Engineering with Computers (2005) 21: 151–163
DOI 10.1007/s00366-005-0319-5

ORIGINAL ARTICLE

Stephan Wagner · Tibor Grasser · Claus Fischer
Siegfried Selberherr

An advanced equation assembly module

Received: 24 October 2003 / Accepted: 9 February 2005 / Published online: 11 November 2005

Abstract We present an advanced equation assembly module which has been developed for the simulation of semiconductor devices based on the Finite Boxes discretization scheme and is currently used in the general purpose device and circuit simulator MINIMOS-NT. Such simulations require the solution of a specific set of nonlinear partial differential equations which are discretized on a grid. The resulting nonlinear problem is solved by a damped Newton algorithm that demands the solution of a linear equation system at each step. The presented module is responsible for assembling these systems and takes into account several requirements of the simulation process. The underlying concepts, namely the representation of boundary conditions, physically motivated variable transformation, preelimination and numerical conditioning, are presented together with some examples.

Keywords Finite Boxes · Linear equation systems · Boundary conditions · Interface conditions · Device simulation · Circuit simulation

1 Introduction

The Finite Boxes discretization method is employed in various kinds of numerical tools and simulators for fast and accurate solution of nonlinear partial differential equation (PDE) systems. The resulting discretized problem is then usually solved by damped Newton iterations which require the solution of a linear equation system at each step. The extensibility and effectiveness of any simulator highly depends on the capabilities of the core modules responsible for handling the linear equation systems. We present an advanced equation assembly module which has been implemented in the device and circuit simulator MINIMOS-NT.

MINIMOS-NT [1] is a general purpose device and circuit simulator that has been developed at the institute for microelectronics over a period of 12 years. Besides the basic semiconductor equations [2], several different types of transport equations can be solved. Among these are the hydrodynamic equations which capture hot-carrier transport [3, 4], a six moments transport model [5], the lattice heat flow equation to cover thermal effects like self-heating [6], and the circuit equations to connect single devices to a circuit [7], both electrically and thermally. 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. This demands a sophisticated system handling the equation assembly, in order to keep the simulator design flexible. To implement such a system, the requirements will be identified and generalized.

A crucial aspect is also the requirement of assembling and solving complex-valued linear equation systems. For that reason the module is able to handle both real-valued and complex-valued contributions and systems.

2 The analytical problem

In order to analyze the electronic properties of an arbitrary semiconductor structure under all kinds of operating conditions, the effects related to the transport of charge carriers under the influence of external fields must be modeled. In MINIMOS-NT carrier transport can

S. Wagner (✉) · T. Grasser
Christian Doppler Laboratory for TCAD in Microelectronics
at the Institute for Microelectronics, TU Vienna,
Gußhausstr. 27–29, 1040 Wien, Austria
E-mail: Wagner@iue.tuwien.ac.at
Tel.: +43-1-58801
Fax: +43-1-58801

S. Selberherr
Institute for Microelectronics, TU Vienna,
Gußhausstraße 27–29/E360, 1040 Wien, Austria

C. Fischer
Firma Dr. Claus Fischer, Gustav Fuhrichweg 24/1,
2201 Gerasdorf bei Wien, Austria

be treated by the drift-diffusion and the hydrodynamic transport models.

Both models are based on the semiclassical Boltzmann transport equation, which is a time-dependent partial integro-differential equation in the six-dimensional phase space. By the so-called method of moments this equation can be reduced to an infinite series of equations. Keeping only the zero and first order moment equations (with proper closure assumptions) yields the basic semiconductor equations. Considering two additional moments gives the hydrodynamic model [8].

The basic semiconductor equations, as given by Vanroosbroeck [9], consist of the Poisson equation, the continuity equations for electron and holes as well as the current relations for both carrier types. The unknown quantities of this equation system are the electrostatic potential, ψ, and the electron and hole concentrations, n and p, respectively. C Denotes the net concentration of the ionized dopants, ε is the dielectric permittivity of the semiconductor, and R is the net recombination rate. The heat-flow equation and thus the lattice temperature T_L is added to account for thermal effects. In the hydrodynamic case the carrier temperatures are allowed to be different from the lattice temperature, adding two more quantities, which are the electron and hole temperatures T_n and T_p.

Basically, a device structure can be divided into several segments to enable simulation of advanced heterostructures and to properly account for all conditions (which may cause very abrupt changes) at the contacts and interfaces between these segments, respectively. See Fig. 1 for an illustration of this concept. Every segment represents an independent domain D in one, two, or three dimensions where the PDEs are posed. the equations are implicitly formulated for a quantity x as $F_{(x)} = 0$ and termed control functions. In order to fully define the mathematical problem, suitable boundary conditions for contacts, interfaces, and external surfaces have to be applied.

Generally, such a system cannot be solved analytically, and the solution must be calculated by means of numerical methods. This approach normally consists of three tasks:

1. The domain D is partitioned into a finite number of subdomains D_i, in which the solution can be approximated with a desired accuracy.
2. The PDE system is approximated in each of the subdomains by algebraic equations. The unknown functions are approximated by functions with a given structure. Hence, 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 derivation of the discrete problem several methods can be applied. We deal here with point residual methods: the Finite Difference method based on rectangular grids or the Finite Boxes (box integration) method allowing general unstructured grids. In the case of orthogonal rectangular grids both methods yield the same discretization.

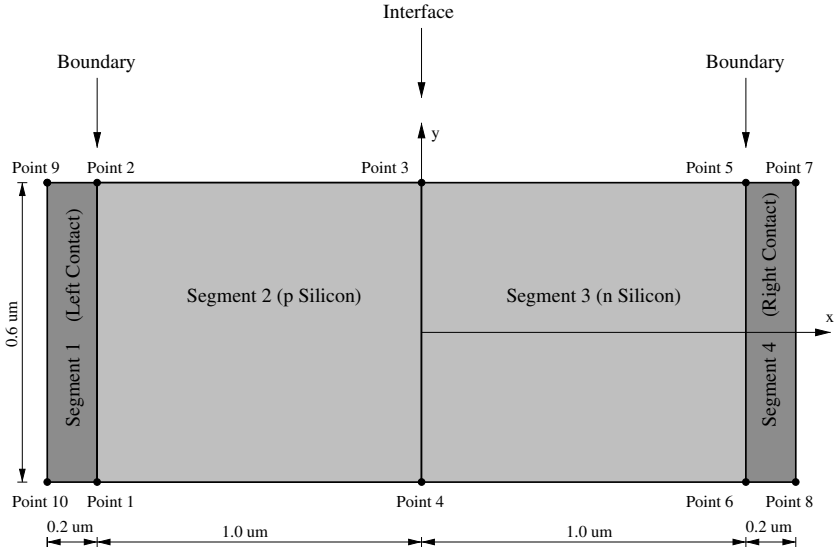

Fig. 1 Illustration of the segment concept: simple diode

3 Discretization

Nonlinear partial differential equations of second order can appear in three variants: elliptic, parabolic, and hyperbolic PDEs. The Poisson equation as well as the steady-state continuity equations form a system of elliptic PDEs, whereas the heat-flow equation is parabolic. To completely determine the solution of an elliptic PDE boundary conditions have to be specified. Since parabolic and hyperbolic PDEs describe evolutionary processes, time normally is an independent variable and an initial condition is additionally required. Hence, also the transient continuity equations are parabolic.

