as the file names. But there are no double quotes for `<{SN}>`, which actually stands for an integer. Basically, the script processes strings only, and all non-string values specified as arguments are automatically converted to strings. However, this is no restriction for system handling text-based input-decks.

The last point is how the arguments and their values are registered. This should be demonstrated for the example:

```
p.newVariable("SN",      getSN)

p.newConstant("NAME",    "Test")

p.newConstant("AUXDIR",  "<{CWDIR}>/aux/",      "dir")
p.newConstant("LOGDIR",  "<{CWDIR}>/log/",      "dir")
p.newConstant("OUTDIR",  "<{CWDIR}>/out/",      "dir")

p.newConstant("ALLDAT",  "<{NAME}>_<{FLATB}>_<{PREFIX}>")

p.newConstant("AUX",     "<{AUXDIR}>/<{ALLDAT}>_<{AUXNR}>_<{AUXNAME}>")

p.newConstant("LOG",     "<{LOGDIR}>/<{ALLDAT}>.log",      "log")
p.newConstant("CRV",     "<{OUTDIR}>/<{NAME}>_<{FLATB}>.crv")
p.newConstant("DEVOUT",  "<{OUTDIR}>/<{ALLDAT}>_out.pbf")
```

Although this does not look like the ultimate selling point at the first glance, there are quite many arguments for such a small parameterized simulation. Obviously it would have been possible to skip some of them. But this example is also intended to be a template for further applications where the underlying concepts might be useful.

Starting with the variable *SN*, its values are coming from a PYTHON function, namely *getSN*. The function itself is shown below. Then, nine constants are defined: a name, three directories, one constant being used as intermediary, the name convention for auxiliary files, and finally the three constants which were used in the template input-deck. The following statements characterize the argument system:

- All values may contain arguments itself – they are subsequently refined, but there is a maximum depth of 20 to avoid endless loops.
- All arguments representing directories should have the third optional parameter *dir*. The library can then create the directories in case they do not exist.
- Constants are constant in terms of their definition only and therefore they are always strings. They can indeed stand for a constant value as seen for *<{NAME}>*, but they can also represent different values as used for *<{CRV}>*. The definition of a curve file is constant, but it contains variables so that the actual value changes.
- There are two ways for defining values for variables: either an array or a PYTHON function. By choosing the latter option, the user can take advantage of the complete PYTHON language and libraries.
- Similar to the directories, all log files should have the optional qualifier *log*. By giving the optional qualifier, the library is able to detect which log file was actually used for a process. This is particularly convenient in heterogenous network environments if the output is piped to log files stored at different locations. Such a setup is easy with host-specific arguments.
- The way of setting up the arguments is often ambiguous.

Note the different definition of *<{CRV}>*. In contrast to the other constants, it does not include the unique tool prefix. While setting up tool flows, the output of one tool is normally used as input for another tool. In the example, the MINIMOS-NT output curve files are the inputs for the merge function which is supposed to combine them. This can be easily achieved if the definition does not depend on the tool prefix.

The auxiliary files are created after all arguments have got their respective values. The stream editor is used to transform the templates to auxiliary files. The user can define an argument *<{AUX}>*, which contains the template for a name of an auxiliary file (see the example above). If such a variable is missing, a unique name is generated by combining *<{FLATB}>*, *<{PREFIX}>*, the number of the auxiliary and its name. The special-purpose arguments *<{FLAT}>* and *<{FLATB}>* contain the current meta-level and combination (for example *M1-C1*), and can be extended by a certain number of variable values (cf. *maxVarDepth*). A list of all special-purpose arguments is given in Table C.1.

C.3.5 Process Management System

The library is basically process-oriented and so the complete implementation is based on this idea. Therefore, it is useful if the user keeps the idea of creating processes in mind, which are finally executed on one host. The underlying simulations differ from each other by an arbitrary number of argument values. How many processes are potentially started?

Name	Value
AUX	Template for the name of an auxiliary file (not generated if existing)
AUXNAME	Name of the currently processed template file (not generally available)
AUXNR	Number of the currently processed template file (not generally available)
AUXx	Name of the x-th auxiliary file
CMD	The command line of the tool and its options
COMBNUM	The current combination number
CWDIR	The current working directory during initialization
FLAT	Current meta-level and combination (extendible)
FLATB	Like <{FLAT}>, but all points are replaced by underscores
HOST	The name of the selected host (is used for the ssh command)
HUNAME	The <i>uname</i> string of the selected host
HVARx	The x-th host-dependent variable
METANUM	The current meta-level number
MODE	The current mode
NICE	The nice level of the selected host
PREFIX	The prefix of the current tool
PROGRAM	The name of the current tool
SUBMODE	The current sub-mode

Table C.1: List of all *special-purpose arguments*, which are automatically generated by the library.

This depends on the user configuration, which can be completed now:

```
def getSN(name, currProcess): return currProcess.procComb * 10

p.numCombinations = 7
```

The function `getSN` was already referenced by argument <{SN}> and should return the number of steps. This number should obviously depend on the processing state. In the example, the number of steps is the tenfold value of the current combination number. Note that the integer value is automatically converted to a string. Furthermore, it should be again emphasized that functions like `getSN` are native PYTHON code without any restrictions coming from the library. The library provides the following information in the function parameters: as first parameter the name of the argument the library is requesting a value, since one function could be used for several variables. The second parameter is the current process (the name of the parameter is up to the user: here `currProcess` was chosen) and an instance of the class `seiprocess`, which stores all information on this specific process.

But there is also a second way for defining values of variables: the array. The library automatically uses the combination number as index for the array. If the array is too small, the combination number modulo the array length is taken. The alternative configuration for <{SN}> could simply look as follows:

```
p.newVariable("SN", [ 10, 20, 30, 40, 50, 60, 70 ])
```

Four members of the process class are essentially defining the state of processing:

Combination: A combination is a set of values of all variables. Each variable gets one particular value in each combination. Thus, all combinations create one axis of a two-dimensional parameter space. The class member for the number of combinations is *numCombinations*, which is seven for the example above. Thus, there are seven states on this axis.

Meta-Level: This level creates the second axis of the two-dimensional parameter space. All combinations are repeated a specific number of times. The default value for the corresponding class member `metaMultiplier` is one, which is not modified in the example above. Note that further dimensions are of course possible, but it was decided to provide them on a multiplicative basis. The user is so responsible to derive the dimension by analyzing the meta-level.

Mode: As stated above, each tool is assigned to a mode. Thus, all tools create a set of active modes. Note that a tool can be disabled by specifying a negative mode.

Sub-Mode: In addition, each tool can have several sub-modes. This means, that each tool can be subsequently processed more than one time.

Since this looks a bit complicated, it is inevitable to note that the library can only be as powerful as its state system. Since the user already specified a set of tools which have to be additionally taken into account, the hierarchy is given by:

1. Loop over all active modes.
2. Loop over all meta-levels.
3. Loop over all combinations.
4. Loop over all tools of the current mode.
5. Loop over all sub-modes.

Inside the fifth loop, an instance of the process class is allocated, a host is selected, all arguments get their values, the auxiliary files are created, and the tool is finally started.

The data stored in the process class is not always complete. There are three levels:

1. Value requesting level: the state information and the real constants are available only.
2. Pre-execution control: besides the state information, the values of all arguments are retrievable by using `getValue`.
3. Post-execution control: all data is available

The library offers a flexible execution control. Before a tool is really executed, the function assigned to the `doExecute` class member is called. It takes the current process as the only parameter. If this function returns zero, the process is not executed. Note that the values of all arguments and the complete state information is available. The user can of course use the complete PYTHON syntax to write this function. The post-execution function `afterExecute` works in a similar way. It takes the complete process information (including the exit code) as only parameter. If it returns zero, the library processing will be terminated as soon as possible. Examples for these two functions are given now:

```
def doExecute(currProcess):
    if currProcess.procMode != 12: return 0
    return 1

def afterExecute(currProcess):
    if currProcess.procExit != 0 and not currProcess.procIsCF:
        print("log file: " + currProcess.procLog)
    return 1
```

The function `doExecute` allows only tools of mode twelve to be executed. The function `afterExecute` prints the log file of the process in the case of a non-zero exit code, but only if the process represents a program and not a callable function.

C.3.6 The Host Management System

The host management system offers the possibility to distribute the load over several CPUs and/or over a network (see Section 5.3.1). The user can specify an arbitrary number of hosts which are subsequently used to execute processes. The system retrieves information about the load average of each host, which can be used to select or reject hosts. By using the *once* parameter of the host, the user can influence the way how hosts are selected. This should be clarified by an example. It is supposed that more than eight processes should be executed, and four hosts (*tcad01* – *tcad04*) are registered allowing two processes at the same time. Depending on the parameter *once*, the following host selection sequences are possible:

```
once = 1: tcad01 => tcad02 => tcad03 => tcad04 =>
          tcad01 => tcad02 => tcad03 => tcad04 => waiting...

once = 2: tcad01 => tcad01 => tcad02 => tcad02 =>
          tcad03 => tcad03 => tcad04 => tcad04 => waiting...
```

Hence the parameter *once* can be used to prioritize some hosts, for example to keep the first processes local or to fully employ the local resources before remote ones are allocated.

Two classes are responsible for the host management: whereas the *seihost* class represents one particular host and stores all information about it, the *hostManagement* class administers the list of all hosts and organizes the communication to them.

As already mentioned above, some *commands* can be restricted to particular hosts. The following example should demonstrate this:

```
tcad01 = p.hostman.newHost("tcad01", 0, 1, 1, [ "powerpc" ])
tcad02 = p.hostman.newHost("tcad02", 0, 2, 2, [ "powerpc" ])
tcad03 = p.hostman.newHost("tcad03", 0, 2, 1, [ "powerpc" ])
tcad04 = p.hostman.newHost("tcad04", 0, 2, 2, [ "powerpc" ])

scmd1 = seilib.seicmdssh("source <{CWDIR}>/seisetup;
                        nice -n <{NICE}> <{CMD}> > <{LOG}>",
                        [ tcad01, tcad04 ])
```

Four hosts are registered, but the command *scmd1* is restricted to *tcad01* and *tcad04*. Note that in the restriction array the return values of *newHost* are used. If there is no host for such a restricted process available, it is simply postponed. Thus, while the process is waiting for an appropriate host, other unrestricted processes can still be executed on hosts *tcad02* and *tcad03*. However, postponed processes have priority over new processes for the hosts they are waiting for.

Since a new shell is opened for all spawned processes, the user has to ensure the setting of the environment. Therefore, a setup script can be sourced as shown in the example above. In addition, the user is responsible for taking the *nice* level into account. The library offers a wide variety of possibilities in the parameter space, but the user is totally free in applying them.

C.3.7 Class Diagram

The diagram of the major SEILIB classes is depicted in Figure C.1. In the center, the main class *seiclass* is shown. This class combines own functionalities and those of other classes in order to provide the complete set of features. Therefore, it is the base class for all SEILIB applications.

In addition, an auxiliary class provides basic functions such as the administration of command line arguments, exit codes, output functions etc. The name of this class is *Auxiliary* and a global instance *seilib.auxiliary* is publicly available.

