Making a PROPHET

Conor Rafferty[1], **R. Kent Smith**

Abstract: The PROPHET simulator is a software system for solving partial differential equations (PDEs) in time and 1,2 or 3 space dimensions. When equipped with appropriate modules, it can be configured as a process simulator or a device simulator, with application to modeling semiconductor fabrication processes and transistor behavior. The simulator is designed with three main goals: efficiency, geometric flexibility, equation extensibility. The first two distinguish it from canned packages such as Mathematica, which do not easily allow the use of arbitrary shapes or grids and are not tuned to solve systems with 10^5 or 10^6 unknowns. The third distinguishes it from previous application-specific simulators for semiconductor problems, such as PISCES or SUPREM-4. The simulator has been used in production for several years to predict semiconductor manufacturing processes.

1 Introduction

The PROPHET simulator is a software system for solving partial differential equations (PDEs) in time and 1,2 or 3 space dimensions. (Ordinary differential equations can be solved as a degenerate case). The PDEs are discretized using either finite elements or finite volume methods in space and with implicit methods in time, reducing the differential equations to a large system of algebraic equations. At each timestep the algebraic equations are solved by Newton's method. The matrix resulting from the linearization is solved by sparse iterative or direct methods.

The simulator is intended for use in production applications, and is therefore designed with maximum efficiency as a goal. The matrix assembly code takes advantage of the vector hardware capabilities available on machines such as the Cray YMP, allowing the solution of 3D problems in reasonable time. Two dimensional problems are routinely solved on a workstation. For most problems, the computational bottleneck has been the linear solver. A compact interface to linear solvers has been implemented, and both our internal solver BLSMIP [Bank, Rose, and Fichtner (1983)] as well as several external solvers such as PETSc [ftp://info.mcs.anl.gov/pub/tech_reports/reports/ANL9511.ps.Z] have been tested with PROPHET .

A second major concern is to solve problems on arbitrary geometries. To this end, the PDE solver makes no assumptions about the domain but uses arbitrary combinations of elements to describe its shape, such as triangles and quadrilaterals in 2D, tetrahedra, bricks or prisms in 3D. There are relatively limited built-in grid generation facilities, but arbitrary grids can be read from other sources and calculations performed on those domains.

The final major design goal is to allow new equations to be specified and solved by a user or a model developer who may not be familiar with numerical methods. The ability to add new equations and even new types of equations to the simulator is particularly important for process simulation, where there is no consensus on the underlying equations describing solid-state diffusion, and new models are frequently proposed. As an example of formulating equations, the system described in Eq. 1 to solve for field Y in materials silicon and oxide with Y continuous at the interface between silicon and oxide, can be described with the following syntax. The coefficient γ and the grid definition are not part of the system definition *per se*, and would appear later in the input file.

$$\nabla\cdot(\varepsilon\nabla Y)+4KY^2=0 \tag{1}$$
$$Y=0 \quad \text{at odxide/exposed surface}$$
$$\partial Y/\partial X=-\gamma Y \quad \text{at silicon/bulk surface}$$

```
system name=test
  + sysvars=a
  + tmpvars=asquare
  + term0=-1*box_div.diffusion(epsfield,a|a)@{silicon,oxide}
  + term1=4.0*nodal.prod(asquare,kfield|a)@{silicon,oxide}
  + term2=dirichlet.default_dirichlet(0|a)@{oxide/exposed}
  + term3=interface.radiation(a|a)@{silicon/bulk}
  + term4=constraint.continuity(a|a)@{silicon/oxide}
  + nterm=5
  + func0=prod(a,a|asquare)@{silicon,oxide}
  + nfunc=1
```

There is one line of description for each term in the PDE, and one line for each boundary condition. The overall system is described in terms of a number of building blocks, each of which takes as input either primary variables or temporary variables which can be created on the fly as functions of other variables. The above example shows the use of a Laplacian operator, nodal operator, Dirichlet boundary condition, flux boundary condition, an interface jump condition (albeit the trivial one), and generation of a temporary field. Many other building blocks are available, such as the $\partial/\partial t$ operator, central and upwind drift operators, Scharfetter-Gummel operator, and new building blocks can easily be constructed.

The intention is that the code can be used by three categories

[1] Bell laboratories, Lucent technologies, Room 1E-338, 600 Mountain Ave., Murray Hill, NJ 07974

of users: application users, model developers, and primary developers. Application users are primarily interested in running the code off-the-shelf but are interested in having some control over its execution, such as the choice of linear method. Model developers are interested in adding new equations, either by combining existing operators or building new operators according to specification. The primary developers are concerned with the underlying database, the implementation of the assembly library and the linear solvers, the grid structure, and so on.

Maintaining an uncluttered, coherent interface for the model developer is particularly important, since it is frequently the case that models must be developed by researchers who are experts in the physical mechanisms but less familiar with numerical methods. Since such users come from a variety of backgrounds, familiarity with any particular programming language is not assumed, and a subroutine binding to any language is possible.

