ELSEVIER

Computational Materials Science 12 (1998) 289–308

COMPUTATIONAL MATERIALS SCIENCE

Continuum based modeling of silicon integrated circuit processing: An object oriented approach

Mark E. Law [a,*], Stephen M. Cea [b,1]

[a] *Department of Electrical Engineering, University of Florida, Gainesville, FL 32611-6120, USA*
[b] *TCAD Department, Intel Corporation, Hillsboro, OR, USA*

Abstract

Continuum based integrated circuit process modeling is the dominant tool used to investigate and understand integrated circuit (IC) development. This paper describes the commonly used models for implantation, diffusion, and material growth. In addition, the supporting numerical techniques are described. This paper focuses on the implementation in object oriented code, Florida Object Oriented Process Simulator (FLOOPS). The software architecture is described for implementing models and numerics. A number of process examples are introduced and discussed. © 1998 Elsevier Science B.V. All rights reserved.

1. Introduction

Integrated Circuit (IC) process modeling is a fast changing field, driven by new technology development. As technology developers continue to design new processes, software modeling codes must be adapted to model these new fabrication steps. In addition, technology shrinks can invalidate old models and require substantial new modeling and physics to be added to the software. As a final complication, the grid needs to be dynamic and revised with each process step since the structure is being changed at each step. This requires software packages to be easy to develop and adapt, because the code never becomes static.

To address these challenges, we developed FLOOPS (FLorida Object Oriented Process Simulator). FLOOPS uses C++ and class hierarchy to represent not only the mesh and materials, but the physical models as well. This allows new models to be implemented easily, since they can be derived from similar models. It also allows the grid code to be separated from the physics fairly completely, because the physics modeling code only needs to interact with the grid at a fairly abstract level. This allows simultaneous development of new physical models along with advanced grid algorithms to handle the developing structural information.

This paper describes FLOOPS and the models contained for use in simulating advanced semiconductor processing. Because the end user is a technology developer, trade-offs have to be made with respect to accuracy, CPU time, and ease of use. Although atomic-scale modeling offers more accuracy and physical insight, continuum modeling will continue to be the major approach for most industrial engineers. Each of the major modeling capabilities (implant, diffusion, and material growth) are described along with the mesh and material representation.

* Corresponding author. Tel.: +1 904 392 6459; e-mail: law@tcad.ee.ufl.edu.
[1] E-mail: scea@ptdcs2.intel.com.

0927-0256/98/$ – see front matter © 1998 Elsevier Science B.V. All rights reserved.
PII: S0927-0256(98)00020-2

2. Mesh and material classes

Any finite element or finite volume simulator needs to discretize space into unique positions and elements that connect the position. The numerical accuracy of the final solution depends on the placement on connections, so this becomes a critical piece of the simulation package. In IC process simulation, it is further complicated by the fact oxidation and silicidation processes move the material boundaries. Also, typical users wish to simulate phenomena in one, two, and three dimensions depending on what effects they are examining. It is very useful to have a single multi-dimensional simulator so that the user can have confidence that the same models are being used in every case.

Most finite element solvers employ only a node element table. This representation makes it more difficult to implement multi-dimensional simulation, since it becomes hard to access edge information in reliable way. For example, in a two-dimensional simulation edges are kept in an implied manner as an indexable pair of nodes in a triangle. Traversing all edges once becomes a difficult chore, and becomes very hard in three-dimensions for a tetrahedral element. FLOOPS stores a richer set of objects, including all the dimensional pieces. This takes more storage than the traditional approach, but makes it easy to traverse edges in any dimension.

2.1. Object description

Fig. 1 shows the overall structure of the mesh, data, and grid representation. The FieldServer object contains methods that allow access to all of the grid and spatial information. FLOOPS typically has one active fieldserver that is a global variable that is operated on by all user commands. The FieldServer contains a list of Coordinate objects, Mesh objects, and Data objects. The Coordinate object represents a single position in space. The Coordinate object includes pointers to one or more Node objects. Each Mesh object incident on a coordinate will have a different node, so that

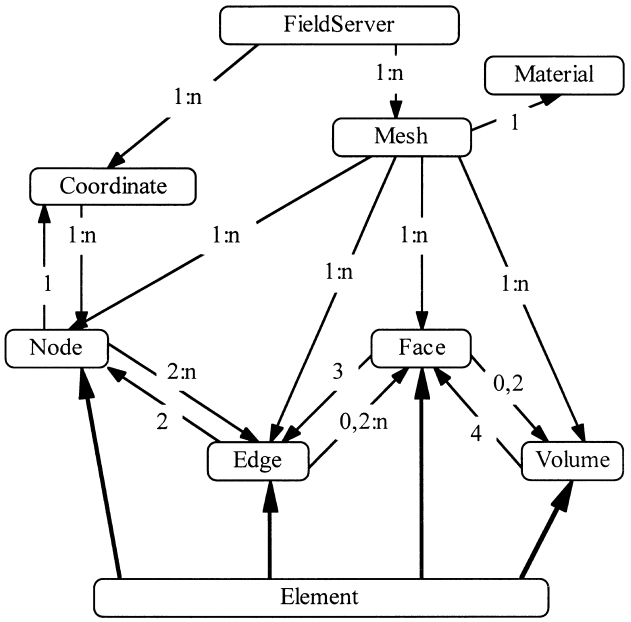

Fig. 1. Class hierarchy for the objects that represent grid and data in the simulator. Derivation is indicated by the bold arrow. When one object contains a collection of other objects, this is indicated by regular arrows, and the range above the line indicated the size of the collection.

boundary Coordinates point to multiple Node objects and internal Coordinates point to only one Node object. This approach is the same as used in SUPREM-IV [1], and allows for abrupt changes in the solution values when the material changes. This is an important requirement for process simulation, since dopant concentrations are discontinuous across material boundaries.

Each Mesh object is made up of a pointer to a material, a list of nodes, edges, faces, volumes, and one or more boundary representations. Each mesh corresponds to a single material, and multiple meshes may exist with the same material in a single fieldserver object. This happens frequently when regions are etched and form discontinuous pieces. Meshes may include an internal hole so that voids can be represented, and is the reason that there may be one or more boundaries of the mesh. Each mesh object contains all of the internal grid representation pieces in addition to the external boundary.

There are also special meshes implemented to represent material interfaces. These meshes are embedded objects that are of dimension one smaller than the overall simulation. For example, in two-dimensional simulation, interface meshes are one-dimensional lines. There are two main reasons for interface meshes. First, they allow easy specification of boundary conditions. As will be discussed in more detail later, equations are associated with mesh and solution information, and having an interface mesh allows a boundary condition to be associated with a mesh rather than the edge of a mesh. The second reason is that there are cases where interfacial segregation occurs as a phase, and these meshes allow storage of dose information directly on the interface.

Each Node object contains a pointer to the single coordinate that represents the physical location of the node. The node also contains a list of all the edges incident upon it. These edges are ordered, so that the near and far end can be determined uniquely. An Edge in the mesh is made up of two nodes, and a list of all faces incident. A Face contains pointers to three or more Edges, each of which is ordered in a counterclockwise fashion. Each Face contains a list of all Volumes incident. Volumes are made up of four or more Faces.

Each of the grid objects (Node, Edge, Face, Volume) contain pointers to the objects of one greater and one lesser dimension. The pointers to the objects of one lower dimension define the object, and the pointers to objects of one greater dimension are used to maintain traversal routines. For example, an edge contains pointers to two nodes, and to a list of faces incident on the edge. In a one-dimensional simulation, an edge has an empty set of faces incident. There is no direct list of nodes contained in a face, although the information is available through the ordered edges.

Nodes, Edges, Faces, and Volumes are derived from the Element base-class. This parent class is used to define virtual functions that operate for refinement, removal, and other operations. Any algorithm written on the element class will work for any dimensional simulation. Also, any algorithm that is written for edges will work in any dimension, since they are available in all dimensions. For example, if an edge is split, the edge notifies all faces attached of the change, and each face notifies all volumes attached of the change. In this way, first order dimensional independence of the code is achieved.

In addition to connectivity and position information, it is necessary to store data on each grid object. For example, finite-volume coupling coefficients are stored for each edge, and doping concentration is needed at each node. To handle data storage, Data classes are used. The Data classes support four different data types; char, integer, double, and three-dimensional vector. Each of these types is derived from a common base class. Each Element contains a unique pointer that identifies the location of its data. The Data class contains a large contiguous block of memory, that can be indexed by the pointers in each Element. In this way, data fields are stored in local memory proximity. A loop over all the values of boron concentration is fairly efficient, because all of the boron data is stored in one large memory block.