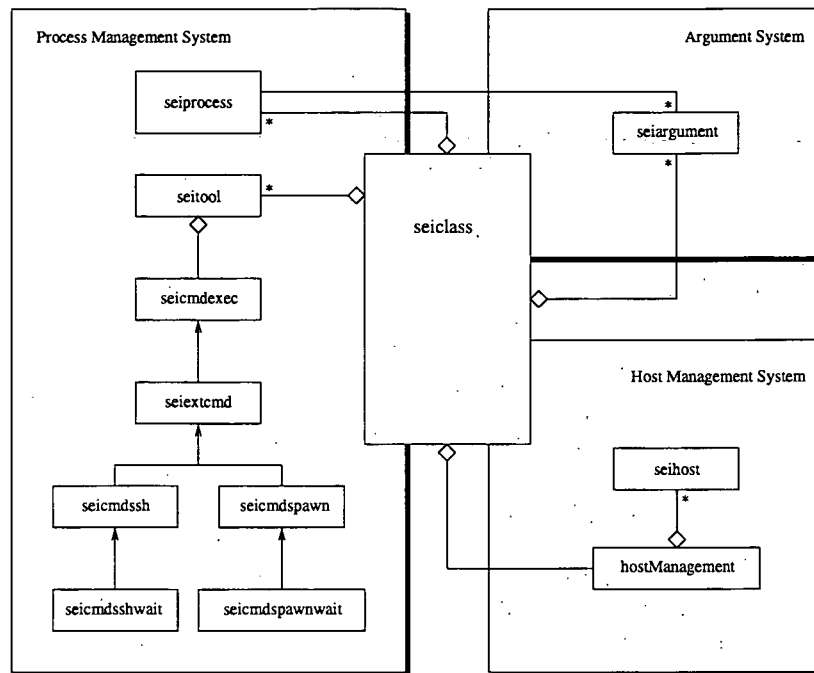


Figure C.1: SEILIB class diagram.

C.3.8 Example Application: The Minimos-NT Test

The formerly single-threaded MINIMOS-NT test system was parallelized mainly for two reasons:

1. New tests have been added to the test suite so that the execution time on a single CPU computer exceeds user-friendly limits. Such a situation could result in skipped tests and therefore less quality and effectiveness of the test.
2. With the IBM cluster, new hardware resources are available which can be employed for spawning multiple processes.

So the platform of the complete test system was changed from Linux to AIX, which did not have any consequences for the PYTHON code since the interpreter is also available for AIX. Furthermore, the new platform has additional advantages regarding the numerical representation (see Appendix C.4.1).

Instead of parallelizing the existing test system, a new test system has been developed as a SEILIB application. Since both systems are based on the same ideas, the adaptation of the old tests was very simple. Due to extended features of SEILIB, new tests can be written in a much more flexible and easier way.

In Figure C.2, the class diagram of the new MINIMOS-NT test system is shown. The main class `Example` is derived from `seiclass` and is responsible for processing one test. The process and host management systems are directly available, the argument system is processing the settings found in the test script.

There is also one test which does not fit entirely in the concept. This test is intended to evaluate all MINIMOS-NT examples as found in a `exa` directory. So the specialized class `ExaExample` was written in order to process all input-decks found in this directory and to provide all features known from the other tests.

The new test system is not only able to parallelize the execution of one test, but also to parallelize the execution of all test scripts found in the test directory. This functionality is regarded as a test itself, and

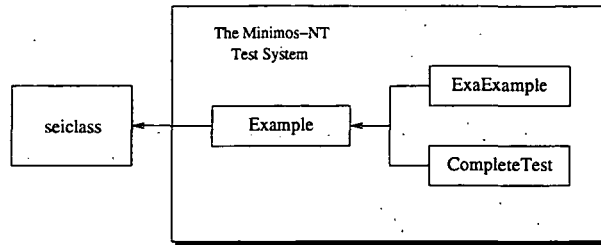


Figure C.2: Class diagram of the MINIMOS-NT test system based on the SEILIB library.

therefore a specialized class `CompleteTest` derived from `Example` was implemented. At this point of the test system a cascaded application of the SEILIB can be seen, which is an important aspect regarding the flexibility and stability of the library.

C.3.9 The Optimization System

Another proof of concept can be conducted based on the next considerations. If the set of the implemented SEILIB subsystems is analyzed, interesting similarities to the SIESTA framework [232] can be found. Basically, SEILIB is able to process template input-decks, employs a process and host management system, and allows to define tool flows consisting of an arbitrary number of arbitrary tools.

In contrast to the SIESTA system, SEILIB was designed to process a fixed number of combinations in a pre-defined and deterministic way. However, this should be no obstacle for providing an optimization feature by implementing a new optimizer class derived from `seiclass`. The optimization system is a highly-specialized SEILIB application which can be employed as a full alternative for SIESTA optimization models.

The description of the optimizer system starts with some details on the technical implementation: each supported optimizer provides an interface program to the actual optimizer system. The optimizers themselves and their interfaces are not strictly speaking part of the SIESTA framework. SIESTA starts the interface program as a separate process and communicates with this process by writing to standard input and reading from standard output. This part can be easily reproduced in PYTHON.

Each optimization model includes one or more so-called free parameters which are varied by the optimizer in order to find a minimum of a given scalar (cost or score function) or vector (cf. *Levenberg-Marquart* optimizer) target. The free parameters are defined by specifying a default, minimum, and maximum value. The variation of the free parameters depends on the results of the respective simulations. So during the optimization process, the interface class of the chosen optimizer requests one or more variable combinations per step. For each of these combinations, the defined tool flow must be processed after the values of the free parameters have been replaced in the template input-decks. The respective results are collected and written to standard output.

In the SEILIB context, the free parameters are arguments with boundaries. Furthermore, in the core results were not of interest at all up to now. For that reason, also slight extensions of the core modules were necessary and implemented. However, each request of the interface class represents a new setup of a pre-defined number of combinations. Thus, the concept fits perfectly in the already existing system and very few adaptations were necessary for providing the optimization setups with all features known from SEILIB and SIESTA. In fact, the largest part of the optimization system deals with preparing the settings for the optimizers.

SEILIB-based optimizations are based on one of four optimizer classes, which are derived from a base optimizer class (see the class diagram in Figure C.3).

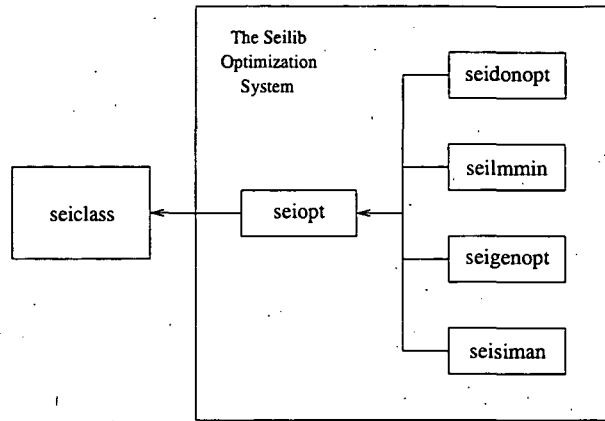


Figure C.3: Class diagram of the SEILIB optimization system.

The following four systems are currently supported:

1. Levenberg-Marquardt Algorithm (LMMIN) by Moré et al., 1980
2. Donlp2 (DONOPT) by Peter Spellucci, 1995
3. Adaptive Simulated Annealing (ASA, SIMAN) by Lester Ingber, 1993-2000 [103]
4. A C++ Library of Genetic Algorithm Components (GALIB) by Matthew Wall, MIT [234], 1995-1999

Note that these optimizer systems are not part of the SEILIB code, but merely coupled to SEILIB. Thus, although their code is redistributed and provided, the license agreements of the respective code or package (see more information in [232]) have to be considered.

In the following, a short example should be discussed, which employs the DONOPT optimizer:

```
p = seiopt.seidonopt()
```

Besides of normal arguments, free parameters are defined, for example:

```
p.newVariable("a", p.getVariable, [ 1.0, -10.0, 10.0 ] )
p.newVariable("b", p.getVariable, [ 0.0, -10.0, 10.0 ] )
p.newVariable("c", p.getVariable, [ 0.0, -10.0, 10.0 ] )
```

Free parameters are variables and get their values from the optimizer. The name of the method is `getVariable` (`getValue` is already used in the argument system). The optional third argument was used for *dir* and *log* before, now it is used to specify the default, minimum, and maximum value of the variable in an array.

The most critical point of the optimizer model is the feedback path of the result. The tool flow is processed depending on the input values requested by the optimizer. In order for the results to be written back to the optimizer, the user is responsible for

1. collecting the results from the correct output files,
2. passing the either scalar or vector-valued result to the `setResult` method of the current process, and
3. finally calling the optimizer method `writeResult`.

A frequently used place for doing this is the `afterExecute` method. If more than one tool is involved, the user must furthermore ensure that this is done for the last tool in the tool flow only.

For example:

```
currProcess.setResult(scalarResult)
p.writeResult(currProcess)
```

Now the definition of the optimization model is done – the only missing part is the configuration of the optimizer. Whereas DONOPT and LMMIN provide configuration methods, the settings of the GENOPT and SIMAN optimizers have to be done directly by accessing the public members of the classes.

The advantages of the SEILIB over the SIESTA system are:

- Higher flexibility while defining tool flows.
- Increased performance of the script processing.
- Scripts can be combined with the model and optimizer setup.
- More user interaction and flexibility during execution.
- Simpler extraction of results from output files.
- Complete system is based on PYTHON syntax and library.

The disadvantages are:

- The optimization model is less strictly defined.
- No compatibility to SIESTA models.
- Not directly included in further developments of the SIESTA framework.

C.4 The Minimos-NT Test

MINIMOS-NT is always under development since there are always new ideas for additional features. However, in order to preserve the already achieved efforts, their functionalities have to be tested on a regular basis.

The MINIMOS-NT test (*mmnttest*) is a feature-based test: a given set of simulations is run and the produced output files are compared with references. In case differences are detected, the reason for them should be found out. The test is based on the following ideas:

- Before new features enter the CVS repository of MINIMOS-NT, their independence from already tested features has to be proven.
- As new features enter the CVS repository of MINIMOS-NT, appropriate tests have to be constructed and added to *mmnttest*.

C.4.1 Numerics

Unfortunately, comparing output files is a cumbersome task. The main reason for difficulties is the limited representation of floating point numbers. The output files heavily depend on

- the compiler,
- the compile mode,

- the machine(s) used for compiling and linking, and
- the machine used for running the tests.

Therefore, the *mmnttest* references have to be created multiple times to cover the range of development environments used at the institute.

Note that all *vprojects* have to be compiled with the same mode in order to ensure consistent numerical representations. In addition, compiling and linking should be restricted to one machine only, even if the full paths are specified in the configuration file.

The problem of the numerical representation has been partly resolved by changing the platform to AIX. This has one particular reason: The main development platform of the institute is Linux. So here a variety of compilers and compilation modes can be found, because each developer has personal preferences which are widely accepted and supported due to the academic context. But once the development is ready to be ported to AIX, all members of the development team use the same compiler and (almost) the same compiler configurations. Since AIX is not the major development platform, it is also easier to recommend respective guidelines.

Since the test can be run on more than one platform, also inter-platform comparisons can be made. These analyses can detect implementation errors which become obvious by running a binary compiled and linked by another compiler. As the SEILIB test system conveniently allows to repeatedly access the results files, several checks can be additionally performed during a post-processing of a test run. These checks may also include comparisons with references generated for other platforms.

C.4.2 Nomenclature and Definitions

As stated above, *mmnttest* is a feature-based test:

- A *feature* is a single MINIMOS-NT functionality, in most cases as described in the documentation.
- A *test* combines a set of related MINIMOS-NT features, for example stepping functions or optical model functionalities.

A test consists of general settings such as the name or id as well as of different models.

- A *name* is the name of the template input-deck.
- An *id* is a unique three-character abbreviation of the test.
- *devices* is a list of one or more input pif files.
- *schemes* is a list of one or more iteration schemes.
- *args* is a list of variables equally used for all models.
- *models* is a list of one or more model combinations.

