EXTRACTION OF TOPOGRAPHY DEPENDENT ELECTRICAL CHARACTERISTICS FROM PROCESS SIMULATION USING SIMPL, WITH APPLICATION TO PLANARIZATION AND DENSE INTERCONNECT TECHNOLOGIES

by

Edward W. Scheckler

Memorandum No. UCB/ERL M89/72

8 June 1989

COVER PAGE

EXTRACTION OF TOPOGRAPHY DEPENDENT ELECTRICAL CHARACTERISTICS FROM PROCESS SIMULATION USING SIMPL, WITH APPLICATION TO PLANARIZATION AND DENSE INTERCONNECT TECHNOLOGIES

by

Edward W. Scheckler

Memorandum No. UCB/ERL M89/72

8 June 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

EXTRACTION OF TOPOGRAPHY DEPENDENT ELECTRICAL CHARACTERISTICS FROM PROCESS SIMULATION USING SIMPL, WITH APPLICATION TO PLANARIZATION AND DENSE INTERCONNECT TECHNOLOGIES

by

Edward W. Scheckler

Memorandum No. UCB/ERL M89/72

8 June 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Extraction of Topography Dependent Electrical Characteristics from Process Simulation using SIMPL, with Application to Planarization and Dense Interconnect Technologies.

Edward W. Scheckler

Department of Electrical Engineering and Computer Sciences

University of California, Berkeley, California 94720, U.S.A.

ABSTRACT

This project demonstrates the use of SIMPL-2 (SIMulated Profiles from the Layout) and SIMPL-DIX (Design interface with X windows) as an interface to other process and device simulators. An interface to RACPLE for analyzing topography dependent parasitic resistances and capacitances is implemented. Enhancements to SIMPL to call the non-planar etch simulation capabilities of SAMPLE are also presented. These integrated CAD tools are applied to a patterned photoresist planarization process, and to VLSI Hopfield neural networks. It is found that the patterned photoresist planarization process shows a relatively high tolerance to reasonable misalignments. VLSI neural networks show significant topography dependent RC parasitic delays which increase as the square of the number of neurons. Based on experience gained as a result of this work, several suggestions for the future of SIMPL are offered.

December 19, 1988

Dedicated to my family with thanks for their continued love and support throughout my education.

Acknowledgements

I have been fortunate to have the support of many people in pursuing this research. First, I would like to thank my research adviser, Professor Andrew R. Neureuther, for his valuable comments, his continuing encouragement, and the generosity he has shown with his time. Professor Neureuther's seemingly limitless energy, and his profound ability are matched by the concern he shows for his students and his enthusiasm for new ideas. His positive outlook and attitude continues to be a source of inspiration and I look forward to future collaborative efforts.

Several students are also deserving of thanks. Alex Wong's work with SIMPL, his ability as a programming expert, and his ideas about this project have been invaluable. Without his contribution, much of this project would not have been possible. Simon Koh got me started with SIMPL and helped me through those first months working with these programs. Joe Wu, the author of SIMPL-DIX, is to be thanked for his hard work in writing a good program. Sherman Kwok has always been available for help with SAMPLE. Don Lyons provided valuable discussions of process technologies. Other students of Professors Neureuther and Oldham, especially Rich Ferguson, Nelson Tam, Kenny Toh, John Gamelin, Bill Bell, Bill Partlo, Dean Drako, Pantas Sutardja, Rama Sutardja, Carl Galewski, and Gino Addiego, have contributed to an environment of teamwork and camaraderie that has made work here more enjoyable.

I would also like to thank the numerous visitors from industry who have provided stimulating discussions in the general area of process and device modeling. Special thanks are given to Simon Polak of the Nederlandse Philips Bedrijven, B.V. in Eindhoven, the Netherlands, who provided me with an opportunity to see how CAD tools are used in industry and challenged me with probing questions on the nature of our group's research efforts at UC-Berkeley.

Thanks to Professor W.G. Oldham for reviewing this report.

The financial support of AFOSR JSEP grant F49620-87-C0041, SEMATECH COE grant 88-MC-500, and the California State MICRO Program grant 88-092 is gratefully acknowledged.

Table of Contents

1. Introduction

1.1. Background

Process and device design for modern integrated circuits depends on a complex interrelationship of manufacturing techniques, physical phenomena, device requirements, and designer experience. With increases in integrated circuit complexity and decreases in device feature dimensions, computer-aided-design software has come to play an important role in product development. It is now practically inconceivable that an integrated circuit could be designed without the help of circuit simulators such as SPICE [Nag75] and layout routing tools such as TimberWolf [Sec85]. Most commercial enterprises involved in chip-making have robust software and computing environments available for IC simulation, routing, and design. Such is however not the case for process and device modeling. Despite the many successes of device simulators such as PISCES [Pin84]] and MINIMOS [Sel80], and process simulators such as SAMPLE [Old79] and SUPREM [Ho83a], process and device simulation is not nearly as well established in industry as circuit simulation and layout automation. This is evidenced by the fact, that whereas it is now nearly impossible to design a circuit without a circuit simulator, it is still possible for device and process engineers to do their jobs without availing themselves of the software capabilities at hand.

Process and device modeling is a difficult task given the variety and complexity of physical phenomena which must be considered. Even when the physics of a certain situation can be reduced to a handful of equations, often algorithms for numerical solution require computing power beyond that readily available. The challenge for process and device simulation involves developing accurate models, determining parameter values for those models, and then implementing algorithms which perform the necessary calculations in a reasonable amount of time. To achieve these goals, many existing simulators depend on simplifying assumptions which restrict their applicability and accuracy. SUPREM-III [Ho83b] only handles diffusion and oxidation in one dimension. SAMPLE-1.6a [Add85] requires planar layers for lithography and etching simulation in two dimensions. Additionally, many of the physical models used by process simulators are heavily dependent on empirical results. SAMPLE, for example, requires experimentally derived development rate equations and curve-fitting parameters for each photoresist material it is required to simulate. The empirical nature of many of the models limits the range over which input parameters may be varied and thus limits the usefulness of the simulation software.

New versions of these programs, however, are closing the gap between process simulation and process reality. SAMPLE-1.7a [SAM88] includes models for non-planar etching, and advanced resist chemistries. A program for development rate parameter extraction, PARMEX [Bel88], will soon be available for rapidly characterizing photoresists for SAMPLE. SUPREM-IV [Raf86] handles diffusion in two dimensions. CREEP [Sut87] is a new program for oxidation, annealing and reflow simulation. COMPOSITE [Lor85] includes many improved etching models as well as providing an extensive library of topography, diffusion and implantation simulation programs.

Advances in process simulation are moving in the direction of complete three-dimensional modeling. Already, the program SPLAT [Toh88] is capable of simulating the variation of light intensity throughout a field on a wafer in to a microlithography system, including the effects of lens aberrations. Future work will involve simulating resist dissolution and pattern transfer in three dimensions in order to give a more complete physical picture of process phenomena. The advances in physical modeling for process simulators, coupled with the recent availability of powerful "super-minicomputers" indicates that this trend towards improved process simulation will continue.

1.2. Problem of Process CAD Integration

One problem that has arisen as a direct result of the increased number of simulation programs available, is the difficulty of passing data from one simulator to another. Production line processes depend on the interaction of several different processing steps. Existing process simulators generally concentrate on only a certain fraction of possible processes. For many research and design applications this is sufficient, but there is now a more pressing need for integrating the many available CAD programs for process simulation. It is also desirable to use process simulators to create input data for device simulators so that the electrical characteristics of a given device can be examined with respect to process variations. Additionally, process simulators should be connected to the layout so that the effect of design rules and mask alignment on topography can be considered.

The latter problem of connecting the layout to the simulator has already been addressed by SIMPL-1 and SIMPL-2 [Gri84], [Lee85]. Both of these programs were designed to generate a device cross section along a cut-line on the layout based on a process flow description. SIMPL-2 can represent a device cross section using arbitrary polygons and is therefore quite general when it comes to describing a device geometry. SIMPL-2 lacks sophisticated process models, however, and is thus of primary interest to layout designers but not to process engineers. SIMPL-2 offers rigorous simulation only in the case of metal deposition, in which case SAMPLE is called. SIMPL-2 also lacks many convenient features from a user's point of view. The graphics are slow and it is not possible to manipulate the display easily.

SIMPL-DIX [Wu88a] solves many of the problems of graphics and user friendliness by providing a design interface using the X window system [Sch86]. SIMPL-DIX also supports a variety of features for analyzing the effects of mask misalignment (WORST), problems due to the layout and geometric effects (HUNCH), and characteristics of the simulated device profile itself (CRITIC). Furthermore, SIMPL-DIX combined with SIMPL-2 is well positioned to fill the need for an integrated design environment, providing for the flow of information between process simulators as well as acting as a front-end input generator for device simulators. Some initial work in integrating the SIMPL programs with CREEP, SUPREM-III and PISCES-II has recently been reported [Wu88b]. Figure 1.1 presents a summary of existing and proposed tools integration using SIMPL.

1.3. Project Overview

It is the intent of this project to demonstrate the feasibility and flexibility of SIMPL as an interface to process and device simulators. The usefulness of this approach will be demonstrated as SIMPL is applied to a few selected problems suited for analysis with these programs. SIMPL has been connected with RACPLE [Lee83] for analyzing topography dependent resistive and capacitive parasitics.

SIMPL is also used for displaying the results of non-planar etch simulation using SAMPLE-1.7a. This report details the new algorithms and interface code needed to integrate RACPLE and SAMPLE non-planar etch with SIMPL. These integrated CAD tools are applied to problems of multilevel planarization schemes and dense interconnect technologies. Finally, based on experience derived from working on this project, several suggestions for improving the SIMPL programs are offered.

2. A General Approach to Software Integration with SIMPL

2.1. SIMPL-DIX and SIMPL-2

SIMPL-DIX and SIMPL-2 are two separate programs which, together with SIMPL-1, form a suite of programs known generically as SIMPL (SIMulated Profiles from the Layout). It is worthwhile to understand the capabilities and purposes of these programs in a general way before considering their specific implementations. SIMPL-2 was originally developed to allow a layout designer to rapidly determine the cross section of a device which would result from a particular layout. To achieve this, SIMPL-2 uses arbitrary polygons to represent device cross sections. Crude, but fast models for simulating deposition, etching, exposure, development, oxidation, and ion implantation are available. SIMPL-2 is capable of displaying two-dimensional process effects such as "bird's beak" oxidation, lateral diffusion under a mask edge, and undercut in etching. A link to SAMPLE for rigorous simulation of metal deposition is included to describe sidewall and step coverage accurately. Using SIMPL-2, it is possible to generate a realistic cross section of a device such as the Berkeley CMOS inverter [Gib86].

SIMPL-DIX was developed to provide a high-level graphics interface and convenient user environment for programs like SIMPL-2. A major goal of SIMPL-DIX is to provide an environment for integrating dissimilar process and device simulation programs. SIMPL-DIX uses the X window system for displaying graphics information and for generating a menu-driven user interface. SIMPL-DIX currently invokes SIMPL-2 to create the database needed to describe a device cross section. SIMPL-DIX also maintains its own database for describing all polygons in a profile, but this database is primarily intended for storing graphics information and thus lacks many of the features of the SIMPL-2 database. For example, SIMPL-DIX does not maintain data describing polygons adjacent to a given polygon, whereas SIMPL-2 contains the material type and relative location of all polygons sharing a boundary or vertex point.

In addition to the link with SIMPL-2, SIMPL-DIX has a number of internal tools to assist the designer in performing and analyzing simulations. This points to the ability of SIMPL-DIX to analyze a profile or layout generated by other programs. The HUNCH feature allows a designer to specify logical operations between masks or sets of mask to identify locations where topographical problems are anticipated. The CRITIC feature is currently being developed to allow aspects of a device profile to be investigated automatically.

The SIMPL programs together form a suitable design environment for process and device simulation. SIMPL-DIX provides a convenient graphics and user interface, whereas SIMPL-2 maintains the most complete data base for describing device profiles. The original goal of SIMPL to link process simulation with the device layout can now be extended to include integration of other device and process simulators.

2.2. Profile Data Representations

Several schemes exist for representing device cross sections. Most can be classified as belonging to either one of two types: linked polygons or multiple layers. SIMPL-2 uses linked polygons to describe a device profile. This technique offers the advantage of being quite general. Nearly all device profiles can be conveniently described in this way. Polygons can be added, removed, or altered by using a suite of subroutines to manipulate the database. Figure 2.1 shows the basic setup of the linked polygon data structure.

An alternative approach to representing device cross sections is to use multiple layers. This method has certain computational advantages for simulating processes such as etching, resist dissolution, and deposition. SAMPLE and COMPOSITE use multiple layers to represent device profiles. Generally, a layer must span the entire length of a simulation window. In regions where no material of a given layer type exists, it is possible to specify a layer of zero thickness. Some programs, like COMPOSITE, do not require that a layer span the entire simulation window, and instead maintain an additional flag at each vertex in the layer to describe whether the point is the starting or ending point of a layer. Figure 2.2 shows how layers can be used to describe a profile in SAMPLE.

In order to integrate programs, it is necessary to translate data between different data representations. It is not difficult to translate a polygonal data representation into a series of layers. This can be achieved by writing the surface of the profile as one layer, removing one surface polygon, writing the new surface as the next layer, and so on down to the substrate. Going from layers to polygons, however, is somewhat more difficult. Two contours, describing one polygon, must be identified and merged, with extraneous points being removed. Fortunately, there are many special cases where it is not necessary to translate an entire profile back and forth between different simulators. In the case of layer deposition, only the surface layer need be sent, and only the new deposited layer must be returned. In the case of etch simulation, usually the entire cross section must be sent, but only the new resulting surface contour need be returned. In some special cases, it is a trivial matter to include this new contour in the original profile. In other more general cases, all polygons above the new contour must be removed, and polygons that intersect it must be clipped. This case is not trivial to implement. Integrating programs which share similar databases, such as COMPOSITE and SAMPLE, or SIMPL and CREEP, is conceptually simple but translating among profile representation formats can become tedious and time-consuming.

A standard Profile Interchange Format (PIF) has been proposed [Duv88] but has not yet been fully implemented. PIF is a polygon based data representation which could be used for passing data among different programs. It would still be necessary to translate a PIF profile representation into that used by a particular program, but the difficulties of coordinating different formats would be streamlined with such a standard. The problem of tools integration would be greatly eased by a standard format for describing device cross sections. The basic concepts of data translation would still be necessary, but their implementation would be greatly simplified.

2.3. SIMPL as a User Interface

Another area to be addressed involves the convenient input of data to a simulation program. SIMPL is a menu driven environment which greatly facilitates its use. A user need not be intimately familiar with the specifics of SIMPL in order to get started with it. This menu feature can be extended to other simulators called by SIMPL. Input files for a program such as SAMPLE can be built up automatically through a series of questions presented by SIMPL. The less time someone has to spend learning how to use a program, the more likely that person is to use it. If less time is spent writing input files, more time can be devoted to using the program or doing other work. The advantages of a menu driven user interface are clear.

SIMPL-DIX offers several routines for creating menu options on the display, and for prompting data input from a user. It is a relatively easy task to create new menu options for SIMPL and instructions for doing so are clearly spelled out in the thesis on SIMPL-DIX [Wu88a].

2.4. Displaying Data with SIMPL

It is similarly straight-forward to display data using SIMPL. All of the data structures required by the X window system are already set up in SIMPL-DIX. Routines for locating points in the display viewports are easily used. Basic X window function calls can be used to display numerical data on the screen, highlight certain regions of the display, or draw new information to the screen. The only challenge in displaying data with SIMPL is deciding precisely where on the screen to put the data so that it is most useful and most cleanly represented. Figure 2.3 is an example of displaying data associated with individual polygons in a SIMPL cross section.

2.5. Unix Interface

SIMPL-DIX uses the interprocess communication (IPC) facilities in the Berkeley UNIX 4.3BSD release [Sec85] to provide a connection to SIMPL-2. This method of communication is useful for passing information to an interactive program. It is also possible to communicate with programs through files, by creating an input file for a given program and then using a Unix system call to run that program. The method using IPC is more sophisticated in that it allows multiple process to be run simultaneously. Unix system calls, however, are easier to implement and are adequate in situations where it is not necessary to run multiple processes. SIMPL uses both methods of inter-program communication. IPC is used for the SIMPL-DIX to SIMPL-2 interface since both programs are run simultaneously and short streams of data, such as process commands, are continuously sent between them. Unix system calls are used to run SAMPLE since large streams of data, such as profile contours, must be sent back and forth. Additionally, nothing is gained by using IPC in this case, since SIMPL must wait for the data from SAMPLE before continuing.

3. Interface to RACPLE for Parasitic Evaluation and SAMPLE 1.7 for Nonplanar Etching.

3.1. The RACPLE Program

RACPLE (Resistance and Capacitance of Profiles in Lithography and Etching) is a post processor for the SAMPLE program. It calculates the effective ratio of length vs. depth for a thin film deposited over a nonplanar surface. The effective number of lateral squares (length/depth) can be used to determine the resistance along the film or the capacitance across the layer. For a planar conducting layer, the resistance is given by

$$R = (resistivity)(L/D)(1/W)$$

the capacitance for a planar dielectric is given by

$$C = (permittivity)(L/D)W$$

RACPLE calculates the effective *L/D* for a thin film by locating critical features in the profile and then dividing the film into small sections for which the resistance can be approximated. This approach has an accuracy of better than 5% when compared with a numerical solution of the Laplace equation. Care should be taken though, since RACPLE does not include the effects of fringing fields for determining capacitance. RACPLE should only be used for capacitance simulation if the dielectric is thin compared to its length and width. Resistance calculation is accurate for a wide range of geometries. More information on RACPLE is contained in the original report on the program [Lee83b].

RACPLE is consistent with the basic philosophy of SIMPL in providing rapid evaluation of a device profile with a minimum of computational effort. Despite its limitations, RACPLE is a useful program for topography dependent electrical parameter extraction, and is a useful addition to the SIMPL design environment.

3.2. SIMPL - RACPLE Interface.

SIMPL-DIX is used to generate input files for RACPLE, call RACPLE to calculate parasitics, and then display the results along with the device profile. The interface extracts each polygon in the SIMPL-DIX data base, creates an input file for each polygon, and then runs RACPLE.

RACPLE uses the SAMPLE plotting file format (f77punch7) as its input data representation. The interface takes the polygon as given by SIMPL-DIX and writes it as a top contour and a bottom contour in a SAMPLE format plot file. The polygon as represented in SIMPL-DIX is a linked list of coordinate pairs which traverse the polygon boundary in a clockwise direction. There is, however, no preferred location for the first vertex in the list. The entire polygon must be read in order to determine which points make up the top contour and which make up the bottom contour. Figure 3.1 gives the shape of a typical generic polygon. The left and right edges need not be vertical, but this is often the case for a polygon which touches the edge of the simulation window. The interface traverses the polygon to determine the relative location of the first vertex in the list, as well as the character of the left and right edges. The head vertex is classified as belonging to one of eight types as shown in Figure 3.2, either a corner point 0, 2, 4, or 6 or an intermediate point 1, 3, 5, or 7. From this information, the interface arranges the points into a top and bottom layer which are written to a file in SAMPLE plot file format. Figure 3.3 shows how the example polygon of Figure 3.2 is split up into two layers.

The routines used by RACPLE expect a high density of points to describe a profile contour, even if the profile is a line segment. SIMPL, in order to save space, eliminates redundant points from a polygon, so that a line segment is described only by its end points. To ensure computational accuracy with RACPLE, the interface inserts additional points in the contours until no pair of adjacent points are separated by more than about 1/20th the length of the contour. Failing to do this can result in errors of as much as 75% for typical profiles, since RACPLE cannot split up the profile correctly if the layers lack a sufficient number of points.

As each input file is created, it is run with RACPLE and the results added to a file called, appropriately enough, RACPLE_RESULTS. The interface uses the Unix system command to call RACPLE since this method is convenient to implement and sufficient for the purposes of this interface. Once all the polygons have been evaluated by RACPLE, the results can be displayed. The interface first determines the dimensions of the display viewport using global variables in SIMPL-DIX. Each piece of data generated by RACPLE is associated with a particular polygon which has one point in its upper left corner. SIMPL-DIX displays the RACPLE data along the top and bottom of the profile viewport and draws a line from the data to the corresponding polygon. To avoid a sloppy display with many crossing lines, the data is first sorted by the x-value of the upper left vertex of the polygon associated with that piece of data. For polygons that have upper left corners with the same x-coordinate, an additional sort by the y-coordinate is performed. The code for most of the routines used in the RACPLE interface is listed in Appendix B as file dix_action4.c. The contents of this file are compiled as part of SIMPL-DIX.

RACPLE is called from the CRITIC menu in SIMPL-DIX which is in turn part of the TOOLS option of the main SIMPL-DIX menu, of the version of SIMPL-DIX currently in use in our research group. RACPLE was included as a CRITIC option because it is a program for criticizing and analyzing profiles, and thus fell naturally into that category of programs.