2.2. Supported grid methods

Support from mesh adaption and refinement is critical for process simulation applications. FLOOPS employs a variety of algorithms to

represent moving boundaries from oxidation and silicidation [2]. These algorithms operate on edges, so to first order they are dimensionally independent.

Edges are examined to see if they meet a growth-based refinement criteria. The user can specify the desired grid spacing in a growing region. Edges that are attached to a node at a growing interface are split when longer than this criteria. Any edge that passes this test is then split at the midpoint of the edge. This generates one new node and requires reconnection and splitting of all the attached higher dimensional grid objects.

Grid removal is accomplished by identifying a edge to be removed based on the rate the edge is shrinking. The ends of this node are considered candidates for removal. If one end is not on an interface (completely internal to the material), that end can be removed by reconnection. Reconnection involves reconnecting edges from the node to be removed to the other node on the edge. For edges which span a material and have both nodes on interfaces, the surrounding volume is transferred to the growing material.

Mesh refinement and removal is also performed based on estimates of the numerical error associated with solution of the partial differential equations [3–5]. This leads to refinement of the mesh in areas of high-error, and coarsening in areas of greater accuracy. There are three primary steps involved in this process. First, estimate the numerical error associated with the current discretizations. Second, use this error to guide local refinement of the grid, both addition and subtraction of elements. Third, make sure the overall grid quality is not significantly degraded by the local refinement. These steps lead to well-defined meshes that control error to a user set tolerance.

3. Implementation of differential equations

The finite volume and finite element code makes use of a common set of base classes that represent solution values and capabilities, partial differential equations, and solution of linear systems. These same objects are used to implement the finite-element solver for material growth and the companion device simulation to FLOOPS, FLOODS [6]. Implementation of new physical systems are relatively straightforward, because of the rich support of the base classes. This allows quick derivation of classes to represent new physical phenomena. Fig. 2 shows a hierarchy of objects for a finite element solver.

The overall handling of the system of partial differential equations is handled by the a Solver class. It maintains a matrix (solutions by materials) of PDE's and has methods to load the linear Jacobian. This classes controls construction of the system Jacobian. This solver class handles time step integration, nonlinear solution via Newton's method, and assembly of the Jacobian.

Each Solver class (as shown in Fig. 2) contains a list of Solution objects and Materials objects over which the problem is to be solved. At the intersection of each solution and material, a pde object can be associated. This allows different pde's to be used for each solution and material type. Since all pde's have a common base class, the solver does not have to know any of the particulars of the physics. Each Solution object knows how to query model selection information and select the correct type of pde for each material.

3.1. Objects for solution variable, models, and parameters

The Solution object contains a SolutionID object base class which holds the basic information about a particular solution variable, for example, boron, interstitials, or velocity. The SolutionID object contains the type identifier and name, and allows unique identification of particular solution variables. The class performs another critical function of determining when a solution is needed numerically, and allocation of the correct partial differential equation classes to be used in that solution. The NeedSol member determines if numerical solution is required by checking to see if the variable is in the structure or if the appropriate models flags or ambient have been selected. The type of model selected is also used by the MakePDE member, which allocates partial differential equation classes for a given material. There are derived classes for each of the different solution

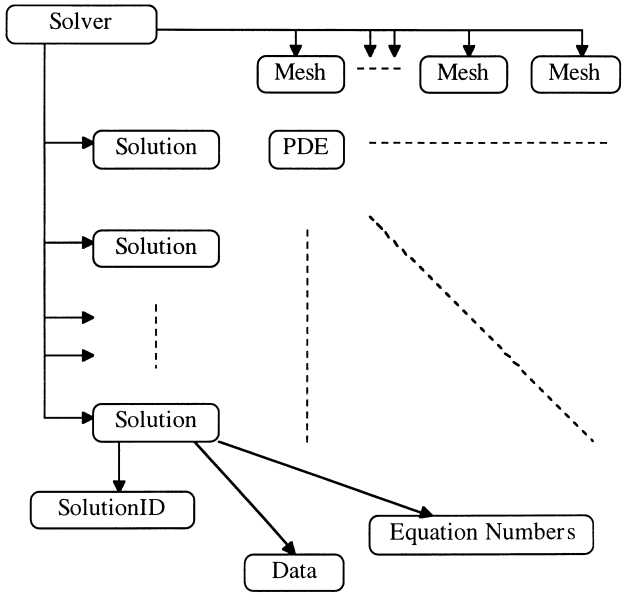

Fig. 2. Each solver class contains a matrix of solutions and partial differential equations. Each Solution, in turn, contains data fields and a solution identifier.

types in the code – Velocity, Stress, Dopants, Defects, Loops, Clusters, Traps, and Potential.

The Solution classes contain the solution values in data fields. In diffusion simulation, three values in time are required at each node to implement the commonly used TR-BDF time discretization scheme [7] for dopants and defects. Three data fields are allocated for that purpose. Solution classes also contain an equation number data field to list the index of each grid object in the global matrix assembled. This helps perform the scatter operation from the small dense Jacobian for each element to the larger, sparse matrix for the entire system.

Parameters and model selections are stored locally by in the partial differential equations and solutionID classes. These classes use a tcl based database to store the parameter and model defaults. The tcl database can be edited during session to change parameters and defaults. Long term storage of these selections is performed by user-editable tcl scripts. These scripts are demand loaded, e.g. silicon interstitial parameters are not read until required by the program. A tk based visual browser and editor are under development.

Since flexibility is required, materials and solutions can be created on the command line. Specification of the relevant parameters are then the only requirement. For example, FLOOPS was developed for silicon technology but has been extended with user commands to also simulate diffusion for HgCdTe [8,9].

3.2. Objects for differential equations

Partial differential equation classes are responsible for implementing the discretized physical equations. These are assigned one per solution variable and material, as shown in Fig. 2. Each pde is responsible for reading user parameters, assigning equations, and assembly the Jacobian terms.

Each PDE object has a method to assign equations and initialize itself. Parameters are read from the tcl database. Equation numbers are allocated and stored in the associated solution class. In this fashion, global numbering of solutions is achieved. Each pde object allocates the number of equations necessary for the material and solution type. This is communicated back to the solver object, which passes the information to the matrix

codes so that the matrix can allocate size. This same routine is responsible for locating variables that the pde is dependent upon. For example, the dopant full couple pde is dependent on the potential and point defect concentrations. It needs to locate these solutions and equations for use during assembly.

The primary member functions of the pde object are the assembly functions. For finite-volumes, there are two that are called – one taking an edge and one taking a node. For finite-elements, only one assembly is called on the element. In the diffusion solver, the discretization is finite-volume based. For material growth processes, the discretization is finite element based.

For finite volume PDE's, the edge based assembly accumulates the flux terms required for the equation. In some cases, these routines are empty when there is no flux for the differential equation. For example, this is the case for defect clusters which are assumed immobile. Quantities for the fluxes that are required from each node are typically precomputed to save CPU time. Nodal values are not recomputed for each edge. The node based assembly routine is responsible for nodal quantities, like recombination fluxes. There are also nodal based routines to assemble the time derivative terms. All of these terms are computed and stored in an small dense Jacobian matrix (the stiffness matrix, to borrow the term from mechanical engineering). These matrices contain both symbolic information as well as numeric. Since each pde has access to the global equation numbers, these are stored into the dense Jacobians.

Finite element pde's operate in a similar fashion. Each pde assembles the finite element small, dense Jacobian on an individual element. Global equation numbers are stored. Since each pde allocates its own equations, higher order elements can be simply included by allocating edge, face, or volume equations. We have used six node triangles in the past, but primarily rely on linear elements for most problems currently.

3.3. Objects for boundary conditions

Boundary conditions are handled in a very similar way to that of body PDE's. Each interface mesh can be assigned a pde object for a solution. These pdes on the interface represent boundary conditions. For finite volume codes, they are called node by node. For finite element codes they are called on the interface element of highest dimension (in two dimensions, that would be an edge).

For flux type boundary conditions (surface recombination velocity), the flux terms are integrated and added to the other flux terms in the equations. Neumann boundary conditions at reflecting boundaries are simply not assembled. Dirchelet boundary conditions present more of a challenge. These boundary conditions need to be able to over ride earlier assembly and clear the matrix row. This is accomplished by mapping into a new variable. For example, in device simulation, we would like to fix the quasi-fermi level and still know the current. This is accomplished by adding a variable for current to balance the flux equation, and then adding a new equation to fix the quasi-fermi level.