For *all* devices and *all* iteration schemes *all* model combinations are performed. The small optical test is given as example:

```
import    seitestlib
example = seitestlib.Example()

example.newConstant("NAME",      "Optical")
example.newConstant("NAMEID",    "OPT")
example.newConstant("DEVICE",    "Diode-PIN")
```



```

example.newConstant("SCHEME", "DD")
example.newConstant("LVL", 1)

example.numCombinations = 2

example.newVariable("ID", [ "01", "02" ])
example.newVariable("J0", [ "1e12", "1e15" ])
example.newVariable("alpha", [ "0.5", "0.7" ])

```

C.4.3 Test Levels

In contrast to former *mmnttest* versions, a level system was introduced. A level is defined by the argument `<{LVL}>`. Generally, it was decided to provide nine levels, which are defined as follows:

Level	Execution Time	Started
1	< 30s	always
2	< 60s	always
3	< 120s	always
4	>= 120s	always
5	< 30s	not always
6	< 60s	not always
7	< 120s	not always
8	>= 120s	not always
9	deactivated	never

Table C.2: Definition of the test levels.

The ninth level is reserved for deactivated models, for example unsuccessful simulations. The other two blocks are divided into four categories each based on the execution time. This time refers to a specific computer, compiler, and mode. At the moment there are over 800 tests available, thus, a complete run takes several days on a single computer. This would be obviously very inconvenient for fast tests during development. In addition, not all tests have to be run really every time. Despite some models are maybe still important in context of full tests, they can be skipped in order to speed-up pre-check-in tests. So before changes may be committed to the CVS repository, a full level 1–4 test should be performed. Level 5 to 8 are additionally tested by automated test runs, which are started independently of feature check-ins.

The `--level` option of the test script requires a range or level numbers, all separated by commas, for example:

- `--level 1`: selects level 1.
- `--level 1-4`: selects level 1,2,3,4.
- `--level 1,3-5,7,8`: selects level 1,3,4,5,7,8

C.4.4 File Structure

Core of the test is the main library `seitestlib.py`, which provides two main classes: `Example` and `CompleteTest`. The former is derived from `seiclass`, the latter from `Example` (see Figure C.2). The new test library is therefore a full SEILIB application. This library is imported and used by each test script file. To run all tests automatically, the driver script `mmnttest.py` can be used.

Each test consists of several files:

- the test script file in `mmnttest` which contains all settings and model combinations,
- the test template file which is a parameterized input-deck stored in `mmnttest/tpl.ipd`,
- the test device file stored in `mmnttest/in.pif`,
- maybe test data files which are stored in `mmnttest/data` (see the stepping test for an example).

The following directories, which may further depend on the host platform, are created by the library if they do not exist:

- `in.ipd`: the actual input-decks. Their creation is based on the template in `tpl.ipd`.
- `in.pbf`: the input-pbf files, which are the converted pifs from `in.pif`.
- `out.crv`: the output curve files of the tests.
- `out.log`: the log files contain the MINIMOS-NT output of the respective test.
- `out.pbf`: the output pbf files of the tests.
- `out.pif`: the output pif files of the tests.

Obviously, there are connections between the general settings and filenames:

- the name specifies the template input-deck in the directory `tpl.ipd`,
for example `tpl.ipd/Optical.ipd.tpl`
- one device is specifying the input pif file in the directory `in.pif`,
for example `tpl.ipd/Diode-PIN.pif`

In the former versions, the input device names were connected with the test name, which is actually not necessary. Furthermore, this connection avoids a quite useful sorting by device classes such as diodes. In the new system, the device name must start with a class and continue with an optional description.

Since the test system combines input information to filenames, several conventions for naming test files must be adhered to. Devices start with a general device description, for example *Diode*, the first letter is capitalized. To further specify the device, for example the material *Si-Si*, an appropriate string is concatenated, separated by a hyphen. So it is easy to see all already existing devices of a specific category and to pick one for a new test if appropriate. Of course, the extension has to be `pif`.

Templates start with a capital letter, the name describes the test. The extension has to be `ipd.tpl`. Note that the name of the test is connected with the general setting name in the test script.

Test scripts start with a specific prefix, which is *mmnt-* for all tests of MINIMOS-NT, and a specific name of the test script. It is recommended to use the same name as for the template. The extension is `.py`.

As already shown above, the test script `test-optical.py` contains:

```
example.newConstant("NAME", "Optical") # connected with the template
example.newConstant("DEVICE", "Diode-PIN") # connected with the device
```

C.4.5 Model Identification System

A system was defined to create unique file names for the actual input and output files. To identify a test, two strings are used:

1. The `<{NAMEID}>` of the test itself, which is now a three-character abbreviation, for example *IMD*.
2. The `<{ID}>` of the model, which is given as first argument of a model

Why is `<{ID}>` a separate argument and not automatically generated by taking the combination numbers into account? First, related tests can so have a relationship expressed by the combination number. Second, skipped or added tests do not destroy already generated references (note that the combination number is part of the filenames). Based on the idea of the former versions, the device name and the iteration scheme are added to the filename.

C.4.6 The References

References are created by specifying the `--genref` option together with a new reference directory: `--refdir x`. The directory `x` should be an existing directory with a descriptive name. This directory is not automatically created if it does not exist. However, the shell script automatically creates the following four subdirectories if they do not exist: `x/ref_crv`, `x/ref_pif`, `x/ref_pbf`, and `x/ref_log`.

Since the output log contains information about the simulation time and date, the output log of the simulations is not automatically compared. However, it might be interesting for manual comparisons.

Appendix D

Calculation of Additional Extrinsic Parameters

Based on the extrinsic Y-parameters (see Section 3.5), the S-, H-, Z-, and A- (ABCD-) parameters are calculated:

$$\underline{D_S} = (1 + \underline{Y_{11}})(1 + \underline{Y_{22}}) - \underline{Y_{12}}\underline{Y_{21}}, \quad (\text{D.1})$$

$$\underline{D_H} = (1 - \underline{S_{11}})(1 + \underline{S_{22}}) + \underline{S_{12}}\underline{S_{21}}, \quad (\text{D.2})$$

$$\underline{D_Z} = \underline{Y_{11}} \underline{Y_{22}} - \underline{Y_{12}}\underline{Y_{21}}, \quad (\text{D.3})$$

$$\underline{S_{11}} = \left((1 - \underline{Y_{11}})(1 + \underline{Y_{22}}) + \underline{Y_{12}}\underline{Y_{21}} \right) / \underline{D_S}, \quad (\text{D.4})$$

$$\underline{S_{12}} = (-2\underline{Y_{12}}) / \underline{D_S}, \quad (\text{D.5})$$

$$\underline{S_{21}} = (-2\underline{Y_{21}}) / \underline{D_S}, \quad (\text{D.6})$$

$$\underline{S_{22}} = \left((1 + \underline{Y_{11}})(1 - \underline{Y_{22}}) + \underline{Y_{12}}\underline{Y_{21}} \right) / \underline{D_S}, \quad (\text{D.7})$$

$$\underline{H_{11}} = \left((1 + \underline{S_{11}})(1 + \underline{S_{22}}) - \underline{S_{12}}\underline{S_{21}} \right) / \underline{D_H}, \quad (\text{D.8})$$

$$\underline{H_{12}} = (2\underline{S_{12}}) / \underline{D_H}, \quad (\text{D.9})$$

$$\underline{H_{21}} = (-2\underline{S_{21}}) / \underline{D_H}, \quad (\text{D.10})$$

$$\underline{H_{22}} = \left((1 - \underline{S_{22}})(1 - \underline{S_{11}}) - \underline{S_{12}}\underline{S_{21}} \right) / \underline{D_H}, \quad (\text{D.11})$$

$$\underline{Z_{11}} = \underline{Y_{22}} / \underline{D_Z}, \quad (\text{D.12})$$

$$\underline{Z_{12}} = -\underline{Y_{12}} / \underline{D_Z}, \quad (\text{D.13})$$

$$\underline{Z_{21}} = -\underline{Y_{21}} / \underline{D_Z}, \quad (\text{D.14})$$

$$\underline{Z_{22}} = \underline{Y_{11}} / \underline{D_Z}, \quad (\text{D.15})$$

$$\underline{A}_{11} = -\underline{Y}_{22}/\underline{A}_{21} , \quad (D.16)$$

$$\underline{A}_{12} = -1/\underline{Y}_{21} , \quad (D.17)$$

$$\underline{A}_{21} = (\underline{Y}_{12}\underline{Y}_{21} - \underline{Y}_{11}\underline{Y}_{22})/\underline{Y}_{21} , \quad (D.18)$$

$$\underline{A}_{22} = -\underline{Y}_{11}/\underline{Y}_{21} . \quad (D.19)$$

Note that the extrinsic Y-parameters are multiplied by the characteristic impedance. If the user wants MINIMOS-NT to calculate for example intrinsic H-parameters based on the formulae above, not only the parasitic elements must be set to zero, but also the characteristic impedances of the ports have to be set to one. In case of a mixed-mode simulation, these additional two-port parameters are not provided. This is due to the fact that they were originally bound to the parasitic circuit which does not make sense if the simulation is anyway based on dynamic boundary conditions imposed by a circuit.

However, these calculations are in fact post-processing steps, which could also be done in the context of the generalized MINIMOS-NT post-processing interface which was recently introduced (see Appendix C.1). Although the chosen approach is coherent and straightforward, it might be a useful alternative to exempt the MINIMOS-NT output functions of the various sets, to move the optimization to the post-processing interface and to provide the transformation formulae in the input-deck. This is not only a matter of implementation and design, but also of usability.

In addition, a *Graphical User Interface* for editing curve files has been developed. The MDI system (*Multiple Document Interface*) is able to visualize the curve data values in tables and offers the specific functionality to transform two-port parameters into other representations. The user is able to select specific ranges and may also edit the complete file, which is particularly required in case manufacturers provide measurement data in different formats. The program can also be started in batch-mode as used in a loop for optimizing the transformation to extrinsic parameters.

Appendix E

Matrix Storage Formats

Since the linear equation systems are arising from discretized partial differential equations, only a few elements of the $n \times n$ entries of the system matrices are unequal to zero. In Table 5.1 the share of nonzero entries is between 3.24 ‰ and 0.04 ‰. In order to reduce the memory consumption required for storing such sparse matrices, specific matrix storage formats are employed.

The assembly and in-house solver module use the storage format called MCSR [178], which stands for *Modified Compressed Sparse Row*. An analogous concept is the MCSC format, which stands for *Modified Compressed Sparse Column*. The order of the latter format is not row-oriented, but column-oriented, which significantly speeds up deleting of a complete column. The PARDISO solver module requires the matrix in a format called CSR, which stands for *Compressed Sparse Row*. The SAMG package requires a format with the same name, but a different variant of it.

E.1 Modified Compressed Sparse Row Matrices

In general an MCSR or MCSC matrix is very well suited for sparse matrices with the diagonal elements all non-zero, which is a basic requirement for the in-house solver module (see Section 4.9). It consists of two parallel arrays of equal length but different data types. One array contains all indices (`idx`), the other all values (`val`).

The template MCSR class encapsulates all data members required for storing MCSR matrices and provides the interface to this data. The dimension of the system matrix is n . The number of non-zeros in the matrix is `nnzall`.

Note that all existing elements count as non-zero elements, even though their actual value might be zero. In context of *Newton* adjustment, it makes definitely sense to assembly also zero entries in order to reserve the space for elements which are later required. This can happen if transient contributions or ignored derivatives are later added.

The n diagonal elements of the matrix are stored in the first n array elements of the value array. The value of the i -th position `val[i]` is located in the i -th row and i -th column of the square matrix. The off-diagonal elements are stored in the array positions after the diagonal elements in the `val` array. In the case of a MCSR matrix the off-diagonals are sorted by the row indices. All off-diagonals of one row are stored sequentially. The `val` array contains these sequences for all rows beginning with row number 0. There is no space or separator between the sequences. The last off-diagonal is stored at `val[nnz]`.

