Automatic Generation of Thin Film Process Flows – Part I: Basic Algorithms

Mohammed H. Zaman†,Member, IEEE, Edwin T. Carlen‡, and Carlos H. Mastrangelo‡,Member, IEEE

Abstract—This paper is the first in a series of two papers describing the algorithms used in the development of MISTIC (Michigan Synthesis Tools for Integrated Circuits). MISTIC is a planar device process compiler that generates process flows for thin film devices from schematics of their structure. This software uses a laboratory specific database of process recipes to produce process flows for a specific set of laboratory resources (furnaces, etchers, lithography equipment, etc.) and generates process statistics that help to choose the most suitable process flow in a comparative manner. The process compiler is augmented by several auxiliary modules: a device builder, process viewer, and database editor thus forming a self-contained process design environment. This paper concentrates on the algorithms used to construct process flows from schematic device representations. The compiler algorithms first extract a directed graph representation of the device organization stored in the form of a restricted square boolean matrix. This matrix is used to generate linear ordered lists of device layers which serve as footprints for the construction of process flows. Process flows are then constructed from these lists through a series of conversions, expansions, and insertions of process steps. The theoretical foundations for the algorithms can be found in [1].

I. INTRODUCTION

In the brief history of IC manufacturing, design automation tools have played a major role in the rapid manufacturing of improved VLSI circuits and devices. Most of these tools have focused on ECAD (Electronic Computer Aided Design) for physical design of VLSI circuits, layout generation, simulation, testing, and verification. With the proliferation of VLSI ECAD tools in recent years, computer-aided design (CAD) of integrated circuits has become one of the most important design tools in the IC industry.

Like integrated circuit design, semiconductor device process development is a large and complex task that can benefit from design automation tools. Technology CAD (TCAD) is essential for the simulation of device performance, fabrication, and yield estimation. Although several specialized process simulation tools such as SUPREM [2], DEPICT [3], and SAMPLE [4] are widely used, TCAD has not seen the equivalent parallel growth of ECAD. Most of this work has been highly focused on the accurate simulation of specific processes with very little work done in automated process flow design. One of the reasons for this lag is the fact that until just a few years ago there were only a few commercial device processes available, namely CMOS and bipolar, which were developed by a combination of trial and error and extensive simulations.

Today there is an increasing need for application specific process design as opposed to process design problems which are focused on scaling of known structures. These application specific processes are essentially customized device processes that vary in structure for each different application [5]. Fabrication processes for EERAM, EPROM, DRAM, and MEMS devices are typical examples of these. This wide variety of devices leads to longer processes with increased complexity. It is now common to use process flows with 20 to 25 mask levels compared to the conventional 6-mask CMOS process. In longer device processes, trial and error methods are too time consuming and inefficient; hence a faster structured methodology is desirable.

The use of a structured representation has been suggested [5]. In this scheme, formal representations of wafer state and individual process steps are used to describe the effects of individual steps on the wafer thus providing a unifying framework for the specification of process flows. A system of this kind (SPEC) was implemented [6, 7] as part of the Microelectronics Manufacturing Science and Technology program [8] at Texas Instruments Incorporated. SPEC composes process flows from machine independent equipment, step, sequence, and wafer specification hierarchies. Research has also been performed in partitioning the task of process flow design into a number of abstraction levels [9]. Frameworks provide the basic glue connecting fundamentally different steps but the proper selection of steps and the overall assembly of the flow requires an expert operator. For more rapid process development, the operator must be eliminated and these two operations must be automated.

Recently [1], a unified method for the automatic generation of application specific process flows has been developed. This procedure, referred as process compilation, utilizes the basic device topology as its input and generates complete process flows as output. This methodology has been implemented in the MISTIC process compiler. In MISTIC, the step selection and flow assembly are determined from the nature of the device structure, the projection constraints of the lithographic process, and the processing capabilities of a particular laboratory. This paper gives an overview of the basic topological algorithms of MISTIC while Part II describes the calculation of process step parameters and figures of merit.

The description of the compilation methods begins in section III with a discussion of the mathematical representation of the device structure. Section IV discusses the creation of device layer sequences from the mathematical representation. Section V discusses how basic process flow steps are constructed from layer sequences, and section VI describes the construction of complete process flows from the basic steps.

II. SOFTWARE STRUCTURE AND GENERAL METHODOLOGY

The fundamental principles used in process compilation are based on abstract concepts such as the fact that a given device consists of many components which must be fabricated to meet certain specifications. These components are then organized in a specific order. The nature of the components and their order determine the selection of individual process steps and ultimately the process flow. The general method used for flow construction
is shown in Fig. 1.

First, the component order is extracted from the device schematic thus producing a series of constraints in the order of the process steps in the flow. Linear ordered component lists are next generated by sorting individual components. Since the construction of each component requires specific steps, a corresponding list of basic steps is generated from the component list. This skeleton process flow is next expanded with additional patterning steps. Finally, specific process step parameters are determined from individual component specifications through a set of rules or calculated using numerical solvers. For example, diffusions are specified by junction depths and peak concentrations thus MISTIC employs a numerical inverse diffusion solver in time schedules for all diffusion components. The procedure generates many different possible process flows for a given device therefore a figure of merit is calculated to rate the quality of each process flow. The sections below discuss the details involved at each step of the flow construction method.