3.4. Linear solver support

Linear solvers are decoupled from the assembly process so that different linear solver packages can be included easily. Fig. 3 shows the relationship between the pde objects and the linear solvers. Data is transferred from the assembly methods of the pde objects to the linear solvers through the stiff object. Each stiff object contains the small, dense Jacobian and global equations or each element type assembled. These are fairly transparent objects, and can be read and written as if they were two-dimensional arrays.

Each linear solver package consumes data from the rest of the simulator through the stiff objects. Each stiff object must be scattered from the small dense representation into the large sparse internal matrix representation. Nearly all packages allow this type of access to the data structures, and some will allow construction by vector. The stiff objects can contain vector data and equation arrays to speed operation.

Each linear solver must define three objects. The first is the matrix storage structure, and is responsible for storing the large sparse object. The second is a preconditioner object, which precon-

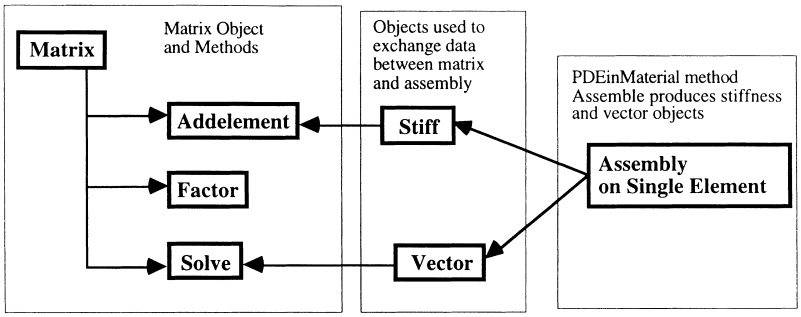

Fig. 3. Major methods of the partial differential equation object, and its interaction with the linear solvers.

ditions and scales the matrix for iterative solution. Finally, an iterator must be defined that takes the matrix, preconditioner, and right-hand side and returns a solution vector. For the special case of direct solves, the preconditioner is a complete LU factorization and the iterator is a simple for/back solve. To date, six different linear solvers have been used in FLOOPS, although only two are currently supported. These are UMF [10] and some hand coded routines implementing zero-fill LU decomposition and BCGS iteration.

4. Dopant implantation and annealing

FLOOPS exploits the modular nature the code to allow easy implementation of advanced models for dopant implantation and annealing. It has proved to be a viable platform for cross university research, as a successful collaboration between the University of Florida and the University of Texas at Austin has resulted in advanced implantation models being provided. The collaboration has been facilitated by the modularity of FLOOPS. Advanced defect, both point and extended, as well as dopant models for annealing are provided.

4.1. Implantation models and implementation

The implantation models in FLOOPS are based on integrating the point response across the surface of the wafer. Before integrating the implant, the surface is preprocessed. Colinear/planar pieces are identified and unioned into larger patches. This allows speedier processing across the surface integration, so each node receives a contribution from each surface piece. Impenetrable masks are identified and eliminated from the surface, so that useless integration can be avoided. Patches on the surface that are overhung or parallel with the implant beam are also eliminated.

Each surface element that remains after preprocessing is assigned a point response class for the implant. This point response is integrated across the surface of the element. Since analytic solutions are available only for cases where the surface is perpendicular to the beam and rectangular, in general this integration must be performed numerically. Adaptive quadrature is used to integrate in these cases. For implants in a two-dimensional simulation, the integration must be performed in one dimension. For implants in three-dimensional simulation, the integration must be performed in two dimensions. This latter integration can be very time consuming. Fig. 4 compares the CPU time requirements for 3D and 2D implants. 3D implantation times for typical structures can be quite large, on the order of hours for today's workstations.

The reduced set of surface elements are each assigned a point response class. Point responses are described with the base class Implant, and one implant type is allocated per reduced surface patch. In most implant cases, the point response is a moment based approximation to the profile. The most commonly used are the dual-pearson models developed by Tasch and co-workers. Because the point response function is a class with derived elements, it is relatively easy to implement new versions of appropriate point responses. The

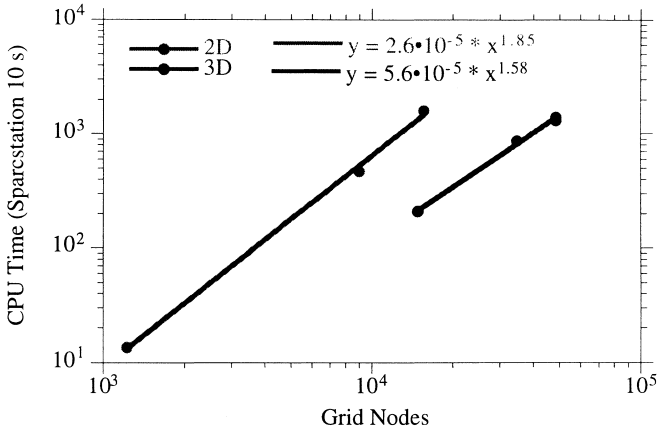

Fig. 4. Relative CPU time comparison for implants into two- and three-dimensional structures as a function of the number of nodes in the structure.

Dual-Pearson models are implemented by using a derived implant class. This class is constructed to contain two individual Pearson point response classes. The results of the individual point responses are summed.

4.2. Annealing models

One of the driving forces for modular models is the ability to easily implement advanced models of annealing and diffusion. Currently, the code supports three levels dopant modeling – constant diffusivity, Fermi-level dependent diffusivity, and full dopant-defect pairs. Point defects and extended defects (dislocation loops and 311's) are also supported. There is also support for different models of dopant activation and for dopant-defect clustering. Boundary conditions are handled with PDE classes that are assigned to interface mesh materials.

This hierarchical system of objects makes it possible to implement new physical models and concepts quite easily. For example, the 311 defect has become the popular explanation for transient enhanced diffusion behavior [11]. A first-order model for this defect was implement in approximately one man-week [12]. Although the model should not be considered a finished product (further work on tuning and behavior is required), the software architecture to support the model was implemented quickly and completely.

4.2.1. Dopants

Both the dopant chemical and active concentration are solved for numerically, even when the active concentration is a simple analytic function of the chemical concentration. This is to streamline the code for the more complex models. This approach does make the simplest models less efficient, but we feel this is a good trade-off. As time progresses, the more complex models will be required for nearly all stages of the process. For each dopant, therefore, two differential equations are solved:

$$\begin{aligned} C_{\mathrm{A}} &= f(C_{\mathrm{A}}, C_{\mathrm{C}}, C_{\mathrm{I}}, C_{\mathrm{V}}, \ldots), \\ \frac{\partial C_{\mathrm{C}}}{\partial t} &= \nabla J(C_{\mathrm{A}}, C_{\mathrm{I}}, C_{\mathrm{V}}, \Psi, \ldots), \end{aligned} \tag{4.1}$$

where C is the concentration with subscripts A is for active, C for chemical, I for interstitials, and V for Vacancies. The Ψ is for potential, f is a function for the active concentration and can include differential operators, and J is the flux of dopant which depends on the model specified.

To help compute the fluxes, an class for discretization of the $a\nabla b$ operator is provided. The routing returns the parent the value of the operator and the derivatives of the that value with respect to a and b. In this way, higher order flux discretization [13,14] can be provided simply as derived classes of the base operator class. Efficiency is also improved by having each dopant model precompute the required values of a and b

at each node. This insures that nodal quantities are computed only once.

For the most advanced dopant-defect pair model, an additional dopant equation is solved to compute the value of the substitutional dopant concentration. In most cases, the substitutional dopant concentration is equal to the active concentration. However, at short times after a large number of point defects have been created from an implant, the number of pairs can become quite large [15,16]. This reduces the substitutional concentrations as a function of the active concentration. The dopant-defect pair model has a flux given by [15–17]:

$$J_{\mathrm{AX}} = D_{\mathrm{AX}} C_{\mathrm{A}} \frac{C_{\mathrm{X}}}{C_{\mathrm{X}}^{*}} \nabla \ln\left(C_{\mathrm{A}} \frac{C_{\mathrm{X}}}{C_{\mathrm{X}}} \frac{n}{n_{\mathrm{I}}} \right), \tag{4.2}$$

where the subscript AX refers to dopant A-defect X pair, A is the substitutional dopant, and X is the defect. D is the equilibrium, Fermi-level dependent diffusivity, and n is the electron concentration with subscript I indicating the temperature dependent intrinsic concentration. The scaled electron concentration handles the electric field effect terms in the equations. The superscript * refers to the equilibrium dopant concentration. This flux is also added to the defect equations, to account for motion of the pairs. This can be done simply within the class, since the class allows the flux term to be added to the stiffness matrix in any position. In this case, the flux and its derivatives are computed once, and summed into both the dopant and defect row of the stiffness element. This model quite obviously maps well into the $a\nabla b$ operator class, and can be efficiently computed. It can also be extended simply to include the effect of mechanical strain on the dopant flux [18].