The index array is required to store the column indices of the respective values. This array runs in parallel to `val`. First, the column numbers can be found in the positions `idx[n+1 . . . nnz]`. Second, the row numbers and the border between two sequences/rows are needed. This information is found in the lower

part of the index array which runs in parallel to the diagonal elements in the value array. For the i -th row the begin index of the sequence stored in the upper part is found at $idx[i]$. Hence the end of the sequence equals the beginning of the next row $idx[i+1]$. To enable a consistent treatment of all rows, especially in loops, it is convenient to leave $val[n]$ unused, to keep the arrays parallel.

Using the MCSC format means that the off-diagonals are grouped by column. All off-diagonals of a column build up a sequence. Hence, the MCSC can be considered the transposed matrix stored in MCSR format.

As an example, the following 4×4 matrix should be stored in a MCSR structure [228]:

$$\begin{array}{c|cccc} \text{row/col} & 0 & 1 & 2 & 3 \\ \hline 0 & 2 & 1 & 0 & 0 \\ 1 & 0 & 4 & 3 & 5 \\ 2 & 7 & 0 & 6 & 0 \\ 3 & 0 & 0 & 0 & 8 \end{array}$$

First of all the four diagonal entries 2, 4, 6 and 8 are stored in the lower part of the value array - and then the four off-diagonals 1, 3, 5 and 7 in the upper part. nnz is eight, two arrays with a dimension of nine have to be allocated. The lower part of the value array contains the four diagonal entries, the next entry is left out and then the off-diagonals are stored sequentially. Hence, the value array looks like (u stands for *unused*):

$$\begin{array}{c|cccccccccc} \text{pos} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \text{val} & 2 & 4 & 6 & 8 & u & 1 & 3 & 5 & 7 \end{array}$$

The lower part of the index array contains the starting index of every row in the upper part, the upper part the original column indices:

$$\begin{array}{c|cccccccccc} \text{pos} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \text{idx} & 5 & 6 & 8 & 9 & 9 & 1 & 2 & 3 & 0 \end{array}$$

The access to this structure is straightforward. For instance, the entry at the position $([row, col])$ $[2, 2]$ should be derived. For a diagonal entry, the entry can be directly derived: $val[2]=6$. The next example is the off-diagonal $[1, 3]$, hence the index array is needed. The off-diagonal sequence of row number 1 starts at position $idx[1]=6$. In case there are off-diagonal entries at all, their column indices have to be compared with 3. So the number of off-diagonal entries of the first row is derived. The starting index of the sequence of the next row is $idx[2]=8$, hence there are two off-diagonals ($idx[2]-idx[1]=8-6=2$). The first one belongs to column $idx[6]=2$, so the next one has to be checked, $idx[7]=3$, which is the right position within the array. The last step is to read the value $val[7]=5$. If there is no value for an existing position stored, it is assumed to be zero.

E.2 Compressed Sparse Row - Variant 1

As already mentioned above, PARDISO requires the system matrix stored in a format called CSR, *Compressed Sparse Row*. Since there are several variants, this one is called *Variant 1* in this work.

The main difference between MCSR and CSR regards storing of diagonal entries. In MCSR an array of length n is used to store all diagonal entries, even if there are zero ones. The CSR format does not store all diagonal entries, and requires therefore also the index information for these entries. For that reason

all entries of all rows are stored sequentially in the value array of length `nnz` and all column indices are stored in the parallel array `ja`. The only information missing is the starting index of each row, which is stored in the array `ia` of length `n + 1`. The CSR representation of the matrix above looks as follows:

```
pos 0 1 2 3 4 5 6 7
val 2 1 4 3 5 7 6 8
ja  0 1 1 2 3 0 2 3
ia  0 2 5 7 8
```

E.3 Compressed Sparse Row - Variant 2

The difference between *Variant 1* and *Variant 2* is the place of the diagonal entry within the value sequence of each row. Whereas *Variant 1* stores the diagonal entry in ascending order with off-diagonal entries, *Variant 2* requires the diagonal entry to be the first one of each row followed by the off-diagonal entries. Therefore, the matrix example is represented as follows:

```
pos 0 1 2 3 4 5 6 7
val 2 1 4 3 5 6 7 8
ja  0 1 1 2 3 2 0 3
ia  0 2 5 7 8
```

E.4 Matrix Storage Format Conversion

Since the CSR format is used in context of external solver modules only, the MCSR class provides a method for converting the already stored MCSR matrix to the respective CSR variant.

This method takes two boolean arguments: the first one, `fortran`, can be used to adapt the index array for the use in FORTRAN code. It is a well-known fact that indexing in FORTRAN starts with 1, whereas with 0 in C/C++. For that reason, the indices have to be incremented by one in case the array is passed to a function written in FORTRAN. The second flag `sortedDiag` indicates whether to store the diagonal sorted (*Variant 1*) or at the beginning of the row (*Variant 2*). The respective arrays are stored within the class and respective direct access methods to the CSR data are provided.

Bibliography

- [1] ABAQUS, Inc., Providence, RI. *ABAQUS*, January 2005. <http://www.hks.com/>.
- [2] Advanced Computing Technology Center, IBM. *Engineering and Scientific Subroutine Library (ESSL)*, January 2005. http://www.research.ibm.com/actc/Optimized_Math_Libraries.html.
- [3] Advanced Micro Devices. *AMD Core Math Library*, January 2005. http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_2282,00.html.
- [4] Agilent Technologies, Inc., Palo Alto, CA. *Advanced Design System ADS*, 2003.
- [5] Agilent Technologies, Inc., Palo Alto, CA. *Guide to Harmonic Balance Simulation in ADS*, 2003.
- [6] Agilent Technologies, Inc., Palo Alto, CA. *E8882A Harmonic Balance Simulator*, 2004. <http://eesof.tm.agilent.com/products/e8882a-a.html>.
- [7] ANSYS, Inc., Canonsburg, PA. *ANSYS*, January 2005. <http://www.ansys.com/>.
- [8] Applied Wave Research, Inc., El Segundo, CA. *Microwave Office 2004*, 2004. <http://www.mwoffice.com/products/mwoffice/>.
- [9] N. Arora. *MOSFET Models for VLSI Circuit Simulation*. Springer, 1993.
- [10] U. Ascher, P.A. Markowich, C. Schmeiser, H. Steinrück, and R. Weiss. Conditioning of the Steady State Semiconductor Device Problem. Technical Report 86-18, University of British Columbia, 1986.
- [11] T. Ayalew. *SiC Semiconductor Devices Technology, Modeling, and Simulation*. Dissertation, TU Vienna, 2004. <http://www.iue.tuwien.ac.at/phd/ayalew/>.
- [12] T. Ayalew, S. Wagner, T. Grasser, and S. Selberherr. Numerical Simulation of Microwave MESFETs in 4H-SiC Fabricated Using Epitaxial Layers on Semi-Insulating Substrates. In *Proc. 5th European Conference on Silicon Carbide and Related Materials*, pages 76–77, Bologna, Italy, September 2004.
- [13] E. Anderson Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, third edition, 1999.
- [14] Z. Bai, D. Day, J. Demmel, and J. Dongarra. *A Test Matrix Collection for Non-Hermitian Eigenvalue Problems (Release 1.0)*, October 1996.
- [15] R.E. Bank and D.J. Rose. Global Approximate Newton Methods. *Numer.Math.*, 37:279–295, 1981.
- [16] R.E. Bank, D.J. Rose, and W. Fichtner. Numerical Methods for Semiconductor Device Simulation. *IEEE Trans.Electron Devices*, ED-30(9):1031–1041, 1983.

- [17] R. Barrett, M. Berry T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks of Iterative Methods*. SIAM, Philadelphia, PA, 1994.
- [18] Basic Linear Algebra Subprograms. *Basic Linear Algebra Subprograms (BLAS)*, January 2005. <http://www.netlib.org/blas/>.
- [19] M. Benzi, C.D. Meyer, and M. TUMA. A Sparse Approximate Inverse Preconditioner. *SIAM J.Sci.Comput.*, 17(5):1135–1149, 1996.
- [20] BIPSIM, Inc., London, Ontario. *BIPOLE3*, September 2004. <http://www.bipsim.com/mainframe.html>.
- [21] L.S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammerling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users’ Guide*. Siam, Philadelphia, 1997.
- [22] K. Bløtekjær. Transport Equations for Electrons in Two-Valley Semiconductors. *IEEE Trans.Electron Devices*, ED-17(1):38–47, 1970.
- [23] Boeing Math Group. *BCSLIB-EXT Iterative Solver*, January 2005. <http://www.boeing.com/phantom/iiss/beamsit.html>.
- [24] Boeing Math Group. *Intelligent Iterative Solver Service (IISS)*, January 2005. <http://www.boeing.com/phantom/iiss/>.
- [25] R. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. Dongarra. Matrix Market: A Web Resource for Test Matrix Collections. *The Quality of Numerical Software: Assessment and Enhancement*, pages 125–137, 1997.
- [26] Boost. *Basic Linear Algebra: uBLAS*, November 2004. <http://www.boost.org/libs/numeric/ublas/>.
- [27] Boost. *The Boost C++ Libraries*, November 2004. <http://www.boost.org/>.
- [28] T.J. Bordelon, X.-L. Wang, C.M. Maziar, and A.F. Tasch. Accounting for Bandstructure Effects in the Hydrodynamic Model: A First-Order Approach for Silicon Device Simulation. *Solid-State Electron.*, 35(2):131–139, 1992.
- [29] I.N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik, 3. überarbeitete und erweiterte Auflage*. Verlag Harri Deutsch, Frankfurt am Main, Thun, 19997.
- [30] E. Buturla, P. Cottrell, B. Grossman, and K. Salsburg. Finite-Element Analysis of Semiconductor Devices: The FIELDAY Program. *IBM J.Res.Dev.*, 44(1-2):142–156, 2000.
- [31] D.M. Caughey and R.E. Thomas. Carrier Mobilities in Silicon Empirically Related to Doping and Field. *Proc.IEEE*, 52:2192–2193, 1967.
- [32] H. Ceric, A. Hoessinger, T. Binder, and S. Selberherr. Modeling of Segregation on Material Interfaces by Means of the Finite Element Method. In *Proc. 4th IMACS Symposium on Mathematical Modeling*, pages 445–452, Vienna, Austria, February 2003.
- [33] J. Cervenka. *Three-Dimensional Mesh Generation for Device and Process Simulation*. Dissertation, Technische Universität Wien, 2004.
- [34] R. Chandra, L. Dagum, D. Korh, D. Maydan, J. McDonald, and R. Menon. *Parallel Programming in OpenMP*. Academic Press, San Diego, London, San Francisco, 2001.