Applying the Finite Boxes discretization scheme [2] the equations are integrated over a control volume (subdomain, usually obtained by a voronoi tesselation) D_i which is associated with the grid point P_i. For this grid point a general equation for the quantity x is implicitly given as

$$f_{x_i}^{S} = \sum_j F_{x_{i,j}} + G_i = 0, \tag{1}$$

Where j runs over all neighboring grid points in the same segment, $F_{x_{i,j}}$ is the flux between points i and j, and G_i is the source term (see Fig. 2).

grid points on the boundary ∂D are defined as having neighbor grid points in other segments. Thus, (1) Does not represent the complete control function f_x, 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 ∂D, The Neumann condition, which specifies the normal derivative, and the linear combination of these conditions giving an intermediate type:

$$\mathbf{n} \cdot \text{grad}\, x + \sigma x = \delta. \tag{2}$$

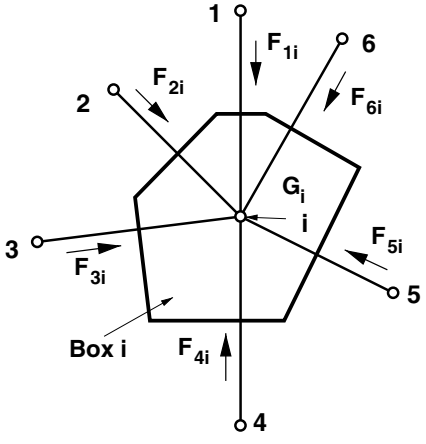

Fig. 2 Box i with six neighbors

Generally, the form of these conditions depends on the respective boundary models, and the conditions of which depend 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 differ 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 interdependency between the solution variables such as thermionic field emission interface conditions [10, 11] have also been implemented.

A method has been under development to implement segment models calculating the interior fluxes and their derivatives independently of 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.

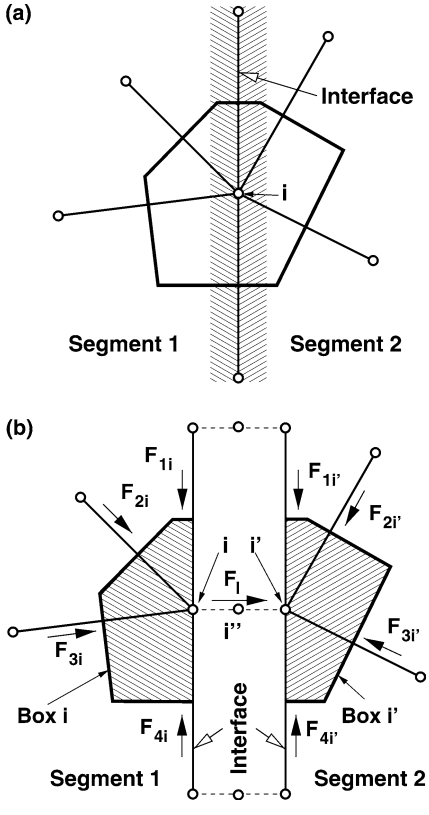

Fig. 3 Splitting of interface points: Interface points as given in **a** are split into three different points having the same geometrical coordinates **b**

3.1 Interface conditions

To account for complex interface conditions, grid points located at the boundary of the segments (see Fig. 3a) Have three values, one for each segment (see Fig. 3b) and a third point located directly at the interface which can be used to formulate more complicated interface conditions like e.g., interface charges. However, to simplify notation these interface values will be omitted in our discussion and only the two interface points, i and i', are used.

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

$$f_{x_i} = f^{\mathrm{S}}_{x_i} + F_{x_{i,i'}} = 0, \tag{3}$$

$$f_{x_{i'}} = f^{\mathrm{S}}_{x_{i'}} + F_{x_{i,i'}} = 0. \tag{4}$$

The intermediate type of interfaces Eq. (2) and thus also the two other types of interfaces are generally given in linearized form by:

$$\alpha(x_i - \beta x_{i'} + \gamma) = F_{x_{i,i'}}, \tag{5}$$

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 Equations 4 to 3, 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 can be used not only for grid point P_i, but also for $\mathrm{P}_{i'}$.

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 (Eq. 5) 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 Sect. 5. 4).
- 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 Sect. 5).

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 $\sum_i f^{\mathrm{S}}_{x_i}$ and introduced as a solution variable because some interface models require the cross-interface electric field strength to determine 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 $\vec{E}_n$ has to be calculated using neighboring points when unstructured two- or three-dimensional grids are used.

See Fig. 4 for an illustration of these concepts. The transformations are set up to combine the various segment contributions with the boundary system.

3.2 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 spacial multi-dimensional segments. Furthermore, all fluxes over the boundary are handled as additional solution variables F_{C} (e.g., Contact charge Q_{C} for Poisson equation, contact electron current $I_{n\mathrm{C}}$ for the electron continuity equations, or H_{C} as the contact heat flow).

For explicit boundary conditions one gets

$$f_{x_i} = f^{\mathrm{S}}_{x_i} + F_{x_{i,C}} = 0, \tag{6}$$

$$f_{F_{\mathrm{C}}} = F_{\mathrm{C}} + \sum\nolimits_i f^{\mathrm{S}}_{x_i} = 0, \tag{7}$$

with i running over all segment grid points.