III. CONSTRUCTION OF DEVICE MATRICES WITH BOUND SOLUTION SET

The first step in the compilation procedure is the construction of a suitable mathematical representation of the device organization. The compiler first reads a device description file and creates several matrices which contain information about the mutual order between the device components. Eventually these matrices lead to the process steps necessary to construct the device.

The device matrices are created in a manner consistent with the lithographic process which allows changes near the surface only. This is accomplished by first establishing binary order relationships between any two adjacent components (i.e. share a common boundary) and any two components which must be fabricated in a specific order. From these relations, the order between any two components is calculated and stored in a boolean matrix. The assembly of the device matrices is performed in a manner that yields a bounded solution set (finite number of solutions) or design space. These bounds are established using a combination of simple estimates for the solution set cardinality and matrix reduction techniques that maintain the number of solutions at a manageable level. Figure 2 depicts the general matrix construction algorithm, where each function within the algorithm is discussed in the following sections of the paper.

A. Adjacency Matrix Representation of Device

The device component organization is specified by the set of order relationships between its individual components. Finding this order is important because it essentially determines the sequence in which each of the device components must be constructed [1]. The order relationships are determined by comparing each polygonal component in the list with each other as follows. Order relations are obtained between any two components based upon material and topological tests. If, for example, the construction of component \( y \) requires a temperature that component \( x \) cannot safely withstand, then it is necessary to form \( x \) before \( y \). This simple compatibility test is readily performed upon examination of the respective material recipes. Similarly, in the topological sense, if component \( y \) partly or completely rests on top of another component \( x \), then in general \( y \) must be constructed after \( x \). Therefore \( x \) precedes \( y \) or in short \( x \prec y \). These binary relations are transitive and irreflexive. Figure 3 de-
all relations can be represented in terms of a directed graph. Figure 4 represents a directed graph of the CMOS device shown in Fig. 3, where each vertex of the graph represents a component and an arc between any two components indicates an order relation. The idea of using graphs to specify the structure of complex multicomponent objects is not new. Similar connection graphs have been used in the construction of automated assembly plans [10]. The graph information is stored in an $n \times n$ square, boolean matrix called the adjacency matrix $A$, where $n$ is the number of device components. In this matrix, element $a_{ij} = 1$, iff component $x_i \prec \text{component } x_j$ and is 0, otherwise.

Order relations and adjacency matrices can also be defined on groups of components that are constructed at the same time from the same layer. Given any two component groups $g_a = \{x_1, \ldots, x_l\}$ and $g_b = \{x_m, \ldots, x_n\}$ then $g_a \prec g_b$ if for any $x_i \in g_a$, $x_j \in g_b$ there exist at least one relationship $x_i \prec x_j$. Thus the relationship between the groups is found as the logical OR of all relations between any two components from each group.

The algorithm shown in Fig. 5 generates the adjacency matrices for a set of polygon groups. The comparison between polygonal components depends on their types. Any two non-diffusion type components have an order relation when they have physical contact at any of their boundaries. This condition is detected by direct comparison of the component sides. The direction of the relation arc is however determined by decomposing the component boundaries into its “top” and “bottom” segments. These segments are determined in relation to an outward substrate normal direction. The precedence relation between two components $p_a \prec p_b$ is true when $|\text{top}(p_a) \cap \text{bot}(p_b)| > 0$ and $|\text{bot}(p_a) \cap \text{top}(p_b)| = 0$, where the vector operation $\vec{A} \land \vec{B}$ results in a vector containing the minimum number of overlapping elements between vectors $\vec{A}$ and $\vec{B}$. The first condition is indicative of a non-zero physical overlap between the top of polygon $p_a$ and the bottom of polygon $p_b$ while the second assures a well defined order. This simple order resolution scheme can be extended to resolve order relations between components that are not in direct contact [11] (such as suspended components) as well as peculiar components where there is no clearly identifiable top or bottom. Examples of elaborate order resolution schemes can be found in [12, 11].

Diffusion-type components are treated differently. This restricted set of diffusions obeys two order rules: (1) If two components are of diffusion type, no relation is established in the topological sense since they can both share the same physical space. (2) If a non-diffusion component (e.g., the substrate) encloses a diffusion, then the non-diffusion component precedes the diffusion.

- **Function GetAdjacencyMatrix**($G$)
  - Input: $G$ a set of sequentially labeled polygonal component groups
  - $n \leftarrow \text{NumberOfGroups}(G)$;  
  - $A \leftarrow \text{CreateMatrix}(n, n)$;  
  - $\forall g_a \in G$ do  
    - $\forall g_b \in G$ do  
      - if $(g_a \neq g_b)$ begin  
        - $i \leftarrow g_a\.label$;  
        - $j \leftarrow g_b\.label$;  
        - $A[i, j] \leftarrow 1$;  
      - end if  
    - end for  
  - end for  
  - return ($A$);  

Fig. 5. Algorithm for calculating the adjacency matrix from device schematic

B. Condensation of Component Groups

In an actual device, some of its components may belong to the same layer, e.g., components of an oxide layer grown in the LOCOS technique or components of a layer separated by
etch operations. In order to establish this hidden relation, the compiler attempts to merge or condense different components into groups belonging to the same layer. Graph condensations are useful means for controlling the magnitude of the problem. Whenever two components or groups are condensed into a larger one, the dimension of the device matrix is reduced by one and the number of relations is in general increased hence yielding a smaller solution set.