- [35] R. C. Clarke and J. W. Palmour. SiC Microwave Power Technologies. *Proc. IEEE*, 90(6):987–992, 2002.
- [36] T. Clees and K. Stüben. Algebraic Multigrid for Industrial Semiconductor Device Simulation. In *Proc. Challenges in Scientific Computing*, pages 110–130. Springer, Berlin, 2002.
- [37] T. Clees, K. Stüben, and S. Mijalković. Application of an Algebraic Multigrid Solver to Process Simulation Problems. In *Proceedings 2000 International Conference on Simulation of Semiconductor Processes and Devices*, pages 225–228. IEEE, 2000.
- [38] Computational Electronics Research Group, University of Purdue. Nanomos, 2002. http://www.nanohub.org/simulation_tools/nanosmos_tool_information.
- [39] Computer Science Department, University of Basel, Switzerland. *Parallel Sparse Direct Linear Solver PARDISO - User Guide Version 1.0*, 2003.
- [40] Comsol, Inc., Burlington, MA. *FEMLAB*, January 2005. <http://www.comsol.com/>.
- [41] Cree. Cree’s Silicon Carbide RF Products, 2003. http://www.cree.com/Products/rf_sicproducts.asp.
- [42] Crosslight, Inc. *APSYS*, 2005. <http://www.crosslight.com/downloads/downloads.html>.
- [43] E. Cuthill and J. McKee. Reducing the Bandwidth of Sparse Symmetric Matrices. In *ACM Conf.*, pages 157–172, 1969.
- [44] T.A. Davis, J.R. Gilbert, S.I. Larimore, and E.G. Ng. A Column Approximate Minimum Degree Ordering Algorithm. *ACM Trans. Mathematical Software*, 30(3):353–376, 2004.
- [45] M.J. Deen and T.A. Fjeldly. *CMOS RF Modeling, Characterization and Applications*. World Scientific, 2002.
- [46] J. Demel. *JANAP – Ein Programm zur Simulation von elektrischen Netzwerken*. Dissertation, Technische Universität Wien, 1989.
- [47] Department of Computer and Science and Engineering, University of Florida. *Unsymmetric Multi-Frontal Package (UMFPACK)*, January 2005. <http://www.cise.ufl.edu/research/sparse/umfpack/>.
- [48] Department of Electrical Engineering and Computer Sciences, University of Berkeley, Berkeley, CA. *BSIM4.4.0 MOSFET Model*, March 2004.
- [49] H.A. Van der Vorst. BI-CGSTAB: A Fast and Smoothly Converging Variant of BI-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J.Sci.Stat.Comput.* 13(2), pages 631–644, 1992.
- [50] P. Deuflhard. A Modified Newton Method for the Solution of Ill-Conditioned Systems of Nonlinear Equations with Application to Multiple Shooting. *Numer.Math.*, 22:289–315, 1974.
- [51] H.J. Dirschmid. *Mathematische Grundlagen der Elektrotechnik*. Vieweg, 1986.
- [52] H.J. Dirschmid. *Matrizen und Lineare Gleichungen*. Manz, 1998.
- [53] J.J. Dongarra and J.R. Bunch. *LINPACK User’s Guide*. SIAM, Philadelphia, 1990.
- [54] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. Van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, 1998.
- [55] I. Duff, R. Grimes, and J. Lewis. *User’s Guide for the Harwell-Boeing Sparse Matrix Collection*, October 1992.

- [56] I.S. Duff and J.A. Scott. A Parallel Direct Solver for Large Sparse Highly Unsymmetric Linear Systems. *ACM Trans.Mathematical Software*, 30(2):95–117, 2004.
- [57] M. Enciso-Aguilar, F. Aniel, P. Crozat, R. Adde, H.-J. Herzog, T. Hackbarth, U. König, and H. v. Känel. DC and High Frequency Performance of 0.1 μm n-type Si/Si_{0.6}Ge_{0.4} MODFET with $f_{\text{max}} = 188$ GHz at 300 K and $f_{\text{max}} = 230$ GHz at 50 K. *Electron.Lett.*, 39(1):149–151, 2003.
- [58] W.L. Engl and H. Dirks. Numerical Device Simulation Guided by Physical Approaches. In B.T. Browne and J.J. Miller, editors, *Numerical Analysis of Semiconductor Devices and Integrated Circuits*, volume I, pages 65–93, Dublin, 1979. Boole Press.
- [59] W.L. Engl and H. Dirks. Functional Device Simulation by Merging Numerical Building Blocks. In B.T. Browne and J.J. Miller, editors, *Numerical Analysis of Semiconductor Devices and Integrated Circuits*, volume II, pages 34–62, Dublin, 1981. Boole Press.
- [60] M. Galassi et al. *GNU Scientific Library Reference Manual*. Network Theory Limited, 2002.
- [61] M.A. Heroux et al. *An Overview of the Trilinos Package Architecture*. Sandia National Laboratories, 2003.
- [62] Fachbereich Elektrotechnik, Hochschule für Technik und Wirtschaft Dresden (FH). *SIMBA – Dreidimensionale numerische Simulation elektronischer Bauelemente*, 2005. <http://www.htw-dresden.de/~klix/simba/welcome.html>.
- [63] R.D. Falgout, J.E. Jones, and U.M. Yang. The Design and Implementation of Hypre, a Library of Parallel High Performance Preconditioners. In *Numerical Solution of Partial Differential Equations on Parallel Computers*. Springer, 2004. (to appear).
- [64] S. Filippone and M. Colajanni. PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices. *ACM Trans.Mathematical Software*, 26(4):527–550, 2000.
- [65] C. Fischer. *Bauelementsimulation in einer Computergestützten Entwurfsumgebung*. Dissertation, Technische Universität Wien, 1994. <http://www.iue.tuwien.ac.at>.
- [66] C. Fischer and S. Selberherr. Optimum Scaling of Non-Symmetric Jacobian Matrices for Threshold Pivoting Preconditioners. In *Intl. Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits NUPAD V*, pages 123–126, Honolulu, 1994.
- [67] International Technology Roadmap for Semiconductors. International Technology Roadmap for Semiconductors - 2001 Edition, 2001. <http://public.itrs.net>.
- [68] G. Freeman, B. Jgannathan, S.J. Jeng, J.-S. Rieh, A.D. Stricker, D.C. Ahlgren, and S. Subbanna. Transistor Design and Application Considerations for >200-GHz SiGe HBTs. *IEEE Trans.Electron Devices*, 50(3):645–655, 2003.
- [69] G. Freeman, J.-S. Rieh, B. Jagannathan, Z. Yang, F. Guarin, A. Joseph, and D. Ahlren. SiGe HBT Performance and Reliability Trends Through f_T of 350 GHz. In *Proc. International Reliability Physics Symposium*, pages 332–338, Dallas, TX, 2003.
- [70] A. Gehring and H. Kosina. Wigner-Function Based Simulation of Classic and Ballistic Transport in Scaled DG-MOSFETs Using the Monte Carlo Method. In *International Workshop on Computational Electronics*, Purdue, 2004.
- [71] A. George and J.W.H. Liu. An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems. *SIAM J.Numer.Anal.*, 15(5):1053–1069, 1978.

- [72] A. George and D.R. McIntyre. On the Application of the Minimum Degree Algorithm to Finite Element Systems. *SIAM J.Numer.Anal.*, 15(1):90–112, 1978.
- [73] Gesellschaft für Computer Simulationstechnik GmbH, Darmstadt. *Maxwell's Equations by Finite Integration Algorithms (MAFIA)*, January 2005. <http://www.cst.de/>.
- [74] I. Getreu. *Modeling the Bipolar Transistor*. Elsevier, Amsterdam, 1978.
- [75] N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer. An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix. *SIAM J.Numer.Anal.*, 13(2):236–250, 1976.
- [76] R. Girvan. *Partial Differential Equations - Differential Approaches*, December 2001. <http://www.scientific-computing.com/review4.html>.
- [77] B.V. Gokhale. Numerical Solutions for a One-Dimensional Silicon n-p-n Transistor. *IEEE Trans.Electron Devices*, 17(8):594–602, 1970.
- [78] G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, second edition, 1989.
- [79] N.I.M. Gould and J.A. Scott. A Numerical Evaluation of HSL Packages for the Direct Solution of Large Sparse Symmetric Linear Systems of Equations. *ACM Trans.Mathematical Software*, 30(3):300–325, 2004.
- [80] H. Grad. On the Kinetic Theory of Rarified Gases. *Comm. Pure and Appl.Math.*, 2:311–407, 1949.
- [81] T. Grasser. *Mixed-Mode Device Simulation*. Dissertation, Technische Universität Wien, 1999. <http://www.iue.tuwien.ac.at>.
- [82] T. Grasser. Non-Parabolic Macroscopic Transport Models for Semiconductor Device Simulation. *Physica A*, 349(1/2):221–258, 2005.
- [83] T. Grasser, C. Jungemann, H. Kosina, B. Meinerzhagen, and S. Selberherr. Advanced Transport Models for Sub-Micrometer Devices. In *Proc. Simulation of Semiconductor Processes and Devices*, Munich, Germany, September 2004.
- [84] T. Grasser, R. Kosik, C. Jungemann, H. Kosina, and S. Selberherr. Non-Parabolic Macroscopic Transport Models for Device Simulation Based on Bulk Monte Carlo Data. *J.Appl.Phys.*, pages 1–10, 2005. (in print).
- [85] T. Grasser, H. Kosina, M. Gritsch, and S. Selberherr. Using Six Moments of Boltzmann's Transport Equation for Device Simulation. *J.Appl.Phys.*, 90(5):2389–2396, 2001.
- [86] T. Grasser, H. Kosina, C. Heitzinger, and S. Selberherr. Characterization of the Hot Electron Distribution Function Using Six Moments. *J.Appl.Phys.*, 91(6):3869–3879, 2002.
- [87] T. Grasser, H. Kosina, and S. Selberherr. On the Validity of the Relaxation Time Approximation for Macroscopic Transport Models. In *Proc. Simulation of Semiconductor Processes and Devices*, Munich, Germany, September 2004.
- [88] T. Grasser and S. Selberherr. Fully-Coupled Electro-Thermal Mixed-Mode Device Simulation of SiGe HBT Circuits. *IEEE Trans.Electron Devices*, 48(7):1421–1427, 2001.
- [89] T. Grasser, T.-w. Tang, H. Kosina, and S. Selberherr. A Review of Hydrodynamic and Energy-Transport Models for Semiconductor Device Simulation. *Proc.IEEE*, 91(2):251–274, 2003.

- [90] M.A. Green and J. Shewchun. Application of the Small-Signal Transmission Line Equivalent Circuit Model to the A.C., D.C. and Transient Analysis of Semiconductor Devices. *Solid-State Electron.*, 17(9):941–949, 1984.
- [91] M. Gritsch. *Numerical Modeling of Silicon-on-Insulator MOSFETs*. Dissertation, Technische Universität Wien, 2002. <http://www.iue.tuwien.ac.at/phd/gritsch>.
- [92] M.J. Grote and T. Huckle. Parallel Preconditioning with Sparse Approximate Inverses. *SIAM J.Sci.Comput.*, 18(3):838–853, 1997.
- [93] A. Gupta. Recent Advances in Direct Methods for Solving Unsymmetric Sparse Systems of Linear Equations. *ACM Trans.Mathematical Software*, 28(3):310–324, 2002.
- [94] M.S. Gupta. Power Gain in Feedback Amplifiers, a Classic Revisited. *IEEE Trans.Microwave Theory and Techniques*, 40(5):864–879, 1992.
- [95] M.S. Gupta. What Is RF? *IEEE Microwave Magazine*, pages 12–16, December 2001.
- [96] O. Heinrichsberger. *Transiente Simulation von Silizium-MOSFETs*. Dissertation, Technische Universität Wien, 1992.
- [97] K.A. Hennacy, Y.-J. Wu, N. Goldsman, and I. Mayergoyz. Deterministic MOSFET Simulation Using a Generalized Spherical Harmonic Expansion of the Boltzmann Equation. *Solid-State Electron.*, 38(8):1489–1495, 1995.
- [98] Hewlett Packard. *Test and Measurement Application Note 95-1, S-Parameter Techniques*, 1995.
- [99] C.W. Ho, A.E. Ruehli, and P.A. Brennan. The Modified Nodal Approach to Network Analysis. *IEEE Trans.Circuits and Systems*, CAS-22(6):504–509, 1975.
- [100] H. Hofmann. *Das elektromagnetische Feld*. Springer, 1986.
- [101] P. Huguet, P. Auxemery, G. Pataut, P. Fellon, D. Geiger, and H. Jung. Space Evaluation of P-HEMT MMIC Process PH15. In *Proc. European Space Components Conf.*, pages 199–204, Noordwijk, 2000.
- [102] M. Ida, K. Kurishima, N. Watanabe, and T. Enoki. InP/InGaAs DHBTs with 341-GHz f_T at High Current Density of over 800 kA/cm². In *IEDM Tech.Dig.*, pages 776–779, Washington, D.C., 2001.
- [103] L. Ingber. Very Fast Simulated Re-Annealing. *Mathematical Computer Modelling*, 12:967–973, 1989. http://www.ingber.com/asa89_vfsr.ps.gz.
- [104] Institut für Grundlagen der Elektrotechnik und Elektronik, Technische Universität Dresden. DEVICE, 2000. http://www.iee.et.tu-dresden.de/~schroter/Device/Doc/device_descr.
- [105] Institut für Mikroelektronik, Technische Universität Wien, Austria. Minimos-NT 2.1 User's Guide. <http://www.iue.tuwien.ac.at/software/minimos-nt>, 2004.
- [106] Institut National de Recherche en Informatique et en Automatique, Rocquencourt. *Scilab - A Scientific Software Package*, January 2005. <http://scilabsoft.inria.fr/>.
- [107] Intel. *Intel Math Kernel Library 7.2*, January 2005. <http://www.intel.com/software/products/mkl/>.
- [108] inuTech - Innovative Numerical Technologies, Nürnberg. *DIFFPACK*, January 2005. <http://www.diffpack.com/>.