At Schottky contacts explicit boundary conditions apply. The semiconductor contact potential ψ_{s} is fixed

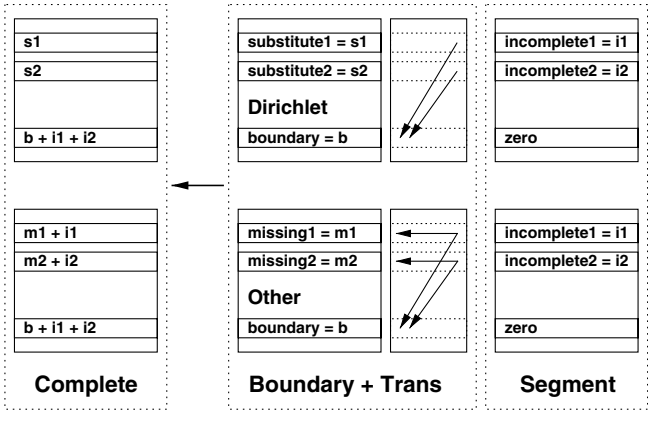

Fig. 4 The complete equations are a combination of the boundary and the segment system. This combination is controlled by the transformation matrix and depends on the interface type

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 potentialψ_{wf}.

$$\psi_s = \psi_C - \psi_{wf}, \quad \text{where } \psi_{wf} = -\frac{E_w}{q}. \tag{8}$$

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

$$f_{x_i} = x_C - h(x_i) = 0, \tag{9}$$

$$f_{F_C} = F_C + \sum\nolimits_i f_{x_i}^S = 0. \tag{10}$$

here, x_C is the boundary value of the quantity, which is a solution variable, whereas (Eq. 10) 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 hydrodynamic 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 [12]), the contact potential at the semiconductor boundary reads

$$\psi_s = \psi_C + \psi_{bi}. \tag{11}$$

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

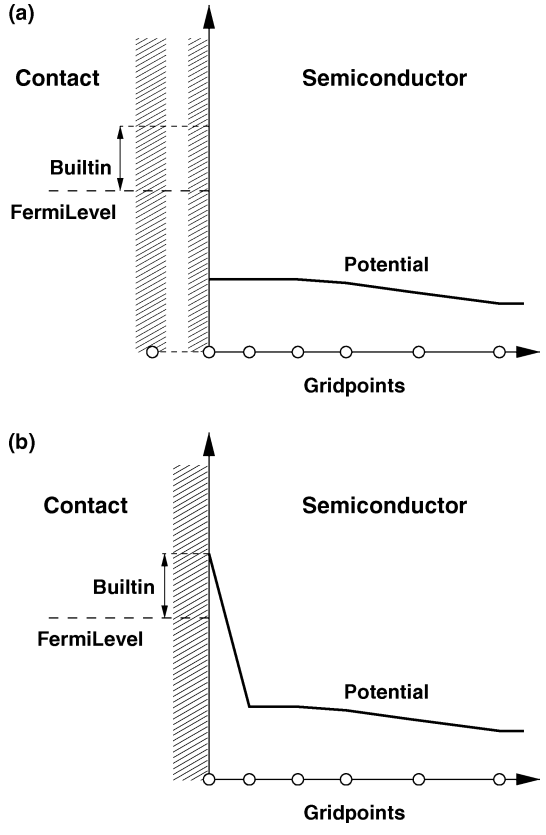

Fig. 5 Effect of a separate potential variable on the initial guess of the potential: **a** with a separate potential variable the potential stays smooth inside the semiconductor region. **b** directly applying the contact potential gives a large discontinuity of the potential

3.2.1 *Separate contact variables*

Having a separate solution variable for the contact voltage avoids numerical problems with large arguments of the bernoulli function B. Using a Scharfetter–Gummel discretization scheme [13] the expression for the current between two grid points i and j reads

$$I_{i,j} = C_1(B(\Delta)n_j - B(-\Delta)n_i)x \quad \text{with } \Delta = C_2(\psi_j - \psi_i) + C_3 \tag{12}$$

with C_i being material parameters. Applying the contact voltage directly to the boundary grid point could cause large arguments of B and hence numerical problems. This is avoided by having a separate variable for the contact voltage. At the beginning of the iteration procedure the constitutive relation for ψ_C is violated and will only successively be adapted which guarantees numerical stability (see Fig. 5).

The generalized boundary condition is the constitutive relation for the contact potential ψ_C and reads:

$$f_{\psi_C} = \alpha\psi_C + \beta I_C + \gamma Q_C - \delta = 0, \tag{13}$$

where Q_C is the contact charge and $I_C = i_{nc} + i_{pc} + \partial Q_C/\partial t$ the contact current. it should be noted that all these quantities are solution variables which are directly available.

4 Solving of the nonlinear system

MINIMOS-NT organizes the solving of the nonlinear, but discretized control functions $\mathbf{f} = \mathbf{0}$ using a damped newton algorithm (k is the number of the iteration step) [2]:

$$\mathbf{j}^{\mathbf{k}} \cdot \mathbf{x}^{\mathbf{k+1}} = \mathbf{f}(\mathbf{v}^{\mathbf{k}}), \tag{14}$$

$$\mathbf{v}^{\mathbf{k+1}} = \mathbf{v}^{\mathbf{k}} + \mathbf{F}_d\mathbf{x}^{\mathbf{k+1}}, \tag{15}$$

$$\mathbf{J} = -\frac{\partial \mathbf{f}}{\partial \mathbf{v}}, \tag{16}$$

where $\mathbf{J}$ is the jacobian matrix, $\mathbf{f}(\mathbf{v})$ the residual and $\mathbf{x}$ the update or correction vector (solution vector of the linear system) that is then used to calculate the next solution vector $\mathbf{v}$ of the Newton approximation. To avoid overshoot of the solution several damping schemes suggested by Deuflhard [14] or Bank and Rose [15] are providing a damping factor F_d. For each Newton iteration step a linear equation system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ has to be assembled and solved.

4.1 Key demands on the assembly module

The key demands on the assembly module (class) can be summarized as follows:

1. Application programming interface providing methods for
   - Adding values to the segment system
   - Adding values to the boundary system
   - Adding values to the transformation matrix
   - Deleting equations
   - Setting elimination flags
   - Administration of priority information
2. Row transformation: linear combination of rows to extinguish large entries (see Sect. 5. 2).
3. Variable transformation: reduce the coupling of the semiconductor equations (see Sect. 5. 3).
4. Preelimination: eliminate problematic equations by gaussian elimination to improve the condition of the inner system matrix (see Sect. 5. 4).
5. Call of specific plug-ins for
   - Scaling: since a threshold value (tolerance) is used to decide whether to keep or skip an entry, the preconditioner used (incomplete-LU Factorization) requires a system matrix having entries of the same order of magnitude (see Sect. 5. 5).
   - Sorting: reduction of the bandwidth of a matrix to reduce the fill-in (see Sect. 5. 5).
   - Solving the linear equation system (see Sect. 5. 6).

The input of the assembly module are the contributions of the various segment and boundary models implemented in the simulator. The assembly module compiles these values to a linear equation system, which is subsequently transformed in order to improve the condition of the system matrix.

For some simulations, for example derivation of the complex-valued admittance matrix, several linear equation systems differ only in the right-hand-side vector. Thus, the effort for assembling, compiling, preeliminating, sorting, scaling and factorizing of the system matrix actually has to be done only once - and this factored matrix could then be used for all Rhs-vectors. For that reason the module is able to simultaneously assemble several Rhs vectors.

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 get the system matrix on input and return the sorting and scaling (diagonal) matrices which are then applied by the assembly module. The solver module gets the system matrix and all Rhs vectors on input and returns the solution vectors of all inner linear equation systems.