In the subsequent sections, the grid structure, the internal database, and the PDE solver are described. A number of applications in process and device simulation are then given to illustrate the ability to solve relevant technological problems.

2 Grid structure

There is generally a trade-off between updating and accessing a grid structure; between the ease with which new elements, nodes, interface, can be added, and the speed with which they can later be retrieved. Following a chain of pointers to arrive at the coordinates of the first node, then a different chain to arrive at the coordinates of the second node, and so on, is slower than retrieving all coordinates by indexing a single array; however a fixed length array does not allow easy removal of nodes.

We have chosen to use dual representations of the grid, a dynamic representation and a static representation. The dynamic representation is the core representation which is stored between runs; it is minimal and unstructured. Elements of different types, nodes, boundary markers, can be added in any order. Elements are represented in plain finite element form; a list of nodes for each element. There is no hierarchy of elements containing faces, faces containing edges, or edges containing nodes. Nor are there structures describing extended objects such as interfaces or regions; instead each element has a region code and an interface code on its external faces. A distinction is maintained between points and nodes, following the example of SUPREM-IV[Law, Rafferty, and Dutton (1986)]. A point is a location in space; a node is a part of an element where solution data is stored. In the bulk of a region, one node corresponds to each point, but at material interfaces, there are as many nodes as there are materials meeting at the interface. This construct allows a natural representation of the many situations where fields are discontinuous across interfaces, with jump conditions describing the discontinuity (Fig. 1). Fields

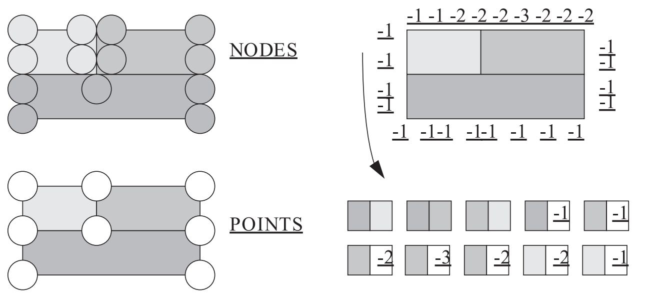


Figure 1 : Grid constructs: (a) Multiple nodes at one point. (b) Automatic generation of interface structures. Ten distinct interfaces exist in this structure.

continuous across the interface are handled in a uniform way, by applying a jump condition of equality, albeit at the (usually small) cost of carrying redundant solution values along the interface.

During the PDE solution phase, where rapid access to grid is desired, a static representation is automatically generated. Elements are reordered into regions, then into similarity groups for vector access (all triangles, all quads, etc). The similarity groups can optionally be subdivided into ‘colors’ where each color contains groups of elements, none of which share nodes, in order to allow vector processing of the color without risk of race conditions. At the end of the solution, the grid is then “unlocked” again for ease in grid manipulation. The cost of locking/unlocking is a tiny fraction of solution or grid generation cost.

There are also subroutines to decorate the core representation with derived information, such as all the neighbors of the elements, and the list of unique edges of the grid (useful for finite volume assembly). In addition, extended interface structures are built automatically at the start of the solver phase. Sets of contiguous elements sharing a common interface are identified and a list of lists of such elements is created, along with their boundary nodes. This structure is convenient for efficiently applying the interface jump conditions and other boundary conditions.

3 Database

PROPHET contains a number of physical models with coefficients which have been calibrated in the past or which require tuning by the simulator user. The coefficients may take the form of numbers, temperature dependent formulae, or tables of measured data. The first function of the PROPHET database is to store such coefficients. In addition it has been found convenient to store all control information for the simulator in the same database, including a description of the user’s input file, a description of the set of equations to be solved, and parameters for the numerical methods. All miscellaneous information

```
root {                                  % top of tree
      userinput {                       % user's parsed input file
            command1 {                  % first command
                  parm1                 % parameters of first command
                  parm2
            }
            command2...                 % second command
      }
      options {                         % control options, e.g.
            timestep                    % whether to print timesteps
      }
      library {                         % coefficient library
            physics {                   % physical coefficients
                  silicon {             % organized by material
                        ni              % e.g. intrinsic number
                        boron {         % solution field in material
                              Dix       % their parameters, e.g.
                        }               % diffusivity
                  }
            math {                      % numerical parameters
                  grid.ratio            % e.g. maximum size ratio of
                                        % adjacent grid cells
                  systems {             % packaged equation systems
                        sidiffusion {   % e.g. diffusion in silicon
                              ...
                        }

            cards {                     % recognized commands
                  solve {
                        time            % parameters of "solve" command
                        temperature
                  }
            cards.defaults {
                  }
      }
}
```

Figure 2 : Top level of PROPHET database.

pertaining to a simulation, as opposed to structured information such as the grid and matrix, is stored in the database.