Component groups can be merged when they meet the following admissibility conditions: (1) they are the same material, (2) they have the same thickness, and (3) there is no order between them. Given the admissible group set \(S(G) = \{g_1, g_2, \ldots, g_j\}\) all \(g_i\)’s may be condensed together at the same time; however, this procedure can yield a condensed device graph which contains cycles [1]. Cycles specify inconsistent order relations hence the resulting graph cannot be sorted. The formation of cycles is a consequence of the fact that \(S\) is a function of the device graph \(G\); hence the admissible groups can change when any of these are condensed (since \(G\) has also changed). Condition (3) prevents the formation of cycles if the admissible groups are condensed in pairs. This is followed by the recalculation of the transitive closure and updated admissible group sets. The pairwise condensation is continued until no admissible groups remain.

In principle, any pair of admissible groups in \(S\) can be condensed in any order. However, an arbitrary condensation may lead to an awkward process. It is natural to assume that the most likely pair to be merged is that which is closest to a common supporting surface. Any group (or component) \(g_i\) is a support for \(g_j\) if \(g_i \prec g_j\). Further, \(g_i\) is a complete support if no group (component) adjacent to \(g_i\) is below \(g_j\). For example, the substrate is a complete support for all device components of a MOSFET, while the field oxide is a support for the metal and polysilicon gates.

Using these definitions, it is reasonable to select the two most likely groups of the set as those that rest at the lowest maximum height \(h_m(g_i, g_j^*)\) from their closest common supporting group \(g_j^*\). The maximal support height is defined as the maximum number of upward arcs in the device graph between the closest common support and the group hence it is indicative of the number of groups sandwiched between them. The height difference

\[
d_s(g_i, g_j) = | h_m(g_i, g_j^*) - h_m(g_j, g_j^*) |
\]

is in effect a measure of the closeness of similar groups with respect to the most likely common surface supporting them. In our scheme, the admissible pair with the lowest average support height

\[
\overline{h}_m = \frac{h_m(g_i, g_j^*) + h_m(g_j, g_j^*)}{2}
\]

is condensed first. If several pairs have the same \(\overline{h}_m\), then the pair closest to the substrate (represented by \(g_o\)) with minimum

\[
\overline{h}_m = \frac{h_m(g_i, g_o) + h_m(g_j, g_o)}{2}
\]

is condensed. In the device there may be more than one admissible group set. The first set used for the condensation is selected from the set height \(H_m^o(S)\) defined as the average height from each of its groups to the substrate.

\[
H_m^o(S) = \frac{1}{n} \sum_{i=1}^{n} h_m(g_i, g_o).
\]

Therefore, any of the sets with the lowest \(H_m^o\) are selected first, followed by the second, and so on. These procedures have been implemented by the two functions in the algorithm shown in Fig. 6. The function for condensing device groups, shown in

\[
\text{function } GetCondensationAdmissibleSets(\text{G})
\]

\[
\begin{align*}
\text{begin} \\
\quad \text{/* a set of sequentially labeled polygonal component groups */} \\
\quad \textbf{begin} \\
\quad \quad \text{/* compare every group with all other groups */} \\
\quad \quad \textbf{foreach} (g_a \in G) \textbf{do} \\
\quad \quad \quad S_a \leftarrow g_a; \\
\quad \quad \quad \textbf{foreach} (g_b \in G, b > a) \textbf{do} \\
\quad \quad \quad \quad \textbf{if} (\text{Admissible}(g_a, g_b, G)) \textbf{begin} \\
\quad \quad \quad \quad \quad S_a \leftarrow S_a + g_b; \\
\quad \quad \quad \quad \textbf{end if} \\
\quad \quad \textbf{end foreach} \\
\quad \textbf{end foreach} \\
\quad \textbf{return} (S); \\
\textbf{end}
\end{align*}
\]

\[
\text{begin}
\quad \text{/* a set of sequentially labeled polygonal component groups */} \\
\quad \textbf{begin} \\
\quad \quad \textbf{foreach} (g_a \in S_k) \textbf{do} \\
\quad \quad \quad \textbf{if} (\text{Admissible}(g_a, g_b, G)) \textbf{begin} \\
\quad \quad \quad \quad \text{CondenseDeviceGroups(AdmissiblePairs}); \\
\quad \quad \quad \text{CondenseDeviceGroups(G);} \\
\quad \text{end if} \\
\quad \textbf{end foreach} \\
\textbf{end}
\]

Fig. 6. Algorithm for (a) finding the admissible sets and (2) condensing the device groups

Fig. 6(b), finds and merges the two groups most likely to be condensed in a recursive fashion until no admissible groups remain. The function for finding admissible groups, shown in Fig. 6(a), finds all sets of groups which meet the admissibility conditions. Function \text{ClosestSupportDistance}(g_a, g_b) \text{ } finds the maximum (finite) distance between \(g_a\) and \(g_b\) for all admissible pairs of the set \(S_k\) using a modified version of Dijkstra’s algorithm [13]. The function \text{PairsLowestToSupport} \text{ } finds the set of most likely pairs for the condensation and stores them in \(P_m\). If \(|P_m| > 1\) then a single pair with the lowest distance to the substrate is selected in \(P^*\). This pair is condensed first, and \text{CondenseDeviceGroups} is then called recursively. Figure
7 shows a possible structure of the group list for the CMOS example shown in Fig. 3 after merging the component groups. For example, group label 5 contains components 9, 10, 11, 12, 13, and 14 all of which were grown with the same oxidation step.