After reverting all transformations and backsubstituting the preeliminated equations, the output of the assembly module is the complete solution vector (or vectors in case of multiple right-hand-side vectors). In addition, the right-hand-side vector(s) are returned which can be used for various norm calculations.

5 Description of the assembly module

Minimos-NT consists of two separate modules responsible for assembling and solving linear equation systems:

1. The assembly module which is directly accessed by the implemented physical models of the simulator, provides an effective application programming interface, various transformation algorithms and the preelimination system. In addition, sorting and scaling plug-ins can be called.
2. The solver module which is plugged into the assembly module, is responsible for solving the so-called inner linear equation system. The module currently used provides a direct (gaussian) method and two iterative solver schemes, Bicgstab [16] and Gmres(m) [17]. Although this work is about the assembly module, a short review of the solver module is given in Sect. 6.

A schematic review of the complete concept is given in Fig. 6. In the upper left corner the Newton iteration control of Minimos-NT is represented. 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 solid lines beginning at the interface, the four matrices $\mathbf{T}_b$, $\mathbf{A}_s$, $\mathbf{A}_b$, and $\mathbf{T}_v$ (see Sect. 5.1) are assembled by using a specific flexible sparse storage class. All diagonal elements are stored in one array, and the off-diagonals, the positions of which are not known in advance, in a balanced binary tree for each row, sorted by column. This allows flexible adding of all entries of the sparse matrices. These structure are then converted to the *modified sparse compressed row* (MCSR) format [18] and are compiled resulting in the complete linear system, which is preeliminated to obtain the inner and the outer system.

The inner one is sorted, scaled, and finally passed to the solver module. After the solution has been calculated, scaling and sorting have to be reversed and the preeliminated equations are solved back. The Newton adjustment levels (dashed lines) reuse already existing MCSR structures, which reduces the assembling effort: the balanced trees may be skipped completely, and during compilation and preelimination much simpler functions (bold boxes) can be used than in the conventional mode (bold boxes with slash). In the next section each step will be described with more details.

5.1 Assembly of the complete linear equation system

The semiconductor device is divided into several segments that are geometrical regions employing a distinct

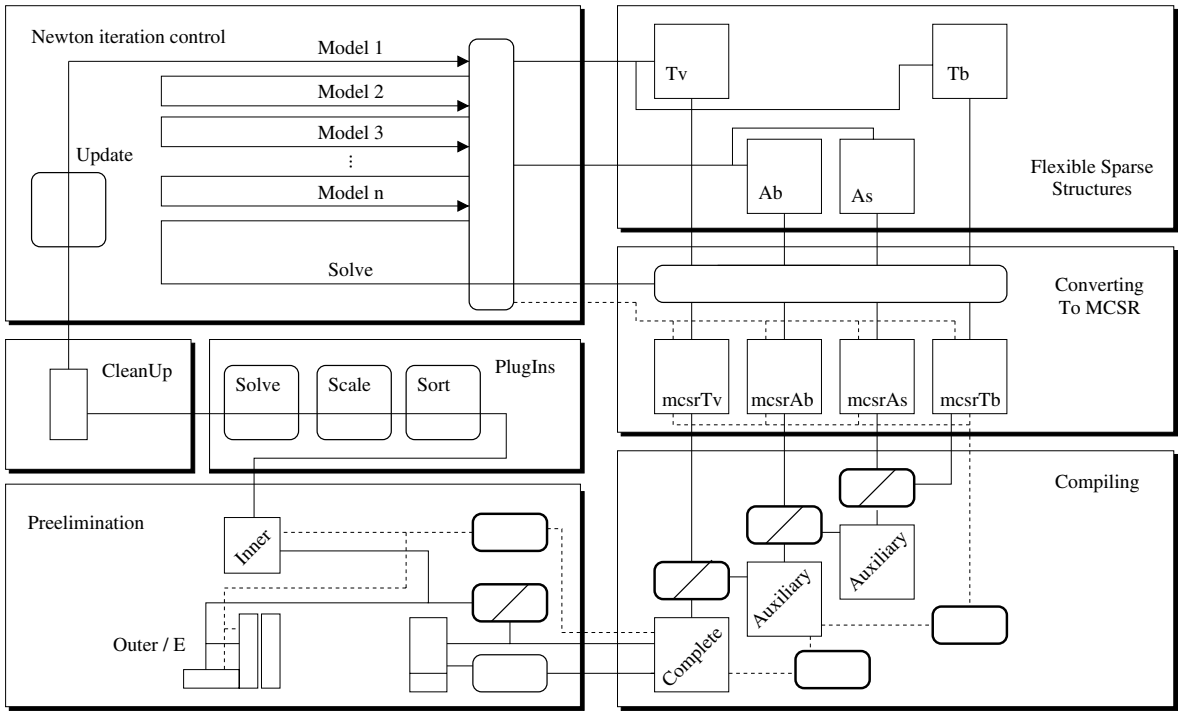

Fig. 6 Schematic assembly overview

set of models. The implementation of each model is completely independent of 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, which are represented by assembly structures for the boundary system which are independent of the segment ones.

Thus, the system matrix **A** or Jacobian matrix 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 **A**. The right hand side vector **B** is treated the same way:

$$\mathbf{A} = \mathbf{A}_b + \mathbf{T}_b \cdot \mathbf{A}_s, \tag{17}$$

$$\mathbf{b} = \mathbf{b}_b + \mathbf{T}_b \cdot \mathbf{b}_s, \tag{18}$$

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}. \tag{19}$$

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 is marked to be solved (the user may request only a subset of all provided models) and secondly the priority of the model is 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 has to be adhered to. In addition, a fourth matrix $\mathbf{T}_v$ is assembled which contains information for an additional variable transformation (see Sect. 5. 3).

During the assembling process, all contributions are added to values stored in balanced binary trees. After the assembly is finished, these trees are converted to the sparse matrix format MCSR since all necessary mathematical operations are defined for these structures. The analogous, column oriented MCSC format is used to speed up column deleting required by the transformation matrix.

During the Newton iterations the structural configuration of these matrices is not modified very often (e.g., on enabling more derivatives), thus, the tree assembly may 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 speeding up the assembly process. However, if an entry in the structure is missing, the conventional assembly procedure can be easily restarted. The so-called newton adjustment addresses not only the assembly matrices, but also the resulting structures of the compilation and preelimination process. A related feature incorporates

the sorting of the inner system matrix into the preelimination process, which allows to skip the sorting process, too.