The data is organized hierarchically. The leaves of the tree are called properties and the organizational branches are called property lists. At the top level, the hierarchy is structured as shown in Fig 2. The top level objects are the user's input file, options for this particular run, and the library. The library is divided into physical and numerical parameters, and the list of recognized commands and their defaults is also stored here. Physical parameters are organized according to material, with solution fields appearing under the materials in which they are to be solved. The description of the PDE system to be solved is stored in the numerical section of the library.

The database supports a notion of inheritance. A property list may contain a property with the name SeeAlso , which is a symbolic link to another property list. If a property is requested on the parent list, but not found, the SeeAlso pointer is followed to see if the referenced list contains the desired property. This process can recurse as necessary. In addition any property list might have more than one SeeAlso pointer (multiple inheritance). This facility makes it convenient to define a new material, or field, with the characteristics of another, while allowing one or two exceptional properties.

The advantages of this storage scheme is uniform and organized access to all simulation parameters, rather than the ad-hoc common blocks and static variables characteristic of older simulators. It also means that new modules can access user-defined parameters without making any arrangements for their storage or transmission elsewhere in the code. Only the subroutine requesting a parameter knows of its existence. Typically a single line of code is required to request a parameter, and the request is answered by one line in the user's input file defining the value of that parameter, or one line in the stored library. The low overhead in defining new parameters encourages module programmers not to hard-code coefficients or decisions, leaving as much as possible mutable outside the

$$\boxed{\frac{\partial C_I}{\partial t}} - \boxed{\nabla\bullet(D_I \nabla C_I)} + \boxed{k\,(C_I C_V - C_I^* C_V^*)} = 0$$
$$\boxed{\frac{\partial C_V}{\partial t}} - \boxed{\nabla\bullet(D_V \nabla C_V)} + \boxed{k\,(C_I C_V - C_I^* C_V^*)} = 0$$

Figure 3 : A block of equations representing point defect diffusion in silicon (without impurities). The block is decomposed into several terms: two laplacian terms, one binary-recombination term, and two transient terms.

binary.

Only the `library` subtree of the database is stored between runs. On startup, a new root is created in memory, and the disk library loaded onto the library branch of the in-memory database. The other top level branches are created and populated based on the input file (`options,userinput`). Database commands in the input file modify the in-memory database, adding new properties, system descriptions, and material parameters.

4 PDE solver

The system of PDE's to be solved is itself described by a property subtree, built on the fly from a `system` command as given in the introduction. To solve a PDE, the subtree is first examined to determine the solution fields, equations, and solution methods. The equations are then discretized using either finite elements or finite volume methods in space, with the choice determined by the operators specified in the system subtree. In time, implicit methods are most commonly used, although explicit methods have also been implemented. The result is to reduce the differential equations to a large system of algebraic equations. At each timestep the algebraic equations are solved by Newton's method. The matrix resulting from the linearization is then solved by sparse iterative or direct methods.

4.1 PDE formulation

A block of partial differential equations is considered as an assembly of operators, called `pdeterms`; for instance a reaction-diffusion system might consist of several reaction terms between various species, a diffusion term for each species, and a transient term for each species. A term-by- term decomposition of the equations for point defect diffusion in silicon is represented schematically in Fig. 3. `PDEterms` can be rectangular, not necessarily square, since they define a number n of outputs in terms of m inputs. Each of these `pdeterms` can be further subdivided into a "geometrical" part (`geoterm`) and a "physical" part (`phyterm`). As an example, the Laplacian operator can be considered as the combination of the divergence operator $\nabla\cdot$ and the flux of point defects $D\nabla C$. The geometrical operator `box_div` chooses a finite volume discretization of the divergence operator, while the operator `fel_div` choose a finite element discretization. In each case, the operator calls a `phyterm` to calculate the flux at the quadrature points (finite elements) or the edge midpoints (finite volume). The construction of the flux from the field values and gradients is *independent* of the discretization and is defined in terms of purely physical constructs such as the value of a field at a point in space and its spatial gradient. The representation of these are simply a scalar and a vector (of length equal to the space dimension), respectively; to calculate the flux at n points in space the same `phyterm` is used both in finite elements and finite differences; the `phyterm` takes as argument `gradsol[1..dim][1..n]` and return as result `flux[1..dim][1..n]`. Thus the *numerical* work of assembling the equations (the discretization), and the *modeling* work of defining fluxes in terms of concentrations and gradients, are separated in `PROPHET` . A model developer writes `phyterms` without knowing whether they will be called from a finite difference or finite element discretization, for a quadrature point in a triangle, a finite volume edge, or the interior of a brick. Space dimension affects only the length of the vectors. The input to the `phyterms` in all cases is a list of field names, field values, field gradients, the space dimension and the region name, and the output is a scalar or vector as well as its derivative with respect to the inputs.

This clean separation is compromised in one case: the Gummel-Scharfetter discretization of the semiconductor device equations. Defined only for edges in a finite volume discretization, the discretization needs not nodal quantities and their gradients, but the net difference along the length of the edge. The divergence operator for this case has been coded to pass the nodal differences rather than the gradients.