The primary disadvantage of this scheme is that the resulting condensed graph, in general, only represents a subset of all possible condensations. This deficiency can be remedied at the expense of a more complex algorithm and substantial computational cost.

C. Estimating Cardinality of the Design Space

A computationally inexpensive a-priori measure of the design space cardinality \(||D|||\) is essential for the detection of a combinatorial explosion. This quantity is used to introduce reduction techniques and additional constraints if \(||D|||\) is too large before attempting the sequencing of the components. A suitable measure of \(||D|||\) is the number of linear extensions. It was shown in [1] that this quantity is bounded by the permanent of its position restrictor matrix \(R\) (i.e. a boolean order matrix based only upon the number of groups above and below). Thus

\[
||D|| \leq \text{per}(R) \leq \text{Ber}(R) = \prod_{i=1}^{N} \left( b_i! \right)^{1/b_i} = \text{Ber}(T + T^T),
\]

where \(\text{Ber}(R)\) is known as Bergman’s bound [14] for the permanent, and \(b_i\) is the number of ones in row \(i\) of \(R\). Matrix \(T\) is the transitive closure of \(G\), and \(T^T\) its transpose. Because only the sum of ones per row is involved, this bound is computationally efficient as it can be calculated in \(N\) steps. Using this a-priori estimation, it is possible to take appropriate measures to control the number of solutions. The pairwise condensation yields a much reduced design space with at least a reduction factor of

\[
||D||_{\text{condensed}} \leq \frac{||D||}{2^m},
\]

where \(m\) is the number of pairwise condensations.

D. Other Means of Controlling Design Space Cardinality

If the number of linear extensions of the device after the condensation is too large, additional constraints must be added to reduce it further. The following techniques are used.

D.1 Reactive Growth Order Based on Thickness

With the exception of pad oxide growth, growth of oxide layers is performed in decreasing order of their final thicknesses because this procedure saves masking steps. By default, MISTIC does not impose this rule; hence for every oxide layer two different processes are generated (masked and unmasked). Thus if a device contains \(m\) reactively grown oxide layers, the design space is multiplied by a factor of \(2^m\). This expansion mechanism is automatically shut off if \(||D||\) exceeds a specified upper limit leaving only the thick-to-thin order active.

D.2 Diffusion Order Based on Paired Diffusion Relations

By default, MISTIC does not assign orders between diffusions. This is done because their order depends on dopant diffusivities and diffusion temperatures which are yet to be determined as well as other field and high-concentration nonlinearities. The effect of leaving these components unordered is a large expansion in \(||D||\). If \(m\) diffusions do not hold any order, \(||D||\) is expanded by a factor of \(m!\). In devices that have many diffusions this becomes a problem. For example, if \(m = 7\), then there exists at least \(8! = 40,320\) feasible solutions. The factorial growth can be controlled by establishing approximate orders between the diffusions based upon causality in the drive-in times since none of these times can be negative. All diffused components in MISTIC are formed by an implantation step followed by a drive-in cycle. These components are specified in terms of their junction depth \(x_j\) and peak concentration \(N_p\). An approximate order can be established between two diffusions under the assumptions that implants are shallow and final profiles are Gaussian-like. These assumptions are often too coarse for finding accurate parameters but usually provide enough information to establish a precedence order.

E. Modification of the Transitive Closure Matrix for Implant Through Thin Layers

In thin-film devices, dopants are often implanted through thin layers. The ordering scheme of Section II creates an arc from the diffusion to the layer thus precluding the formation of the diffused layer after the growth. This is rectified by selectively removing these arcs while maintaining the rest of the order relations intact. The procedure followed here is slightly different than that described in [1].

A dielectric component group \(g_i\) is transparent to an implant when its thickness is less than a threshold value \(t(g_i) \leq d_i\). The set \(C(g_j)\) of component groups that immediately cover diffusion \(g_j\) is first found from the adjacency matrix. In order to allow the deposition of these transparent groups to occur prior to the diffusion, the device graph is modified. First, all incoming arcs in the diffusion vertex are extended to all vertices immediately above it. Let \(W(C)\) represent the subset of source vertices in the subgraph containing vertices \(g_i \in C\). For each \(g_i \in W(C)\), if \(g_i\) is transparent, the arc between the diffusion and \(g_i\) is removed and an arc is added from the diffusion to all elements above \(g_i\) which would prevent the ion implantation from taking place, namely those elements above \(g_i\) which are in a direct line of sight normal to the diffusion. This blocking relation thus takes the place of the contact relation used in Section II. For example, Fig. 8 shows how the adjacency relations are changed. In the original adjacency graph of Fig. 8(b), there is an arc from the diffusion \(g_1\) to oxide \(g_2\). Since \(g_2\) is thin, it is in direct contact, and there is no layer between \(g_1\) and \(g_2\), hence the implantation
can take place through $g_2$. In the case of diffusions, the upward arcs from $g_4$ to those above $g_1$ are already present (due to the contact relation). Next the $g_1$ to $g_2$ arc is removed. The last step consists of adding arcs for all components above $g_2$ that block $g_1$; hence creating an arc from $g_1$ to $g_5$. The modified adjacency graph is shown in Fig. 8(c).