- [109] B.M. Irons. A Frontal Solution Program for Finite Element Analysis. *Int.J.Numer.Meth.Eng.*, 2:5–32, 1970.
- [110] ISE Integrated Systems Engineering AG, Zürich, Switzerland. *DIOS-ISE, ISE TCAD Release 8.0*, July 2002.
- [111] ISE Integrated Systems Engineering AG, Zürich, Switzerland. *DESSIS-ISE, ISE TCAD Release 9.0*, August 2003.
- [112] C. Jacoboni and P. Lugli. *The Monte Carlo Method for Semiconductor Device Simulation*. Springer, Wien-New York, 1989.
- [113] B. Jgannathan, M. Khater, F. Pagette, J.-S. Rieh, D. Angell, H. Chen, J. Florkey, F. Golan, D.R. Greenberg, R. Groves, S.J. Jeng, J. Johnson, e. Mengistu, K.T. Schonenberg, C.M. Schnabel, P. Smith, D.C. Ahlgren, G. Freeman, K. Stein, and S. Subbanna. Self-Aligned SiGe NPN Transistor With 285 GHz f_{\max} and 207 GHz f_T in a Manufacturable Technology. *IEEE Electron Device Lett.*, 23(5):258–260, 2002.
- [114] J.S. Rosen. *General ANSYS Overview*, January 2005. <http://www.andrew.cmu.edu/user/jsrosen/thermalFL/overview.htm>.
- [115] C. Jungemann and B. Meinerzhagen. *Hierarchical Device Simulation: The Monte-Carlo Perspective*. Springer, Wien-New York, 2003.
- [116] C. Jungemann, B. Neinhüs, and B. Meinerzhagen. Full-Band Monte Carlo Device Simulation of a SiGe/Si HBT with a Realistic Ge Profile. *IEICE Trans.Electron.*, E83-C(8):1228–1234, 2000.
- [117] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
- [118] R. Klima, T. Grasser, and S. Selberherr. The Control System of the Device Simulator Minimos-NT. In *Proc. 2nd WSEAS Intl. Conf. on Simulation, Modelling and Optimization*, pages 281–284, Skiathos, Greece, September 2002.
- [119] C.H. Koelbel, G.L. Steel Jr., and M.E. Zosel. *The High Performance Fortran Handbook*. Scientific and Engineering Computational, MIT Press, Cambridge, MA, 1994.
- [120] S.J. Koester, K.L. Saenger, J.O. Chu, Q.C. Ouyang, J.A. Ott, K.A. Jenkins, D.F. Canaperi, J.A. Tornello, C.V. Jahnes, and S.E.Steen. Laterally Scaled Si-Si_{0.7}Ge_{0.3} n-MODFETs with $f_{\max} > 200$ GHz and Low Operating Bias. *IEEE Electron Device Lett.*, 26(3):178–180, 2005.
- [121] S.J. Koester, K.L. Saenger, J.O. Chu, Q.C. Ouyang, J.A. Ott, M.A. Rooks, D.F. Canaperi, J.A. Tornello, C.V. Jahnes, and S.E.Steen. 80 nm gate-length Si/Si_{0.7}Ge_{0.3} n-MODFET with 194 GHz f_{\max} . *Electron.Lett.*, 39(23):1684–1685, 2003.
- [122] H. Kosina, M. Nedjalkov, and S. Selberherr. Theory of the Monte Carlo Method for Semiconductor Device Simulation. *IEEE Trans.Electron Devices*, 47(10):1898–1908, 2000.
- [123] H. Kosina and S. Selberherr. A Hybrid Device Simulator that Combines Monte Carlo and Drift-Diffusion Analysis. *IEEE Trans.Computer-Aided Design*, 13(2):201–210, 1994.
- [124] H. Kosina and Ch. Troger. SPIN — A Schrödinger-Poisson Solver Including Nonparabolic Bands. *VLSI Design*, 8(1-4):489–493, 1998.
- [125] K.S. Kundert. Introduction to RF Simulation and Its Application. *IEEE J.Solid-State Circuits*, 34(9):1298–1319, 1999.

- [126] H.P. Langtangen. *Computational Partial Differential Equations, Numerical Methods and Diffpack Programming, Lecture Notes in Computational Science and Engineering*. Springer Verlag, 1999.
- [127] S.E. Laux. Techniques for Small-Signal Analysis of Semiconductor Devices. *IEEE Trans. Electron Devices*, ED-32(10):2028–2037, 1985.
- [128] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.*, October 1997.
- [129] C.D. Levermore. Moment Closure Hierarchies for Kinetic Theories. *J.Stat.Phys.*, 83(1):1021–1065, 1996.
- [130] B. Li, S. Prasas, L.-W. Yang, and S.C. Wang. A Semianalytical Parameter-Extraction Procedure for HBT Equivalent Circuits. *IEEE Trans. Microwave Theory and Techniques*, 46(10):1427–1435, 1998.
- [131] N. Li, Y. Saad, and E. Chow. Crout Versions of ILU for General Sparse Matrices. *SIAM J.Sci.Comput.*, 25(2):716–728, 2003.
- [132] J.J. Liou and F. Schwier. RF MOSFET: Recent Advances, Current Status and Future Trends. *Solid-State Electron.*, 47(11):1881–1895, November 2003.
- [133] Livermore Software Technology Corporation Corp., Livermore, CA. *LS-DYNA*, January 2005. <http://www.lstc.com/>.
- [134] M. Lundstrom. *Fundamentals of Carrier Transport*. Cambridge University Press, 2000.
- [135] W. Ma, S. Kaya, and A. Asenov. Scaling of RF Linearity in DG and SOI MOSFETs. In *Proc. Intl.Symp. on Electron Devices for Microwave and Optoelectronic Applications*, pages 255–260, Orlando, FL, 2003.
- [136] P.A. Markowich, C.A. Ringhofer, and C. Schmeiser. *Semiconductor Equations*. Springer Verlag, 1990.
- [137] A.H. Marshak and K.M. van Vliet. Electrical Current in Solids with Position-Dependent Band Structure. *Solid-State Electron.*, 21:417–427, 1978.
- [138] A.H. Marshak and C.M. VanVliet. Electrical Current and Carrier Density in Degenerate Materials with Nonuniform Band Structure. *Proc.IEEE*, 72(2):148–164, 1984.
- [139] Mathematical and Computational Sciences Division, Information Technology Laboratory, National Institute of Standards and Technology. *The Matrix Market*. <http://math.nist.gov/MatrixMarket/info.html>.
- [140] Mathematics and Engineering Analysis Unit, Boeing Phantom Works: *SParse Object Oriented Linear Equations Solver (SPOOLES)*, January 1999. <http://www.netlib.org/linalg/spooles/spooles.2.2.html>.
- [141] MathWorks, Natick, MA. *MATLAB and Simulink Products*, February 2005. <http://www.mathworks.com/>.
- [142] H. Mau. *Anpassung und Implementation des Energietransportmodells zur vergleichenden Simulation mit dem Drift-Diffusions-Modell an SiGe-Heterobipolartransistoren*. Dissertation, Technische Universität Ilmenau, 1997.

- [143] C.C. McAndrew, J.A. Seitchik, D.F. Bowers, M. Dunn, I. Getreu, M. McSwain, S. Moinian, J. Parker, D.J. Roulston, M. Schröter, P. van Wijnen, and L.F. Wagner. VBIC95, The Vertical Bipolar Inter-Company Model. *IEEE J.Solid-State Circuits*, SC-31(10):1476–1483, 1996.
- [144] J.R.F. McMacken and S.G. Chamberlain. CHORD: A Modular Semiconductor Device Simulation Development Tool Incorporating External Network Models. *IEEE Trans.Computer-Aided Design*, 8(8):826–836, 1989.
- [145] R. Menon and L. Dagum. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science & Engineering*, pages 1:46–55, 1998.
- [146] M.S. Mock. An Initial Value Problem from Semiconductor Device Theory. *SIAM J.Math.Anal.*, 5(4):597–612, 1974.
- [147] L.W. Nagel. SPICE2: A Computer Program to Simulate Semiconductor Circuits. Technical Report UCB/ERL M520, University of California, Berkeley, 1975.
- [148] Nanoelectronics Group at Raytheon TI Systems. Nemo, 1999. <http://www.cfdrc.com/nemo/>.
- [149] M. Nedjalkov, R. Kosik, H. Kosina, and S. Selberherr. A Wigner Equation for Nanometer and Femtosecond Transport Regime. In *Proc. 1st IEEE Conference on Nanotechnology*, pages 277–281, Maui, USA, October 2001. IEEE.
- [150] M. Nedjalkov, R. Kosik, H. Kosina, and S. Selberherr. Wigner Transport through Tunneling Structures – Scattering Interpretation of the Potential Operator. In *Proc. Simulation of Semiconductor Processes and Devices*, pages 187–190, Kobe, Japan, September 2002.
- [151] M. Nedjalkov, H. Kosina, S. Selberherr, and I. Dimov. A Backward Monte Carlo Method for Simulation of the Electron Quantum Kinetics in Semiconductors. *VLSI Design*, 13(1-4):405–411, 2001.
- [152] netlib. *The Netlib Repository – Netlib*, 2005. <http://www.netlib.org>.
- [153] Network for Computational Nanotechnology. nanoHub, 2005. <http://www.nanohub.org>.
- [154] P. G. Neudeck. SiC Technology. In *The VLSI Handbook, The Electrical Engineering Handbook Series*, W.-K. Chen, Ed. Boca Raton, Florida: CRC Press and IEEE Press, pages 6.1–6.24, 2000.
- [155] Numerical Algorithms Groups. *Numerical Algorithms Group*, September 2004. http://www.nag.co.uk/main_engineering.asp.
- [156] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations*. Wiley, Chichester, 1992.
- [157] Open Systems Laboratory, Indiana University Bloomington. *Blitz++ - Object-Oriented Scientific Computing*, November 2004. <http://www.oonumerics.org/blitz/>.
- [158] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*. Wissenschaftsverlag, Mannheim–Leipzig–Wien–Zürich, 1993.
- [159] P.S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufman, San Francisco, 1997.
- [160] V. Palankovski and R. Quay. *Analysis and Simulation of Heterostructure Devices*. Springer, Wien, New York, 2004.
- [161] V. Palankovski, S. Wagner, T. Grassler, R. Schultheis, and S. Selberherr. Direct S-Parameter Extraction by Physical Two-Dimensional Device AC-Simulation. In *Proc. International Symposium on Compound Semiconductors*, pages 303–306, Lausanne, Switzerland, 2002.