5.2 Row-transformation

The complete linear equation system is built from an original system (segment system), which is the main matrix $\mathbf{A}_s$ and the main right hand side vector $\mathbf{b}_s$, both of them representing cumulated fluxes and their derivatives to the system variables. Basically, the fluxes are calculated from segment models which are the models for the interior of discretized regions. The matrix is a linear superposition of very small matrices, one for each flux, with a few non-zero elements only. Consequently, the same superposition applies for vector $\mathbf{b}_s$. 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 $\mathbf{b}_s$ in the places appropriate for the variables that are assigned to the boxes. In the context of the newton method, matrix $\mathbf{A}_s$ contains the negative derivatives of the values in $\mathbf{b}_s$ to the system variables, so that the change $d\mathbf{x}$ in the variables leads to a change $-\mathbf{A}_s \cdot d\mathbf{v} = d\mathbf{b}_s$ in the right hand side. Considering $\mathbf{b}_s$ as a function of the variable vector $\mathbf{v}$, one can write:

$$\mathbf{b}_s = \mathbf{b}_s(\mathbf{v}), \tag{20}$$

$$\mathbf{A}_s = -\frac{\mathbf{db}_s}{\mathbf{dv}}. \tag{21}$$

The boundary conditions will enforce some special physical conditions at the boundaries. The control functions of boxes along the boundary 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 main system.

The boundary conditions are therefore implemented by two elements: a boundary system ($\mathbf{A}_b$ and $\mathbf{b}_b$) and a transformation matrix $\mathbf{T}_b$. The purpose of the matrix $\mathbf{T}_b$ is the forwarding of the fluxes of the main system to their final destination or their resetting if they are not required. The system of $\mathbf{A}_b$ and $\mathbf{b}_b$ represents additional or substitutional parts of the final equation for the variables at the boundaries. Again, the entries in the matrix $\mathbf{A}_s$ are the negative derivatives of the right hand side vector $\mathbf{b}_b$ to the variable vector $\mathbf{v}$:

$$\mathbf{b}_b = \mathbf{b}_b(\mathbf{v}), \tag{22}$$

$$\mathbf{A}_b = -\frac{\mathbf{db}_b}{\mathbf{dv}}. \tag{23}$$

The various transformation submatrices for each kind of boundary condition are summarized in Table 1:

5.2.1 The complete linear system

The full system is assembled from the segment system and the boundary system in the following way:

$$\mathbf{b} = \mathbf{b}_b + \mathbf{T}_b \cdot \mathbf{b}_s, \tag{24}$$

$$\mathbf{A} = \mathbf{A}_b + \mathbf{T}_b \cdot \mathbf{A}_s, \tag{25}$$

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}, \tag{26}$$

$$(\mathbf{A}_b + \mathbf{T}_b \cdot \mathbf{A}_s) \cdot \mathbf{x} = \mathbf{b}_b + \mathbf{T}_b \cdot \mathbf{b}_s. \tag{27}$$

As stated above, vector $\mathbf{x}$ represents a linear change of the variables vector $\mathbf{v}$. With Eq. 28 and the new variables vector $\mathbf{v}_n$ from Eq. 29, the new value $\mathbf{b}_n$ of the vector function $\mathbf{b}(\mathbf{v})$ will be described by the linear approximation in equation Eq. 30.

$$\mathbf{x} = \mathbf{dv}, \tag{28}$$

$$\mathbf{v_n} = \mathbf{v} + \mathbf{x}, \tag{29}$$

$$\mathbf{b_n}(\mathbf{v_n}) = \mathbf{b} + \frac{\mathbf{db}}{\mathbf{dv}} \cdot \mathbf{dv} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x} = \mathbf{0}. \tag{30}$$

5.3 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 $\mathbf{T}_v$ is assembled and multiplied with the system matrix.

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 [19].

The transformation expressed by matrix $\mathbf{T}_v$ is given by Eq. 31:

$$((\mathbf{A}_b + \mathbf{T}_b \cdot \mathbf{A}_s) \cdot \mathbf{T}_v) \cdot (\mathbf{T}_v^{-1} \cdot \mathbf{x}) = (\mathbf{b}_b + \mathbf{T}_b \cdot \mathbf{b}_s). \tag{31}$$

For compactness the following substitutions will be used hereinafter:

Table 1 Transformation submatrices with $t_{x,y}$ as entry in the transformation matrix

Type	Other				Dirichlet			
Interface	$t_{x,y}$	x_i	$x_{i'}$		$t_{x,y}$	x_i	$x_{i'}$	
	x_i	1	0		x_i	0	0	
	$x_{i'}$	0	1		$x_{i'}$	0	0	
Contact	$t_{x,y}$	x_{i_1}	x_{i_2}	F_C	$t_{x,y}$	x_{i_1}	x_{i_2}	F_C
	x_{i_1}	1	0	0	x_{i_1}	0	0	0
	x_{i_2}	0	1	0	x_{i_2}	0	0	0
	F_C	1	1	0	F_C	1	1	0

$$\tilde{A} = ((A_b + T_b \cdot A_s) \cdot T_v), \tag{32}$$
$$\tilde{x} = (T_v^{-1} \cdot x), \tag{33}$$
$$\tilde{b} = (b_b + T_b \cdot b_s). \tag{34}$$

5.4 Preelimination

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 that is necessary to make the linear equation system solvable with an iterative solver. 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 that this simply prevents iterative linear solvers from converging.

One solution to this problem which occurs only at the boundary variables that are affected in this way by the boundary conditions, is to apply gaussian elimination to these variables/equations before the system is passed on to the linear solver. After the iterative solver has converged, the eliminated variables are calculated by backsubstitution into the eliminated equations. This process is a partial gaussian factorization of the matrix. Here, it is called preelimination.

Before they can be eliminated, the equations of this type are sorted to the back 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. The equation causing a possibly ill condition have to be marked for preelimination.

The outer system is removed from the linear equation system and later solved by gaussian elimination, the inner system (with an improved condition) is passed on to an iterative solver. See Fig. 7 [12] for an illustration of this concept.

The resulting system is given by Eq. 35:

$$(E \cdot P \cdot \tilde{A} \cdot P^T) \cdot (P \cdot \tilde{x}) = (E \cdot P \cdot \tilde{b}). \tag{35}$$

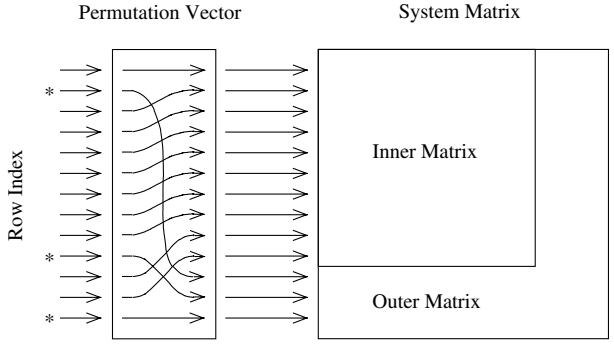

Fig. 7 All equations marked for preelimination (*asterisk*) are moved to the outer system matrix, the others remain in the inner one

here, $\mathbf{P}$ is the permutation matrix with its inverse equal to its transposed matrix $\mathbf{P}^T$. $\mathbf{E}$ is a matrix of elimination coefficients obtained as the lower matrix $\mathbf{L}$ of a gaussian elimination of the permuted system matrix. $\mathbf{E}$ contains non-zero off-diagonals in the outer parts only, the inner matrix up to the row/column index that narrows the section passed to the linear solver is a strict unity matrix.