4.2.2. Defects

Point defects are accounted for in a manner similar to the dopants. Each defect flux is assembled using an edge by edge assembly. The dopant-defect pair fluxes are computed by the dopant. The defect fluxes are computed including the electric field, and can be represented with [17,19]:

$$J_{\mathrm{X}} = D_{\mathrm{X}} C_{\mathrm{X}}^{*} \nabla \frac{C_{\mathrm{X}}}{C_{\mathrm{X}}^{*}}, \tag{4.3}$$

where the subscripts hold their meaning from Section 4.2.1. The equilibrium defect concentration is a function of potential, so this does account for the electric field effects on the charged defects. It is interesting to note that in this flux formulation, the prefactor is just the self-diffusion component due to the particular defect. This is why, in many cases, the exact values of D_{X} and C_X^* do not significantly affect the answer as long as the product is constant.

Also included in FLOOPS are models for {311} defects [12] and dislocation loops [20–22]. These extended defects are interstitial storage mechanisms that influence the diffusion of dopants. As these defects dissolve, they release interstitials which contribute to the enhanced diffusion of dopants that are interstitial diffusers. They also can absorb interstitials and thereby reduce the diffusion.

The {311} loop model handles both evolution and nucleation processes [12]. The nucleation model is based on homogenous nucleation mechanisms. Capture and release of interstitials is included. The {311} model solves for two moments – the number of {311} defects and the number of interstitials contained in the defects.

The dislocation loop model includes the strain from the loops, and their evolution during a variety of ambient conditions [20–22]. Unfortunately, a nucleation model does not exist for dislocations, and the initial distribution data must be entered. From that point, however, a dislocation distribution is solved for. This includes the average, minimum, and maximum size as well as the total interstitial concentration. Since it is frequently the largest loops which can cause device degradation [23], the maximum size is explicitly solved. The code assumes a triangular distribution of loops.

4.3. Examples

4.3.1. Oxidation enhanced diffusion

Oxidation of the silicon surface injects interstitials into the crystal [24]. This enhances the diffusion of interstitial based diffusing species, e.g. boron and phosphorus. Fig. 5 shows the junction depth of an implanted boron layer under a wet ambient at 1000°C. Two different conditions are plotted. First, the point defect models were

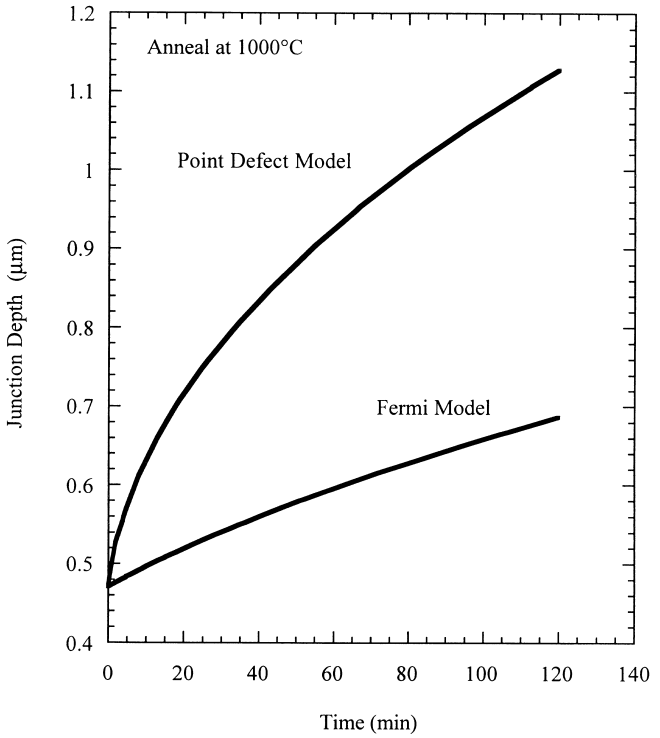

Fig. 5. Plot of boron junction depth for an oxidizing (H_2O) ambient at 1000°C as a function of time. Two models are compared – with and without point defects. Oxidation enhanced diffusion is quite evident, as the defect model shows a consistently deeper junction.

disabled, and the diffusion was computed using the simple Fermi level dependent model. Second, the point defect models are enabled and the diffusion is computed as enhanced by the interstitial injection from the oxide. As can be seen, this creates a substantially deeper junction.

Oxidation enhanced diffusion is a straightforward application of the models described earlier. In this case, the user selects the modeling level, and the defect and dopant PDE class allocations are changed inside the diffusion solver. The majority of the code remains the same and is not impacted by the model change. Boundary condition PDE's for the defects are available that compute both the surface recombination and velocity dependent injection of interstitials [15,25–27].

4.3.2. Dissolution of {311} defects

In simulating transient enhanced diffusion (TED), it is important to be able to compute the number of interstitials contained in {311} defects. The {311} defects store interstitials and release them at a rate that is similar to the duration of TED [11]. It is important to model both the nucleation and growth of these defect structures as well as their dissolution. Fig. 6 shows the comparison of measured {311} interstitial and defect dose [11] with simulations from FLOOPS [12].

This simulation read the total Frenkel pair computation from UT Marlowe [28,29]. An additional interstitial population was added to account for the silicon added during the implant. The {311} defect model included simulation of both the number of defects and the number of interstitials. The vast majority of the defect population recombines, leaving a population of interstitials roughly equal to the implanted dose. These precipitate onto nuclei to form {311} defects. Subsequent dissolution is controlled by the binding energy of an interstitial to a {311} defect, and was set to 2.0 eV for this simulation.

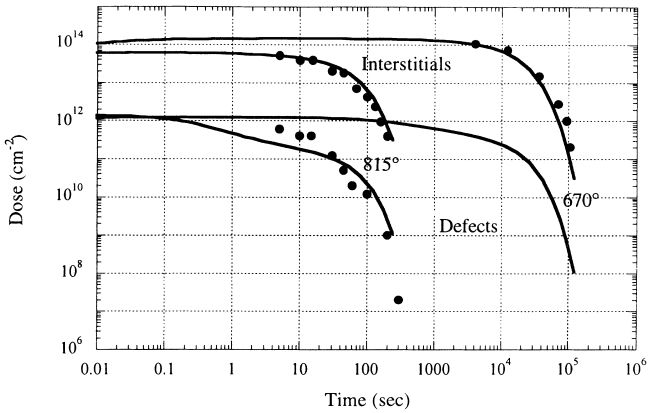

Fig. 6. Data [11] and simulation of the number of interstitials and number of {311} defects for two different temperature anneals.

4.3.3. Growth of loops during oxidation

During oxidation, interstitials are injected which can be captured by dislocations. The dislocations can be measured directly using plan-view transmission electron microscopy, and the interstitial count in the loops can be obtained. At 900°C, in an inert ambient, the dislocation loops do not show appreciable growth or shrinkage. Fig. 7 shows the growth in dislocations as measured [21] and simulated [20] by FLOOPS. Two different implant conditions are shown which correspond to two different initial loop conditions. Also simulated is the evolution of the loops distribution which includes Ostwald ripening behavior [22]. Initial loop distributions were measured and entered into FLOOPS as the starting condition.

5. Material growth

5.1. Models

Modeling reactive diffusive growth processes consists of performing two steps that are repeated at each time step of the simulation. First, solve for the diffusion and reaction of the reactant through the growing layer. Secondly, account for the

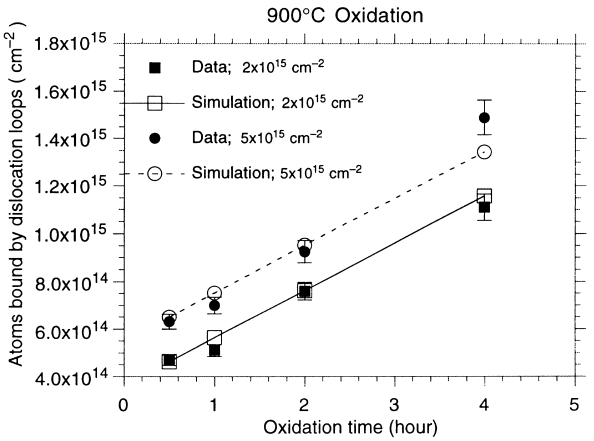

Fig. 7. Data and simulation for dislocation loops evolution in a dry oxygen ambient at 900°C. At this temperature under an inert ambient, dislocation loops do not show appreciable growth or shrinkage.

volume changes caused by the growth and solve for the stress caused by these volume changes in the surrounding layers. The grid is then updated and the next time step is solved. These two steps can depend on each other, for example the stress caused by the flow could affect the diffusion of the reactant as in oxide growth. The flow chart for solving for stress dependent material growth is shown in Fig. 8. This section will begin by looking at a general reaction and diffusion system and how it is applied to titanium silicide and silicon dioxide growth. The material models for solving for the stress used in FLOOPS are discussed. Examples of oxide and silicide growth are then presented.