Currently MISTIC allows for implants through all transparent layers, with $d_t$ being a preset parameter. MISTIC can also be optionally set to allow only processes where the transparent layers fully cover the diffusion opening, hence $\|W(C)\| = 1$, in order to prevent the formation of a stepped implant.

IV. THE LAYER SEQUENCE

The bounded device matrix contains all the basic order constraints between the component groups as shown from step 3 of Fig. 1. In thin film devices, each of its components (groups) are constructed one at a time in a sequential manner. Therefore, the next step in the compilation consists of finding all linear vertex lists (or linear extensions) consistent with the matrix order. This procedure is known as topological sorting [15]. After the vertex sorting, group structures are converted to layers which hold additional information used at later stages in the compilation. Each of the sequenced layer lists is a high-level abstraction of its corresponding detailed process flows. The flow steps are hence obtained by expanding the layer list as discussed below. Reactive growth layer sequencing is discussed in this section as a practical example.

A. Generation of Linear Extensions

The linear extensions can be generated in a variety of ways [16, 15, 17–20]. MISTIC uses a modified version of Steiner’s sequential algorithm [1] which does not create any duplicate or incorrect lists. Because of its recursive, depth first nature the algorithm completes each linear extension before attempting to generate the next; hence it does not require much storage. The algorithm requires two lists of nodes. List node_list contains nodes not yet included in linear extension lin_ext. Initially, node_list contains nodes ordered by their identifying label, and lin_ext is null. In each invocation the beginning node of node_list is selected and the list minus this node is duplicated in new_node_list. The algorithm next attempts to insert the selected node at all admissible positions in lin_ext according to $T$. If an admissible position $x$ is found, then a new augmented linear extension new_lin_ext is formed and the function is called recursively. When node_list is exhausted, a linear extension is completed and expanded into a process flow. Each of the nodes in the linear extensions is next converted into a layer structure. The layer data structure has the same attributes as the corresponding group and a list of segments identifying the regions where the group block is the line of sight. This information is later used in the lithography steps. This layer sequence specifies a skeleton process flow since each layer type indicates the steps required to form it. For example, a deposited layer requires a deposition step, a diffused layer requires an ion implantation and subsequent diffusion of the dopants, and so on.

Figure 9 shows a layer sequence for a typical MOSFET. The layer list does not indicate the placement of etching steps nor any other steps that arise from layers that are missing in the final device structure, for example, pad oxide, and sacrificial layers. MISTIC inserts these missing steps according to predetermined rules. The sections below discuss some of these rules.

B. Reactive Growth Schemes

MISTIC implements reactive growth in three basic schemes
shown in Fig. 10 obeying the following rules: (1) the growth of thick layers is always influenced by thinner ones disregarding their order because doing so reduces the overall reactive-growth time and eliminates unnecessary masking steps, (2) if the growth of thick layers precedes that of thinner ones then the thin growth area is either etched back or protected by a barrier. (c) if the thicker growth follows the thinner one, the thin layer must be protected with a barrier.

Fig. 10. Reactive growth schemes used in MISTIC which yield the same structure. If the thinner growth follows the thicker one, the growth area is either (a) etched back or (b) protected by a barrier. (c) if the thicker growth follows the thinner one, the thin layer must be protected with a barrier.

The various reactive growth schemes result in a corresponding multiplication of the process flow set. MISTIC duplicates each layer list and attaches a boolean flag named protect to each reactive growth layer in the sequenced layer list. For each reactively grown layer this flag determines how other reactive areas of subsequent thinner growths and no desired growth are treated during the growth. These two type of regions may be either protected by a barrier deposited at any point prior to the growth or back etched at any point later determined by the topology of subsequently deposited layers. The recursive algorithm shown in Fig. 11 shows how a single layer list is duplicated and expanded into several lists according to the reactive growth schemes discussed above.

After the generation of these lists, the auxiliary barrier layer is added. The insertion of this layer demands the presence of a reactive surface. This condition however cannot be completely determined until the masks for the growth are known. These masks consist of a linked list of line segments specifying the device regions where the external environment acts on the wafer surface. Since MISTIC allows both barrier and etch back schemes, two types of masks are required a growth mask $M_G$ and an etch back mask $M_E$. The growth mask represents the regions subject to the current growth environment hence it determines the pattern of the barrier layer. The growth mask is constructed from line segments from three types of layers: (1) partially grown layers, (2) unprotected layers of following thinner growths, and (3) other unprotected reactive layers. Similarly, the etch back mask represents the regions where the growth in the reactive unprotected regions is removed, namely segments of type (2) and (3).