5.5 Sorting and scaling of the inner linear equation system

Matrices arising from the discretization of differential operators are sparse, because only neighboring points are considered. To reduce memory consumption, only the non-zero elements are stored (see MCSR format). During a factorization of $\mathbf{A}$ into an upper and lower triangular matrix $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$, additional matrix elements termed fill-in [2] become non-zero. The profile $p(\mathbf{A})$ is a measure for this fill-in

$$p(\mathbf{A}) = \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} \mathbf{m_i}, \tag{36}$$
$$m_i = i - \min_{a_{i,j} \neq 0}(j), \tag{37}$$

and the bandwidth of the matrix is $\max_i(m_i)$. since storing of p($\mathbf{A}$) requires additional memory, a transformation is applied to reduce the bandwidth and thus the profile. The standard module provided by default obtains the sorting matrix $\mathbf{R}_s$ (similar to $\mathbf{P}$) by a Cuthill–McKee-based algorithm. It contains in each row a single entry of 1 and is applied in such a way that rows and columns are equally swapped (to keep the diagonal dominance).

To provide the (ILU-) preconditioner with a normalized representation of the matrix, a scaling of all values has to be performed. The standard algorithm used by default works with a two-stage strategy [20]: In the first stage, the matrix is scaled such that the diagonal elements are one. The second stage attempts to suppress the off-diagonals while keeping the diagonals at unity. The resulting scaling matrices $\mathbf{S}_r$ and $\mathbf{S}_c$ are diagonal matrices, and $\mathbf{R}_s^T$ equals $\mathbf{R}_s^{-1}$. With $\mathbf{A}_i \cdot \mathbf{x}_i = \mathbf{b}_i$ as the inner system, the effect of sorting and scaling is given in Eq. 38:

$$(\mathbf{S}_r \cdot (\mathbf{R}_s^T \cdot \mathbf{A_i} \cdot \mathbf{R}_s) \cdot \mathbf{S}_c) \cdot (\mathbf{S}_c^{-1} \cdot (\mathbf{R}_s^T \cdot \mathbf{x_i})) = \mathbf{S}_r \cdot (\mathbf{R}_s^T \cdot \mathbf{b_i}). \tag{38}$$

5.6 The solver module

The solver module is responsible for calculating the solution vector of the inner linear equation system passed by the assembly module. There are several approaches to obtain this vector: a gaussian solver

factorizes the matrix with a complete LU factorization, followed by a forward and backward substitution. Iterative solvers use successive approximations of this vector to obtain more accurate solutions to a linear equation system at each step [1000].

The progress of convergence of an iterative method depends mainly on the spectrum of the system matrix **A**. An important way to handle ill-conditioned matrices ($\kappa(\mathbf{A}) \gg 1$) is to precondition the matrix **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. In the particular case of semiconductor equations, proper precondition is a must.

After the scaling (providing a normalized system) there are two main functions involved in iterative solving: a preconditioner (here Incomplete-LU factorization) and a iterative equation solver scheme (here either BICGSTAB or GMRES(m)).

Adding a preconditioner to an iterative solver causes extra cost, both once for the setup and per iteration for the algorithm. The trade-off between construction/application and gain in convergence speed should be considered. According to [12], 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 administrated by the equation system and adapted after each solver call, cf. Fig. 8 for an illustration of the hierarchical concept.

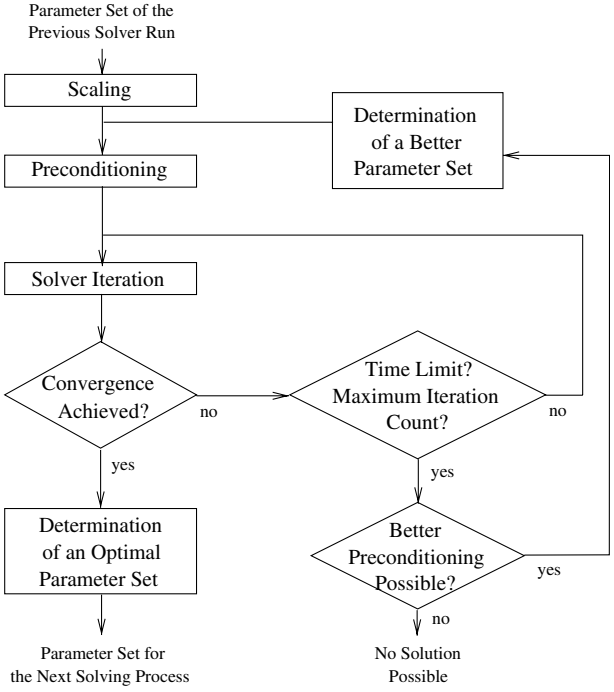

Fig. 8 The hierarchical concept

6 Mixed-mode simulation

Several works dealing with circuit simulation using distributed devices (that are devices for which the semiconductor equations are solved) have been published so far (22–24 or 25). Most publications deal with the coupling of device simulators to SPICE. This results in a two-level Newton algorithm since the device and circuit equations are handled subsequently. Each solution of the circuit equations gives a new operating point for the distributed devices. The device simulator is called to calculate the resulting currents and the derivatives of these currents with respect to the contact voltages. The other approach is called full-Newton algorithm as it combines the device and circuit equations into one single equation system. This equation system is then solved applying Newton algorithm. In contrast to the two-level Newton algorithm, where the device and circuit unknowns are solved separately in a decoupled manner, the complete set of unknowns is solved simultaneously. However, the problem of calculating the device currents and their derivatives remains. As the device current is normally calculated in a post-processing step the derivatives are lost and have to be calculated by other techniques which is cumbersome and time-consuming [22], [25]. With the equation assembly features presented here this problem can be solved in a very efficient and elegant way. Since the contact currents are available as solution variables their derivatives with respect to the contact voltages are contained in the system matrix.

For the assembly of the circuit matrix MINIMOS-NT employs the modified nodal approach (MNA). However, to be compatible with the constitutive relation formulation used for the semiconductor equations ($\mathbf{f}(\mathbf{x}) = \mathbf{0}$) the constitutive relation for the device compact models are formulated in the same way rather than directly calculating linearized circuit elements.