5.1.1. Diffusion and reaction

The formation of both SiO_2 and $TiSi_2$ are modeled using a Deal-Grove [30] like diffusion and reaction model. In this model, a reactant diffuses through the growing layer from the source of reactant to the reacting interface. At the reacting interface, the molecules react to form the new material. There are three fluxes associated with this model: (1) the flux of reactant from the source to the interface between the source and growing layer, F_1; (2) the flux of the reactant as it diffuses through growing layer, F_2; (3) the flux of the reacting species as it is consumed by the reaction to form the new layer of the growing material. In the steady state condition, the three fluxes will be equal and any one of them could be the rate limiting step.

F_1 is defined as:

$$F_1 = k_t(C^* - C_s), \tag{5.1}$$

where C^* is the equilibrium concentration of reactant at the source interface, C_s is the reactant concentration across the interface and k_t is the transfer coefficient across the interface. If k_t is large, C_s will be equal to C^* and this flux will not be a rate limiting step. This is implemented in FLOOPS using a boundary condition PDE similar to those employed for surface segregation of dopants.

The flux of atoms across the growing layer F_2 is given by:

$$F_2 = -D\frac{(C_s - C_r)}{x_g}, \tag{5.2}$$

where D is the diffusion coefficient of the reactant in the growing layer, C_r is the reactant concentration at the reacting interface and x_g is the thickness of the growing layer. This flux is implemented with a steady state diffusion PDE, and can be either discretized in a finite volume or finite element approach as necessary to link with the material flow.

The flux at the reacting interface is:

$$F_3 = k_s C_r, \tag{5.3}$$

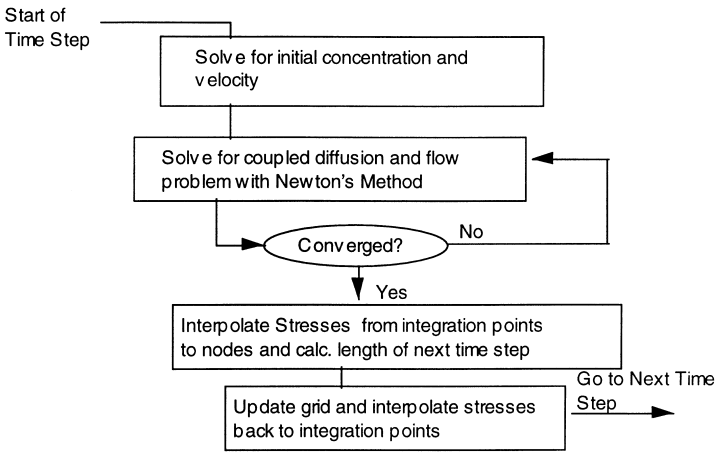

Fig. 8. Flow chart for the global control of the material growth processes.

where k_s is the reaction rate coefficient. In steady state the three fluxes can be set equal and the growth rate dx_g/dt can now be calculated by defining N_1 as the number of molecules of reactant incorporated into a unit volume of growing material. The growth rate is proportional to this flux.

In FLOOPS, general PDEs are assembled for each of these three fluxes. The growth of SiO_2 and $TiSi_2$ can be modeled by applying the PDEs to the materials or interfaces for each of the three fluxes, constant diffusivity, source of the reactant and reacting interface. Table 1 shows which material and material interfaces that each PDE are assigned to for either $TiSi_2$ or SiO_2 growth. The reactant for SiO_2 growth is an oxidant (either wet or dry) and for $TiSi_2$ growth it is silicon.

The diffusivities and reaction rates used for the diffusion of reactant can depend on stress. This has been seen for both oxide [31,32] and titanium silicide [33] growth. The stress generated by thermal oxidation is known to reduce both the reaction rate, k_s in Eq. (5.3), and the diffusivity, D in Eq. (5.2). This was reviewed extensively by Rafferty [32].

A compressive stress normal to the Si/SiO_2 interface will reduce the reaction rate k_s by

$$k_s = k_{s0} \exp\left(-\sigma_{nn}\frac{V_r}{kT}\right), \tag{5.4}$$

where k_{s0} is the reaction rate from the Deal Grove model, σ_{nn} is the stress normal to the Si/SiO_2 interface and V_r/kT is the activation volume divided by Boltzmann's constant and the temperature in K. The derivation of this expression was first done by Kao [31].

The diffusivity is influenced by pressure. The diffusivity as a function of pressure is given as

$$D = D_0 \exp\left(-\frac{PV_d}{kT}\right), \quad P > 0, \tag{5.5}$$

and either $D = D_0$ for $P < 0$ or

$$D = \min(D_{max}, D_{enh}), \quad P < 0, \tag{5.6}$$

where D_0 is the diffusivity with zero stress, P is the hydrostatic pressure, D_{max} is the maximum diffusivity, D_{enh} is the D from Eq. (5.5) with $P < 0$. The term D_{max} is used to clamp the increase in diffusivity due to tensile pressure. Stress dependent diffusivity is used in FLOOPS for both silicon dioxide and titanium silicide growth.

5.1.2. Stress and material flow

After the reaction takes place there is a volume change in the materials involved in the growth. This volume change can cause deformation and/or stress in the materials involved in the reaction as well as any surrounding materials (including the substrate). These stresses can influence the growth rate, cause defects in the silicon substrate or create voids in polycrystalline films. It is very important to account for the stress caused by film growth as well as the stress from other sources, for example, thermal mismatch. This section will discuss stress and some models that can be used to solve for the stress in growing layers and any surrounding layers.

Stress in thin films can be divided into two categories: intrinsic and extrinsic. Extrinsic stresses are caused by the differences between thermal expansion coefficients of the thin film an the substrate as the wafer is subjected to thermal cycling. There are many sources of intrinsic stress. Intrinsic stresses can be caused by the volume change caused by the growth of thin films like oxides and silicides. It also arises from precipitation, annihilation of defects, phase transformation, grain growth and structural changes. The main focus of this work is to look at intrinsic stresses due to volume change and their affects on growth. Thermal stresses and other intrinsic stresses can also be included in the viscoelastic models used in FLOOPS.

FLOOPS supports several material models. In linear viscous materials, the stress is linearly proportional to the strain rate and can be expressed as:

$$\sigma = D\dot{\varepsilon}, \tag{5.7}$$

Table 1
PDEs assigned to different materials for the solution of the reactant during either titanium silicide or silicon dioxide growth

Growth system	Constant diffusivity	Source of reactant	Reacting interface
$TiSi_2$	$TiSi_2$	Silicon/$TiSi_2$	Titanium/$TiSi_2$
Oxide	SiO_2	Gas/SiO_2	Silicon/SiO_2

where D now represents the tensor of viscosities for the material. For a simple one-dimensional viscous element, $D = 2\mu$, where μ is the viscosity. Viscous materials can be incompressible which means that the material's volume cannot change even though it's shape does. Fluids are the materials that most often exhibit viscous behavior.

The behavior of viscoelastic materials lies between viscous and elastic. A Maxwell viscoelastic material can be represented by a viscous and elastic element in series. This model can exhibit either viscous or elastic behavior depending on the time scale of the observation in relation to the relaxation time of the element. The behavior of the element is given by:

$$\frac{\dot{\sigma}}{G} + \frac{\sigma}{\mu} = \dot{\varepsilon} \tag{5.8}$$

whose solution for constant growth rate and viscosity is:

$$\sigma = \mu\dot{\varepsilon}\left(1 - \mathrm{e}^{-1/\tau}\right), \tag{5.9}$$

where μ is the viscosity, t is the time and τ is the relaxation time for a linear Maxwellian viscoelastic material:

$$\tau = \mu/G \tag{5.10}$$

Here G is the shear modulus which is:

$$G = \frac{Y}{2(1+v)}, \tag{5.11}$$

where Y and v are the Young's modulus and Poisson's ratio of the material.