- [162] V. Palankovski, S. Wagner, and S. Selberherr. Numerical Analysis of Compound Semiconductor RF Devices. In *Proc. GaAs IC Symposium*, pages 107–110, San Diego, CA, 2003. (invited).
- [163] Parallel Algorithms and Optimization Group, Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications, Toulouse. *MUMPS: A Multifrontal Massively Parallel Sparse Direct Solver*, July 2003. <http://www.enseeiht.fr/lima/apo/MUMPS/>.
- [164] A. Prechtl. *Vorlesungen über die Grundlagen der Elektrotechnik, Band 2*. Springer, Wien–New York, 1995.
- [165] W. H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1997.
- [166] W. L. Pribble, J. W. Palmour, S. T. Sheppard, R. P. Smith, S. T. Allen, T. J. Smith, Z. Ring, J. J. Sumarkeris, A. W. Saxler, and J. W. Milligan. Applications of SiC MESFETs and GaN HEMTs in Power Amplifier Design. *IEEE MTT-S Digest*, 3:1819–1822, 2002.
- [167] Python Software Foundation. *Python*, January 2005.
- [168] T. Quarles, A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, March 1994.
- [169] R. Quay. *Analysis and Simulation of High Electron Mobility Transistors*. Dissertation, Technische Universität Wien, 2002. <http://www.iue.tuwien.ac.at/phd/quay>.
- [170] R. Quay, R. Kiefer, F. van Raay, H. Massler, S. Ramberger, S. Muller, M. Dammann, M. Mikulla, M. Schlechtweg, and G. Weimann. Integration of a 0.13- μm CMOS and a High Performance Self-Aligned SiGe HBT Featuring Low Base Resistance. In *Intl. Electron Devices Meeting*, pages 673–676, San Francisco, 2002.
- [171] B. Razavi. CMOS Technology Characterisation for Analog and RF Design. *IEEE J.Solid-State Circuits*, 34(3):268–276, 1999.
- [172] J.-S. Rieh, B. Jagannathan, H. Chen, K. Schonenberg, D. Angell, A. Chinthakindi, J. Florkey, F. Golan, D. Greenberg, S.-J. Jeng, M. Khater, F. Pagette, C. Schnabel, P. Smith, A. Stricker, K. Vaed, R. Volant, D.C. Ahlgren, G. Freeman, K. Stein, and S. Subbanna. SiGe HBTs with Cut-off Frequency of 350 GHz. In *Intl. Electron Devices Meeting*, pages 771–774, San Francisco, 2002.
- [173] J.-S. Rieh, B. Jagannathan, H. Chen, K. Schonenberg, S.-J. Jeng, M. Khater, D.C. Ahlgren, G. Freeman, and S. Subbanna. Performance and Design Considerations for High Speed SiGe HBTs of $f_T/f_{\text{max}}=375\text{GHz}/210\text{GHz}$. In *Proc. International Conference on Indium Phosphide and Related Materials*, pages 374–377, Santa Barbara, CA, 2003.
- [174] C. Ringhofer, C. Schmeiser, and A. Zwirchmayer. Moment Methods for the Semiconductor Boltzmann Equation in Bounded Position Domains. *SIAM J.Numer.Anal.*, 39(3):1078–1095, 2001.
- [175] J.G. Rollins and J. Choma. Mixed-Mode PISCES-SPICE Coupled Circuit and Device Solver. *IEEE Trans.Computer-Aided Design*, 7:862–867, 1988.
- [176] G.-C. Rota. Twelve Problems in Probability no One Likes to Bring Up. In *Algebraic Combinatorics and Computer Science*, pages 57–93. Springer Italia, Milan, 2001.

- [177] F.M. Rotella, G. Ma, Z. Yu, and R.W. Dutton. Modeling, Analysis, and Design of RF LDMOS Devices Using Harmonic Balance Device Simulation. *IEEE Trans.Microwave Theory and Techniques*, 48(6):991–999, 2002.
- [178] Y. Saad. SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations. Technical report, RIACS, NASA Ames Research Center, Moffett Field, CA 94035, May 1990.
- [179] Y. Saad and M.H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J.Sci.Stat.Comput.*, 7(3):856–869, 1986.
- [180] Sandia National Laboratories. *The Trilinos Project*, January 2005. <http://software.sandia.gov/Trilinos/>.
- [181] O. Schenk and K. Gärtner. Solving Unsymmetric Sparse Systems of Linear Equations with PAR-DISO. *Future Generation Computer Systems*, 2003. Accepted, in press.
- [182] O. Schenk, K. Gärtner, and W. Fichtner. Efficient Sparse LU Factorization with Left-Right Looking Strategy on Shared Memory Multiprocessors. *BIT*, 40(1):158–176, 2003.
- [183] O. Schenk, S. Röllin, and A. Gupta. The Effects of Unsymmetric Matrix Permutations and Scalings in Semiconductor Device and Circuit Simulation. *IEEE Trans.Computer-Aided Design*, 23(3):400–411, 2004.
- [184] I. Schnyder, M. Rohner, E. Gini, D. Huber, C. Bergamaschi, and H. Jackel. A Laterally Etched Collector InP/InGaAs(P) DHBT Process for High Speed Power Applications. In *Indium Phosphide and Related Materials*, pages 477–480, Williamsburg, VA, 2000.
- [185] D. Schroeder. *Modelling of Interface Carrier Transport for Device Simulation*. Springer, 1994.
- [186] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall, 1996.
- [187] R. Schultheis, N. Bovolon, J.-E. Müller, and P. Zwicknagl. Modelling of Heterojunction Bipolar Transistors (HBTs) Based on Gallium Arsenide (GaAs). *Intl.J. of RF and Microwave Computer-Aided Engineering*, 10(1):33–42, 2000.
- [188] H.R. Schwarz. *Numerische Mathematik*. Teubner, Stuttgart, 1997. 4. Auflage.
- [189] F. Schwierz and J.J. Liou. *Modern Microwave Transistors: Theory, Design, and Performance*. Wiley, New Jersey, 2003.
- [190] J.A. Scott. Parallel Frontal Solvers for Large Sparse Linear Systems. *ACM Trans.Mathematical Software*, 29(4):395–417, 2003.
- [191] J.A. Scott. MA57 – A Code for the Solution of Sparse Symmetric Definite and Indefinite Systems. *ACM Trans.Mathematical Software*, 30(2):118–144, 2004.
- [192] S. Seeger and K.H. Hoffmann. The Cumulant Method for Computational Kinetic Theory. *Continuum Mech. Thermodyn.*, 12:403–421, 2000.
- [193] S. Selberherr. *Analysis and Simulation of Semiconductor Devices*. Springer, Wien–New York, 1984.
- [194] S. Selberherr. Device Modeling and Physics. *Physica Scripta*, T35:293–298, 1991.
- [195] 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.

- [196] G. Sewell. TWODEPEP, a Small General-Purpose Finite Element Program. *Angewandte Informatik*, 4:249–253, 1982.
- [197] G. Sewell. PDE2D: Easy-to-Use Software for General Two-Dimensional Partial Differential Equations. *Advances in Engineering Software*, 17(2):105–112, 1993.
- [198] K. Shinohara, Y. Yamashita, A. Endoh, I. Watanabe, K. Hikosaka, T. Matsui, T. Mimura, and S. Hiyamizu. 550 GHz f_T Pseudomorphic InP-HEMTs with Reduced Source-Drain Resistance. In *Proc. 61st Device Research Conference*, pages 145–146, Salt Lake City, UT, 2003.
- [199] K. Shinohara, Y. Yamashita, A. Endoh, I. Watanabe, K. Hikosaka, T. Matsui, T. Mimura, and S. Hiyamizu. 547-GHz f_T In_{0.7}Ga_{0.3}As-In_{0.52}Al_{0.48}As HEMTs with Reduced Source and Drain Resistance. *IEEE Electron Device Lett.*, 25(5):241–243, 2004.
- [200] Silvaco, Santa Clara, California. *Atlas User's Manual - Device Simulation Software*, December 2002.
- [201] T. Simlinger. *Simulation von Heterostruktur-Feldeffekttransistoren*. Dissertation, Technische Universität Wien, 1996. <http://www.iue.tuwien.ac.at>.
- [202] B.T. Smith, J.M. Boyle, and J.J. Dongarra. *Matrix Eigensystem Routines - EISPACK Guide*. Springer, 1976.
- [203] Software & Analysis of Advanced Materials Processing Center, University of Florida. FLOODS and FLOOPS, 2002. <http://www.swamp.tec.ufl.edu/~floods/>.
- [204] A. Stach. *Simulation von MOSFET-Schaltungen*. Diplomarbeit, Technische Universität Wien, 1995.
- [205] Stanford TCAD Group. *PISCES-2ET*. Stanford University, Stanford, CA, 1994. <http://www-tcad.stanford.edu/tcad.html>.
- [206] Stanford TCAD Group. *PISCES-2H-B*. Stanford University, Stanford, CA, June 1997. <http://www-tcad.stanford.edu/tcad.html>.
- [207] Stanford TCAD Group. *Stanford TCAD Tools*, 2004. <http://www-tcad.stanford.edu/tcad.html>.
- [208] M.B. Steer, J.W. Brandler, and C.M. Snowden. Computer-Aided Design of RF and Microwave Circuits and Systems. *IEEE Trans.Microwave Theory and Techniques*, 50(3):996–1005, 2002.
- [209] R. Stratton. Diffusion of Hot and Cold Electrons in Semiconductor Barriers. *Physical Review*, 126(6):2002–2014, 1962.
- [210] D. Streit, R. Lai, A. Oki, and A. Gutierrez-Aitken. InP HEMT and HBT Technology and Applications. In *Proc. Intl.Symp. on Electron Devices for Microwave and Optoelectronic Applications*, pages 14–17, Manchester, UK, 2002.
- [211] A.D. Stricker, J.B. Johnson, G. Freeman, and J.-S. Rieh. Design and Optimization of a 200 GHz Sige HBT Collector Profile by TCAD. *Applied Surface Science*, 224/1-4:324–329, 2004.
- [212] B. Stroustrup. *C++ Programming Language*. Addison-Wesley, 1997.
- [213] K. Stüben and T. Clees. *SAMG User's Manual Release 21c*. Fraunhofer Institute for Algorithms and Scientific Computing, 2003.
- [214] Synopsis, Freemont, CA. *Davinci, Three-Dimensional Device Simulation Program, Version 2002.4*, February 2003.