Flux boundary conditions are treated uniformly in this system. Another term added to the system list specifies an interface `geoterm` which in turn calls a `phyterm` describing the flux at an interface in terms of the field values at the interface. The input to an interface `phyterm` is the same as for a bulk `phyterm` except that the field value arrays are double length, to provide values on both sides of an interface, and two interface names instead of a single region name are provided.

Dirichlet boundary conditions and jump conditions call for separate handling. Since the updates at Dirichlet nodes are always zero, the appropriate (off-diagonal) rows and columns in the matrix and residual vector are zeroed out. Jump conditions are handled by first assembling the matrix and residual vector as if there were no jump conditions. Then in a cleanup pass, all the PDE terms are transferred into the matrix row of the lowest numbered node number, and the jump conditions are written into the rows corresponding to the other nodes at the same location, with derivatives of the jump equations with respect to nodal values being added to the matrix. Arbitrary nonlinear jump conditions involving combinations of variables

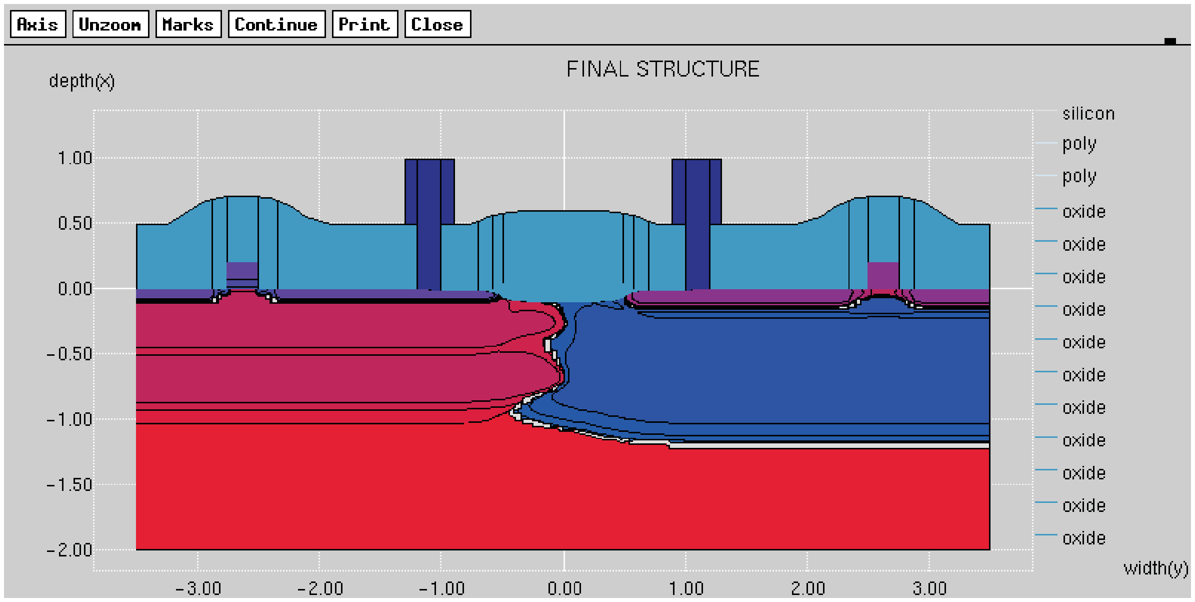


Figure 4 : Large diffusion example. A full CMOS process flow for two transistors (nMOS and pMOS) as well as the isolation between them is simulated in under an hour on a workstation.

at the interfaces are therefore handled straightforwardly. The jump conditions are implemented by `phyterms` with the usual calling sequence.

4.2 *Auxiliary fields*

It is frequently convenient to define auxiliary fields and reuse them more than once in the formulation of a system of PDEs. Not only does this simplify the writing, it avoids unnecessary recomputation. For instance, the traditional impurity diffusion equation of boron in silicon is written in Eq. 2-4.

$$\frac{\partial C_c}{\partial t} = \nabla \cdot D_B(\psi)\,[\nabla C_a - C_a \nabla \psi] \qquad \text{Diffusion equation} \quad (2)$$

$$C_a + \beta C_a^m = C_c \qquad \text{Solubility relationship} \quad (3)$$

$$\psi = -\log\left(\frac{C_a}{2n_i} + \sqrt{\left(\frac{C_a}{2n_i}\right)^2 + 1}\right) \qquad \text{Normalized Potential} \quad (4)$$

There is one differential equation (the diffusion equation) for one unknown, the *chemical concentration* C_c of boron. The flux of boron is however most conveniently defined in terms of its *active* concentration C_a, and the potential ψ, which are defined by the two auxiliary equations. PROPHET allows the definition of such variables, and eliminates them automatically as the matrix is being constructed. Derivatives such as ∇C_a are eliminated by applying the chain rule. Such elimination reduces the size of the matrix to be solved, conserving storage and CPU time.