3.3. SIMPL interface to run SAMPLE 1.7 for Nonplanar Etching

SAMPLE release 1.7a includes programs for simulating the etching of nonplanar layers [Lyo88]. To provide an interface to these routines, an additional command was added to SIMPL-2 to create a SAMPLE input file for nonplanar etch simulation. To begin, the top of the profile, where the topography meets air, is traversed by SIMPL-2 and the points describing this contour are saved in an array. The polygon containing the topmost vertex is deleted, unless that polygon has a lowest vertex lower than the lowest vertex of any other polygon that makes up the surface. Figure 3.4 shows some cases of polygons making up surface layers, and which would be deleted. Once a polygon is deleted, the new surface is traversed and saved as the next layer. Once the substrate is reached, all of the layers are written to a SAMPLE input file as parameters for "nonplanar" statements. The database in SIMPL-2 is restored by rebuilding the original topography from data which had been stored in a cross-section data file.

SIMPL-2 then prompts for the etch rates of the materials represented by each layer. Finally, SIMPL-2 requests the etch time for the simulation. Usually, SIMPL requests the etch or deposition distance and the calculates the corresponding time, but for multiple materials with different etch rates, the resulting etch distance is known only after the simulation is run. With the etch rates and geometries of each of the layers, the interface creates a complete input file for running the SAMPLE etch machine. Using a Unix system call, SAMPLE 1.7a is run with the input file and the results are stored in a SAMPLE format plot file. SIMPL-DIX can display the results of the etch simulation in the profile viewport as shown in Figure 3.5. Currently, SIMPL-2 cannot take the result of the etch simulation and update its profile data base, except in one special case: if the etch contour consists of only one material from end to end, all the polygons above it can be removed, and the etch contour used as the surface of a new polygon to replace the polygon currently cut by the etch contour. This is certainly not a general interface, but it can be useful as will be seen in a later example. A general interface will be available in the near future, but is not a part of this project. The routines used for the nonplanar etch interface are listed in Appendix B as F77Layers.c. These routines are compiled as part of SIMPL-2.

4. SIMPL for Analysis of a Patterned Photoresist Planarization Process

4.1. Process Overview

A standard technique for planarization is etch back of a resist-coated dielectric layer [Ada81]. In this technique a dielectric film is deposited over the topography to be planarized. An organic film like photoresist is then spun on in such a way as to planarize the entire surface of the layer. The combination organic film and dielectric are then etched in a plasma environment that has been designed to produce equal etch rates in both materials. The limitations of this process have been documented [Sti87]. The thickness of the deposited film is a function of feature density. As the distance between the features increases, the planarizing effect of the film is lost.

A planarization process using a sacrificial fill layer of patterned photoresist has been proposed which solves planarization problems encountered in both intermetal dielectric for a 1.2 micron 256K SRAM technology and trench isolation for a 0.8 micron 1M SRAM technology [She88]. Photoresist is used to fill valleys in the deposited film. After the dielectric (e.g. boro-silicate glass) is deposited on the topography, a photoresist is spun on with a nominal thickness equal to the step height of the underlying topography. The resist is the patterned in such a way as to remain in areas where conformal coverage of the dielectric is expected to occur. A second layer of resist is now deposited on top. A high degree of planarization exists for the last deposition step since most of the nonplanar regions were filled in by the patterned photoresist. The etch-back proceeds from this point leaving a planar dielectric layer covering the topography.

Misalignment of the mask used to pattern the first layer of photoresist can be expected to counteract the planarizing benefits of this process and degrade device performance. However, as will be shown using process simulation, this masking step has noncritical dimensions and alignment requirements. Adding masking steps adds to the cost of the process, but some of this cost may be saved given the relatively high tolerance of this masking step to misalignment error.

4.2. SIMPL Simulation of the Patterned Photoresist Planarization Process

This patterned photoresist planarization process can be simulated using SIMPL. The substrate SI is chosen to represent some underlying topography. In this case, the topography consists of 1.6/1.4 micron lines and spaces next to a 14 micron region with no lines. The height of the lines is 0.85 microns. Figure 4.1 shows this topography. 1.5 microns of glass (PSG) is deposited using isotropic deposition. 0.75 microns of resist (RST) is deposited vertically on top of the dielectric as shown in Figure 4.2. This resist is patterned with the mask NB leaving a box of resist as shown in Figure 4.3. 1.0 microns of a second resist is deposited as shown in Figure 4.4. After using SAMPLE to simulate the etch back, the resulting profile is as shown in Figure 4.5. To complete the process, a metal layer is deposited on top of the dielectric as shown in Figure 4.6. The complete SIMPL-2 process file for this simulation is listed in Appendix A as process.rbx1

This process is sensitive to misalignment of the mask used to pattern the resist (NB). The WORST feature of SIMPL-DIX is used to shift the mask NB to the right. The result of shifting the mask NB to the right by 0.6, 0.9, 1.2, 1.8 microns and then simulating the process is shown in Figures 4.7, 4.8, 4.9, and 4.10 respectively. Also, if the mask NB is originally too small due to excessive process bias, the result is shown in Figure 4.11. At a certain point, the effect of mask misalignment is to greatly diminish the planarizing advantage of the process. After 0.7 microns misalignment, the problems are severe.

4.3. Results - RACPLE Analysis

To get a more quantitative measurement of the effect of mask misalignment in this process, RACPLE can be used to measure the number of lateral squares in the dielectric layer. Figure 4.12 is a plot of lateral squares of PSG versus misalignment. For no misalignment and misalignment of 0.6 microns, RACPLE measures about 8.2 lateral squares for the PSG layer. At 0.7 microns misalignment, the RACPLE measurement jumps to 12.4 and continues to climb to 13.9 for a misalignment of 0.9 microns. For 1.1 microns misalignment RACPLE measures 12.28 squares, and for more than 1.2 microns misalignment, the measurement settles at 11.33 lateral squares and remains there for misalignment of up to 2.4

microns. Similarly, for the profile of Figure 4.11, in which the mask NB is biased too small, RACPLE calculates 11.83 lateral squares for the dielectric layer.

It is apparent that the misalignment or mask size error to create a noticeable degradation of device performance is on the order of 50% of the minimum linewidth for this example. This supports the claim that the masking step has noncritical dimensions. The degree to which the masking step tolerates error has been shown here with SIMPL. One interesting feature of Figure 4.12 is the sudden jump in the number of lateral squares at 0.7 microns misalignment which settles at around 1.2 microns misalignment. One cause of this effect is the fact that at above 1.2 microns misalignment, the thinning of the dielectric on the left side of the structure is compensated by an excess of dielectric on the right side. At 0.7 to 0.9 microns misalignment, there is a serious thinning on the left side, but no noticeable thickening on the right. The capacitance across the dielectric on the left side is actually quite high for all cases of misalignment greater than 50% of the minimum linewidth, but is compensated on the right side if the mask is misaligned enough.

Using SIMPL, it has been shown that mask misalignment tolerance is good up to about 0.6 microns for this case, but results in serious performance degradation for further misalignment. The effects of altering layer thicknesses, or changing feature dimensions and spacings can be expected to affect the misalignment tolerance. These effects can be investigated with SIMPL.

4.4. Comments on Approach

This approach to analyzing the above planarization process can identify certain trends using elementary models. It would be interesting to include additional process effects and more comprehensive analysis of the exact electrical nature of the device. Microloading effects which would cause a variation in etch rate across an individual die are not accounted for by the etch simulation models. As a result, certain topography effects which might be problematic in real devices are not seen in this simulation. Additionally, RACPLE gives only an estimate of the electrical properties of the profile. A complete analysis of the multiple parasitic capacitances and even inductances would be of interest.

5. Application to VLSI Hopfield Neural Networks

5.1. SIMPL for Neural Network Analysis

It was originally proposed that the linking of SIMPL process simulation with RACPLE analysis could be used to investigate parasitic resistive and capacitative loading effects in highly interconnected computational structures such as neural networks. SIMPL and RACPLE have been applied to an analysis of parasitic loading in a structure similar to that developed at AT&T Bell Laboratories [Jac86]. It will be shown that layout and topographical features in processing do have an impact on neural network performance.

5.2. Overview of a Reported VLSI Neural Network

The properties of highly interconnected arrays of amplifiers have generated much interest for their potential use in a new class of computing circuits. The properties of such networks have much in common with biological information processing systems (brains) in that they are massively parallel and fault tolerant. The basic network configuration is shown at the top of Figure 5.1. Several amplifiers are connected such that each amplifier output is available as input to any of the other amplifiers. The actual feedback connection is made with a resistor, and the pattern of resistors in the network determines the behavior of the entire circuit.

The basic operation of the circuit can be described as analogous to the motion of a particle through a potential energy field in a multidimensional space. This analogy holds if the matrix of interconnect resistances is symmetric. The output voltages V_i of each of the N amplifiers are independent coordinates in space. The amplifier gain characteristics are assumed to be symmetric around $V=0$. For this case, the energy function is

$$E=\sum_{i<j} T_{ij} V_i V_j + \frac{1}{r_i} \int g^{-1}(V) dV$$

where $V=g(u)$ is the transfer function of the amplifiers, $T_{ij}=T_{ji}=1/R_{ij}$ are the coupling resistors between the amplifiers, $1/R_i=\sum_j 1/R_{ij}$ and $T_{ii}=0$ [Hop84]. If the system is put into any particular state by applying voltages at the inputs, the energy function gradient will cause the circuit to relax to a stable state which is close to the initial state. This type of circuit can be used as a content-addressable memory [How87]. The search operation is done in a fully parallel way and the time to reach a solution is determined by the speed of the amplifiers and the time constant of the resistor network.

At the bottom of Figure 5.1 is a circuit diagram showing the basic make-up of a single neuron. The resistances and capacitances, in addition to the resistive weight, are due to parasitics in the VLSI implementation of the network. To study these effects, a single interconnect element was fabricated with SIMPL.

5.3. SIMPL Simulation of a Neural Network Element

Using electron-beam lithography, a 12x12 resistor matrix that fits into a 6x6 micron square was fabricated by a group at AT&T Bell Laboratories [Jac86]. To generate a device cross section using SIMPL, the basic geometry of the AT&T device was used. A 0.5 micron thick oxide layer was grown on the substrate. 0.1 microns of tungsten was deposited and patterned into 0.3 micron lines and spaces. 0.1 microns of a polyimide was deposited as an interplanar dielectric (using SIMPL anisotropic deposition with a 20 degree source angle. A resistor hole was created and filled with polysilicon. Finally, a 0.1 micron layer of nickel was deposited (using SIMPL vertical deposition). The resulting cross section is shown in Figure 5.2. The cut line was positioned such that the profile generated represents one period (0.5 microns) of the tungsten line. A second simulation for a cross-over point with no contact is shown in Figure 5.3. The SIMPL process file for Figure 5.3 is listed in Appendix A.

5.4. Parasitic Analysis

For a planar nickel layer which is 0.5 microns long, 0.1 microns thick and 0.2 microns wide, The number of lateral squares is 5 and the overall resistance is

$$R=7.85\Omega\text{–cm}\times 5\times\frac{1}{0.2\mu\text{m}}$$

The inter-metal capacitance is on the order of

$$C=3.9\times 8.85x\,10^{-14}\frac{\text{F}}{\text{cm}}\times 3\times 0.2\mu\text{m}$$

Using RACPLE to measure the resistance of the nickel electrode, the number of lateral squares is 7.085

instead of 5, giving an increase of 40%. Measuring the inter metal capacitance by running RACPLE over the width of the tungsten electrode only, gives 4.7 lateral squares instead of 3, an increase of more than 56%. Additional capacitances from the electrode to the substrate are negligible in comparison with the inter-metal capacitance. Using these values in the above equations, the total RC of the upper electrode is about 1.0 picoseconds. If there are a thousand such tungsten lines, the total RC delay of the nickel line is about 1 microsecond. If we assume an amplifier delay on the order of a microsecond, clearly the interconnect delay time is an important mechanism. Topography related effects can have a serious impact on neural network circuit performance, since the total distributed RC delay constant of the counter electrode increases as the square of the number of neurons. With planarization techniques, the interconnect delay can be reduced to improve circuit performance.

5.5. Comments on this Approach

SIMPL and RACPLE are in place for studying topography and process dependent effects in novel circuits such as neural networks. As demonstrated above, basic trends in the characteristics of novel devices can be investigated with integrated process and device simulation. It is worth pointing out that industry is not yet developing VLSI neural networks which use processes much different from those used for conventional CMOS chips. The interest in neural networks now is in getting the chips to work and finding applications for them [Hec88]. Still, many new architectures and designs for VLSI neural networks are being proposed, and tools such as SIMPL and RACPLE can be used to provide an initial assessment of some of the electrical issues involved.

It would be interesting to perform a full three-dimensional analysis of the interconnect structure, including ones with resistive weights. Process simulation should generate the device geometry in three dimensions to provide the right link between process flow and device analysis. The nature of the circuit poses some interesting challenges as well. It has been proposed that the stability properties of Hopfield neural networks can be related to the properties of individual neurons [Mic87]. This synthesis of system theory and device technology should also be brought to the analysis. Rigorous software for process and device simulation will be needed for problems of dense interconnect networks in general, and has a ready application in neural networks.

6. Future Directions for SIMPL

6.1. Profile Data Management

In the area of profile data management, two questions must be addressed. First, what data should be maintained? Second, how should this data be transferred to places where it is needed? For SIMPL, the first question can be answered this way: SIMPL should maintain a description of a device profile which completely describes the device geometry, and materials involved. The profile data description as it now exists handles most of this information. However, some additions are needed to completely describe a profile. Currently, SIMPL only describes the net charge of dopant contained at a particular location in the substrate. This should be changed so that the species of dopant is included. Such information is needed for diffusion simulation with programs like SUPREM. SIMPL should also maintain information about impurities in materials other than the silicon substrate. No information is currently maintained about impurities present in a gate oxide, for example. Another addition to the profile description which is still needed is the ability to describe a floating island of material which is completely surrounded by another material. There are ways around this, usually involving dividing up the surrounding material into two parts. Ideally however, the SIMPL data base would handle a floating island of material as a normal case.

The second question involves how data is transferred. Currently, this is one of the major bottlenecks which noticeably slows down the performance of SIMPL. SIMPL-DIX and SIMPL-2 use files to transfer profile information back and forth. The time spent writing files can be significant when large pieces of data must be transferred. It is easy to write programs that communicate through files, but it is not very efficient from a user's point of view. In this day of powerful computer work-stations, with several megabytes of RAM storage available, SIMPL should avoid communication with files. Instead, data should be stored in memory common to both SIMPL-DIX and SIMPL-2.

Likewise, in the area of communication with other programs, it is desired to get around the need to use files. This is not an easy problem to solve yet, especially for programs written in different programming languages. There is some promise, however, that a PIF data base will one day become available. Eventually it will be possible to send a program a pointer to the root of a data tree instead of a file full of data. Without a PIF parser this is not yet a possible alternative, but it should be pursued.

6.2. Tools Integration with SIMPL

There is already strong interest in industry for an integrated design environment for process and device simulation. SIMPL has the potential for filling this gap, but work remains to be done. SIMPL-DIX is a suitable framework for creating menu driven interfaces to various programs, but it is not always an easy environment in which to do development work. For a programmer wishing to integrate a new simulator into SIMPL-DIX, it would be useful to have some sort of library of standardized routines from which to build the application interface. Many routines which fit that description already exist in SIMPL-2 and SIMPL-DIX but they are not organized in an efficient manner. Many must also be rewritten in order to be useful for general applications. As it stands now, it often takes programmers several months to understand the intricacies of SIMPL. Instead of tracking down routines which already exist, it is often easier to write new ones. This only adds to the size and complexity of SIMPL. One might say that SIMPL is becoming so convoluted that the name should be changed to HARD. SIMPL, if fully integrated with programs like SAMPLE, SUPREM and PISCES, would become a powerful design environment. With a standardized approach to integrating new simulation software into its framework, SIMPL would remain at the leading edge in the face of rapidly changing technology. Admittedly, this would be a major undertaking, but industry would be very interested in a design environment which could easily be connected to proprietary device and process simulation software.

Listed in Appendix C are several of the C functions in SIMPL-2 which are useful for tools integration and profile manipulation. This list includes C functions available in the most recent release version of SIMPL-2 as well as new ones written as a part of the ongoing development of SIMPL. A few routines in SIMPL-DIX which were used in this project are also listed. The routines are organized according to their use in tools integration and profile manipulation. It is not an exhaustive list of the functions available in SIMPL but is intended as a first step in organizing the routines in a useful fashion.

Another area to be considered is the possibility of integrating SIMPL with the OCT/VEM environment [Har86]. Many of the software tools needed for integrating SIMPL with other simulators exists in OCT/VEM. Additionally, the issue of upgrading from X windows version 10 to version 11 would automatically be solved if SIMPL is included in OCT/VEM. Once those programs switch to X11, so will SIMPL. The trade-offs and advantages of integrating SIMPL with OCT/VEM should be further explored. Some combination of these options, where SIMPL can be used as a stand-alone environment and where it is also available in OCT/VEM may be the most flexible and useful approach.

A standard PIF format is also a necessity in order to pursue the goal of tools integration. It is possible to develop custom interfaces for each simulation program that becomes available, but the time involved is often great. If every process and device simulation program communicated with PIF, this aspect of tools integration would be trivial.

6.3. Problems in the Implementation of SIMPL

Recent releases of SIMPL suffer from some implementation problems which hamper the effectiveness of the program. The biggest problems involve the internal etch simulation models. Many standard cases are not handled correctly, and the resulting profiles are incorrect. There are also cases where SIMPL-2 incorrectly searches the database, which instead of merely giving a false profile, cause the program to crash unexpectedly. The implantation routines often suffer from overflow problems. The grid which SIMPL-2 uses to store doping concentration information is sometimes allocated incorrectly. The routines are also prone towards developing infinite loops which freeze the program. SIMPL-DIX has some minor problems, generally resulting when some global variable is not reset properly. Occasionally, attempts to re-initialize SIMPL-2 from SIMPL-DIX are not successful.

This lack of robustness takes away from the usefulness and credibility of the program. When a perfectly normal process cannot be simulated by SIMPL, most potential users simply give up. These problems are not due to major shortcomings of the ideas behind SIMPL, but can be traced to problems of implementation and inconsistencies in the software. At some point, these implementation problems will need to be thoroughly investigated and cleaned up.

6.4. Conclusion

SIMPL is a useful tool for studying the complex interrelationship of physical phenomena that go into modern integrated circuit process design. This project has demonstrated how some basic tools can be integrated into the framework of SIMPL, shown how SIMPL can be used to analyze proposed processes, and listed some of the work that must be done to keep SIMPL at the cutting edge of CAD technology for process and device simulation. It is hoped that this report is a useful contribution to this increasingly important field in integrated circuit design and manufacture.

References

[Nag75] L.W. Nagel, *SPICE2 - A Computer Program to Simulate Semiconductor Circuits*, ERL Memo No. UCB/ERL-M250, U.C. Berkeley, May 1985.

[Sec85] C. Sechen, A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package, "*Journal of Solid-State Circuits*, April 1985.

[Pin84] M.R. Pinto, C.R. Rafferty, and R.W. Dutton, *PISCES II: Poisson and Continuity Equation Solver*, Integrated Circuits Lab, Stanford University, Sept. 1984.

[Sel80] S. Selberherr, A. Schutz, and W. Potzl, "MINIMOS - A two-dimensional MOS transistor analyzer, "*IEEE Transactions on Electron Devices*, vol. ED-27, p. 1540, Aug. 1980.

[Old79] W.G. Oldham, A.R. Neureuther, C.Sung, J.L. Reynolds, and S.N. Nandgaonaker, "A general simulator for VLSI lithography, and etching processes: Part-II - application to deposition and etching," *IEEE Transactions on Electron Devices*, vol ED-27, no. 8, pp. 1455-1459, Aug. 1980.

[Ho83a] C.P Ho, J.D. Plummer, S.E. Hansen, R.W. Dutton, "VLSI Process Modeling - SUPREM III, "*IEEE Transactions on Electron Devices*, vol ED-30, pp. 1438-1453, Nov. 1983.

[Ho83b] C.P Ho, S.E.Hansen, *SUPREM-III - A program for Integrated Circuit Process Modeling and Simulation*, Technical Report SEL 83-001, Integrated Circuits Lab, Stanford University, 1983.

[Add85] G. Addiego, T.E. Berger, J.L Reynolds, *SAMPLE User Guide Version 1.6a*, Electronics Research Laboratory, U.C. Berkeley, Feb. 1985.

[SAM88] *SAMPLE User Guide Version 1.7a*, U.C. Berkeley, not yet released.