At high strain rates most materials display nonlinear or plastic flow characteristics. One of the most common nonlinear flow models is the hyperbolic sine or Eyring model. This model has been applied to oxide growth with great success by Rafferty [32]. In this model the viscosity is stress dependent and given by:

$$\mu = \mu_0 \frac{\sigma_s/\sigma_c}{\sinh(\sigma_s/\sigma_c)}, \tag{5.12}$$

where μ_0 is the low stress viscosity, σ_c is the stress above which the material softens and σ_s is the maximum shear stress. This type of nonlinear viscosity can be used for viscous flow and viscoelastic flow of any material in FLOOPS.

5.1.3. Material models in FLOOPS

In FLOOPS the most general material model is the viscoelastic model. Nonlinear or stress dependent viscosity given by Eq. (5.12) is available in the model. This material can exhibit either elastic or viscous behavior depending on the time frame of interest in relation to the relaxation time. For times much less than τ the behavior will be elastic and for times much longer than τ the behavior will be viscous. This general response make this model ideal for process simulation where a wide range of material responses, temperatures and times need to be simulated. Elastic behavior is simulated by using a large viscosity in the viscoelastic model, thereby giving the material a large relaxation time. This has worked well for a number of elastic problems including the stress due to nitride strips where both the nitride and silicon are treated as elastic materials.

5.2. Model implementation

All of the models for material growth are solved using the Finite Element Method. There are two types of PDEs that need to be solved for in material growth, diffusion of reactant, and stress due to the reaction. The code supports both two- and three-dimensional simulations. Linear shape functions are used for all the elements. The nonlinear iterations that arise due to stress dependent growth are solved using Newton's method [32,33].

In both the silicide and oxide growth systems the diffusion of reactant is modeled by solving the steady state diffusion equation. In the flow equations the unknowns are the nodal velocities and the equation governing deformation is Newton's second law. Newton's second law for continuum mechanical problems is given by

$$\begin{Bmatrix} \frac{\partial \sigma_{xx}}{\partial x} & \frac{\partial \sigma_{xy}}{\partial y} & \frac{\partial \sigma_{xz}}{\partial z} \\ \frac{\partial \sigma_{yy}}{\partial y} & \frac{\partial \sigma_{yx}}{\partial x} & \frac{\partial \sigma_{yz}}{\partial z} \\ \frac{\partial \sigma_{zz}}{\partial z} & \frac{\partial \sigma_{xz}}{\partial x} & \frac{\partial \sigma_{yz}}{\partial z} \end{Bmatrix} = 0 \tag{5.13}$$

for a three-dimensional problem. The flow is solved for using the virtual work equation. It is given by

$$\int_{\Omega} \delta\dot{\varepsilon}^{\mathrm{T}}\sigma \, \mathrm{d}\Omega = 0, \tag{5.14}$$

where $\delta\dot{\varepsilon}$ is the virtual strain rate and σ is the stress tensor.

The strain rate and stress tensors can be written in vector form as:

$$\dot{\varepsilon} = \begin{Bmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ \dot{\varepsilon}_{xy} \\ \dot{\varepsilon}_{yz} \\ \dot{\varepsilon}_{zx} \end{Bmatrix}, \qquad \sigma = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{Bmatrix}, \tag{5.15}$$

where the strain rate is the derivative of the velocity. The stress is related to the strain rate through the constitutive relationships of the previous section.

The most general model in FLOOPS is the viscoelastic model described earlier with or without nonlinear viscosity. This model is similar to the one used by Peng [34] and Senez [35,36]. The finite element model can be used to model viscoelastic behavior by including the correct constitutive expression. The correct constitutive relationships are for the deviatoric stress

$$\sigma_{\mathrm{d}} = \mu\left(1 - \mathrm{e}^{-\Delta t/\tau}\right)\bar{D}\dot{\varepsilon} + \sigma_{\mathrm{d0}}\mathrm{e}^{-\Delta t/\tau} \tag{5.16}$$

and for the dilatational or pressure component

$$p = K\Delta t m^{\mathrm{T}}\dot{\varepsilon} + p_0. \tag{5.17}$$

In these equations $\mu\left(1 - \mathrm{e}^{-\Delta t/\tau}\right)$ is the effective viscosity, σ_{do} and p_0 are the residual stresses from the previous time step, and $\tau = \mu/G$ is the relaxation time constant and

$$\bar{D} = \begin{Bmatrix} \frac{4}{3} & -\frac{2}{3} & -\frac{2}{3} & 0 & 0 & 0 \\ -\frac{2}{3} & \frac{4}{3} & -\frac{2}{3} & 0 & 0 & 0 \\ -\frac{2}{3} & -\frac{2}{3} & \frac{4}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{Bmatrix}. \tag{5.18}$$

The element stiffness matrix can now be written as

$$K^{\mathrm{e}} = \int_{\Omega_{\mathrm{e}}} B^{\mathrm{T}} D B \, \mathrm{d}\Omega_{\mathrm{e}}, \tag{5.19}$$

where B is the derivative of the shape function matrix and D is the material matrix which contains the material parameters like effective viscosity and the bulk modulus K. The residual stresses are treated as right-hand side terms of the form

$$\int_{\Omega^{\mathrm{e}}} B^{\mathrm{T}}\sigma_0 \, \mathrm{d}\Omega^{\mathrm{e}}. \tag{5.20}$$

This model can also be used to include stress due to other sources, like thermal mismatch and intrinsic stress.

The class structure for the material growth models are shown in Fig. 9. The base finite element class is at the top level. It is derived from the PDE class described earlier. This base class

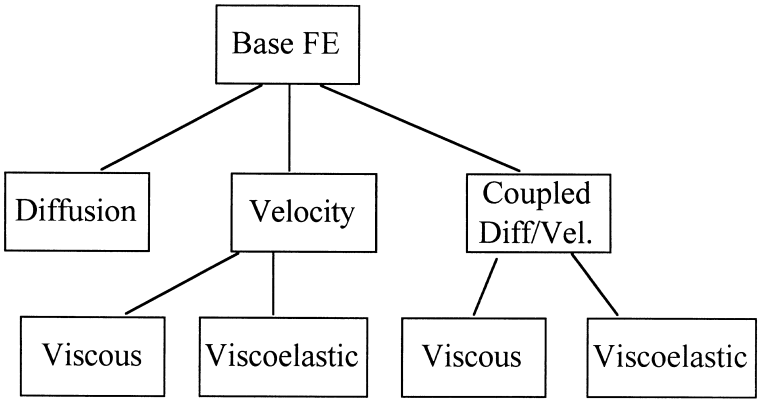

Fig. 9. Class hierarchy of the finite element PDE classes used in material growth processes.

Table 2
PDE's assigned to different materials for the solution of the flow equations during either titanium silicide or silicon dioxide growth

Growth system	Consumed	Flow	Consume and flow
SiO_2	Si	SiO_2, Si_3N_4	Poly
$TiSi_2$	Si	$TiSi_2$, SiO_2	Poly, Ti

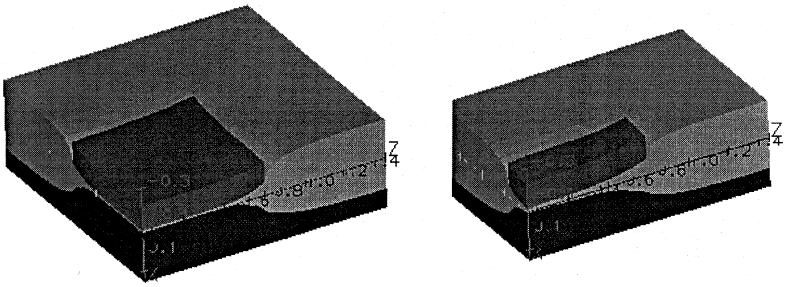

Fig. 10. Three-dimensional oxidation structures using the nonlinear flow and growth models for oxide. The nitride on the right shows considerable more lifting under the tip than the square nitride shape on the left.

contains functions that are used by all the classes derived from it. It calculates shape functions for the supported elements, which are triangles in 2D and tetrahedras in 3D. It also supplies functions that perform the matrix multiplications needed to generate the element stiffness matrix for each PDE. The second level of classes represent the type of equation that is being solved. There are classes which solve for diffusion of reactant, stress with velocity as the unknown and a coupled diffusion and stress solution which is used for stress dependent growth. These classes find the element stiffness matrix for each model depending on the material coefficients and the element geometry. For example for the flow solution they assemble equations of the form of Eq. (5.20). The third level of classes contain functions which vary whether the material model is viscous or viscoelastic, for example the D matrix and the rhs assembly.

Table 2 shows the similarities between the silicide and oxide systems. In both cases, there are three different material types. There are materials that are consumed by the reaction and are treated as nonflowing and distorting materials. There are materials that are flowing in response to the growth. Finally, there are materials which are being both consumed and flowing with respect to the growth forces.