A feature of the system description language is that fields may be marked optional. At the start of a solution phase, the fields existing in the structure are compared with the fields on the solution list, and any optional fields which are not present in the structure dropped from the solution list. This allows a single system description to cover the diffusion of one or half a dozen dopants in silicon.

4.3 *Numerical methods*

The main time integration method is TR-BDF2[Bank, Rose, and Fichtner (1983)]. The time step is automatically chosen based on Milne's method applied to the comparison of TR-TR and TR-BDF2 methods at each timestep, in order to achieve a desired temporal error.

If after solving a given timestep, the temporal error is found to be significantly more than the desired error tolerance, the timestep is rejected and restarted with a shorter step. While this precaution may seem obvious, some older simulators accept any solved timestep and cut back the *next* timestep if the error is large. The latter strategy can propagate poor solution values.

A Newton's method with systematically tightening inner solution tolerance[Bank and Rose (1981)] is used for the solution of the system of nonlinear algebraic equations arising from the discretization.

Each field-to-field coupling block in the Jacobian matrix is stored in Bank-Smith sparse format[Bank, Rose, and Fichtner (1983)], with diagonal entries first, upper triangle following, then lower triangle. The full Jacobian is the catenation of all the sparse blocks, each of which may be null, diagonal, symmetric, or asymmetric.

Table 1 : CPU-time for device simulation

Device: JFET with 2 carriers, full Newton, 2556 nodes; Hardware: Sun UltraSparc 170						
	Total Time	#Newton Loops	#Linear Solves	Solve Time	Assembly Time	Time per solve
PROPHET+BLSMIP	237s	93	93	186s	26s	2.55s
MEDICI	387s	126	71	-	-	5.5s
PROPHET+PETSC	655s	93	93	602s	27s	7.0s

Most of our experience with linear solvers has been with the Bell Labs Sparse Matrix package (BLSMIP)[Bank, Rose, and Fichtner (1983)]. The preferred method for diffusion systems has been found to be ABF[Bank, Chan, Coughran, and Smith (1989)] with an incomplete LU decomposition preconditioner and CGS as the accelerator. For device simulation in 2D, direct solution with a minimum local fill ordering[Vlach and Singhal (1983)] is optimal. For external solvers, mapping from the internal format to external format is generally a small fraction of the solution time, and we have also had good success using the PETSc solver from Argonne Laboratories [ftp://info.mcs.anl.gov/pub/tech_reports/reports/ANL9511.ps.Z].

5 Applications

5.1 Performance

It is a frequent concern that flexibility in setting up the solution system may exact a high price in execution time. In our experience, the flexibility costs only a slight parsing overhead and a few high level if-statements, which are negligible compared to the cost of processing thousands of floating point numbers. As an example of this, a large diffusion example is shown in Fig. 4. This example is the result of simulating a full CMOS process flow for two transistors (nMOS and pMOS) as well as the isolation between them. There were four fields solved over 14,000 nodes and a total of 135 composite timesteps or 405 individual timesteps, each of which required 1-5 Newton iterations. The total execution time was 49 minutes on a Sun UltraSparc-II/300 workstation. This compares very favorably with commercially available tools, which frequently take over an hour to solve even a half transistor.

As a further example, Tab. 1 gives performance comparisons for device simulation using PROPHET and MEDICI, the most widely used commercially supported specialized device analysis tool. The simulation takes a JFET through its on-off curve. The CPU time is entirely dependent on the linear solver used. These results are quite satisfactory, particularly in the light that the device simulation modules are relatively new in PROPHET and do not take advantage of smart initial guesses or Jacobian recycling strategies.

5.2 Process simulation

Accurate modeling of deep submicron devices requires tracking the diffusion not only of the dopants but also of the intrinsic point defects in silicon[Fahey, Griffin, and Plummer (1989)]. The defect-dopant coupled diffusion system is listed in Eq. 5-10. Interstitials I diffuse rapidly and pair with dopants D to create a flux of dopant-defect pairs F_{DI} which transports the otherwise immobile dopants; a similar mechanism applies to dopant-vacancy pairs. An electrostatic potential ψ is generated by the ionized dopants, and the potential gradient (electric field) gives rise to another term in the dopant flux. There are two parallel equations for interstitials I and vacancies V, and one equation for each dopant D. Finally, interstitials may cluster into extended defects containing a concentration C of interstitials.