[Bel88] W.R. Bell II, P.D. Flanner III, C. Zee, N. Tam, A.R. Neureuther, "Determination of quantitative resist models from experiment," *SPIE Advances in Resist Technology and Processing V(1988)*, pp. 382-389, Feb.-Mar. 1988.

[Raf86] C. Rafferty, R.W. Dutton, "SUPREM-IV," *SRC-Berkeley-Stanford Technology Transfer Course*, July 8, 1986.

[Sut87] P. Sutardja, *Finite Element Methods for Process Simulation Application to Silicon Oxidation*, ERL Memo no. UCB/ERL-M88/26, U.C. Berkeley, May 1988.

[Lor85] J. Lorenz, *et al.*, "COMPOSITE - Complete Process Modeling of Silicon Technology," *IEEE Transactions on Electron Devices*, vol. ED-32, pp. 1977-1986, Oct. 1985.

[Toh88] K.K.H. Toh, *Two-Dimensional Images with Effects of Lens Aberrations in Optical Lithography*, ERL Memo No. UCB/ERL-M88/30, U.C. Berkeley, May 1988.

[Gri83] M.A. Grimm, K. Lee, A.R. Neureuther, "SIMPL-1 (SIMulated Profiles from the Layout - version 1," *IEDM Technical Digest*, pp. 255-258, Dec. 1983.

[Lee85] K. Lee, A.R. Neurether, *Symposium on VLSI Technology, Digest of Technical Papers*, pp. 64-65, May 1985.

[Wu88a] H. Wu, *SIMPL-DIX (SIMulated Profiles from the Layout - Design Interface in X)*, ERL Memo No. UCB/ERL-M88/13, U.C. Berkeley, Jan. 1988.

[Sch86] R.W. Scheifler, J. Gettys, "The X Window System," *ACM Transactions on Graphics*," vol. 5, pp. 79-109, April 1986.

[Wu88b] H. Wu, A.S. Wong, Y.L Koh, E.W. Scheckler, A.R. Neureuther, "Simulated Profiles from the Layout - Design Interface in X (SIMPL-DIX)," *IEDM Technical Digest*, Dec. 1988.

[Lee83] K. Lee, Y. Sakai, A.R. Neureuther, "Topography-Dependent Electrical Parameter Simulation for VLSI Design," *IEEE Transactions on Electron Devices*, vol. ED-30, pp. 1469-1474, Nov. 1983.

[Gib86] L. Gibson, *Applications of SIMPL*, ERL Memo No. UCB/ERL-M86/56, U.C. Berkeley, June 1986.

[Duv88] S.G. Duvall, *IEEE Transactions on Computer-Aided Design*, vol. CAD-7, pp. 741-754, July 1988.

[Sec85] S. Sechrest, *An Introductory 4.3BSD Interprocess Communication Tutorial*, Computer Science research Group, Department of Electrical Engineering and Computer Science, U.C. Berkeley, July 1985.

[Lee83b] K. Lee, *Topography-Dependent Step Coverage Resistance Simulation*, SAMPLE Report No. SAMD-7, U.C. Berkeley, March, 1983.

[Lyo88] D.E. Lyons, S.F. Meier, L. Winemberg, A.R. Neureuther, W.G. Oldham, "Simulation of Back of the Line Processes with SAMPLE," *KTI Microelectronics Seminar*, pp.261-281, Nov. 1988.

[She88] D.J. Sheldon, C.W. Gruenschlager, L. Kammerdiner, N.B. Henis, P. Kelleher, J.D. Hayden, "Application of a Two-Layer Planarization Process to VLSI Intermetal Dielectric and Trench Isolation Processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 1, pp. 140-146, Nov. 1988.

[Ada81] A.C. Adams, C.D. Capio, "Planarization of phosphorous-doped silicon dioxide," *Journal of the Electrochemical Society, vol. 128, pp. 423-429, 1981.*

[Sti87] L.C. Stillwagon let al., "Planarization of substrate topography by spin coating," *Journal of the Electrochemical Society*, vol. 134, pp. 2030-2037, 1987.

[Jac86] L.D. Jackel, R.E. Howard, H.P. Graf, B. Straughn, J.S. Denker, "Artificial Neural Networks for Computing," *Journal of Vacuum Science Technology*, vol B4, p. 61, 1986.

[Hop84] J.J. Hopfield, "Neurons with graded responses have collective properties like those of two state neurons," *Proc. Nat. Acad. Sci. USA*, vol 81, p. 3088, 1984.

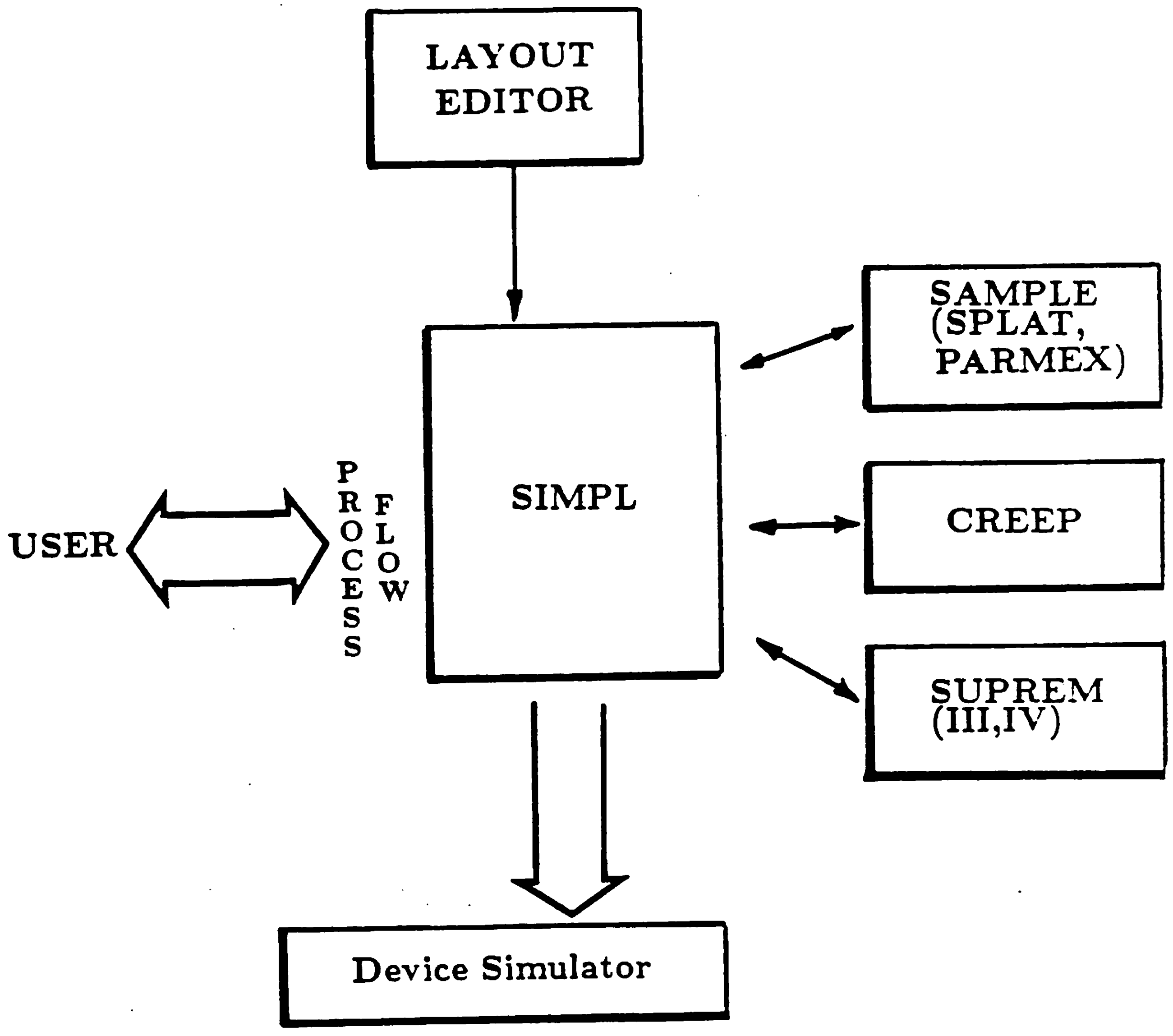
[How87] R.E. Howard, D.B. Schwartz, J.S. Denker, R.W. Epworth, H.P. Graf, W.E. Hubbard, L.D. Jackel, B.L. Straughn, D.M. Tennant, "An Associative Memory based on an Electronic Neural Network Architecture," *IEEE Transactions on Electron Devices*, vol ED-34, pp. 1553-1555, July 1987.

[Mic87] A.N. Michel, J.A. Farrell, W. Porod, "Stability Results for Neural Networks," *Neural Information Processing*, ed. Dana Anderson, American Institute of Physics, New York, 1988.

[Hec88] R. Hecht-Nielsen, Private Communication, 1988.

[Har86] D.S. Harrison, P. Moore, R.L. Spickelmeier, A.R. Newton, "Data management and graphics editing in the Berkeley design environment," *The Proceedings of ICCAD*, pp. 20-24, Nov. 1986.

Figure 1.1. SIMPL as an all-purpose design interface



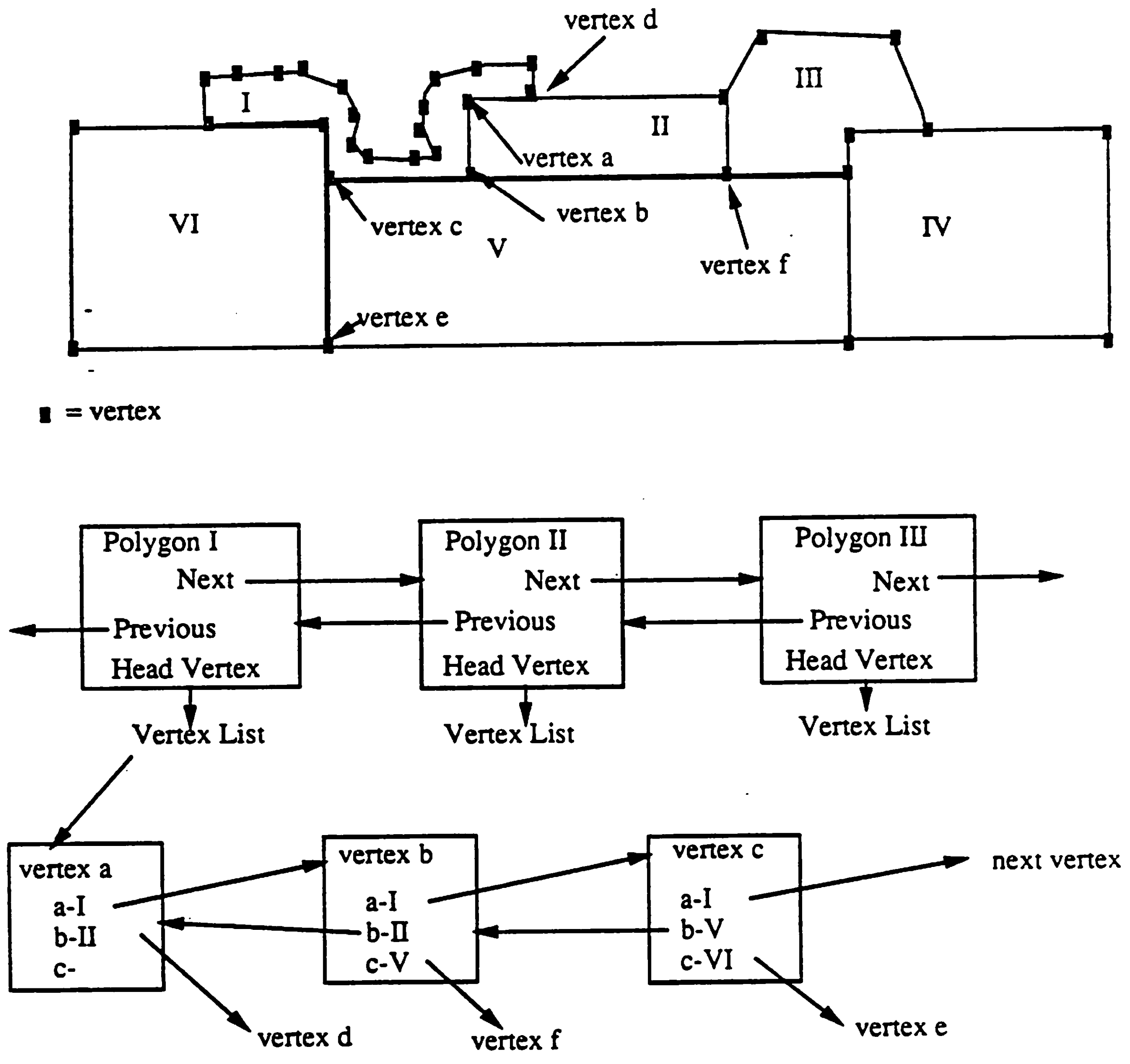


Figure 2.1. Linked polygon data structure

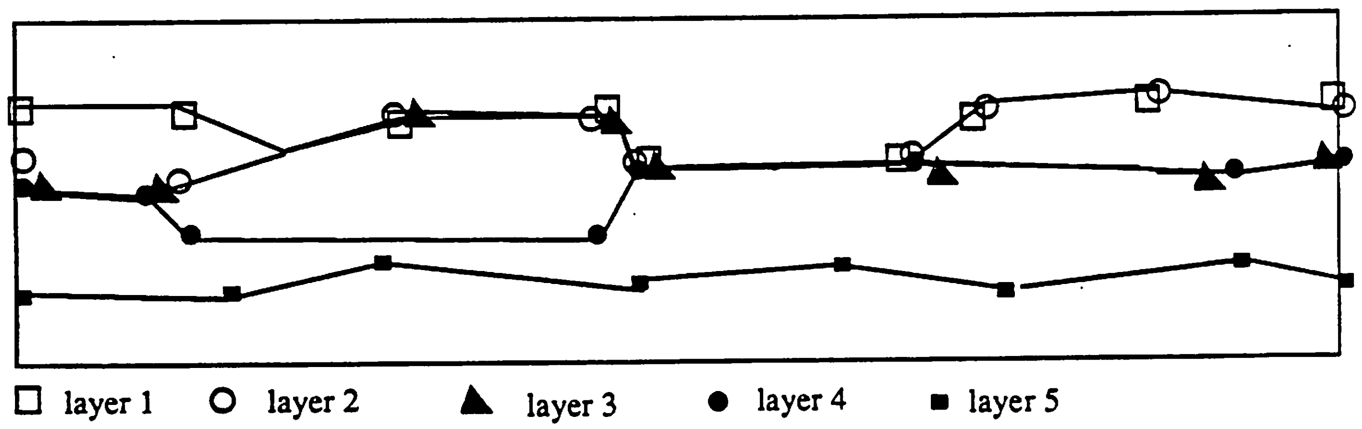


Figure 2.2. Polygons represented as layers

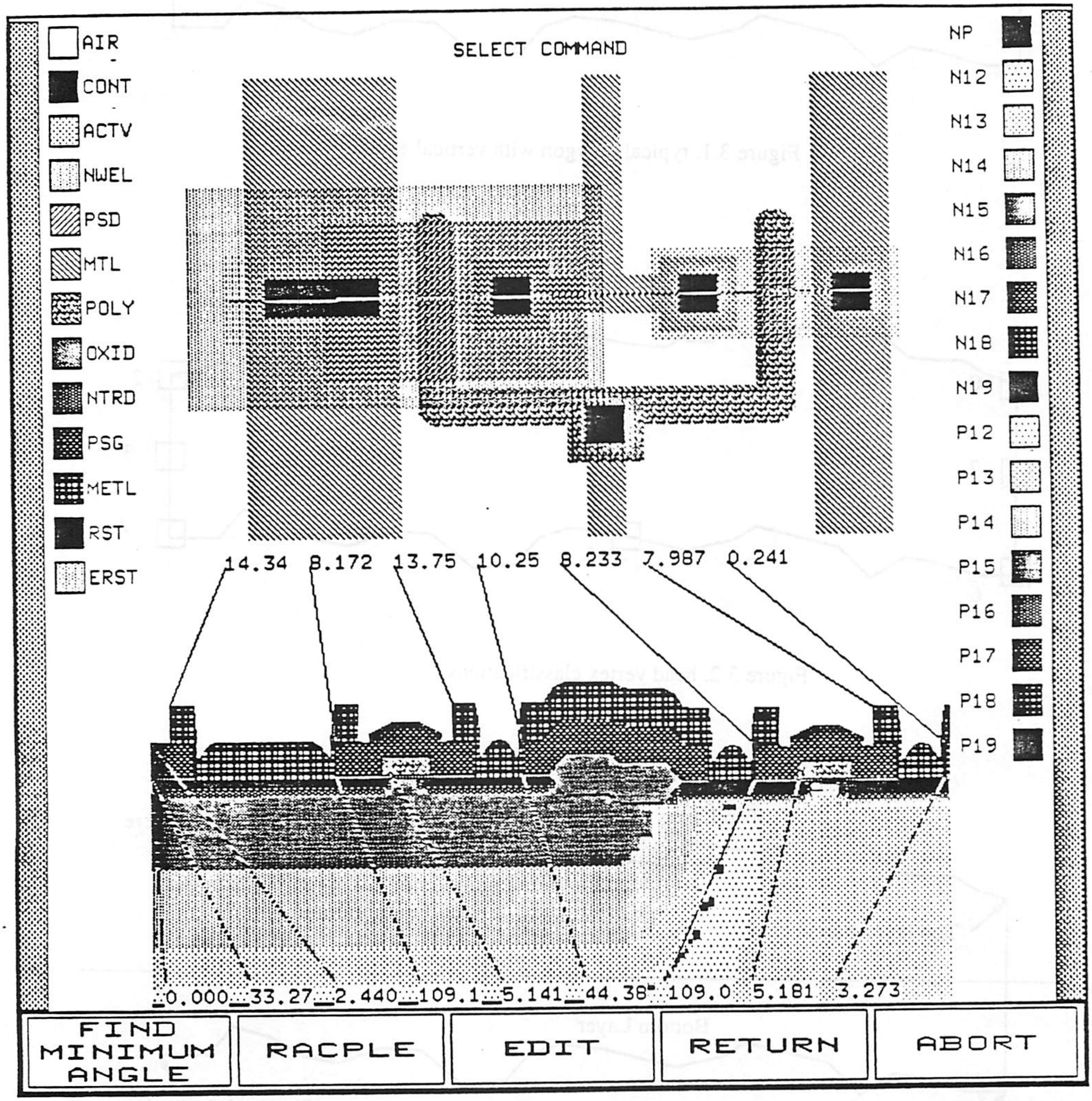


Figure 2.3 Example of displaying RACPLE data with SIMPL

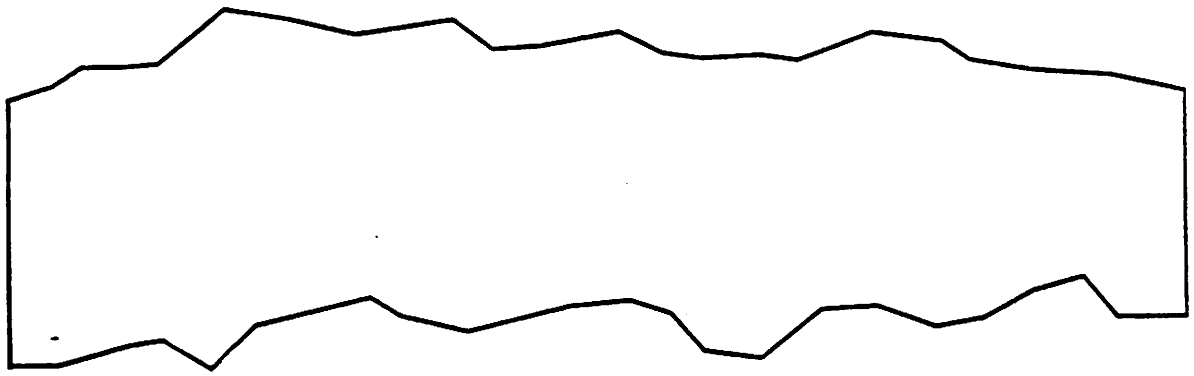


Figure 3.1. typical polygon with vertical edges.