5.3. Three-dimensional oxidation examples

FLOOPS simulations of oxidation use nonlinear stress dependent viscoelastic simulations. The parameters were fit to the cylinder and hole oxide

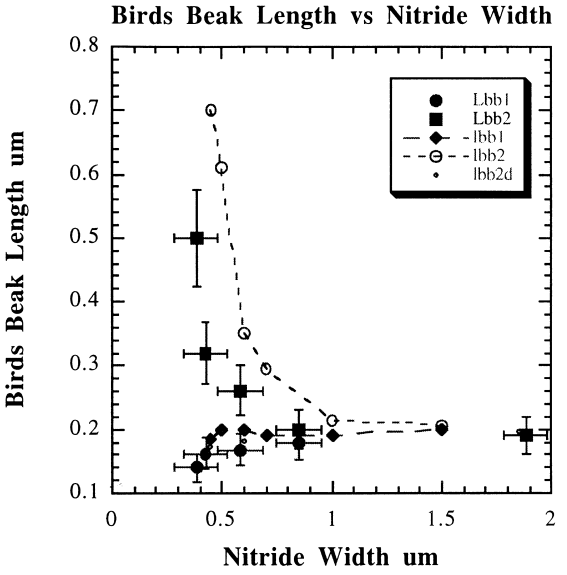

Fig. 11. Plot of the bird's beak length as a function of the nitride shape. Lbb1 is the encroachment at the tip and Lbb2 is the encroachment at the side. Lines are for the simulated results.

growth of Kao [31,37] and various two-dimensional LOCOS structures. These parameters were then used to simulate the three-dimensional LOCOS data from an experiment by Smeys [38]. The experimental structures had a pad oxide thickness of 10 nm while the nitride thickness was 150 nm. After patterning the nitride, 430 nm of field oxide was grown at 1000°C in pyrogenic steam. The oxidation time was 70 min. The nitride was stripped, and the pad oxide was given a timed etch to remove 10 nm of oxide. Top view scanning electron microscopy (SEM) photographs were taken to measure the bird's beak length both under the tip (Lbb2) under the line (Lbb1).

Nitride lines 1.5 μm in length and with widths varying between 1.5 and 0.4 μm were simulated. The field oxide opening was 1.5 μm along both edges. Reflecting boundary conditions were used so $\frac{1}{4}$ of the structure was simulated. Fig. 10 shows the simulated structures for 1.5 and 0.4 μm wide nitride lines. The nitride line is lifted higher for the narrow line which agrees with the data.

In both two- and three-dimensional simulations the bird's beak length is overestimated. The bird's beak length is measured as the length from the edge of the nitride mask to the point were the oxide under the mask is 10 nm. Two-dimensional simulations were used in order to determine the oxide thickness under the nitride that resulted in a bird's beak length of 0.19 μm. The thickness was 15.5 nm. This thickness is used to extract the bird's beak length versus nitride width along the long edge (Lbb2) and under the short edge (Lbb1). The results obtained using this technique compared to the experimental data are shown in Fig. 11. The simulated results match the trend well for both the increase of Lbb2 with decreasing width and the decrease of Lbb1 with decreasing width.

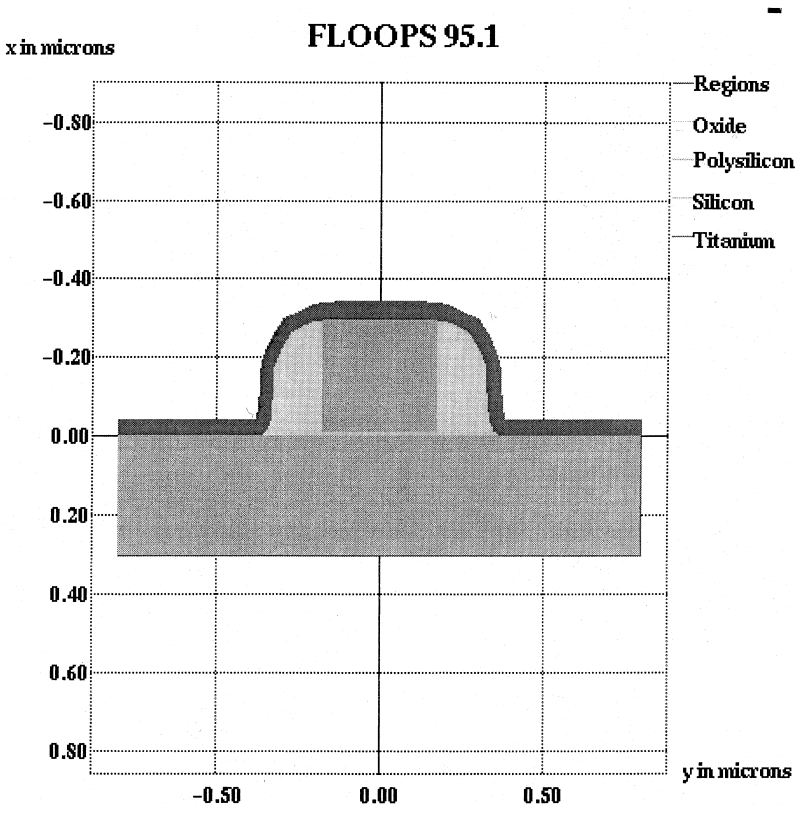

Fig. 12. Salicide structure before silicide formation.

5.4. Titanium salicide simulation

The starting structure before titanium silicide formation is shown in Fig. 12. There is deposited titanium contacting the silicon at the source and drain and polysilicon at the gate. The simulation anneal is 650°C RTA in nitrogen. For this anneal, FLOOPS will simulate both the formation of $TiSi_2$ and TiN. $TiSi_2$ forms in region where the titanium is in contact with either silicon or polysilicon. TiN forms on exposed titanium due to the nitrogen ambient. The addition of the TiN growth capability took less than 3 man days to complete due to the fact that it shares much of its code with the SiO_2 and $TiSi_2$ growth code. The growth kinetics of TiN and $TiSi_2$ are assumed to be diffusion limited, a stress dependent diffusivity is used for the $TiSi_2$ growth. The viscoelastic model is used to model the deformation during silicide growth. The values for Young's modulus and Poisson's ratio were taken from literature. All of the materials are treated as nearly elastic by using a high (1×10^{17}) viscosity. A more detailed review of the model parameters for titanium silicide growth in FLOOPS is given in Cea [33]. The structure after a 650°C anneal for 30 s and strip of the TiN and unreacted titanium is shown in Fig. 13. The bowing of the silicide on the short gate length is produced. It is possible that other stress dependent terms are important for silicide growth for instance a reduction of the transport of silicon across the silicon/$TiSi_2$ interface or plastic flow of the materials present. More experimental evidence of the nature of the stress effects on silicide growth is needed for more quantitative simulation.

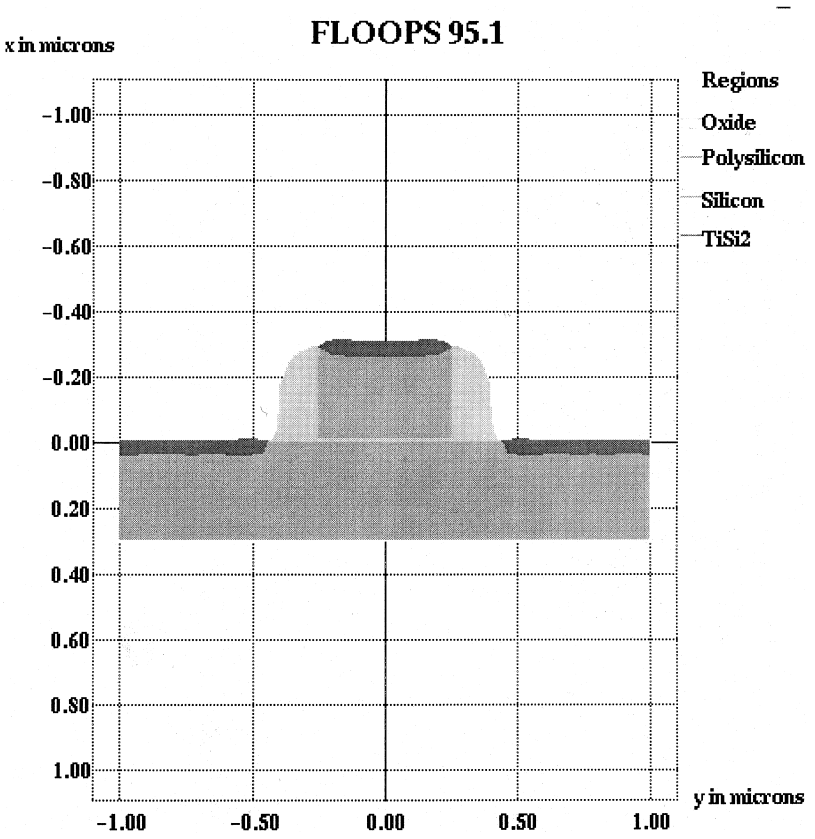

Fig. 13. Salicide after formation. Note the "poly smile" effect in the silicide on poly caused by nonlinear growth of the silicide.