$$\frac{\partial I}{\partial t} = \nabla\cdot(D_I\nabla I) - \nabla\cdot F_{DI} - k_R(IV - I^*V^*) - f_c(I,C) \tag{5}$$

$$\frac{\partial V}{\partial t} = \nabla\cdot(D_V\nabla V) - \nabla\cdot F_{DV} - k_R(IV - I^*V^*) \tag{6}$$

$$\frac{\partial D}{\partial t} = -\nabla\cdot(F_{DI} + F_{DV}) \tag{7}$$

$$F_{DX} = -f_x D_D\left(\frac{X}{X^*}(\nabla D + \xi_D D\nabla\psi) + D\nabla\frac{X}{X^*}\right) \quad \text{for } X = \{I,V\} \tag{8}$$

$$\frac{\partial C}{\partial t} = f_C(I,C) = k_f IC - k_r C \tag{9}$$

$$\nabla\cdot(\varepsilon\nabla\psi) = -q\,(p - n + \xi_D D) \tag{10}$$

The last term of the dopant flux relation for F_D shows that impurities may move under the action of an interstitial gradient. Such a gradient exists at the silicon surface during annealing of implant damage, where surface recombination effectively removes interstitials[Lim, Rafferty, and Klemens (1995)], creating an interstitial sink, while there is a strong source at the location of the most recent implants. The resulting gradient of interstitials leads to a flux of dopant to the surface, where it piles up inhomogeneously and affects the threshold voltage of transistors[Rafferty, Vuong, Eshraghi, Giles, Pinto, and Hillenius (1993)].

An interesting consequence of this effect is that the threshold voltage of a transistor may depend on its layout. Fig. 5 shows a conventional layout, as well as the layout for a power transistor. In the power device, the waffle-gate openings present a weaker source of interstitials relative to the amount of recombining surface available at the corners, as compared to the straight edges of the conventional layout. As a result, there is less interstitial-driven dopant pileup at the corners of the source/drain openings and a path exists for current to flow earlier there than along the straight edges (Fig. 6). Consequently,

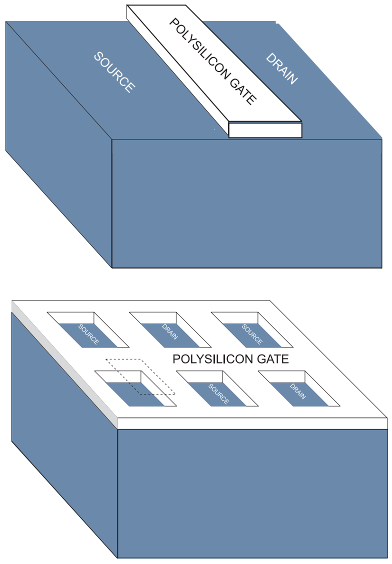


Figure 5 : Gate layout for conventional and power MOSFETs. The waffle structure maximizes current density.

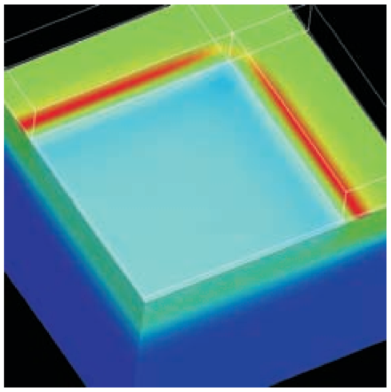


Figure 6 : Boron diffusion profile near corner of implant window. The boron ridge appearing along the gate edge is reduced near the corner and allows carriers to sneak through, lowering the threshold voltage relative to a linear device.

the threshold voltage for such waffle-gate devices is predicted to be lower than for conventional devices; this effect has been measured experimentally[Darwish, Rafferty, Williams, Cornell, and Yilmaz (1995)].

The ability to predict such complex defect-dopant interactions is of great utility in designing dopant profiles in deep submicron technology. The cost of running such simulations, at least in two dimensions, is no more than an hour or so on a workstation. This combination has led to substantial use of simulation and interesting diffusion discoveries by process development engineers[Chaudhry, Rafferty, Nagy, Chyan, Carroll, Chen, and Lee (1997); Kamgar, Vuong, Liu, Rafferty, and Clemens (1997)]. Current usage is of the order of 5,000 hits a month.

5.3 Device simulation

The ever-thinning gate dielectric in MOSFETs has brought quantum effects at the silicon/oxide interface into prominence[Krisch, Bude, and Manchanda (1996)]. The electron and hole wavelengths are no longer short in comparison to the dielectric thickness. Such effects have been modeled in one dimension using a rigorous coupled solution of the Schrödinger and Poisson equations. Extending such models to two dimensions and including transport is a formidable computational challenge.

$$-\nabla\cdot(\varepsilon\nabla\psi) = q\left(p-n+N_D^+-N_A^-\right) \qquad (11)$$

$$\nabla\cdot(n\mu_n\nabla\phi_n) = g-r \qquad (12)$$

$$\nabla\cdot(-p\mu_p\nabla\phi_p) = g-r \qquad (13)$$

$$\phi_n-\psi+\frac{kT}{q}\ln\frac{n}{n_i} = 2b_n\left(\frac{\nabla^2\sqrt{n}}{\sqrt{n}}\right) \qquad (14)$$

$$\phi_p-\psi-\frac{kT}{q}\ln\frac{p}{n_i} = -2b_p\left(\frac{\nabla^2\sqrt{p}}{\sqrt{p}}\right) \qquad (15)$$

The density gradient approach to quantum effects[Ancona and Iafrate (1989)] provides a more compact approach to solving such problems. It modifies the governing PDE's of device simulation in a fundamental way and its inclusion in a specialized device simulation package would require major revisions. However as a set of drift-diffusion PDE's, it can be readily be solved by PROPHET . The steady-state device equations with DG corrections[Rafferty, Biegel, Yu, Ancona, Bude, and Dutton (1998)] are shown in Eq. 11-15. Potential ψ, the quasi-fermi levels ϕ_n, ϕ_p and the phaseless electron/hole amplitudes $\sqrt{n}$, $\sqrt{p}$ become the fundamental solution variables. The generation and recombination rates g, r are functions of the carrier densities n, p and possibly electric field, and the mobilities μ_n, μ_p are functions of doping N_A, N_D and electric field.