In order to clarify the method of construction for both $M_G$ and $M_E$, it is useful to define the following set of quantities. Let $G_i$ represent the set of line segments for the current growth. We are interested in the relations between $G_i$ and all other growths of the process. The set $G_{<i}$ consists of all line segments of all growths following $G_i$. Similarly $G_{>i}$ contains line segments of all growths prior to $G_i$. These two sets are split onto four sub-sets.

$$G_{=} = G_{>i} + G_{<T},$$

$$G_{<} = G_{<i} + G_{>T},$$

where the $i$ and $T$ subscripts correspond to growths with final thicknesses thinner and thicker than that for $G_i$. Finally, let $N_i$ be the set containing the line segments of reactive surfaces with no desired growth during the current growth. Using these definitions, masks $M_G$ and $M_E$ can be expressed in terms of boolean operations between these sets

$$M_G = (G_{<i} + G_{<T} \cdot p_G + N_i \cdot p_G),$$

$$M_E = (G_{>i} \cdot P_G + N_i \cdot P_G),$$

where $p_G \equiv \text{protect}$. At this time in the compilation $G_{<i}$ and $G_{>i}$ are known but $N_i$ is not since the latter is determined by

```plaintext
function MultiplyForGrowthOptions (lsseq_list, growth_serial)
/* lsseq_list is a list of elements where each element contains a list of sequenced layers. New elements are added to the end of this list, if new reactive growth choice seems to be feasible. */
growth_serial indicates the particular oxide growth that is to be considered for different possible reactive growth choices. The reactive growths are assigned serial numbers as 0, 1, 2, etc. */
begin
/* traverse the list */
while (lsseq_list != NULL) do
l ← FindGrowthLayer (lsseq.layers, growth_serial);
if (l = NULL) return (lsseq_list);
/* make protect TRUE for this layer */
l.growthInfo.protection ← TRUE;
/* if thinner growth regions exist or no grow area exist, create a new layer sequence with protect FALSE */
if (ThinnerGrowthExist (l) or NoGrowAreasExist (l)) begin
new layer_list ← DuplicateLayers (lsseq.layers);
end if
l ← FindGrowthLayer (new layer_list, growth_serial);
l.growthInfo.protection ← FALSE;
AddNewLayerSeq (new layer_list, lsseq_list);
end if
lsseq_list ← MultiplyForGrowthOptions (lsseq_list, growth_serial + 1);
return (lsseq_list);
end
```
the profile at the wafer surface which depends on the order of the etching and stripping steps not yet inserted in the process. To approximate \( N_i \), MISTIC makes the initial assumption that all surfaces not covered by the barrier layer during the growth are reactive. This is a reasonable assumption since most of the growth occurs early in the device process

\[
N_i \approx N_A = (G_i + G_r + G_s) \quad (10)
\]

Hence, this is only an approximation since during the growth there may be some parts of the wafer surface that are non-reactive hence \( N_i \subset N_A \). The actual amount of barrier layer present during a growth is later checked and partially corrected during the creation of the initial step list described in the next section. Both masks in Eq. (9) are finally corrected after the etch steps are inserted. Once the approximate masks are calculated, the corresponding barrier layers must be added to the layer list. In principle each protected reactive growth could be performed using a barrier layer deposited just prior to the growth and stripped immediately after it. However, the use of multiple barriers requires repatterning of the same barrier features.

The pattern redefinition results in severe feature misalignments and a low process yield. In order to avoid these errors, MISTIC allows only a single deposition of the barrier for the entire process. This layer must be used by all the protected reactive growth cycles in the process by selectively etching away parts of the barrier that are affected in each growth.

In accordance with the scheme discussed above, a single barrier layer record is added to the layer list prior to the first reactively grown layer for which \( M_G \neq 0 \). It is also customary to grow a thin pad oxide before the barrier if the barrier is deposited on top of a bare silicon surface. This thin pad oxide reduces the stress at the silicon interface hence minimizing the generation of interfacial defects. The pad oxide layer is then inserted before the growth if

\[
G_{st} \cdot p_G + N_i \cdot p_G \neq 0. \quad (11)
\]

Currently MISTIC supports only the basic reactive growth schemes discussed above. Many other more complex schemes are possible [21].

V. THE BASIC STEP SEQUENCE

Each layer sequence is next converted into a sequence of steps, as shown in step 4 of Fig. 1. Figure 12 shows the algorithm used for the creation of the initial step list. For each layer in the layer list this algorithm inserts or adds more steps to the step list corresponding to the operations required to form the layer. This seemingly straightforward procedure is however complicated by the presence of barrier layers. These temporary layers can block access to the surface because they were not considered during the generation of the component sequences. Therefore the placement of patterning and strip steps for the temporary barrier layers must be performed in a way that does not interfere with the original layer order. If this cannot be accomplished, the sequence is discarded. Figure 12 shows how the initial step list is created.

Obviously, the initial layer list is an incomplete representation of the corresponding process flows. This list represents only the process steps that create the layers, but it does not specify the process steps that change or modify them such as etching steps.

A. Insertion of Pad Oxide and \( Si_3N_4 \) Barrier Patterning Steps

Determining the right point of insertion for the pad oxide removal step is critical. During processing, the pad oxide layer can be easily contaminated by photoresist and other contaminants. On the other hand, it is a good idea to keep the pad oxide on the surface as long as possible to prevent contamination of the underlying silicon surface. Therefore, this film is often removed from the wafer just prior to a thin oxidation step. In this blanket etching step the pad oxide layer is completely removed; hence if other oxide layers remain exposed during these steps they lose
a small amount of their thickness which is usually negligible. The algorithm shown in Fig. 12 first detects the presence of the pad oxide and barrier layers and subsequently adds patterning and etch steps for these layers to the step list whenever needed by other subsequent step types. These patterning steps require knowledge of the current state of the barrier, namely in the form of which fragments are present at any given time. This information is stored in a special grid data structure described in detail in section V-B.1.