6. Conclusion

The main objects and capabilities of FLOOPS have been presented. FLOOPS is a modular platform which allows development of new models and grid algorithms to be handled easily. The object-oriented nature of the code allows substantial leverage, e.g, silicidation and oxidation modules share substantial code. New models for recently discovered phenomena can be quickly added to the software. This platform efficiently allows multiple developers to interact across different areas of the code. Several examples demonstrating both capabilities and performance have been presented to demonstrate the effectiveness of this approach.

Acknowledgements

The authors would like to thank the Semiconductor Research Corporation and the National Science Foundation for support of this work.

References

[1] M.E. Law, C.S. Rafferty, R.W. Dutton, SUPREM-IV User's Manual, Stanford University, 1988.
[2] M.E. Law, Grid adaption near moving boundaries in two-dimensions for IC process simulation, IEEE Trans. Computer-Aided Design 14 (1995) 1223.
[3] C.C. Lin, M.E. Law, Mesh adaption and flux discretizations for dopant diffusion modeling, NUPAD-V, Honolulu, 1994, pp. 151–154.
[4] C.C. Lin, M.E. Law, Two-dimensional adaption and flux discretizations for dopant diffusion modeling, IEEE Trans. Computer-Aided Design 15 (1996).
[5] M.E. Law, M. Cerrato, Improved local refinement algorithms for adaptive meshing of process simulation problems, SISPAD, Boston, 1997, p. 233.
[6] M. Liang, M.E. Law, An object-oriented approach to device simulation – FLOODS, IEEE Trans. Computer-Aided Design 13 (1994) 1235–1240.
[7] H.R. Yeager, R.W. Dutton, An approach to solving multiparticle diffusion exhibiting nonlinear stiff coupling, IEEE Trans. Elec. Dev. 32 (1985) 1964–1976.
[8] H.G. Robinson, D.H. Mao, B.L. Williams, S. Holander-Gleixner, J.E. Yu, C.R. Helms, Modeling ion implantation of HgCdTe, J. Electronic Materials 25 (1996) 1336.
[9] S. Holander-Gleixner, H.G. Robinson, B.L. Williams, D.H. Mao, J.E. Yu, C.R. Helms, Modeling of diffusion and ion implantation in mercury cadmium telluride: A comparison to transient enhanced diffusion in silicon, in: Process Physics and Modeling in Semiconductor Technology, Los Angeles, 1996, p. 216.
[10] T.A. Davis, I.S. Duff, An unsymmetric-pattern multifrontal method for sparse {LU} factorization, SIAM J. Matrix Anal. Appl. 18 (1997) 140.
[11] D.J. Eaglesham, P.A. Stolk, H.J. Gossmann, J.M. Poate, Implantation and transient B diffusion in Si: The source of the interstitials, Appl. Phys. Lett. 65 (1994) 2305–2307.
[12] M.E. Law, K.S. Jones, {311} Defect formation and evolution for Si and B implants, in: Process Physics and Modeling in Semiconductor Technology, Los Angeles, 1996, pp. 374–378.
[13] C.C. Lin, M.E. Law, R.E. Lowther, Automatic grid generation and higher order flux discretizations for diffusion modeling, NUPAD-IV, 1992.
[14] R.E. Lowther, A discretization scheme that allows coarse grid-spacing in finite element process simulation, IEEE Trans. Computer-Aided Design, 1989.
[15] H. Park, M.E. Law, Point defect based modeling of low dose silicon implant damage and oxidation effects on phosphorus and boron diffusion in silicon, J. Appl. Phys. 72 (1992) 3431–3440.
[16] M.E. Law, H. Park, P. Novell, Theory of dopant diffusion assuming nondilute concentrations of dopant-defect pairs, Appl. Phys. Lett. 59 (1991) 3488.
[17] M.E. Law, J.R. Pfiester, R.W. Dutton, The effect of implantation damage on arsenic/phosphorus codiffusion, in: International Electron Devices Meeting, San Francisco, 1988.
[18] H. Park, K.S. Jones, J. Slinkmann, M.E. Law, The effect of hydrostatic pressure on dopant diffusion in silicon, J. Appl. Phys. 78 (1995) 3664.
[19] M.E. Law, Two-Dimensional Numerical Simulation of Dopant Diffusion in Silicon, Stanford University, 1988.
[20] H. Park, K.S. Jones, M.E. Law, A point defect based two-dimensional model of the evolution of dislocation loops in silicon during oxidation, J. Electrochem. Soc. 141 (1994) 759–766.
[21] H. Park, H. Robinson, K.S. Jones, M.E. Law, Diffusion-limited interaction of dislocation loops and intersititials during dry oxidation in silicon, Appl. Phys. Lett. 65 (1994).
[22] S. Chaudhry, J. Liu, K.S. Jones, M.E. Law, Evolution of dislocation loops in silicon in an inert ambient – II, Solid-State Electronics 38 (1995) 1313–1319.
[23] P.M. Fahey, S.R. Mader, S.R. Stiffler, R.L. Mohler, J.D. Mis, J.A. Slinkman, Stress-induced dislocations in silicon integrated circuits, IBM J. Res. Dev. 36 (1992) 158–182.
[24] S.M. Hu, Kinetics of interstitial supersaturation during oxidation of silicon, Appl. Phys. Lett. 43 (1993) 449.
[25] S.M. Hu, Kinetics of interstitial supersaturation and enhanced diffusion in short-time/low-temperature oxidation of silicon, J. Appl. Phys. 57 (1985) 4527.
[26] S.M. Hu, Interstitial and vacancy concentrations in presence of interstitial injection, J. Appl. Phys. 57 (1985) 1069.

[27] M.E. Law, R.W. Dutton, Verification of analytic point defect models using SUPREM-IV, IEEE Trans. Computer-Aided Design 7 (1988) 181.
[28] K.M. Klein, C. Park, A.F. Tasch, Monte Carlo simulation of ion implantation into single-crystal silicon including new models for electronic stopping and cumulative damage, in: IEEE International Electron Devices Meeting, San Francisco, CA, 1990.
[29] S.-H. Yang, S.J. Morris, S. Tian, K.B. Parab, J.A.F. Tasch, Monte Carlo simulation of arsenic ion implantation in (100) single-crystal silicon, IEEE Trans. Semiconductor Manufacturing 9 (1996) 49–58.
[30] B.E. Deal, A.S. Grove, General relationship for the thermal oxidation of silicon, J. Appl. Phys. 36 (1965) 3370–3378.
[31] D.-B. Kao, J.P. McVittie, W.D. Nix, K.C. Saraswat, Two-dimensional silicon oxidation experiments and theory, in: IEDM Tech. Digest, 1985, p. 388.
[32] C.S. Rafferty, Stress Effects in Silicon Oxidation – Simulation and Experiments, Stanford University, 1988.
[33] S. Cea, Mutlidimensional viscoelastic modeling of silicon oxidation and titanium silicidation, in: Electrical and Computer Engineering, University of Florida, Gainesville, 1996.
[34] J.P. Peng, D. Chidambarrao, G.R. Srinivasan, Novel: A nonlinear viscoelastic model for thermal oxidation of silicon, COMPEL 10 (1991) 341–353.
[35] V. Senez, D. Collard, B. Baccus, Quantitative 2D stress dependent oxidation with viscoelastic model, SISDEP, Erlangen, 1993, p. 165.
[36] V. Senez, D. Collard, P. Ferreira, B. Baccus, Two-dimensional simulation of local oxidation of silicon: Calibrated viscoelastic flow analysis, IEEE Trans. Electron Devices 43 (1996) 720–731.
[37] D.-B. Kao, J.P. McVittie, W.D. Nix, K.C. Saraswat, Two-dimensional thermal oxidation of silicon I. Experiments, IEEE Trans. Electron Devices, 1987.
[38] P. Smeys, Geometry and stress effects in scaled integrated circuit isolation technologies, in: Electrical Engineering, Stanford University, Stanford, 1996.