The above system was scripted into PROPHET and the quantum coefficients b_p, b_n calibrated to give good fits to capacitance data on large area testers[Rafferty, Biegel, Yu, Ancona, Bude, and Dutton (1998)]. The same coefficients gave good agreement to capacitance data for capacitors with thicknesses ranging from 20Å to 80Å. The model was then applied to the type of situation which would be very expensive to analyze with a full Schrödinger-Poisson analysis: gate oxides with finite roughness. Fig. 7 shows the simulated structure, of which one period was simulated. A comparison of the classical analysis and quantum analysis (Fig. 8) shows that the quantum solution is much less sensitive to roughness, provided the roughness wavelength is shorter than the electron wavelength. This

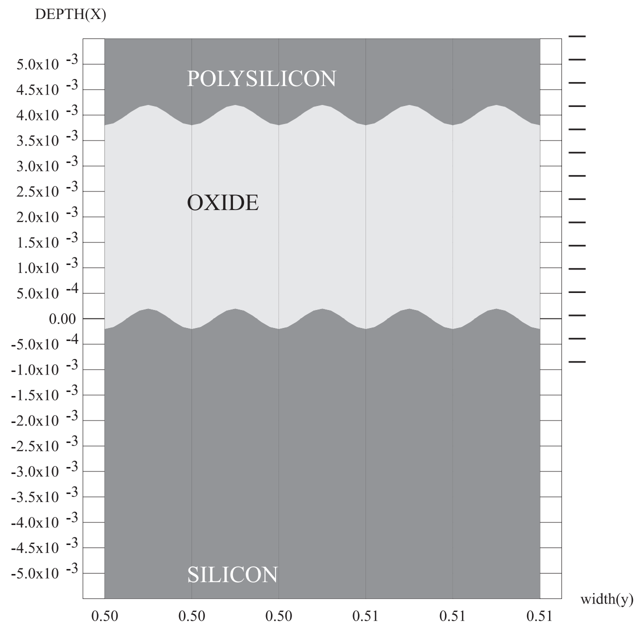


Figure 7 : Gate with rough surface. The amplitude of the roughness is 2Å and the wavelength 20Å. For this calculation the upper and lower surfaces were assumed to vary synchronously; other types of roughness could also be considered.

suggests that in monitoring and controlling surface roughness, most attention should be paid to longer wavelength components.

6 Summary

The PROPHET simulator was designed as a development platform for solving the sets of partial differential equations which arise in semiconductor process and device modeling. It uses a compact equation language and a hierarchical database to allow the rapid prototyping of new systems of equations. Systems can be formulated entirely using existing operators in some cases, and where necessary new physical operators can be added according to a well-defined interface. Both finite element and finite difference discretizations are provided to achieve the necessary accuracy and speed in different applications. The numerics are streamlined for solving large problems, allowing it to answer real-life questions. It has been instrumental in making significant discoveries in diffusion research and device analysis.

Acknowledgement: It is a pleasure to acknowledge contributions to the PROPHET code from many collaborators, particularly Martin Giles, Hong-Ha Vuong, Ashraf Alam, Bryan Biegel, Zhiping Yu, Anand Pardhanani and Abhijhit Bose. We are indebted to our users for their suggestions and their patience, particularly Samir Chaudhry, Hans Gossmann, and Gerhard Hobler. The device simulation work was supported by DARPA contracts DABT63-95-C-0090 and DABT63-96-C-0067 in collaboration with Prof. Robert Dutton at Stanford University and Prof. Graham Carey at University of Texas at Austin, respectively.

References

Ancona, M.; Iafrate, G. (1989): Quantum correction to the equation of state of an electron gas in a semiconductor. *Phys. Rev. B*, vol. 39, no. 13, pp. 9536.

Bank, R.; Chan, T.; Coughran, W.; Smith, R. (1989): The alternate-block-factorization procedure for systems of partial-differential equations. *BIT*, vol. 29, no. 4, pp. 938.

Bank, R.; Rose, D. (1981): Global approximate newton methods. *Numerische Mathematik*, vol. 37, pp. 279.

Bank, R.; Rose, D.; Fichtner, W. (1983): Numerical methods for semiconductor device simulation. *IEEE Transactions on Electron Devices*, vol. ED-30, pp. 1031.