- [215] Synopsis, Fremont, CA. *HSpice Circuit Simulator*, February 2003.
- [216] Synopsis, Fremont, CA. *Medici, Two-Dimensional Device Simulation Program, Version 2002.4*, February 2003.
- [217] S.M. Sze. *Physics of Semiconductor Devices*. Wiley, New York, second edition, 1981.
- [218] V. Temple and J. Shewchun. Exact Frequency Dependent Complex Admittance of the MOS Diode Including Surface States. *Solid-State Electron.*, 16(1):93–113, 1973.
- [219] R. Thoma, A. Emunds, B. Meinerzhagen, H.J. Peifer, and W.L. Engl. Hydrodynamic Equations for Semiconductors with Nonparabolic Band Structure. *IEEE Trans. Electron Devices*, 38(6):1343–1353, June 1991.
- [220] B. Troyanovsky. *Frequency Domain Algorithms for Simulating Large Signal Distortion in Semiconductor Devices*. Dissertation, Stanford University, 1997.
- [221] B. Troyanovsky, F. Rotella, Z. Yu, R. W. Dutton, and J. Sato-Iwanaga. Large Signal Analysis of RF/Microwave Devices with Parasitics Using Harmonic Balance Device Simulation. In *Proc. SASIMI, Fukuoka, Japan*, pages 178–179, 1996.
- [222] R. S. Tuminaro, M. Heroux, S. A. Hutchinson, and J. N. Shadid. *Official Aztec User's Guide: Version 2.1*. Sandia National Laboratories, December 1999.
- [223] University of California Berkeley. *SuperLU - Sparse Gaussian Elimination on High Performance Computers*, January 2005. <http://www.cs.berkeley.edu/~demmel/SuperLU.html>.
- [224] University of Tennessee. *Automatically Tuned Linear Algebra Software*, July 2004. <http://math-atlas.sourceforge.net/>.
- [225] W.V. VanRoosbroeck. Theory of Flow of Electrons and Holes in Germanium and Other Semiconductors. *Bell Syst. Techn. J.*, 29:560–607, 1950.
- [226] Visual Numerics. *IMSL Numerical Libraries Family of Products*, January 2005. <http://www.vni.com/products/imsi/index.html>.
- [227] G.K. Wachutka. Rigorous Thermodynamic Treatment of Heat Generation and Conduction in Semiconductor Device Modeling. *IEEE Trans. Computer-Aided Design*, 9(11):1141–1149, November 1990.
- [228] S. Wagner. The Minimos-NT Linear Equation Solving Module. Diplomarbeit, Technische Universität Wien, 2001.
- [229] S. Wagner, T. Grasser, C. Fischer, and S. Selberherr. A Simulator Module for Advanced Equation Assembling. In *Proc. 15th European Simulation Symposium ESS*, pages 55–64, Delft, The Netherlands, 2003.
- [230] S. Wagner, T. Grasser, and S. Selberherr. Evaluation of Linear Solver Modules for Semiconductor Device Simulation. In *Proc. 5th Intl. Conference on Mathematical Problems in Engineering and Aerospace Sciences ICNPAA*, Timisoara, Romania, June 2004. (in print).
- [231] S. Wagner, T. Grasser, and S. Selberherr. Mixed-Mode Device and Circuit Simulation. In *Proc. 11th Intl. Conference Mixed Design of Integrated Circuits and Systems*, pages 36–41, Szczecin, Poland, June 2004. (invited).

- [232] S. Wagner, S. Holzer, R. Strasser, R. Plasun, T. Grasser, and S. Selberherr. *SIESTA - The Simulation Environment for Semiconductor Technology Analysis*. Institut für Mikroelektronik, 2003.
- [233] S. Wagner, V. Palankovski, T. Grasser, G. Röhrer, and S. Selberherr. A Direct Extraction Feature for Scattering Parameters of SiGe-HBTs. *Applied Surface Science*, 224/1-4:365–369, 2004.
- [234] M. Wall. GALib A C++ Library of Genetic Algorithm Components. *Massachusetts Institute of Technology*, 2000. <http://lancet.mit.edu/ga>.
- [235] E.X. Wang, M. Stettler, S. Yu, and C. Maziar. Application of Cumulant Expansion to the Modeling of Non-local Effects in Semiconductor Devices. In *Proc. Intl. Workshop on Computational Electronics*, pages 234–237, Piscataway, NJ, USA, 1998.
- [236] K. Washio. High-speed SiGe HBTs and Their Applications. *Applied Surface Science*, 224/1-4:306–311, 2004.
- [237] Webplexity. nextnano³, 2004. <http://www.webplexity.de/nextnano3.php>.
- [238] R.C. Whaley, Antoine Petitet, and Jack J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1-2):3–35, 2001.
- [239] P.H. Woerlee, M.J. Knitel, R. van Langevelde, D.B.M. Klaassen, L.F. Tiemeijer, A.J. Scholten, and A.T.A. Zegers van Duijnhoven. RF-CMOS Performance Trends. *IEEE Trans. Electron Devices*, 48(8):1776–1782, August 2001.
- [240] D.L. Woolard, H. Tian, R.J. Trew, M.A. Littlejohn, and K.W. Kim. Hydrodynamic Electron-Transport Model: Nonparabolic Corrections to the Streaming Terms. *Physical Review B*, 44(20):11119–11132, 1991.
- [241] Z. Yu, R.W. Dutton, B. Troyanovsky, and J. Sato-Iwanaga. Large Signal Analysis of RF Circuits in Device Simulation. *IEICE Trans. Electron.*, E82-C(6):908–916, 1999.
- [242] A. P. Zhang, L.B. Rowland, E. B. Kaminsky, J. W. Kretchmer, R. A. Beaupre, J. L. Garrett, J. B. Tucker, B. J. Edward, J. Foppes, and A. F. Allen. Microwave Power SiC MESFETs and GaN HEMTs. In *Proceedings of the IEEE Lester Eastman Conference on High Performance Devices*, pages 181–185, 2002.
- [243] O.C. Zienkiewicz. *The Finite Element Method*. McGraw-Hill, 1977.
- [244] O. Zinke and H. Brunswig. *Hochfrequenztechnik*. Springer, 1999.

Own Publications

Publications in Journals

- [J03] S. Wagner, T. Grasser, C. Fischer, and S. Selberherr, "An Advanced Equation Assembly Module," *Engineering with Computers*, 2005, (in print).
- [J02] J. Park, S. Wagner, T. Grasser, and S. Selberherr, "New SOI Lateral Power Devices with Trench Oxide," *Solid-State Electron.*, vol. 48, no. 6, pages 1007–1015, 2004.
- [J01] S. Wagner, V. Palankovski, T. Grasser, G. Röhrer, and S. Selberherr, "A Direct Extraction Feature for Scattering Parameters of SiGe-HBTs," *Applied Surface Science*, vol. 224/1-4, pages 365–369, 2004.

Publications in Conference Proceedings

- [C18] W. Wessner, S. Wagner, T. Grasser, and S. Selberherr, "Meshing Aspects on Three-Dimensional Fin-Fet Device Simulations," in *Proc. Asia Pacific Microwave Conference (APMC)*, (New Delhi, India), 2004. (cd rom).
- [C17] S. Wagner, T. Grasser, and S. Selberherr, "Physical Modeling of Semiconductor Devices for Microwave Applications," in *Proc. Asia Pacific Microwave Conference (APMC)*, (New Delhi, India), 2004. (cd rom, invited).
- [C16] T. Ayalew, S. Wagner, T. Grasser, and S. Selberherr, "Numerical Simulation of Microwave MES-FETs in 4H-SiC Fabricated Using Epitaxial Layers on Semi-Insulating Substrates," in *Proc. 5th European Conference on Silicon Carbide and Related Materials*, (Bologna, Italy), pages 76–77, Sept. 2004.
- [C15] H. Ceric, R. Sabelka, S. Holzer, W. Wessner, S. Wagner, T. Grasser, and S. Selberherr, "The Evolution of the Resistance and Current Density During Electromigration," in *Proc. Simulation of Semiconductor Processes and Devices*, (Munich, Germany), pages 331–334, Sept. 2004.
- [C14] S. Wagner, T. Grasser, and S. Selberherr, "Performance Evaluation of Linear Solvers Employed for Semiconductor Device Simulation," in *Proc. Simulation of Semiconductor Processes and Devices SISPAD*, (Munich, Germany), pages 351–354, Sept. 2004.
- [C13] S. Wagner, T. Grasser, and S. Selberherr, "Mixed-Mode Device and Circuit Simulation," in *Proc. 11th Intl. Conference Mixed Design of Integrated Circuits and Systems*, (Szczecin, Poland), pages 36–41, June 2004. (invited).
- [C12] S. Wagner, T. Grasser, and S. Selberherr, "Evaluation of Linear Solver Modules for Semiconductor Device Simulation," in *Proc. 5th Intl. Conference on Mathematical Problems in Engineering and Aerospace Sciences ICNPAA*, (Timisoara, Romania), June 2004. (in print).

- [C11] S. Wagner, T. Grasser, and S. Selberherr, "Benchmarking Linear Solvers with Semiconductor Simulation Examples," in *Proc. Intl. Conference on Scientific and Engineering Computation IC-SEC*, (Singapore), June 2004. (cd rom).
- [C10] S. Holzer, A. Sheikoleslami, S. Wagner, C. Heitzinger, T. Grasser, and S. Selberherr, "Optimization and Inverse Modeling for TCAD Applications," in *Proc. Symposium on Nano Devices Technology SNDT*, (Hsinchu, Taiwan), pages 113–116, May 2004.
- [C09] S. Wagner, V. Palankovski, R. Quay, T. Grasser, and S. Selberherr, "Numerical Simulation of High-Speed High-Breakdown Indium Phosphide HBTs," in *Proc. Intl. Workshop on the Physics of Semiconductor Devices*, (Madras, India), pages 836–838, 2003.
- [C08] S. Wagner, T. Grasser, C. Fischer, and S. Selberherr, "A Generally Applicable Approach for Advanced Equation Assembling," in *Proc. International Conference on Software Engineering and Applications SEA*, (Marina del Rey, CA), pages 494–499, 2003.
- [C07] S. Wagner, V. Palankovski, G. Röhrer, T. Grasser, and S. Selberherr, "Numerische Berechnung von Silizium-Germanium Heterostruktur-Bipolartransistoren," in *Beiträge zur Informationstagung Mikroelektronik ME 03*, (Vienna, Austria), pages 383–388, 2003.
- [C06] S. Wagner, T. Grasser, C. Fischer, and S. Selberherr, "Advanced Equation Assembling Techniques for Numerical Simulators," in *Proc. 15th European Simulation and Modeling Conference ESMC*, (Naples, Italy), pages 390–394, 2003.
- [C05] V. Palankovski, S. Wagner, and S. Selberherr, "Numerical Analysis of Compound Semiconductor RF Devices," in *Proc. GaAs IC Symposium*, (San Diego, CA), pages 107–110, 2003. (invited).
- [C04] S. Wagner, T. Grasser, C. Fischer, and S. Selberherr, "A Simulator Module for Advanced Equation Assembling," in *Proc. 15th European Simulation Symposium ESS*, (Delft, The Netherlands), pages 55–64, 2003.
- [C03] S. Wagner, V. Palankovski, T. Grasser, G. Röhrer, and S. Selberherr, "A Direct Extraction Feature for Scattering Parameters of SiGe-HBTs," in *Abstracts Intl. SiGe Technology and Device Meeting*, (Nagoya, Japan), pages 83–84, 2003.
- [C02] V. Palankovski, S. Wagner, T. Grasser, R. Schultheis, and S. Selberherr, "Direct S-Parameter Extraction by Physical Two-Dimensional Device AC-Simulation," in *Proc. International Symposium on Compound Semiconductors*, (Lausanne, Switzerland), pages 303–306, 2002.
- [C01] S. Wagner, V. Palankovski, R. Quay, T. Grasser, and S. Selberherr, "Small-Signal Analysis and Direct S-Parameter Extraction," in *Proc. Intl. Symposium on Electron Devices for Microwave and Optoelectronic Applications EDMO*, (Manchester, UK), pages 50–55, 2002.

Curriculum Vitae

- October 15th, 1976 Born in Vienna, Austria
- May 1995 High School Graduation, BRG XIV Wien
- October 1995 Enrolled in Electrical Engineering
at the Technical University of Vienna
- June 1996 - January 1997 Compulsory Military Service
- October 1998 Enrolled in Business Administration
at the Vienna University for Economics and Business Administration
- December 1998 Passed 1. Diplomprüfung at the
Technical University of Vienna
- November 2001 Received degree of “Diplomingenieur”, comparable to
Master of Science (MSc), in Electrical Engineering
with a concentration in Computer Technology
from the Technical University of Vienna (with honors)
- November 2001 Entered doctoral program at the
Institute for Microelectronics, TU Vienna
- July 2003 - August 2003 Internship at Texas Instruments in
Dallas, TX
- January 2005 Passed 1. Diplomprüfung (BW94) at the
Vienna University for Economics and Business Administration (with honors)