In the case of a mixed-mode simulation Eq. 13 is replaced by

$$f_{\psi_C} = \psi_C - \varphi_C = 0, \tag{39}$$

with ϕ_C being the node voltage of the circuit node connected to the device. This can be interpreted as a zero-valued voltage source connecting the circuit node to the device (see Fig. 9). The constitutive relation for ϕ_C follows from KCL and reads

$$f_{\varphi_C} = \sum\nolimits_i I_{D_i} + I_C = 0, \tag{40}$$

with I_{D_i} being the currents of the compact models connected to the same node. In terms of the familiar MNA stamps

$j_{x,\,y}$	ϕ_C	ψ_C	I_C	RHS
ϕ_C	0	0	-1	$f^k_{\varphi_C}$
ψ_C	1	-1	0	$f^k_{\psi_C}$
I_C	0	0	-1	$f^k_{I_C}$

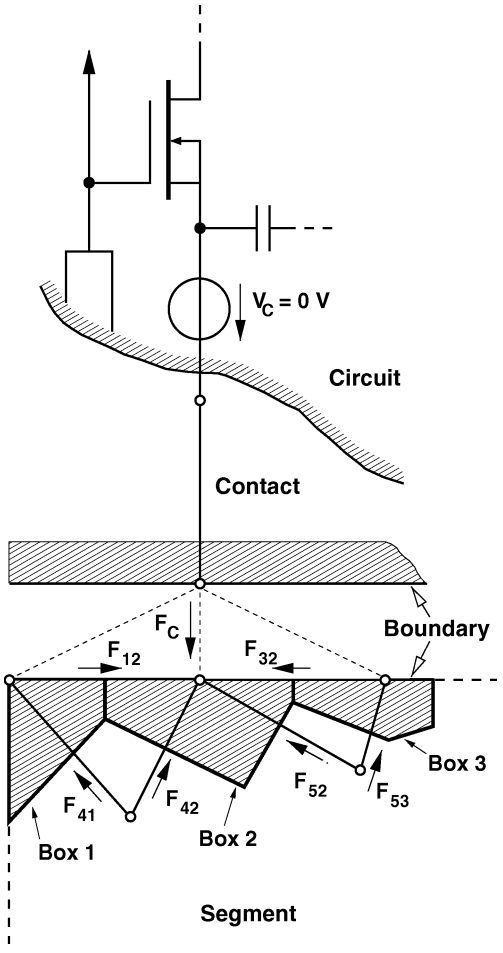

Fig. 9 Contact handling for mixed-mode problems

For the same one-dimensional contact the relevant part of the linear equation system reads

$j_{x,\,y}$	ψ_0	n_0	ψ_C	I_C	ϕ_C	RHS
ψ_0	-1	0	1	0	0	$f^k_{\psi_0}$
n_0	0	-1	0	0	0	$f^k_{n_0}$
ψ_C	0	0	-1	0	1	$f^k_{\psi_C}$
I_C	$-\partial I_{01}/\partial\psi_0$	$\partial I_{01}/\partial n_0$	0	-1	0	$f^k_{I_C}$
ϕ_C	0	0	0	-1	0	$f^k_{\varphi_C}$

Eliminating ψ_0, n_0, ψ_C, and I_C yields the desired result

$j_{x,\,y}$	ϕ_C	RHS
ϕ_C	$-\partial I_{01}/\partial\psi_0$	$-f^k_{\psi_C}+f^k_{I_C}-f^k_{n_0}\cdot\partial I_{01}/\partial n_0$ $-\partial I_{01}/\partial\psi_0\cdot(f^k_{\psi_0}+f^k_{\psi_C})$

7 Conclusions

We presented the concept and implementation of an advanced assembly module successfully applied in the device and circuit simulator MINIMOS-NT. The module provides all conceptional and numerical features required for assembling and solving linear systems arising from semiconductor device and circuit simulation. We developed a formulation which allows the independent treatment of segments, boundaries, and interface models. Boundary and interface conditions are dealt with separately by the boundary and interface models. All fluxes over boundaries are available as solution variables, which simplifies the formulation of boundary conditions and circuit equations. In addition, a physically motivated variable transformation can be applied. Equations can be preeliminated to improve the condition of the system matrix of the resulting linear system. The remaining inner system can be passed to specific sorting and scaling plug-ins and is finally solved by using one of the algorithms provided by the solver module.

The presented concepts result in superior stability of MINIMOS-NT without restricting model implementation and further development. The general approach for treating boundary conditions and several preconditioning measures yield diagonal-dominant linear equation systems well prepared for advanced solver algorithms. As a result boundary conditions for specific operating points can be directly applied without successively stepping to the desired value as is very common even in commercial simulators.

Appendix

As an example consider a simple ohmic boundary condition for the electron concentration $n_i = N_0$ with N_0 being a constant value depending only on the doping of the semiconductor. To further simplify the example we only consider the static Poisson equation and the static continuity equation for electrons. Furthermore, the separate variable for the electron contact current I_{C_n} is omitted. For a one-dimensional device (index 0 refers to the incomplete point at the contact, 1 to the first complete point), the segment constitutive relations read

$$f^S_{\psi_0} = \Psi_{01} + q \cdot n_0 \cdot V_0 \neq 0, \tag{41}$$

$$f^S_{\psi_1} = \Psi_{10} + \Psi_{12} + q \cdot n_1 \cdot V_1 = 0, \tag{42}$$

$$f^S_{n_0} = I_{01} \neq 0, \tag{43}$$

$$f^S_{n_1} = I_{01} + I_{12} = 0, \tag{44}$$

with

$$\Psi_{i,j} = \Psi(\psi_i, \psi_j) = -\Psi_{ji}$$

$$I_{i,j} = I(n_i, n_j, \psi_i, \psi_j) = -I_{ji}.$$

At the boundary, the constitutive relations are

$$f_{\psi_0} = \psi_0 - \psi_C = 0, \quad (45)$$
$$f_{n_0} = n_0 - N_0 = 0, \quad (46)$$
$$f_{I_C} = I_C + f^S_{n_0} = 0, \quad (47)$$
$$f_{Q_C} = Q_C + f^S_{\psi_0} = 0. \quad (48)$$

The boundary constitutive relations will be used to determine the quantity values at the boundary while the segment constitutive relations will be used to build up an expression for the boundary charge Q_C and for the boundary current I_C. This is done by the boundary models which set the appropriate entries in the transformation matrix $\mathbf{T}_B$. The solution vector $\mathbf{x}$ contains the following quantities

$$\mathbf{x} = (\psi_\mathbf{0}, \psi_\mathbf{1}, \ldots \mathbf{n_0}, \mathbf{n_1}, \ldots \psi_C, \mathbf{I}_C, \mathbf{Q}_C, \ldots)^T. \quad (49)$$