Chaudhry, S.; Rafferty, C.; Nagy, W.; Chyan, Y.; Carroll, M.; Chen, A.; Lee, K. (1997): Suppression of reverse short channel effect by high energy implantation. *Technical Digest of the International Electron Device Meeting IEDM*, pg. 679.

Darwish, M.; Rafferty, C.; Williams, R.; Cornell, M.; Yilmaz, H. (1995): Transient enhanced threshold shifts in power MOS transistors. *Technical Digest of the International Electron Device Meeting IEDM*, pg. 233.

Fahey, P.; Griffin, P.; Plummer, J. (1989): Point defects and dopant diffusion in silicon. *Reviews of Modern Physics*, vol. 61, no. 2, pp. 289.

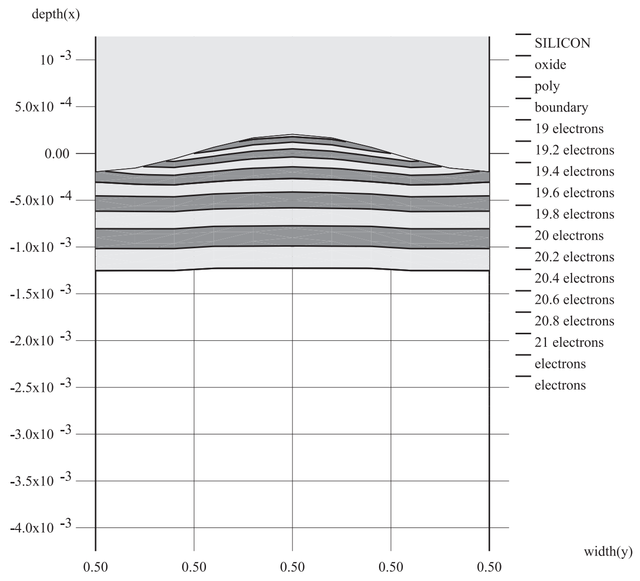
Kamgar, A.; Vuong, H.-H.; Liu, C.; Rafferty, C.; Clemens, J. (1997): Impact of nitrogen implant prior to the gate oxide growth on transient enhanced diffusion. *Technical Digest of the International Electron Device Meeting IEDM*, pg. 695.

Krisch, K.; Bude, J.; Manchanda, L. (1996): Gate capacitance attenuation in MOS devices with thin gate dielectrics. *Electron Device Letters*, vol. 17, no. 11, pp. 521.

Law, M.; Rafferty, C.; Dutton, R. (1986): New n-well fabrication techniques based on 2d process simulation. *Technical Digest of the International Electron Device Meeting IEDM*, pg. 518.

Lim, D.; Rafferty, C.; Klemens, F. (1995): The role of the surface in transient enhanced diffusion. *Appl. Phys. Lett.*, vol. 67, pp. 2302.

Rafferty, C.; Vuong, H.-H.; Eshraghi, S.; Giles, M.; Pinto, M.; Hillenius, S. (1993): Explanation of reverse short channel effect by defect gradients. *Technical Digest of the International Electron Device Meeting IEDM*, pg. 311.



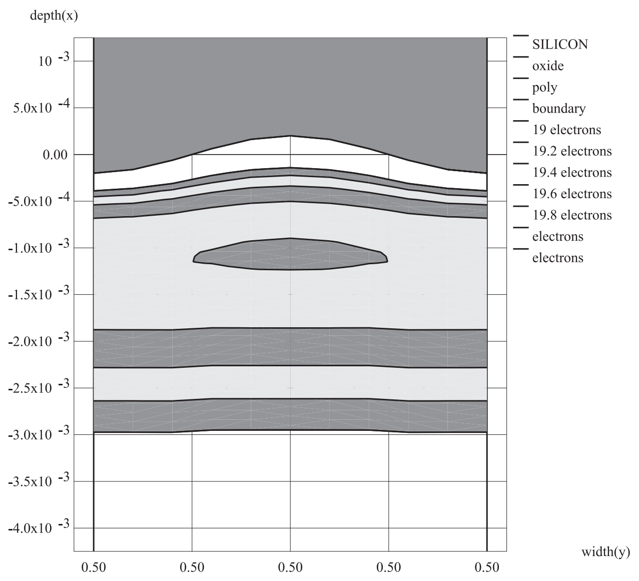


Figure 8 : (a) Classical solution of rough structure shows all electrons collecting at points of maximum potential. (b) Density-gradient quantum mechanical treatment shows much less influence of surface roughness on electron density. The finite electron wavelength prevents clustering of electrons in the nooks and crannies of the surface.

Rafferty, C. S.; Biegel, B.; Yu, Z.; Ancona, M.; Bude, J.; Dutton, R. (1998): Multi-dimensional quantum effect simulation using a density-gradient model and script-level programming techniques. *SISPAD Conference Proceedings*, pg. 137.

Vlach, J.; Singhal, K. (1983): Computer methods for circuit analysis and design. *Van Nostrand Reinhold*, pp. 66–67.