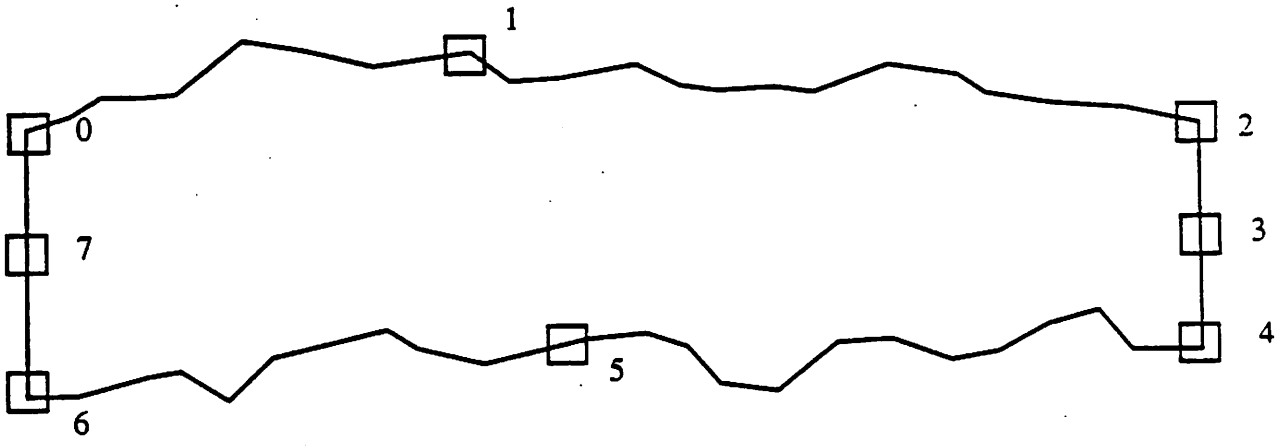


Figure 3.2. head vertex classifications.

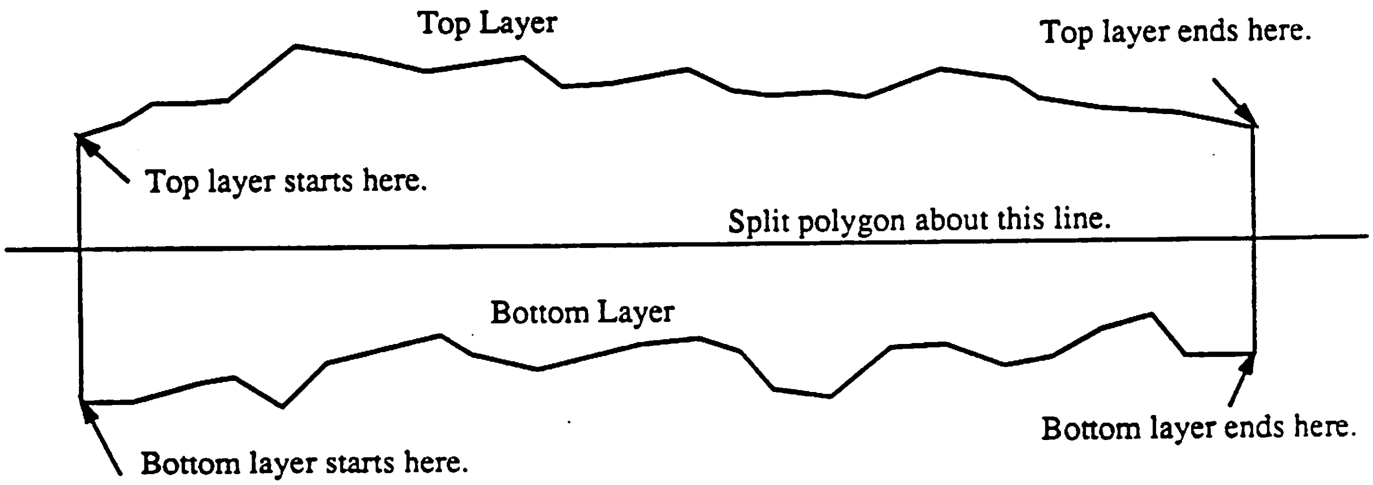
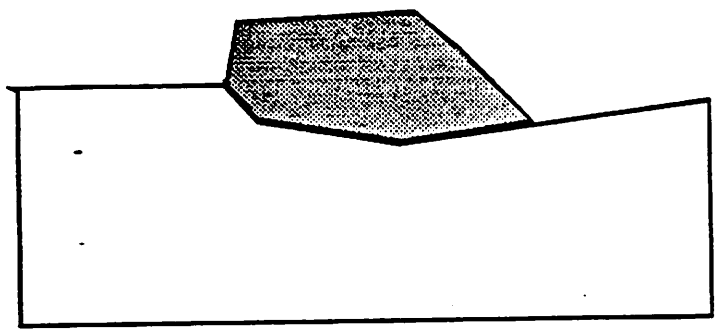
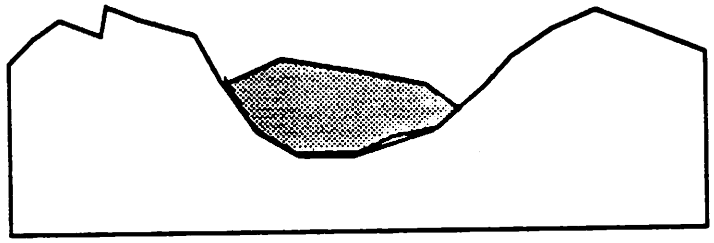


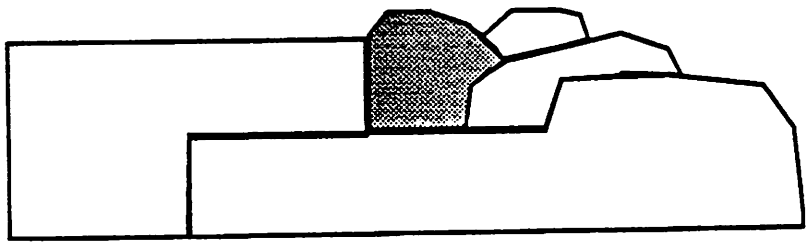
Figure 3.3. polygon split into two layers.



Case 1. One polygon with top vertex equal to highest point on surface.



Case2. Polygon nested in a valley created by another polygon.



Case 3. Multiple polygons with vertexes equal to highest point in surface.

▨ Polygon identified as top polygon.

Figure 3.4. Examples identifying which surface polygon would be deleted in the routines to write SIMPL-2 profiles as a series of layers.

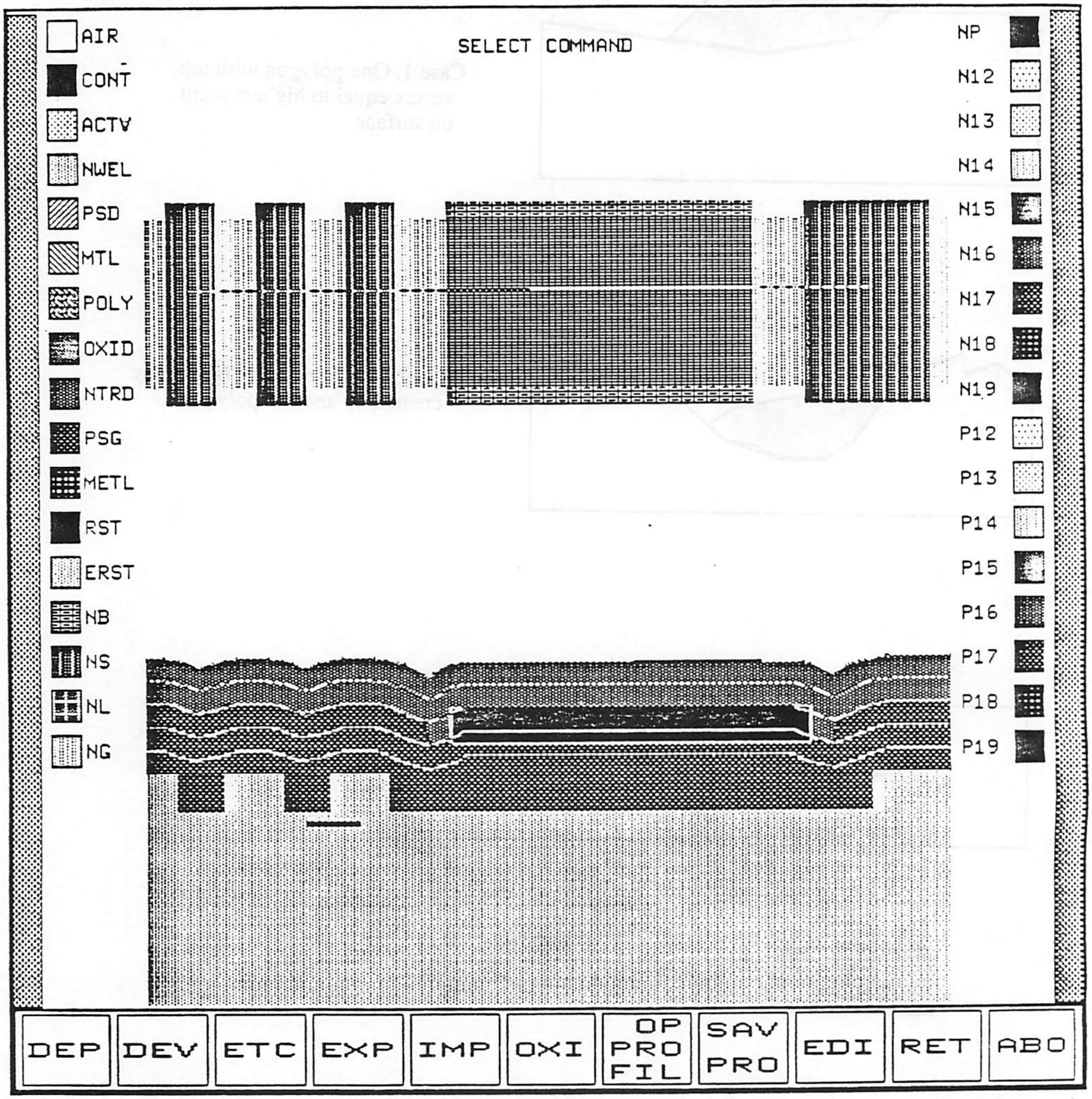


Figure 3.5 Example of displaying SAMPLE non-planar etch with SIMPL

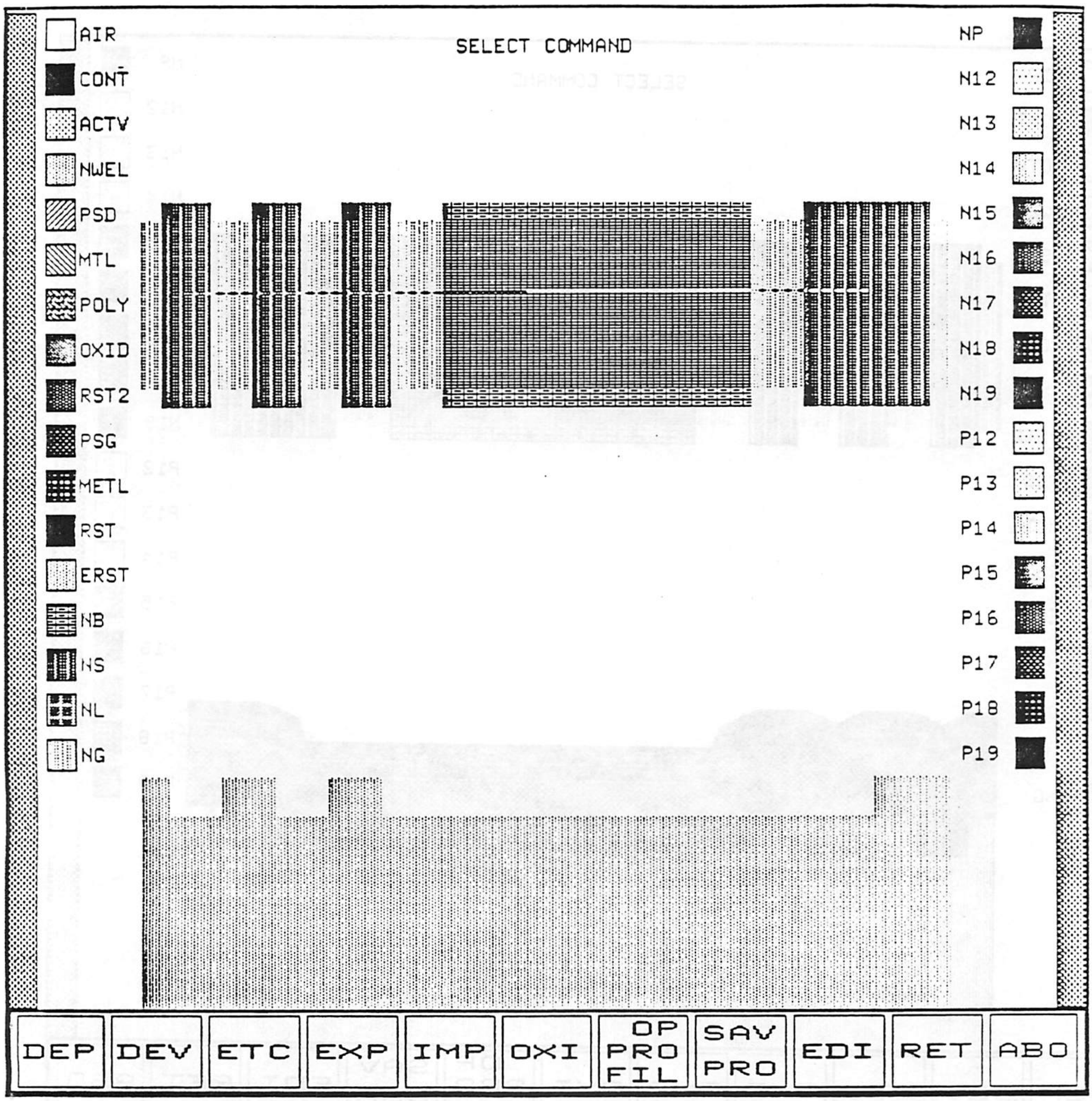


Figure 4.1 Underlying topography, 1.6/1.4 micron lines and spaces

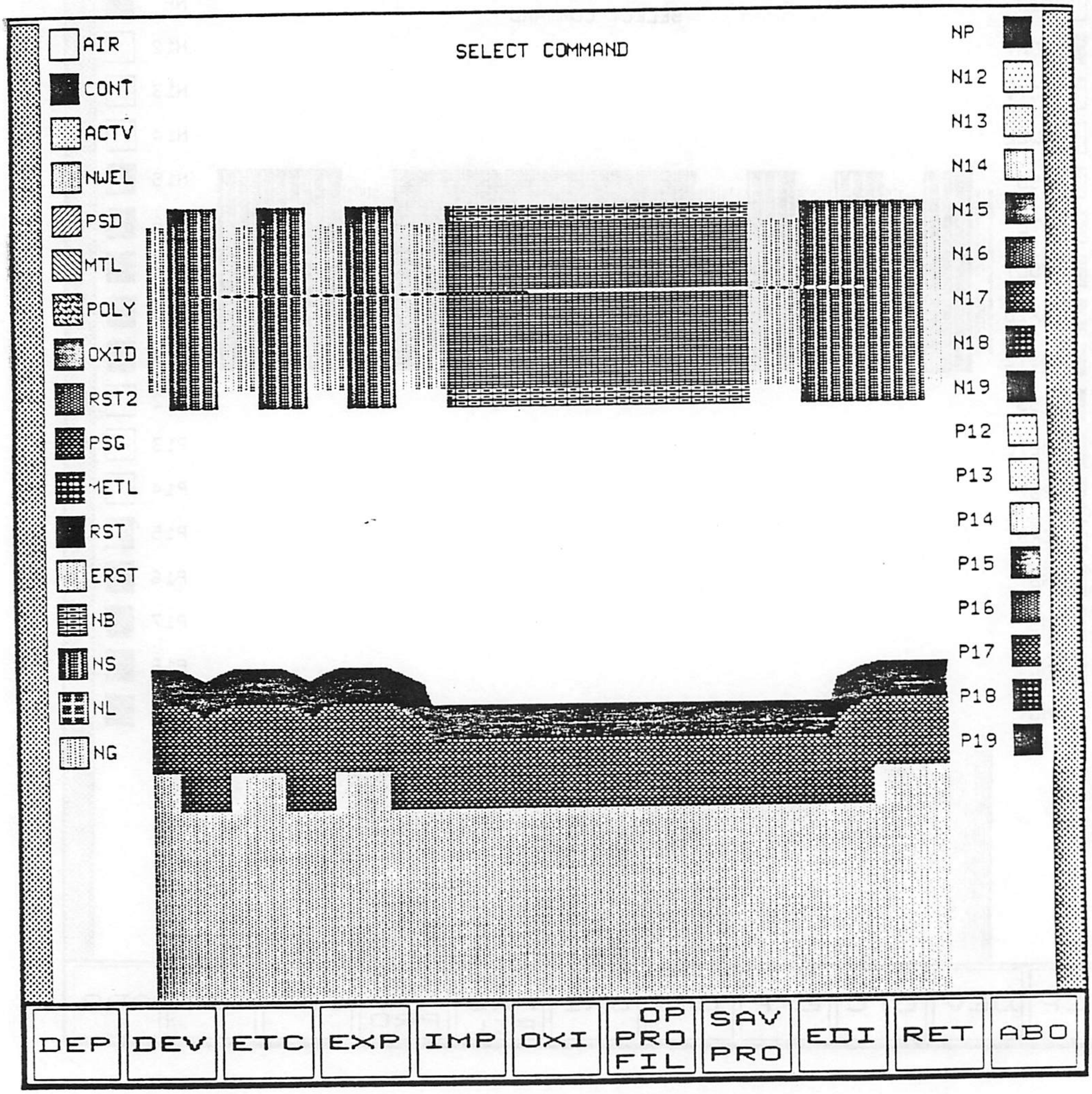


Figure 4.2 After depositing dielectric and first photoresist. . .

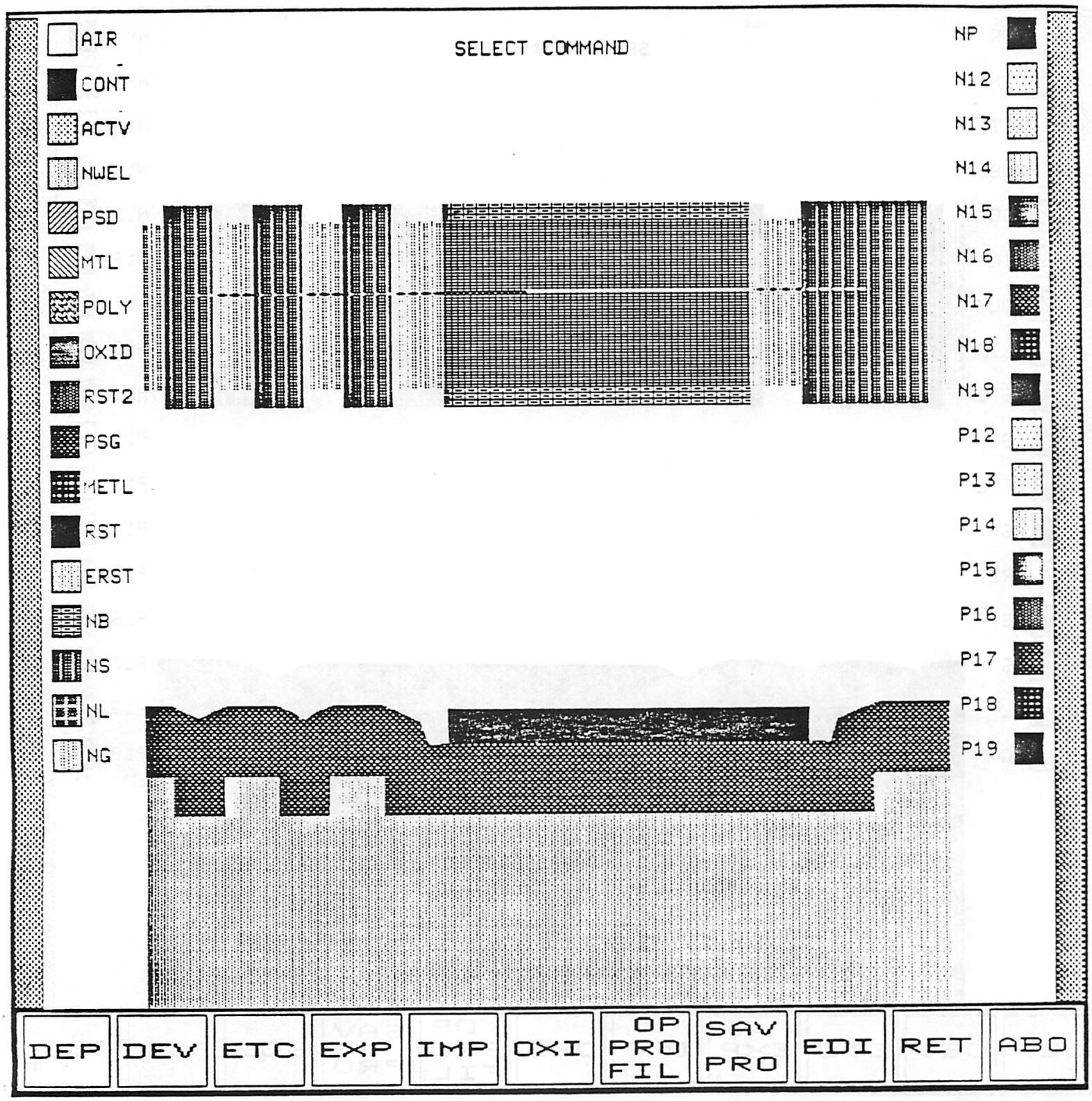


Figure 4.3 After patterning photoresist. . .