For voltage controlled contacts with V_0 applied to the contact one gets (50), when applying the current I_0 to the contact f_{ψ_C} changes to (51).

$$f_{\psi_C} = \psi_C - V_0 = 0, \quad (50)$$
$$f_{\psi_C} = I_C - I_0 = 0. \quad (51)$$

The contact related entries in the three matrices ($\mathbf{A}_b + \mathbf{T}_b$. $\mathbf{A}_s$) are assembled as follows (the complete equations such as Eqs. 42 or 44 are assembled in $\mathbf{A}_s$ only and have $\mathbf{T}_b$ diagonal entries of 1):

$\mathbf{A}_b$	ψ_0	n_0	I_C	Q_C	ψ_C	see	$\mathbf{T}_b$	ψ_0	n_0	I_C	Q_C	ψ_C
ψ_0	-1	0	0	0	0	(45)	ψ_0	0	0	0	0	0
n_0	0	-1	0	0	0	(46)	n_0	0	0	0	0	0
I_C	0	1	-1	0	0	(47)	I_C	0	1	0	0	0
Q_C	0	0	0	-1	0	(48)	Q_C	1	0	0	0	0
ψ_C	0	0	-1	0	0	(51)	ψ_C	0	0	0	0	0

$\mathbf{A}_b$	ψ_0	n_0	I_C	Q_C	ψ_C	see	transferred to
ψ_0	$-\partial\Psi_{01}/\partial\psi_0$	-q·V_0	0	0	0	(41)	Q_C
n_0	$-\partial I_{01}/\partial\psi_0$	$-\partial I_{01}/\partial n_0$	0	0	0	(43)	I_C
I_C	0	0	0	0	0		
Q_C	0	0	0	0	0		
ψ_C	0	0	0	0	0		

The compiled linear equation system for iteration step k is

$\mathbf{A}$	ψ_0	n_0	ψ_C	I_C	Q_C	RHS
ψ_0	-1	0	1	0	0	$f^k_{\psi_0}$
n_0	0	-1	0	0	0	$f^k_{n_0}$
ψ_C	0	0	0	-1	0	$f^k_{\psi_C}$
I_C	$-\partial I_{01}/\partial\psi_0$	$-\partial I_{01}/\partial n_0$	0	-1	0	$f^k_{I_C}$
Q_C	$-\partial\Psi_{01}/\partial\psi_0$	$-$q·V_0	0	0	-1	$f^k_{Q_C}$

As the constitutive relations for the quantities ψ_0, n_0, and I_C are preeliminated one ends up with the following equation for ψ_C:

$j_{x,\,y}$	ψ	RHS
ψ_C	$-\partial I_{01}/\partial\psi_0$	$f^k_{\psi_c} - f^k_{I_C} - f^k_{\psi_0} \cdot \partial I_{01}/\partial\psi_0 + f^k_{n_0} \cdot \partial I_{01}/\partial n_0$

References

1. Institut für Mikroelektronik, Technische Universität Wien, Austria (2004) Minimos-NT 2.1. User's Guide. http://www.iue.tuwien.ac.at/software/minimos-nt
2. Selberherr S (1984) Analysis and simulation of semiconductor devices. Springer, Wien
3. Stratton R (1962) Diffusion of hot and cold electrons in semiconductor barriers. Phy Rev 126(6):2002–2014
4. Bløtekjær K (1970) Transport equations for electrons in two-valley semiconductors. IEEE Trans Electron Devices ED-17(1):38–47
5. Grasser T, Kosina H, Gritsch M, Selberherr S (2001) Using six moments of Boltzmann's transport equation for device simulation. J Appl Phys 90(5):2389–2396
6. Wachutka G (1990) Rigorous thermodynamic treatment of heat generation and conduction in semiconductor device modeling. IEEE Trans Computer-Aided Design 9:1141–1149
7. Grasser T, Selberherr S (2001) Fully-coupled electro-thermal mixed-mode device simulation of SiGe HBT circuits. IEEE Trans.Electron Devices 48(7):1421–1427
8. Grasser T, Tang T, Kosina H, Selberherr S (2003) A review of hydrodynamic and energy-transport models for semiconductor device simulation. Proc IEEE 91(2):251–274
9. VanRoosbroeck W (1950) Theory of flow of electrons and holes in germanium and other semiconductors. Bell Syst Techn J 29:560–607
10. Schroeder D (1994) Modelling of interface carrier transport for device simulation. Springer Berlin Heidelberg New York
11. Simlinger T (1996) Simulation von Heterostruktur-Feldeffekttransistoren. Dissertation, Technische Universität Wien. http://www.iue.tuwien.ac.at.
12. Fischer C (1994) Bauelementsimulation in einer computergestützten Entwurfsumgebung. Dissertation, Technische Universität Wien. http://www.iue.tuwien.ac.at.
13. Scharfetter D, Gummel H (1969) Large-signal analysis of a silicon read diode oscillator. IEEE Trans.Electron Devices ED-16(1):64–77
14. Deuflhard P (1974) A modified newton method for the solution of ill-conditioned systems of nonlinear equations with application to multiple shooting. Numer Math 22:289–315
15. Bank R, Rose D (1981) Global approximate Newton methods. Numer Math 37:279–295
16. Van der Vorst H (1992) 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):631–644
17. Saad Y, Schultz M (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Stat Comput 7(3):856–869
18. Saad Y (1990) A basic tool kit for sparse matrix computations. Moffett Field, CA 94035: Technical Report, RIACS, NASA Ames Research Center
19. Ascher U, Markowich P, Schmeiser C, Steinrück H, Weiss R (1986) Conditioning of the steady state semiconductor device problem. Tech Rep 86-18, University of British Columbia

20. Fischer C, Selberherr S (1994) Optimum Scaling of Non-symmetric jacobian matrices for threshold pivoting preconditioners. In: International workshop on numerical modeling of processes and devices for integrated circuits NUPAD V, (Honolulu), pp 123–126
21. Barrett R, Chan MBT, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C, Van der Vorst H (1994) Templates for the solution of linear systems: building blocks of iterative methods. SIAM, Philadelphia
22. McMacken J, Chamberlain S (1989) CHORD: A modular semiconductor device simulation development tool incorporating external network models. IEEE Trans Computer-Aided Design 8(8):826–836
23. Rollins J, Choma J (1988) Mixed-mode PISCES-SPICE coupled circuit and device solver. IEEE Trans Computer-Aided Design 7:862–867
24. Engl W, Laur R, Dirks H (1982) MEDUSA - a simulator for modular circuits. IEEE Trans Computer-Aided Design Int Circuits Syst 1(2):85–93
25. Mayaram K, Pederson D (1992) Coupling algorithms for mixed-level ciruit and device simulation. IEEE Trans Computer-Aided Design 11(8):1003–1012