If a diffused layer is encountered in the step list and the barrier layer is present, it must be checked if it blocks all the consecutive diffusion steps. In addition, it must be checked that it is etched appropriately by modifying its grid structure and adding the etch or strip steps to the step list prior to the insertion of the diffusion steps.

If a reactive growth layer is encountered in the list and the barrier is present, then function CheckBarrierForGrowth, shown in Fig. 13, determines if it can be used as a barrier. The presence of these barriers influences the placement of steps in the initial step list. There are two different cases involved. If the growth requires the use of a barrier, the algorithm removes the barrier on the segments where the growth must take place. If, however, there is insufficient barrier material to protect the surface then the process is discarded. If the growth does not require the barrier, then the function checks if there is enough barrier present to protect thinner oxide layers, and inserts appropriate etch or strip steps. Since the placement of the barriers and the interaction with the reactive growths is now known, the masks can be corrected.

After the growth step is inserted, the algorithm also checks if the barrier layer is needed for any further reactive growth steps. If this layer is no longer used, it inserts the final barrier and pad oxide stripping steps into the step list. Figure 14 shows the resulting step sequence after insertion of pad oxide and Si₃N₄ barrier removal steps.

B. Insertion of Etching Steps

The next step, step 5 in Fig. 1, is the insertion of etch steps. The insertion of etch steps requires knowledge of the top wafer profile and the layers immediately beneath etched regions. This information determines the set of exposed materials Mₑ needed for the selection of a high selectivity etchant.

In principle, once this information is available, single etching steps can be inserted immediately after depositions. This naive step placement is subject to several problems. If an adequate etchant cannot be found for a given Mₑ, the naive scheme yields an etch failure. In these situations, a successful etch may still be possible if it is divided into several substeps, each substep defining different gaps of the same layer. This procedure splits Mₑ into several smaller subsets hence relaxing the selectivity requirements.

Further, the naive placement may yield processes that have redundant steps. This situation develops when adjacent layers (in the order sense) have the same etch patterns as for example contact holes and via structures that cut through several layers. In these cases two or more etching steps can be condensed into one step. Because of the implicit correspondence between the gaps and etch steps, the etch insertion algorithm must consider not only the presence of the layer but also order relations between gaps of different layers. The gaps hold blocking and visibility relations which essentially determine etch step placements in the step list.

In order to determine Mₑ and the gap relations, MISTIC uses a special grid data structure that describes vertical slices of topologically different adjacent regions of the wafer. This scheme assumes that all points contained within a slice are equivalent since they are subject to the same process conditions. Therefore the grid state is sufficient to describe the state of the wafer at any time.

B.1 The Grid Structure and Ordered Gap List

The grid is a two dimensional data structure used to determine the effects of deposition, growth, and etching on all equivalent regions of the device. The grid is constructed from the basic device geometry as shown in Fig. 15. First all device components are scanned.

Each component contributes two point coordinates corresponding to its horizontal outer boundaries. These coordinates are next sorted in increasing order, and, after elimination of du-

---

**Function CheckBarrierForGrowth**

```plaintext
function CheckBarrierForGrowth(l, bar r, bar yr, pad yr, stp st)
begin
  if (Protected( ) = TRUE ) begin
    /* create barrier etch step */
    gp st ← EtchBarrierAtGrowthSegs(l, b arr);
  if (BarrierStillPresent (b arr) = TRUE ) begin
    AddBarrierAndPadEtchSteps(gp st, bar yr, pad yr, stp st);
  else
    AddBarrierAndPadStripSteps(bar yr, pad yr, stp st);
  end if
end if
/* No barrier at protected segs */
DelProcStepList (stp st);
return FALSE;
end if
/* if not protected */
if (BarrierPresentAtNotAffectSegs(l, b arr)) begin
  if (BarrierPresentAtGrowthSegs(l, b arr)) begin
    gap st ← EtchUndesiredBarrierSegs(l, b arr);
  end if
  if (BarrierStillPresent (b arr) = TRUE )
    AddBarrierAndPadEtchSteps(gap st, bar yr, pad yr, stp st);
  else
    AddBarrierAndPadStriping(bar yr, pad yr, stp st);
end if
/* redundant process, because same process created for ‘protect’ TRUE */
DelProcStepList (stp st);
return FALSE;
end if
CorrectGrowthMask (l, growth info, mask, etch back gaps);
else begin
/* infeasible process, because barrier not present at segments that should be protected */
DelProcStepList (stp st);
return FALSE;
end if
end
end
```

Fig. 13. Algorithm used for preparing barrier layer for growth.
constructed for each layer terminated by a header. To create a replicated points, a one dimensional chain of point records is constructed from the gap list. This procedure is repeated gap is removed from the gap list. This procedure is repeated recursively until the gap list is empty, and the etch insertion is completed. The gap list is constructed by sorting all gaps in increasing distances from the lower leftmost corner of the device starting from the substrate and ending at the device rightmost corner of its top layer.

B.2 Recursive Insertion Algorithm

A gap can be etched in a layer anytime after the layer formation but prior to the deposition of a layer that blocks it as shown in the example of Fig. 16. The blocking relationship between gaps and successive layers may be different for different gaps of the same layer. Therefore, a given layer may be etched several times using multiple masks. This split-mask etch scheme is useful because it relaxes the selectivity requirements for the etchant.