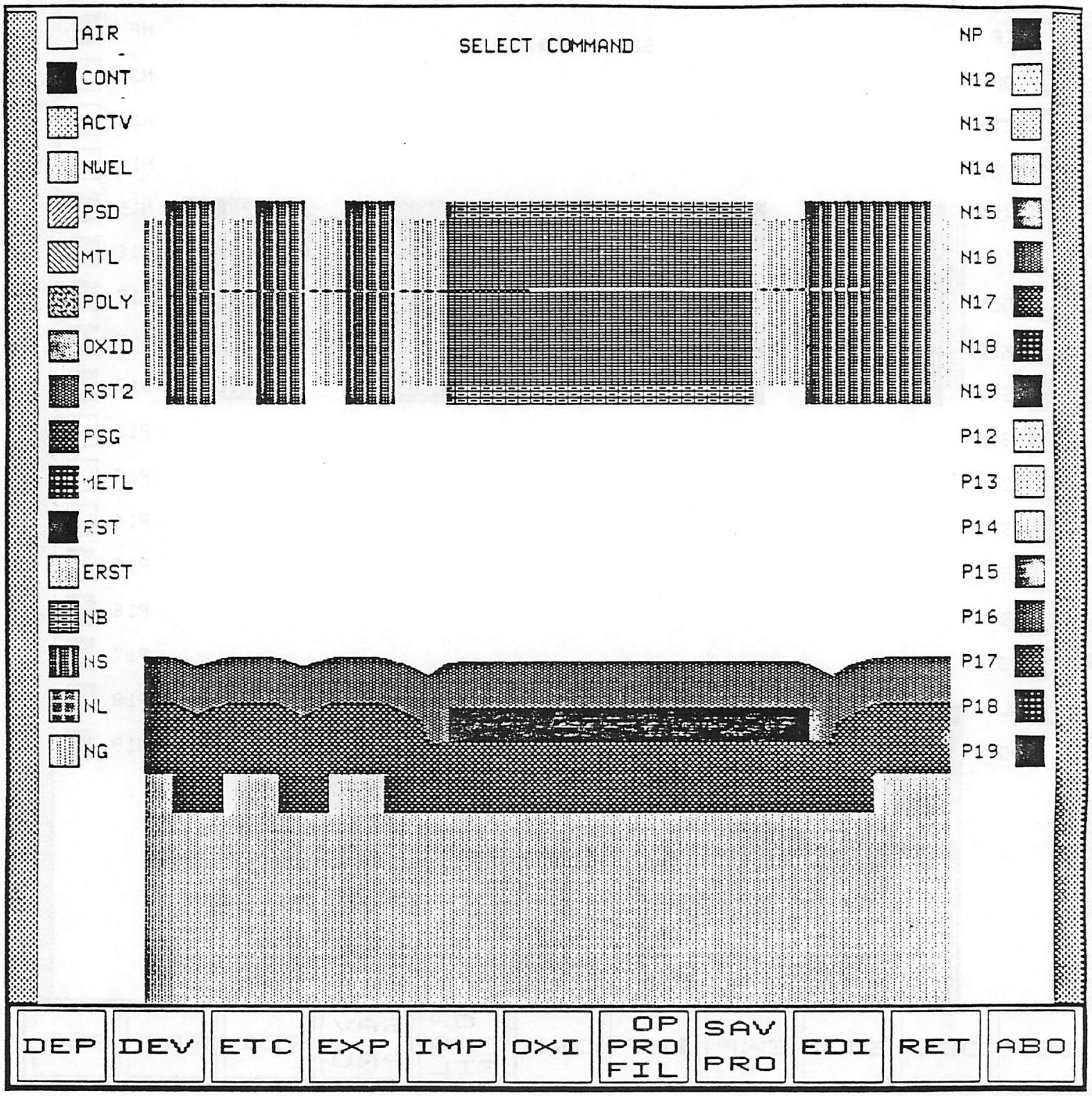


Figure 4.4 After depositing second layer of photoresist. . .

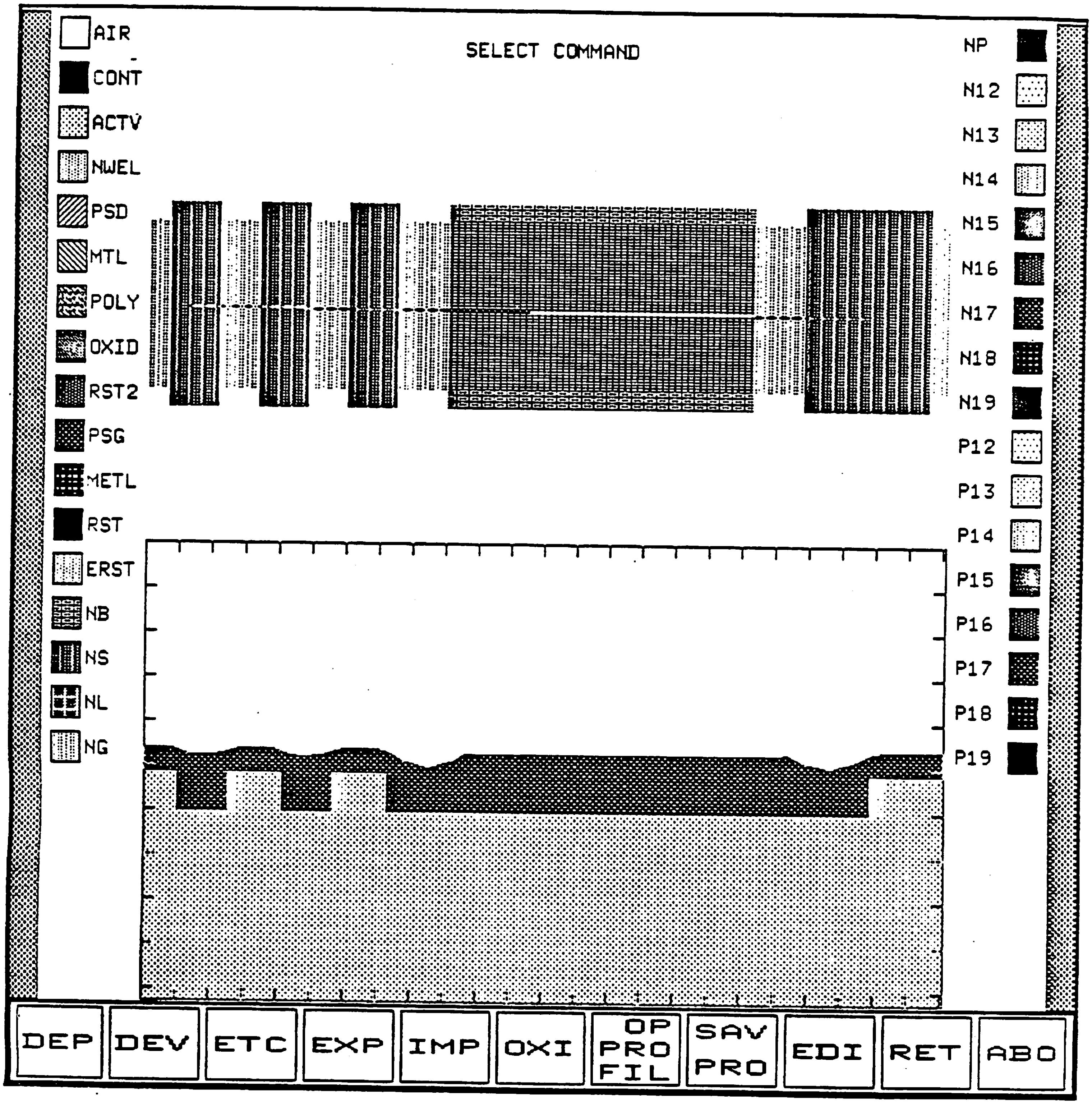


Figure 4.5 Topography remaining after etch-back

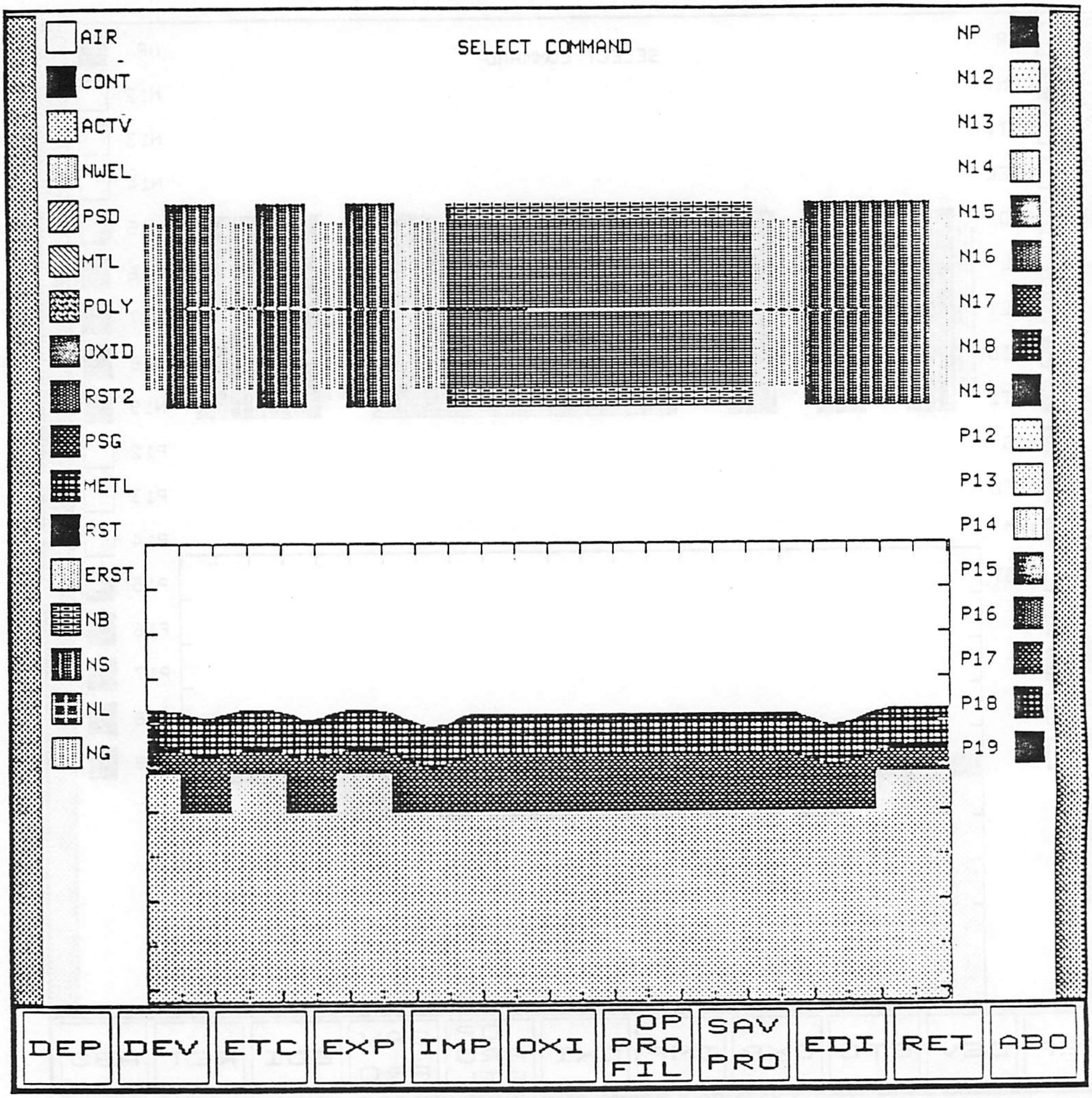


Figure 4.6 Metallization step gives final topography.