At any given time \( n \), the gap list \( E_T \) is partitioned onto several sub-sets. The causal set \( E_c \) is the set of gaps from all layers that are present in the device at time \( n \) and the future gap set \( E_f \) consists of all the rest. Thus

\[
E_T = E_c + E_f. \tag{12}
\]

The causal set is further partitioned into essential \( E_e \) and non-essential \( E_n \) gaps sets.

\[
E_c = E_e + E_n. \tag{13}
\]

A causal gap is essential whenever it is blocked by layer \( l_n \); hence it must be etched at this time while non-essential gaps can be etched at a later time. For example, in Fig. 21 these sets are

\[
E_e(N) = \{ b, d, f \}, \\
E_n(N) = \{ a, c, e \}, \\
E_f(N) = \{ g, h, i, j, k \}, \\
E_e(N + 1) = \{ a, c, d, g, h \}, \\
E_n(N + 1) = \{ e, i \}, \\
E_f(N + 1) = \{ j, k \}.
\]

The etch insertion algorithm first scans the initial step list and constructs \( E_e(n) \) and \( E_n(n) \) for layer \( l_n \). Next it creates two step lists corresponding to two different etching schemes. The first list contains etching steps for \( E_e \) while the second only contains etching steps for \( E_n \), delaying the insertion of \( E_n \). Both of these steps are inserted prior to the deposition step for \( l_n \).

The step list under construction, the ordered gap list, and the grid structure are passed to the recursive function ExpandEtchSteps, shown in Fig. 17. This function inserts each etch step into a newly expanded step list and modifies the grid structure accordingly. Functions InsertEtchSteps and InsertNonEtchStep insert steps into the step list under construction and modify the grid accordingly. Function InsertEtchSteps, shown in Fig. 18, is invoked twice with causal and essential etch gaps generating two process flows for every etch step. It detects etch failures without finding a definite etchant by examining the etch similarity of the materials exposed to the etch. All database materials that do not have selective etchants with respect to one another are called etch-similar.

Function InsertEtchSteps first scans the etch gap list from the last entry to the first and searches for the first gap entry that etches a non-similar material as the last (which is etched first). It
next partitions the list at this point returning the last etch-similar set of gaps that can be formed by the same etch step, and the set is removed from the gap list. The etch-similar gap set list is passed to function \texttt{InsertEtchSimilarMatEtchSteps}. This function first adds a photosist layer to the grid. It next modifies the photosist and the corresponding layers in the grid for all the gaps in the list. The resulting grid contains a representation of the topography at the surface from which the list of materials that were exposed to the etchant is generated. If any of these belong to the same material etch group as the etched layers then the etch was not successful and is signaled as a false return.

If the selectivity check was successful, the gaps are next grouped and assembled into etching steps. When multiple layers are etched, layers are attacked from top to bottom. Since gaps located at the end of the gap list correspond to the top layer, they are assembled into etch steps first. These gaps are then removed from gap list, and the procedure is continued until the gap list is empty. This procedure may generate redundant etching steps which are later condensed into one. Figure 19 shows the expanded step list after insertion of the etch steps as shown in Fig. 14. After the expansion of etch steps, adjacent etch steps that pattern layers of the same material are merged into a single step. This situation occurs, for example, during the etching of contact holes where the same etch removes multiple layers of silicon dioxide.

C. Correction of Reactive Growth Steps

After the insertion of etch steps is completed, the reactive growth barrier and etch back masks are corrected. The main purpose of this correction is the elimination of unnecessary barrier deposition and pad oxide growth steps. A barrier is unnecessary if the regions which are not subject to the growth are non-reactive. From the grid topography, the set $N_i$ containing the line segments of reactive surfaces with no desired growth is accurately calculated, and Eq. (9) are re-evaluated (but not changed). If $M_{G1} = 0$ the corresponding barrier and pad oxide deposition and strip steps are removed from the step list.

D. Insertion of Lithography Steps

A lithography step is required before any masked etching steps. Lithography steps may be unnecessary in some cases. If
the preceding step is an etching step with the same mask set, this etch step does not require a separate lithography. The algorithm for lithography insertion takes this case into account condensing adjacent etch steps that operate on layers of similar materials.

VI. THE PROCESS SPECIFICATION LIST

The basic step sequence provides a qualitative description of the actual process flow, but it does not hold specific step information such as process times, temperatures, and other general recipe information. Each process step in the list may contain a set of sub-steps. Each of these sub-steps may contain a set of written instructions and additional parameters which depend on the step or sub-step operation. This specific step information is determined from a single parameterized recipe which is selected from the compiler database. The generation of the process specification list from the step list is discussed in Part II.

VII. SUMMARY

This paper describes the most important algorithms used in the planar process compiler MISTIC. These algorithms take as input a cross section of a device structure and generates device process flows consistent with lithographic, laboratory resources, and other process constraints. The algorithms discussed in Part I generate process flows which require many parameters. Further, typically many flows are generated for a given device structure; therefore a grading system must be devised. The calculation of the flow parameters, flow figure of merit, and the performance evaluation of the MISTIC system are discussed in Part II.

VIII. ACKNOWLEDGMENTS

We thank Dr. S. B. Crary for numerous helpful discussions. This research was sponsored by the National Science Foundation under grant ECS-9309229.

REFERENCES