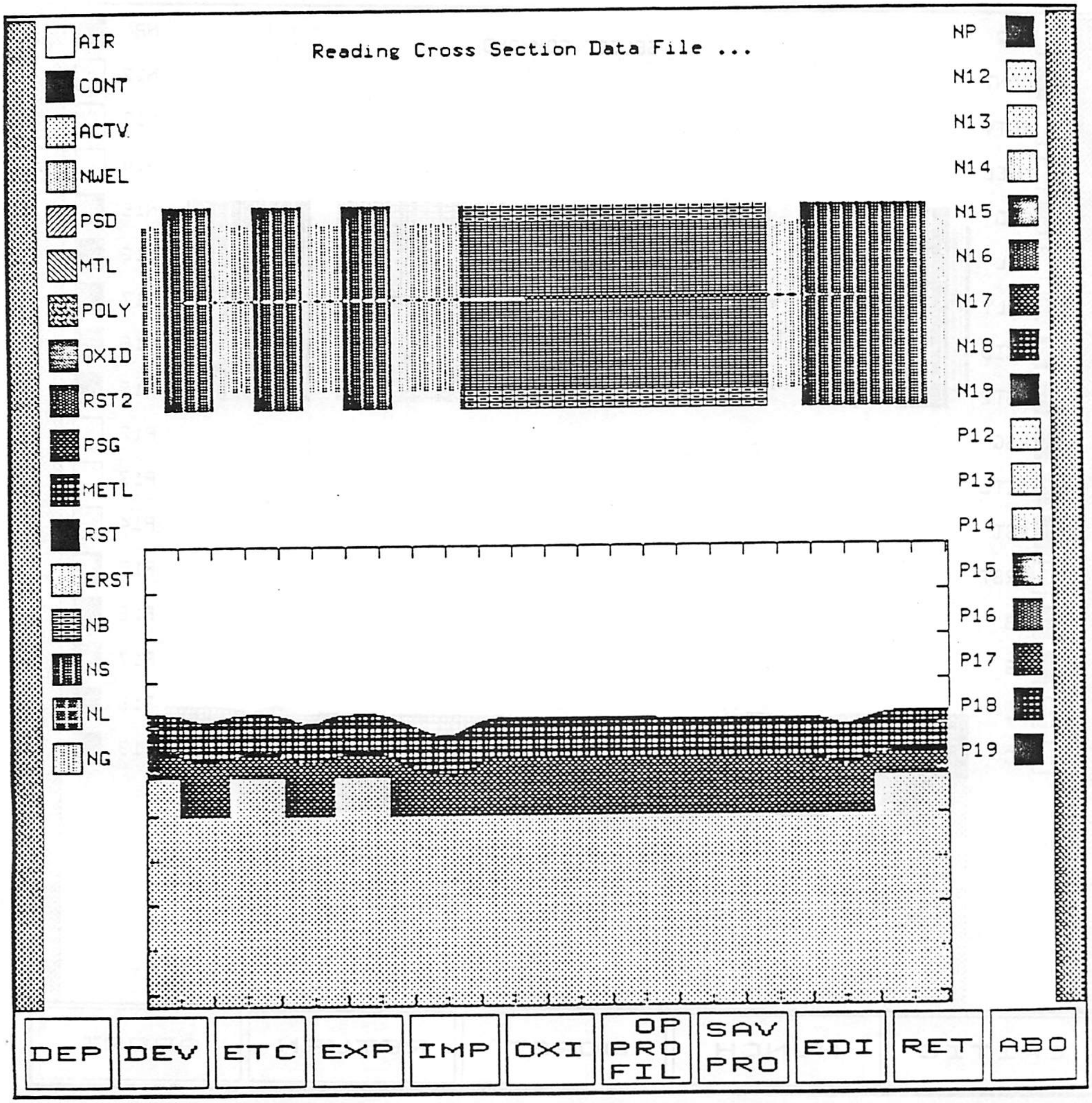


Figure 4.7 Effect of 0.6 micron misalignment

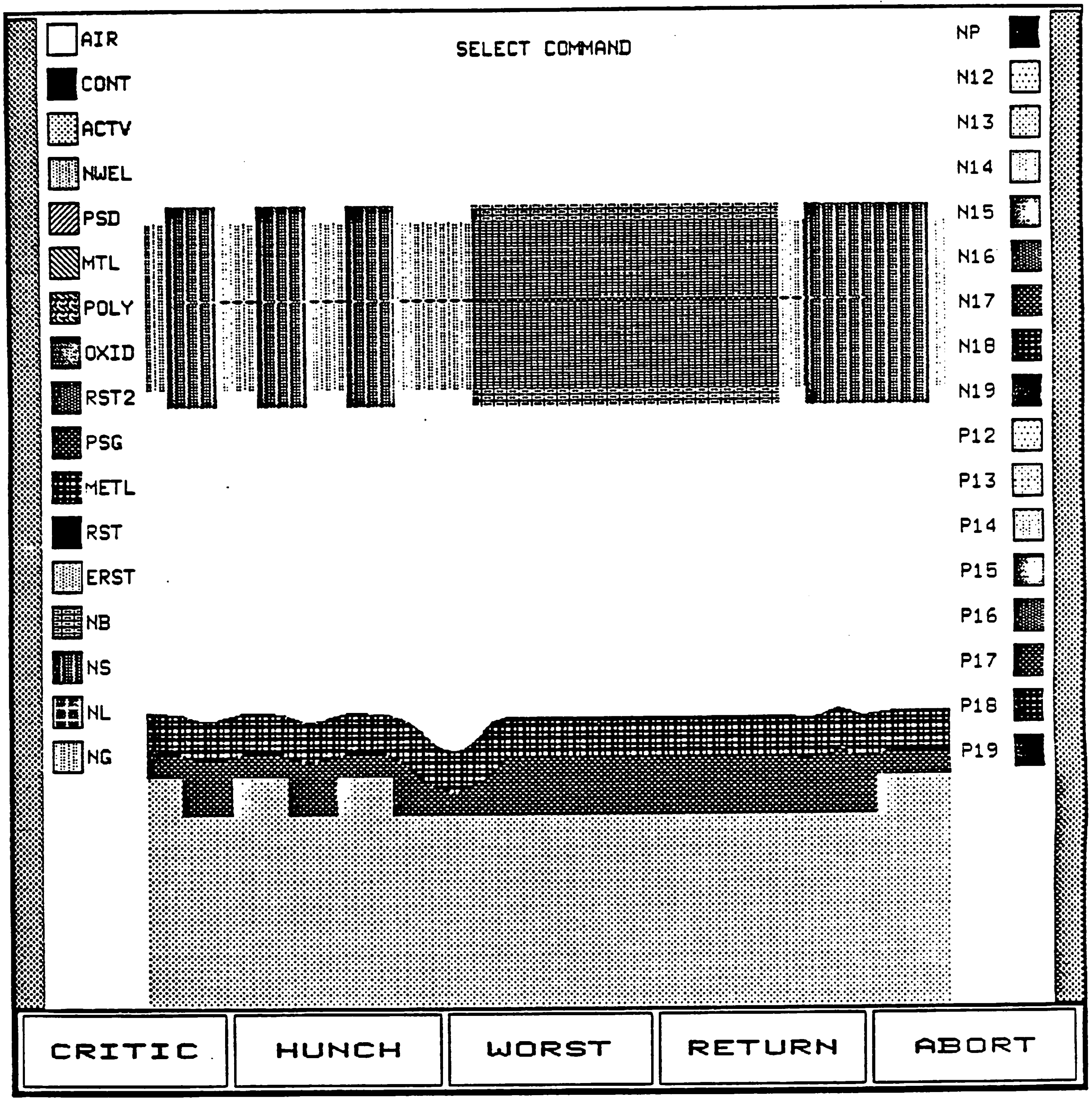


Figure 4.8 Effect of 0.9 micron misalignment

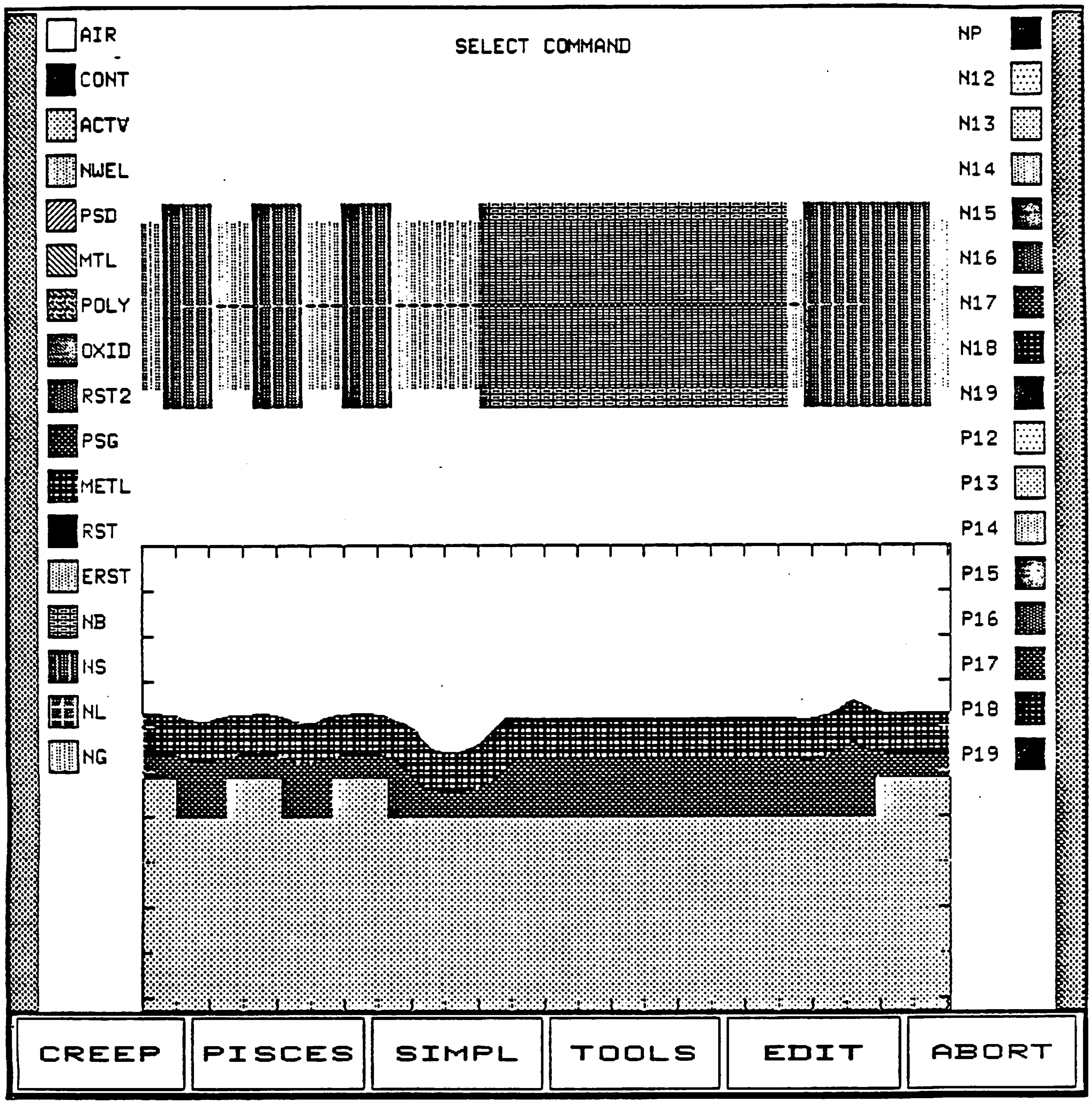


Figure 4.9 Effect of 1.2 micron misalignment

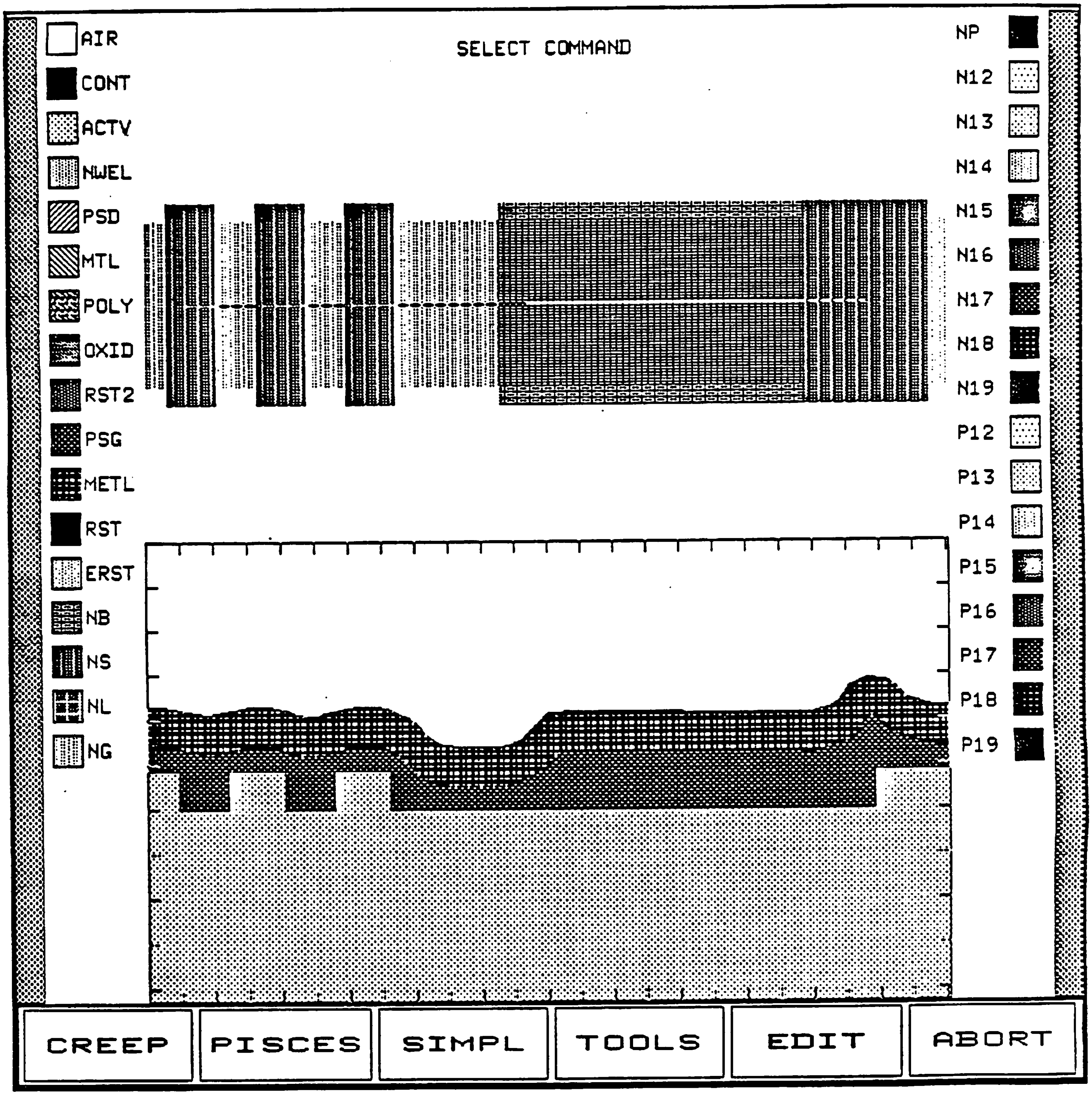


Figure 4.10 Effect of 1.8 micron misalignment

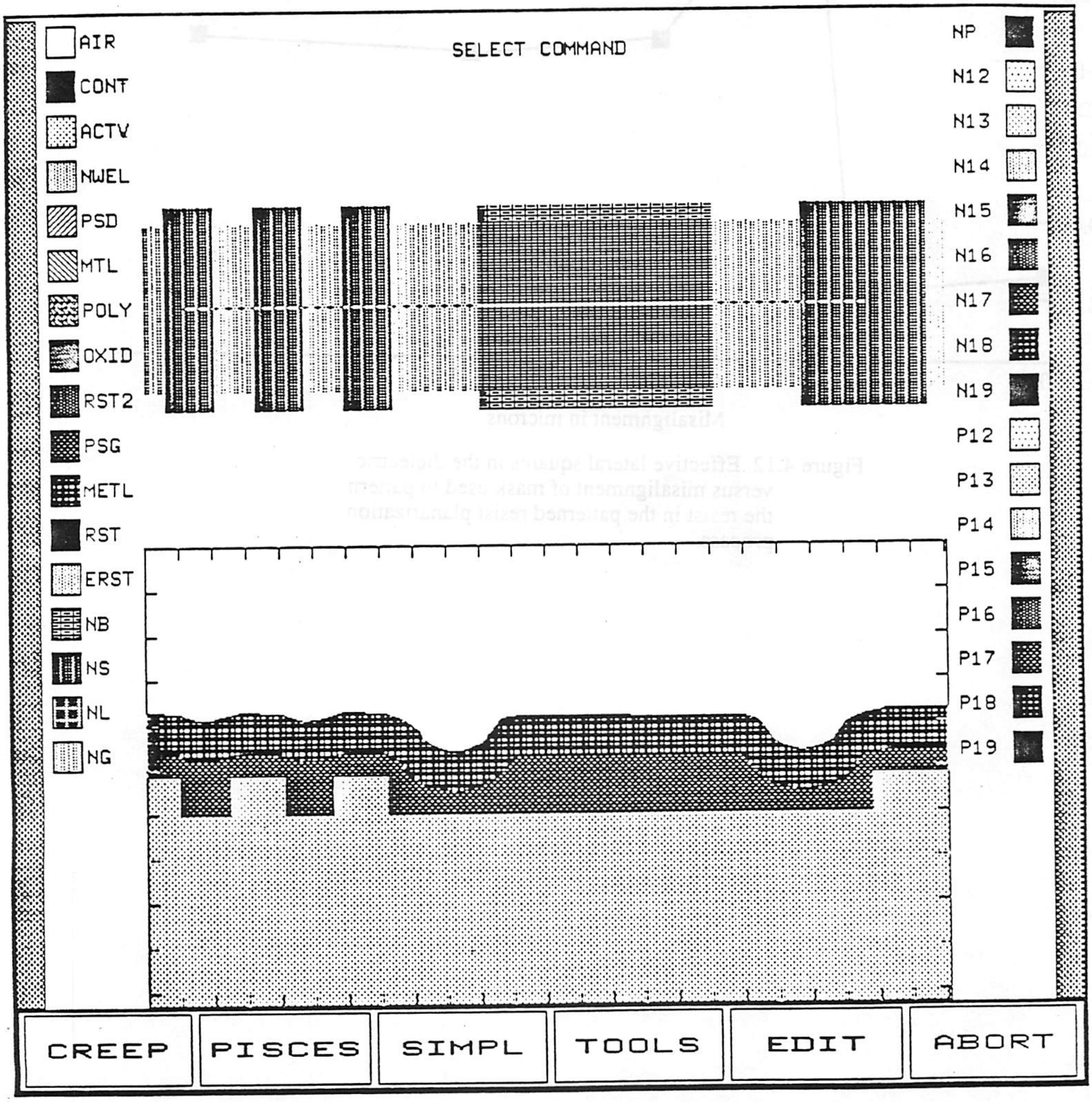


Figure 4.11 Effect of mask NB being too small

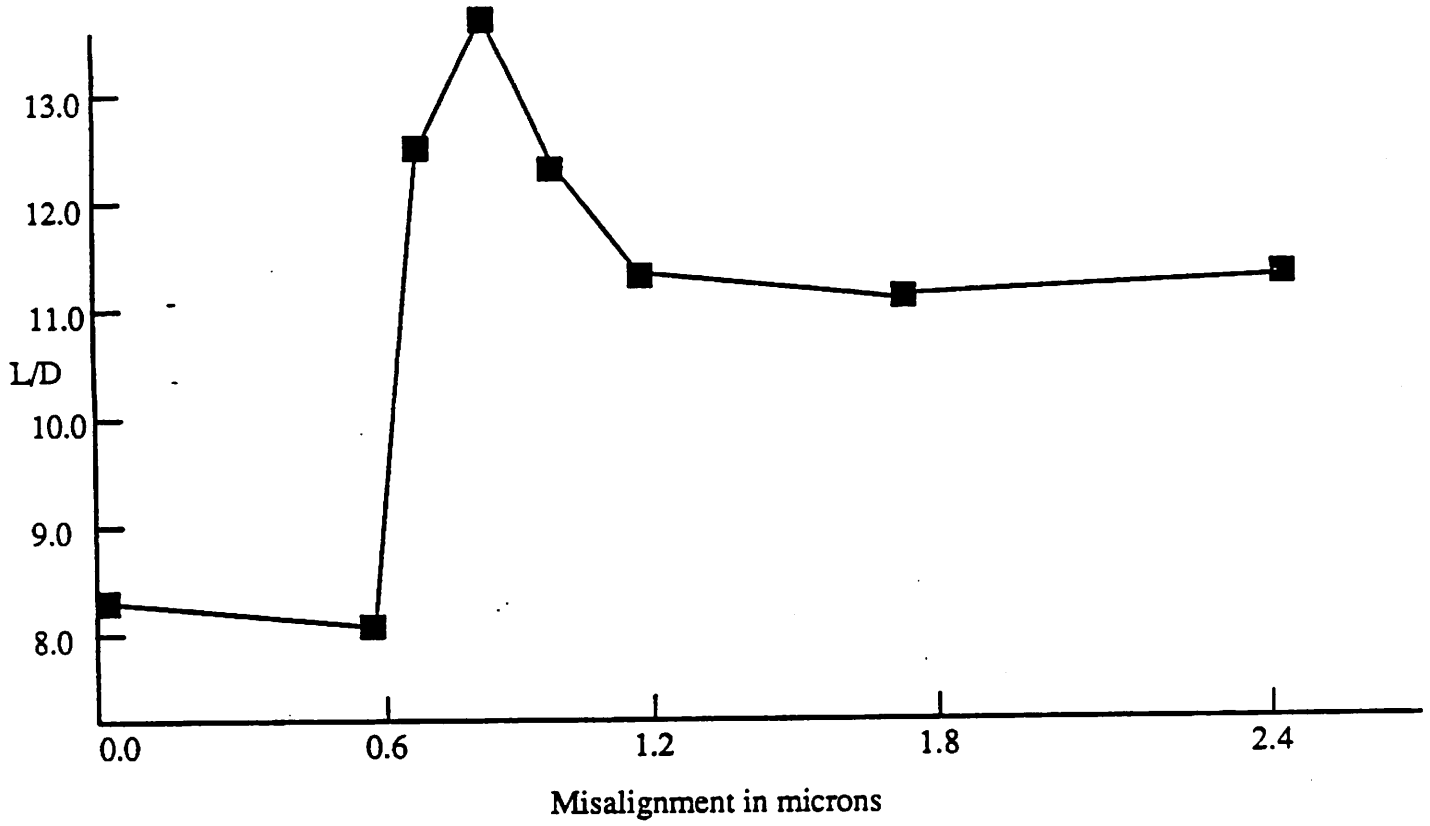
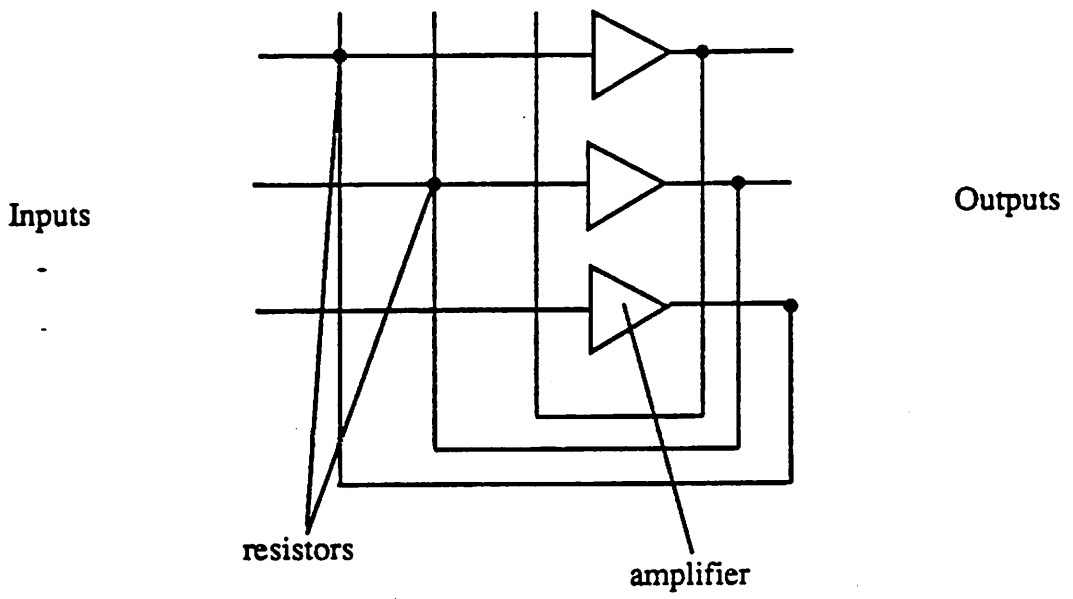


Figure 4.12. Effective lateral squares in the dielectric versus misalignment of mask used to pattern the resist in the patterned resist planarization process.

Basic schemetic of Hopfield neural network:



Schematic of one neuron:

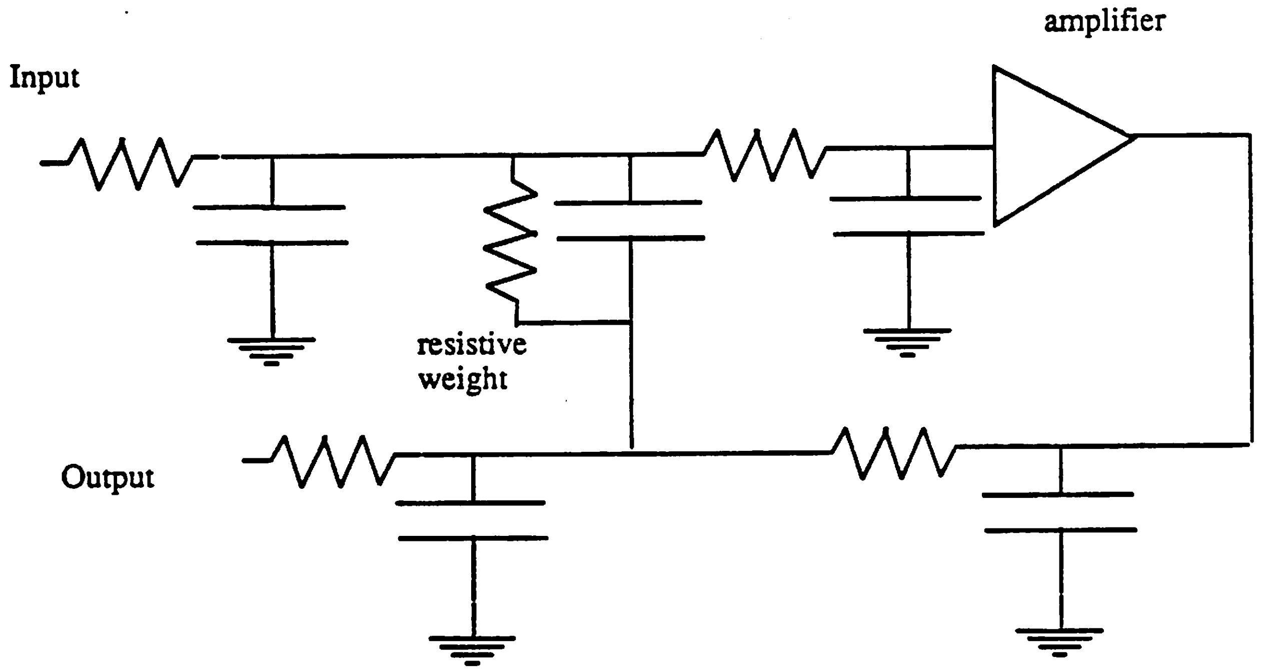


Figure 5.1. Hopfield Neural Network

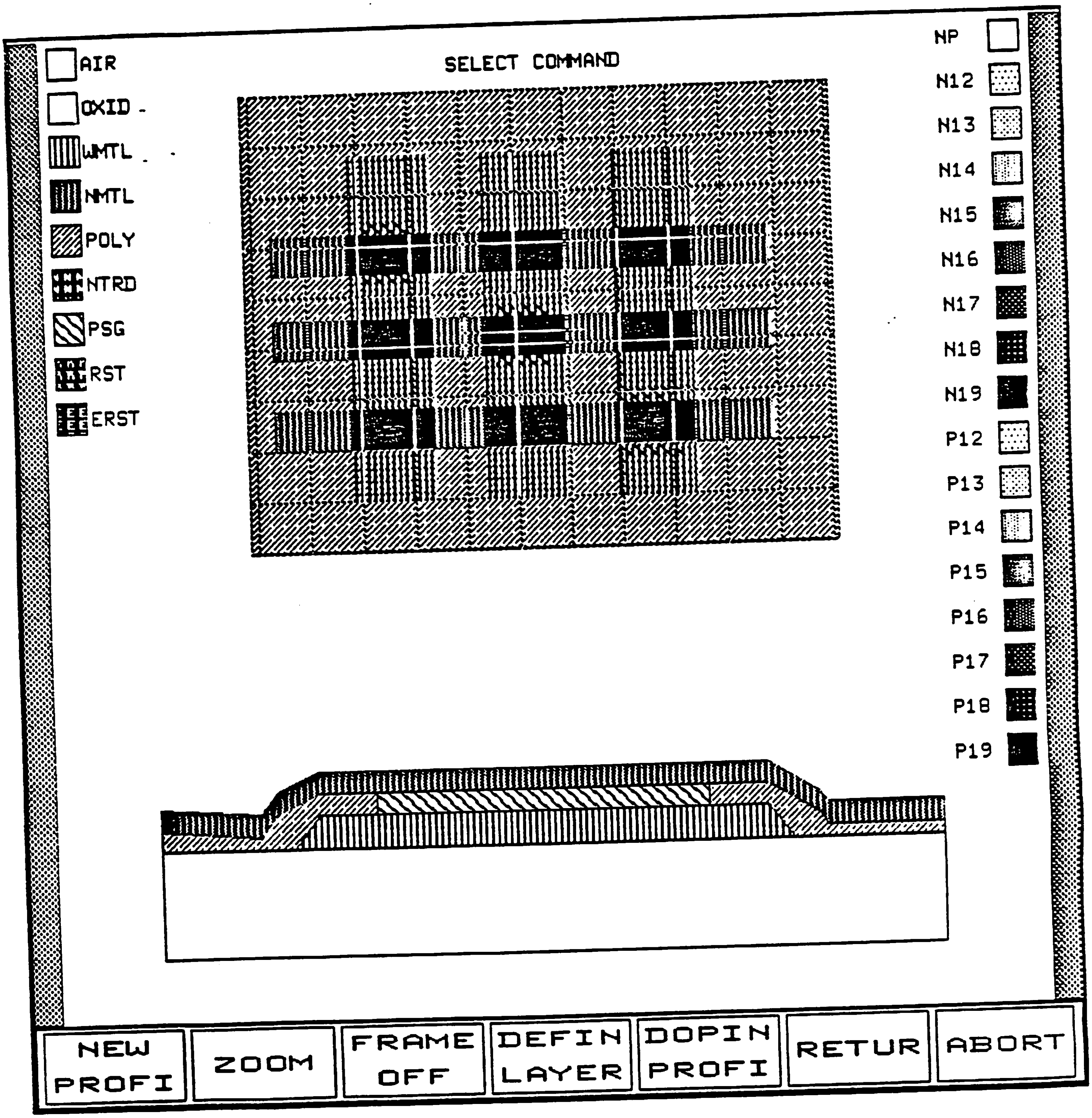


Figure 5.2. One interconnect element of a VLSI Hopfield Neural network. In this example, POLY is polyimide, PSG is polysilicon, WMTL is tungsten and NMTL is nickel. The thickness of each layer is approximately 0.1 microns. The width of the tungsten lines is 0.3 microns, and their spacing is 0.2 microns.

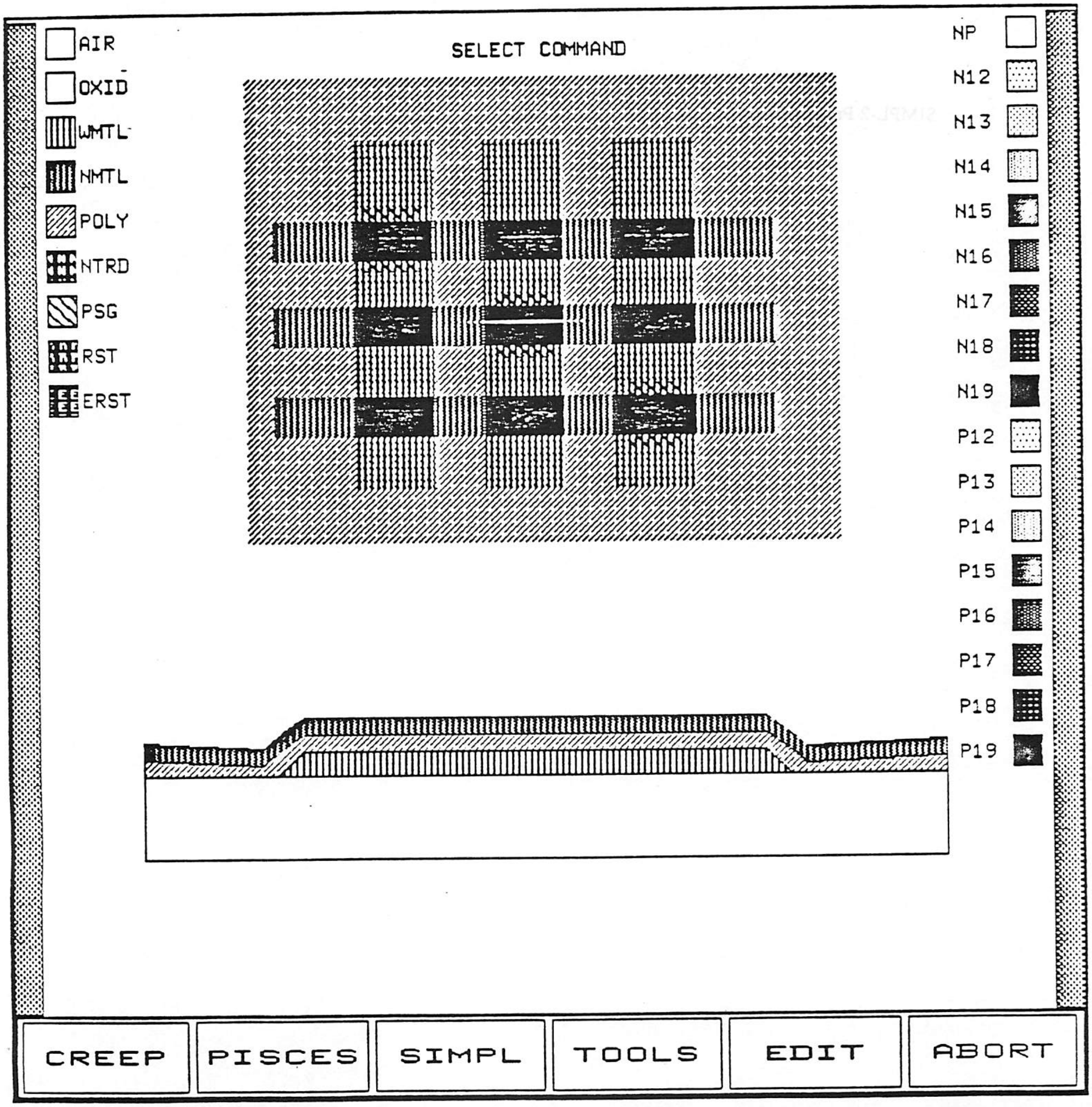


Figure 5.3. Counter electrode crossing base electrode in VLSI Hopfield neural network with no resistive contact.

Appendix A

SIMPL-2 Process Files

process.rbx1

```
*******************************************************************************

LAYOUT FILE :            rbx1.cif

SUBSTRATE TYPE:          p type, concentration 1e14

CUT-LINE COORDINATES : x1 =   -1705, y1 =    271
                       x2 =     596, y2 =    276

*******************************************************************************

* 1 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? rst
THICKNESS OF THE MATERIAL (micro-meter) ? 1.0
ISO, ANISO, VERT, or SAMPLE MENU (I,A,V,or S) ? v
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 2 *

WHICH PROCESS ? EXPO
WHICH MASK ? ns
INVERT THE MASK (yes or no) ? no
NAME OF THE EXPOSED RESIST ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 3 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 4 *

WHICH PROCESS ? ETCH
WHICH LAYER DO YOU WANT ETCH ? si
ETCH ALL (yes or no) ? no
AMOUNT OF VERTICAL ETCH (micro_meter) ? 0.85
RATIO X/Z OF ETCHING (0.0 <= RATIO <= 1.0) ? 0.0
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 5 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? rst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 6 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? psg
THICKNESS OF THE MATERIAL (micro-meter) ? 1.5
ISO, ANISO, VERT, or SAMPLE MENU (I,A,V,or S) ? i
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 7 *

WHICH PROCESS ? DEPO
```

```
NAME OF THE MATERIAL ? rst
THICKNESS OF THE MATERIAL (micro-meter) ? 0.75
ISO, ANISO, VERT, or SAMPLE MENU (I,A,V,or S) ? v
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 8 *

WHICH PROCESS ? EXPO
WHICH MASK ? nb
INVERT THE MASK (yes or no) ? no
NAME OF THE EXPOSED RESIST ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 9 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 10 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? rst2
THICKNESS OF THE MATERIAL (micro-meter) ? 1.0
ISO, ANISO, VERT, or SAMPLE MENU (I,A,V,or S) ? i
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 11 *

WHICH PROCESS ? ETCN
etchrate for RST2, layer 3 (um/sec) ? 0.01
etchrate for RST, layer 2 (um/sec) ? 0.01
etchrate for PSG, layer 1 (um/sec) ? 0.01
etchrate for SI, layer 0 (um/sec) ? 0.001
timestep in seconds ? 50
number of steps ? 4
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 12 *

WHICH PROCESS ? ETCU
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


* 13 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? metl
THICKNESS OF THE MATERIAL (micro-meter) ? 0.85
ISO, ANISO, VERT, or SAMPLE MENU (I,A,V,or S) ? i
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes


WHICH PROCESS ? END
```

```
*************************************************************************************

LAYOUT FILE :           nn.cif

SUBSTRATE TYPE:         p type, concentration 0

CUT-LINE COORDINATES : x1 =    -59, y1 =     12
                       x2 =     30, y2 =     12

*************************************************************************************

* 1 *

WHICH PROCESS ? OXID
OXIDE THICKNESS (micro-meter) ? .5
Xt (micro-meter) ? .5
Xe (micro-meter) ? .25
u1 ? .1
u2 ? .5
u3 ? .9
d1 ? .1
d2 ? .5
d3 ? .9
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 2 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? wmtl
THICKNESS OF THE MATERIAL (micro-meter) ? .1
ISOTROPIC, ANISOTROPIC, OR VERTICAL (I, A, or V) ? v
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 3 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? rst
THICKNESS OF THE MATERIAL (micro-meter) ? .25
ISOTROPIC, ANISOTROPIC, OR VERTICAL (I, A, or V) ? v
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 4 *

WHICH PROCESS ? EXPO
WHICH MASK ? wmtl
INVERT THE MASK (yes or no) ? no
NAME OF THE EXPOSED RESIST ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 5 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT
```

```
* 6 *

WHICH PROCESS ? ETCH
WHICH LAYER DO YOU WANT ETCH ? wmtl
ETCH ALL (yes or no) ? no
AMOUNT OF VERTICAL ETCH (micro_meter) ? .1
RATIO X/Z OF ETCHING (0.0 <= RATIO <= 1.0) ? .1
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 7 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? rst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 8 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? poly
THICKNESS OF THE MATERIAL (micro-meter) ? .10
ISOTROPIC, ANISOTROPIC, OR VERTICAL (I, A, or V) ? a
SPUTTERING SOURCE ANGLE (degrees) ? 20
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 12 *

WHICH PROCESS ? EXPO
WHICH MASK ? psg
INVERT THE MASK (yes or no) ? yes
NAME OF THE EXPOSED RESIST ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 13 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? erst
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

* 14 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? nmtl
THICKNESS OF THE MATERIAL (micro-meter) ? .10
ISOTROPIC, ANISOTROPIC, OR VERTICAL (I, A, or V) ? v
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? WAIT

WHICH PROCESS ? END
```

Appendix B

Source code for the majority of the routines developed for this project.

dix_actions4.c
F77Layers.c

```c
/* dix_action4.c
 *      Fourth part of SIMPL-DIX action routines.
 *
 *      See "command_control.c".
 *
 * Edward W. Scheckler Nov. 15, 1988
 * Copyright (C) 1988  U. C. Berkeley SAMPLE Group
 */


#include <stdio.h>
#include <math.h>
#include <X/Xlib.h>

#include "simpl-dix.h"
#include "simpl.h"
#include "cif.h"
#include "command.h"
#include "display.h"
#include "default.h"


/*
 * External declarations.
 */

char *RACPLE_Path;
extern dixViewport DIX_Viewport[VIEWPORT_SIZE];

extern floatView Profile_View;
extern floatBound Profile_Bound;


extern short Pattern_Size;

extern short Command_Id;
extern short Menu_Id;

extern short Cutline_Status;
extern short Layout_Status;
extern short Pattern_Status;
extern short Profile_Status;
extern short ProfileFrame_Status;


extern FontInfo *Body_FontInfo;
extern Color Background_Color;
extern Color Foreground_Color;
extern simplPolygon *SIMPL_PolygonRt;
static float xleft[50],yleft[50];
static float squares[50];
/*******

Routines to call a version of RACPLE

Added: Oct 24, 1988 EWS

If the file RACPLE_RESULTS already exists, then DIX
assumes that it belongs to the current profile. If not
such a file is created by calling RACPLE for each polygon
in the profile.

********/
```

```c
DoRacple()
{
    FILE *fpp,*fopen();
    simplPolygon *Polygonptr;
    char polyname[NAME_SIZE];
    char junk[80],colon;
    struct float_path *Polypath;
    int i,ii;

    Polygonptr = SIMPL_PolygonRt;
    i = 0;
        if((fpp=fopen("RACPLE_RESULTS","r"))!=NULL){
      if(GetYesOrNo("RACPLE_RESULTS exists. Use it?")==YES){
                /*read the file*/
                        fscanf(fpp,"%s %s",junk,junk);
                        fscanf(fpp,"%s %s %s %s %s %s %s %s %s",junk,junk,
                                junk,junk,junk,junk,junk,junk,junk);
                        for(;;){
                        fscanf(fpp,"%d",&ii);
                        if(ii != i ) break;
                        fscanf(fpp,"%s",polyname);
                        fscanf(fpp,"%f",&xleft[i]);
                    fscanf(fpp,"%f",&yleft[i]);
                        fscanf(fpp,"%f",&squares[i]);
                        i++;
                        }
         fclose(fpp);
        } else {
         fclose(fpp);
                goto makenew;
        }/*GetYesOrNo*/
         } else {
makenew:    fpp = fopen("RACPLE_RESULTS","w");
    fprintf(fpp,"****************RACPLE RESULTS****************\n");
    fprintf(fpp,"Material Type, Head Vertex (x,z), # of Lateral Squares\n");
        while (Polygonptr != NULL) {
        CreateRacpleInput(i,Polygonptr,&xleft[i],&yleft[i]);
        RunRacple(i,&squares[i]);
    sprintf(polyname,"%s",Polygonptr->name);
        fprintf(fpp,"%3d %6s %12.3f %7.3f %9.3f\n",i,
                        polyname,xleft[i],yleft[i],squares[i]);
    Polygonptr = Polygonptr->next;
    i++;
         } /*while*/
    fclose(fpp);
    } /*if*/
        PrintRacpleData(i);
}
/*******

PrintRacpleData(number_of_points)

Routine to sort each row of data by value of
x coordinate and print it to profile viewport.

Revised 11/8/88 EWS

*******/
PrintRacpleData(i)
        int i;
{
        dixViewport view;
        int max_per_row,no_rows,no_col;
        int left_edge,right_edge,top_edge,bottom_edge;
    int wval,hval,xval[50],yval[50],xpoint[50],ypoint[50];
        int index,j,k,l,lm,klm,gap,n,flag;
    float tempx,tempy,temps,prevx;
    int samexl[10],samexr[10],no_same,ltoh[10],htol[10];
    char value[15];
        char prompt_string[80];
    Pixel fore,back;
        FontInfo *font;
        FILE *fp;

        view = DIX_Viewport[1];
        left_edge = GetProfileViewX(view,Profile_View,Profile_Bound.left);
    right_edge = GetProfileViewX(view,Profile_View,Profile_Bound.right);
    top_edge = GetProfileViewY(view,Profile_View,Profile_Bound.top);
        bottom_edge=GetProfileViewY(view,Profile_View,Profile_Bound.bottom);
    sprintf(value,"%f",squares[0]);
    wval = GetTextWidth(value,5,Body_FontInfo);
    hval = (2*TEXT_MARGIN)+Body_FontInfo->height;
        back = Background_Color.pixel;
        fore = Foreground_Color.pixel;
        font = Body_FontInfo;

    max_per_row = ((right_edge - left_edge)/wval) - 3;
    if(i%max_per_row == 0) {
                no_rows = i/max_per_row -1;
        } else {
        no_rows = i/max_per_row ;
        } /*if*/

    for(l=0;l<=no_rows;l++){
                if(l==no_rows && i%max_per_row != 0) {
                        no_col = i%max_per_row;
                } else {
                        no_col = max_per_row;
                }/*if*/
          /******************
                Do a shell sort on each row using x-coordinate
          ******************/
                lm = l*max_per_row;
                for(gap=no_col/2; gap>0;gap/=2)
                        for(j=gap;j<no_col; j++)
                                for(k=j-gap; (k>=0 && xleft[k+lm] >
                                          xleft[k+lm+gap]);k-=gap) {
                                    klm = k + lm;
                                        tempx=xleft[klm];
                                        tempy=yleft[klm];
                     temps=squares[klm];
                                        xleft[klm]=xleft[klm+gap];
                                        yleft[klm]=yleft[klm+gap];
                                        squares[klm]=squares[klm+gap];
                                        xleft[klm+gap]=tempx;
                                        yleft[klm+gap]=tempy;
                                        squares[klm+gap]=temps;
                                }/*for k*/
                /******************
                        Try to eliminate crossing lines
        ******************/
                /*fp = fopen("debug","w");*/
                ltoh[0]=1;
                htol[0]=0;
                prevx = xleft[lm];
        no same = 0;
                flag = 0;
                samexl[no same] = 0;
```

```c
                for(k=0;k<no_col;k++){
                        klm = k + lm;
                        xpoint[klm]=GetProfileViewX(view,Profile_View,
                                        xleft[klm]);
                        ypoint[klm]=GetProfileViewY(view,Profile_View,
                                        yleft[klm]);
                        if(klm < max_per_row){
                                yval[klm] = bottom_edge - TEXT_MARGIN
                                        - Body_FontInfo->height;
                                xval[klm] = left_edge +(wval+wval/3)*klm
                                        + wval/5 + wval*(max_per_row - no_col)/2;
                        } else {
                                yval[klm] = top_edge + (klm/max_per_row -1)*
                                        (Body_FontInfo->height + TEXT_MARGIN);
                                xval[klm] = left_edge + (wval+wval/3)*
                                        (klm%max_per_row)+wval*(max_per_row -
                                        no_col)/2 + wval/5;
                        }
                        if(     k!=0 && xleft[klm]-prevx < 0.001) {
                                samexr[no_same]=klm;
                                if ((l==0 && xpoint[klm]< xval[klm])||
                                                                (l>0 && xpoint[klm]> xval
                                        ltoh[no_same] = 1;
                                        htol[no_same] = 0;
                                        }
                                if ((l==0 && xpoint[klm]> xval[klm])||
                                                                (l>0 && xpoint[klm]< xval
                                        ltoh[no_same] = 0;
                                        htol[no_same] = 1;
                                        }
                                flag = 1;
                                if(k == no_col - 1){
                                        no_same++;
                                        }
                                } else {
                                if (flag == 1) no_same++;
                                flag = 0;
                                ltoh[no_same]=1;
                                htol[no_same]=0;
                                samexl[no_same] = klm;
                                prevx = xleft[klm];
                                }
                } /*for k*/
                for(n = 0; n < no_same; n++){
                /*fprintf(fp,"%i;xl=%i, xr=%i\n",n,samexl[n],samexr[n]);*/
                for(gap=(samexr[n]-samexl[n]+1)/2;gap>0;gap/=2)
                        for(j=gap; j<(samexr[n]-samexl[n]+1); j++)
                                for(k=j-gap; (k>=0 && ((yleft[k+samexl[n]+lm] >
                                        yleft[k+lm+samexl[n]+gap])&&(ltoh[n]==1) ||
                                        (yleft[k+samexl[n]+lm] < yleft[k+lm
                                                +samexl[n]+gap])&&(htol[n]==1)) );k-=gap)
                                                klm = k+lm+samexl[n];
        /*fprintf(fp,"swap %i %i; htol %i ltoh %i\n",klm,gap,ltoh[n],htol[n]);*/
                                                tempx=xleft[klm];
                                                tempy=yleft[klm];
                                                temps=squares[klm];
                                                xleft[klm]=xleft[klm+gap];
                                                yleft[klm]=yleft[klm+gap];
                                                squares[klm]=squares[klm+gap];
                                                xleft[klm+gap]=tempx;
                                                yleft[klm+gap]=tempy;
                                                squares[klm+gap]=temps;
                                                xpoint[klm+gap]=xpoint[klm];
                                                ypoint[klm+gap]=ypoint[klm];
```

```
                                                        xpoint[klm]=GetProfileViewX(view,Profile_
                                                                        xleft[klm]);
                                                        ypoint[klm]=GetProfileViewY(view,Profile_
                                                                        yleft[klm]);

                                        }
                } /*for n*/
                /*close(fp);*/

        }/*for l*/


        for(index=0;index<l;index++){
                sprintf(value,"%f",squares[index]);

                PrintText(view,xval[index],yval[index],
                        wval,hval,value,font,fore,
                        back, CLIP_RIGHT);
        if(index >= max_per_row || l<=max_per_row){
                        yval[index] = yval[index] + Body_FontInfo->height;
                        }
        XLine(view.self,xval[index],yval[index],
                                xpoint[index],ypoint[index],1,1,
                        (back^fore),GXxor,AllPlanes);
        XFlush();

        }/*for loop*/
}
/*********

Create Racple Input

This routine reads a ploygon in the SIMPL-DIX linked list
and decomposes it into an upper and lower layer. The layers
are written to a file which can be used as input to RACPLE.

Revised 10/13/88 EWS

Bugs: Sometimes it fails to print out the second layer
      I think there is a bug in one of the possible
         Cases.

*********/
CreateRacpleInput(filecnt,Poly,xleft,zleft)
    int filecnt;
        simplPolygon *Poly;
    float *xleft,*zleft;
{
    FILE *fp;
    floatPath  *pathptr;
    float xmin,xmax,zmin,zmax,xmaxz,xminz;
    float xmaxzm,xminzm;
    float xhead,zhead;
    float x0[500],z0[500];
    float x1[500],z1[500];
    float x2[500],z2[500];
    int i,j,k;
    int head_loc,degen_edge;
    char filename[80];
    int count,ixmin,ixmax,ixmaxm,ixminm;
    int number_pts1,number_pts2;

/********
find x and z extrema
********/
    xmin = 1000.0;
xmax = -10.0;
xmaxz = -10.0;
xminz = -10.0;
xmaxzm = 1000.0;
xminzm = 1000.0;
    zmin = 1000.0;
zmax = -10.0;
count = 0;
pathptr = Poly->path;
xhead = pathptr->point.x;
    zhead = pathptr->point.y;
while (pathptr != NULL) {
x0[count] = pathptr->point.x;
z0[count] = pathptr->point.y;
            if(pathptr->point.x >= xmax){
                    if (count != 0 && pathptr->point.x == xmax){
                            if(pathptr->point.y >= xmaxz){
                                    ixmax = count;
                                    xmaxz = pathptr->point.y;
                            } else if(pathptr->point.y <= xmaxzm){
                                    ixmaxm = count;
                                    xmaxzm = pathptr->point.y;
                            }
        } else {
                            ixmax = count;
                            ixmaxm = count;
                            xmax = pathptr->point.x;
                            xmaxz = pathptr->point.y;
                            xmaxzm = pathptr->point.y;
                    }
        }/*if pathptr */
    if(pathptr->point.x <= xmin){
                    if(count!= 0 && pathptr->point.x == xmin){
                            if(pathptr->point.y >= xminz){
                                    ixmin = count;
                                    xminz = pathptr->point.y;
                            } else if(pathptr->point.y <= xminzm) {
                                    ixminm = count;
                                    xminzm = pathptr->point.y;

            }
                    } else {
                            ixmin = count;
                            ixminm = count;
                            xmin = pathptr->point.x;
                            xminz = pathptr->point.y;
                            xminzm = pathptr->point.y;

            }
        }/*if pathptr*/
    if(pathptr->point.y >= zmax)
            zmax = pathptr->point.y;
    if(pathptr->point.y <= zmin)
            zmin = pathptr->point.y;
            pathptr = pathptr->next;
    count++;
} /*while*/
/******
            determine relative location of head vertex
    and corner points.
*******/
if((ixmin==ixminm && ixmax==ixmaxm){degen_edge = 3;
    } else if(ixmin == ixminm) {degen_edge = 1;
} else if(ixmax == ixmaxm) {degen_edge = 2;
} else {degen_edge = 0;}
```

```
    if(xhead == xmin && zhead == xminz) {
                        head_loc = 0;
    } else
    if(xhead == xmax && zhead == xmaxz) {
                        head_loc = 2;
    } else
    if(xhead == xmin && zhead == xminzm) {
            head_loc = 6;
    } else
    if(xhead == xmax && zhead == xmaxzm) {
            head_loc = 4;
        }else
    if( ixmin > ixmax && ixminm > ixmaxm ){
                        head_loc = 1;
    } else
    if( ixmin < ixmax && ixminm < ixmaxm ){
            head_loc = 5;
    } else
    if( degen_edge == 0 || degen_edge == 1){
                if(xhead == xmax &&(zhead < xmaxz && zhead > xmaxzm)) {
                head_loc = 3;
        }
    } else
    if( degen_edge == 0 || degen_edge == 2){
        if(xhead == xmin &&(zhead < xminz && zhead > xminzm)) {
                head_loc = 7;
        }
    }


    switch(head_loc) {
    case 0:
    case 6:
    case 7:
            number_pts1 =  ixmax - ixmin +1;
        number_pts2 =  ixminm - ixmaxm +1;
                if(head_loc == 6){
                        number_pts2 = count - ixmaxm +1 ;
        }
        for(k=0;k<number_pts1;k++) {
                        x1[k] = x0[ixmin+k];
            z1[k] = z0[ixmin+k];
        }
        for(k=0;k<number_pts2;k++) {
            if(head_loc==6)ixminm=count;
            x2[k] = x0[ixminm-k];
            z2[k] = z0[ixminm-k];
        }
        if(head_loc == 6){
                        x2[0]=x0[0];
                        z2[0]=z0[0];
                }
        break;
    case 2:
    case 3:
    case 4:
                /* needs patch for case 2*/
        number_pts1 = ixmax - ixmin +1;
        number_pts2 = ixminm - ixmaxm +1;
        for(k=0;k<number_pts2;k++) {
            x2[k] = x0[ixminm-k];
            z2[k] = z0[ixminm-k];
        }
```

```
                for(k=0;k<number_pts1;k++) {
                    x1[k] = x0[ixmin+k];
                    z1[k] = z0[ixmin+k];
                }
                break;
        case 1:
                        number_pts2 = ixminm - ixmaxm + 1;
                        number_pts1 = ixmax + count-ixmin + 1;
                        for(k=0;k<number_pts2;k++) {
                    x2[k] = x0[ixminm-k];
                    z2[k] = z0[ixminm-k];
                }
                for(k=0;k<number_pts1;k++) {
                    if(k < count-ixmin){
                                x1[k] = x0[ixmin+k];
                            z1[k] = z0[ixmin+k];
                    } else {
                     x1[k] = x0[k-(count-ixmin)];
                                z1[k] = z0[k-(count-ixmin)];
                    }
                }
                break;
        case 5:
                    number_pts1 = ixmax - ixmin + 1;
                        number_pts2 = ixminm + (count-ixmaxm) + 1;
                        for(k=0;k<number_pts1;k++){
                            x1[k] = x0[ixmin+k];
                                z1[k] = z0[ixmin+k];
                }
                for(k=0;k<number_pts2;k++) {
                    if(k <= ixminm){
                                x2[k] = x0[ixminm-k];
                                z2[k] = z0[ixminm-k];
                        } else {
                                x2[k] = x0[count-(k-ixminm)];
                     z2[k] = z0[count-(k-ixminm)];
                    }
                }
                break;
          default:
                number_pts1=1;
                number_pts2=1;
                x1[0]=-99.99;
                z1[0]=-99.99;
                x2[0]=-99.99;
                z2[0]=-99.99;
                        break;
        } /*switch*/

          *xleft = xmin;
          *zleft = xminz;
/***********************************************/
/* write layer information to f77 format file */
/***********************************************/
        sprintf(filename,"RACPLE_Input.%d",filecnt);
            fp = fopen(filename,"w");

/*******write first layer to file***************/
        AddPointstoLayer(x2,z2,&number_pts2);
            fprintf(fp,"\n %10.6f %10.6f %10.6f %10.6f\n",
                                    xmin,xmax,zmin-zmax-0.1,0.1);
            fprintf(fp,"%10.6f \n ",2.0);
            fprintf(fp,"%10.6f \n",1.0*number_pts2);
            for (j=0; j<number_pts2; j++)
```

```
                fprintf(fp,"%10.6f %10.6f\n",x2[j],
                                            z2[j] -zmax );

/*******write second layer to file*************/
    AddPointstoLayer(x1,z1,&number_pts1);
    fprintf(fp,"%10.6f \n",1.0*number_pts1);
    for (j=0;j<number_pts1;j++){
        fprintf(fp,"%10.6f %10.6f\n",x1[j],
                                            z1[j]-zmax );
        }
        fclose(fp);

}/*Create Racple Input*/

AddPointstoLayer(xx,zz,nn)
float *xx,*zz;
int *nn;
{
        float eps;
    float minlength;
    int i,j,flg,dbgcnt;
    float xnew[500],znew[500];

    minlength = fabs(xx[*nn-1] - xx[0])/20.0;
    eps = 0.0001;
    xnew[0] = xx[0];
    znew[0] = zz[0];
    j = 1;
    dbgcnt = 0;
    while(dbgcnt<10){
        dbgcnt++;
        j=1;
        flg = 0;
                for(i=1;i<*nn;i++){
          if(fabs(zz[i]-zz[i-1])<eps && fabs(xx[i]-xx[i-1])>minlength){
                flg = 1;
                                xnew[j]=(xx[i]+xx[i-1])/2.0;
                    znew[j]=(zz[i]+zz[i-1])/2.0;
                j++;
                }
                        xnew[j]=xx[i];
                znew[j]=zz[i];
                j++;
        }
        if(flg == 1){
                for(i=1;i<j;i++){
                                xx[i]=xnew[i];
                    zz[i]=znew[i];
             /*****patch to eliminate small sharp spikes
                   from RACPLE input *******/
                if(i<j-1){
                                            if(fabs(xnew[i+1]-xnew[i-1]) < minlength/3.5 &&
                                                    (znew[i-1]<znew[i])&&
                                                    (znew[i+1]<znew[i])){
                                                    zz[i] = (znew[i-1]+znew[i+1])/2.0;

                        }
                    }
                  }
            *nn = j;
        } else return(0);
    }
 }

 /***********
```

```
         This routine calls RACPLE with the appropriate input
         file. Output is sent to RACPLE_Output.#
         The Output file is read and the number of lateral   ,
      squares for the first layer is returned


      revised 10/13/88 EWS

************/
RunRacple(filecnt,output_data)
float *output_data;
int filecnt;
{
    FILE *fp;
    FILE *fp2;
    int idum;
    char str[17],str2[7];
        char command[80];
    char Racple[80];
    char debug[80],outfile[80];
        char message_string[80];

        sprintf(message_string,"Running RACPLE : %i",filecnt);
        Prompt(message_string);

    if(RACPLE_Path != NULL){
                strcpy(Racple,RACPLE_Path);
    } else {
                strcpy(Racple,DEF_RACPLE_PATH);
    }
        /*strcpy(Racple,"/users2/edscheck/DEVICESIM/racple/racple");*/
        sprintf(command,"%s < RACPLE_Input.%i > RACPLE_Output.%i",
                        Racple,filecnt,filecnt);
        system(command);
   /* sprintf(debug,"debug.%i",filecnt);
    fp2 = fopen(debug,"w");
        fprintf(fp2,"%s",command);
    fclose(fp2); */
    sprintf(outfile,"RACPLE_Output.%i",filecnt);
    fp = fopen(outfile,"r");
    fscanf(fp,"%i %17c %f %7c",&idum,str,output_data,str2);
    /*fscanf(fp,"%f",output_data); */
    fclose(fp);
}/*RunRacple*/

/***************/
Display_Nonplanar_Etch()
/***************/
{
        dixViewport view;
        Pixel fore,back;
        FontInfo *font;
    FILE *fp,*fopen();
        float xmx,xmn,ymx,ymn,nlns;
    float no_pts;
    float xx1,yy1,xx2,yy2;
    int i,j,x1,y1,x2,y2;

        view = DIX_Viewport[1];
        back = Background_Color.pixel;
        fore = Foreground_Color.pixel;
        font = Body_Font_Info;
        fp = fopen("SAMPLE_netchf77","r");
        fscanf(fp,"%f%f%f%f",&xmn,&xmx,&ymn,&ymx);
        fscanf(fp,"%f",&nlns);
        for(j=0;j<nlns;j++){
/*****
        read a line segment
*****/
        fscanf(fp,"%f",&no_pts);
        fscanf(fp,"%f%f",&xx1,&yy1);
        yy1 = (Profile_Bound.top  + yy1);
        x1 = GetProfileViewX(view,Profile_View,xx1);
    y1 = GetProfileViewY(view,Profile_View,yy1);
        for(i=1;i<no_pts;i++){
                fscanf(fp,"%f%f",&xx2,&yy2);
        yy2 = (Profile_Bound.top + yy2);
                x2 = GetProfileViewX(view,Profile_View,xx2);
                y2 = GetProfileViewY(view,Profile_View,yy2);
/****
        draw it to screen
****/
                XLine(view.self,x1,y1,x2,y2,1,1,(back^fore),GXxor,AllPlanes);
            XFlush();
                x1 = x2;
        y1 = y2;
        }/*for i*/
    }/*for j*/
  fclose(fp);
}
```

```
/*F77Layers.c
*       Routines used to run non-planar
*       etch simulation with SAMPLE
*
*Edward W. Scheckler   Dec. 1, 1988
*Copyright (C) 1988 U.C. Berkeley SAMPLE Group
*/
#include "SIMPL.h"
#include "SIMPL_Macros.h"
#include "Local.h"
#include <stdio.h>

typedef struct layers {
        float x[500];
    float z[500];
    char layer_type[15];
        int number_pts;
} LAYERS;
static LAYERS layer_array[30];
static int number_layers;

static POLYGONPTR top_poly_list[30];
/**********************/
Run_nonplanaretch()
/**********************/
{
        FILE *fp;
        char command[100];
    char Sample1_7[100];

    GLprintf("initializing for nonplanar etch");
        if(-1 != Write_SAMPLEnplnr()){
    sprintf(Sample1_7,"/users2/edscheck/bin/sample1.7");

#ifdef UNIX
        GLprintf("running SAMPLE1.7 . . .");
    sprintf(command,"%s < SAMPLE_nonplanar > SAMPLE_nOutput",Sample1_7);
    system(command);
        sprintf(command,"mv -f f77punch7 SAMPLE_netchf77");
        system(command);
#endif
    }
}
Run_nonplanar_update(flag)
int flag;
{
    FILE *fptr;
    char YORN[3];
    char command[100];
    if (flag == 1){
            Redefine_Profile();
    } else {
    fptr = fopen("temp_save","r");
    Read_Data2(fptr);
    fclose(fptr);
    }
    sprintf(command,"rm -f temp_save");
    system(command);
}
/**********************/
Redefine_Profile()
/**********************/
{
COORDINATES tp[1000],bt[1000];
int ntp,nbt;
int i,j;
FILE *fp;
float xmx,xmn,zmn,zmx,nlns,nntp;


    fp = fopen("SAMPLE_netchf77","r");
    fscanf(fp,"%f%f%f%f",&xmx,&xmn,&zmn,&zmn);
    fscanf(fp,"%f",&nlns);
    for(j=0;j<nlns;j++){
        fscanf(fp,"%f",&nntp);
        ntp = 1*nntp;
                for(i=0;i<ntp;i++){
          if(j==3){
          fscanf(fp,"%f%f",&bt[i].x,&bt[i].z);
          nbt = ntp;
          } else {
                  fscanf(fp,"%f%f",&tp[i].x,&tp[i].z);
          }
        }
    }
    fclose(fp);

/*      for(i=0;i<layer_array[3].number_pts;i++){
                bt[i].x = layer_array[3].x[i];
        bt[i].z = layer_array[3].z[i];
    }
    nbt = layer_array[3].number_pts;*/


/*   if(layer_array[2].layer_type == NULL)*/
                sprintf(layer_array[2].layer_type,"PSG");

        New_Top_Poly(tp,ntp,bt,nbt,layer_array[2].layer_type);

}/*Redefine_Profile*/

/**********************/
Get_Layers()
/**********************
        This routine extracts the profile stored in SIMPL-2 and
        interprets it as a series of layers which can then be written
        to a SAMPLE plot file in f77punch7 format.
    It is also used to write an input file for SAMPLE nonplanar
    etch

        BUGS - destroys SIMPL-2 data structure
        This has been remedied by saving profile in a data file
        so that it can be reloaded at the end of this routine.
**********************/
{
    FILE *fptr,*fopen();
        int i,j;
        POLYGONPTR top_polygon,tmp1_polygon,top2_polygon;
        POLYGONPTR Find_Polygon(),Find_top_poly();
        VERTEXPTR  top_v,top2_v;
        VERTEXPTR  tmp1_v;
        VERTEXPTR  Get_LeftTop();
        char *Write_savefile();
        char previous[15];
        char Material_name[15];
        char *save_name;
        char s_name[80];
        char YORN[3];
        char layerfile_name[14];
    float xleft[30],xright[30],zleft[30];
        float top_z,top2_z; /* highest point in layer */
    float bottom_of_poly();
/********
        save SIMPL data structure since
        subsequent code will destroy it.
********/
  fptr = fopen("temp_save","w");
  Write_savefile2(fptr);
  close(fptr);
/************************************
        The Layers are extracted by reading the top contour, and
        then deleting  the polygon which make up the top
        contour which also shares the same material type as the top
        most vertex. This continues until the substrate is reached.
************************************/
        i=0;
for(;;)
{    j = 0;
    top2_z=0.0;
        top_z=0.0;
    top_polygon = NULL;
    top2_polygon = NULL;
    tmpl_polygon = NULL;
        tmpl_v = Get_LeftTop();
    top2_v = tmpl_v;
        top_v=tmpl_v;
        while (tmpl_v->xz.x < xmax) {
                if(strcmp(tmpl_v->bMtrl,"AIR")==0) {
                        layer_array[i].x[j] = tmpl_v->xz.x;
                        layer_array[i].z[j++] = tmpl_v->xz.z;
                        strcpy(previous,tmpl_v->aMtrl);
                        Move(&tmpl_v,tmpl_v->aMtrl);
                } else {
                        layer_array[i].x[j] = tmpl_v->xz.x;
                        layer_array[i].z[j++] = tmpl_v->xz.z;
                        if(strcmp(tmpl_v->aMtrl,previous)==0){
                                strcpy(previous,tmpl_v->bMtrl);
                                Move(&tmpl_v,tmpl_v->bMtrl);
                        } else {
                                strcpy(previous,tmpl_v->aMtrl);
                                Move(&tmpl_v,tmpl_v->aMtrl);
                        }
                                }
                if(tmpl_v->xz.z > top_z) {
                                top_z = tmpl_v->xz.z;
                                top_v = tmpl_v;
                        top_polygon = Find_top_poly(top_v);
                        }
        tmpl_polygon = Find_top_poly(tmpl_v);
        if(top_polygon != NULL){
         if(  strcmp(top_polygon->Name,
                        tmpl_polygon->Name)!=0){
                        /* find top of this polygon */
           if(tmpl_v->xz.x > top2_z) {
                                top2_z = tmpl_v->xz.x;
                                top2_v = tmpl_v;
                top2_polygon = Find_top_poly(top2_v);
                                }
           }
        }
                } /*while*/
                /* read in last vertex*/
```

```
                                layer_array[i].x[j] = tmpl_v->xz.x;
                                layer_array[i].z[j++] = tmpl_v->xz.z;

        layer_array[i].number_pts = j   ;

    if(bottom_of_poly(top_polygon) < bottom_of_poly(top2_polygon)){
        top_polygon = top2_polygon;
    }

    strcpy(layer_array[i].layer_type,top_polygon->Name);

        if(strcmp(top_polygon->Name,"SI")==0) {
                                xleft[i]=xmin;
                                xright[i]=xmax;
                                break;
    }
        Find_xmin_and_xmax(top_polygon,&xleft[i],&xright[i],&zleft[i]);
    top_poly_list[i] = top_polygon;
        Delete_Polygon(top_polygon);
        i++;
 }
    number_layers = i+1;
/*      fptr = fopen("temp_save","r");
    Read_Data2(fptr);
    fclose(fptr); */
}
/*********/
float bottom_of_poly(plyptr)
/*********/
POLYGONPTR plyptr;
{
        float bottom;
    VERTEXPTR tmpv;
    int i ;

    if(plyptr == NULL) {
                return(-100.0);
    }
    bottom = 10000.0;
        plyptr->Nvertex = Count_Vertices(plyptr);
        tmpv = plyptr->HeadVertex;
    for(i=1;i< (plyptr->Nvertex); i++){
                if(tmpv->xz.x < bottom) bottom = tmpv->xz.x;
                Move(&tmpv,plyptr->Name);
    }
        return(bottom);
 }

/*********/
POLYGONPTR Find_top_poly(top_v)
/*********/
VERTEXPTR top_v;
{
  POLYGONPTR Find_Polygon(),top_polygon;
  char Material_name[15];

  if  (Other(top_v->aMtrl,"AIR")==TRUE){
     strcpy(Material_name,top_v->aMtrl);
      } else if(Other(top_v->bMtrl,"AIR")==TRUE){
      strcpy(Material_name,top_v->bMtrl);
     } else if((Other(top_v->cMtrl,"AIR")==TRUE)  {
     strcpy(Material_name,top_v->cMtrl);
     } else {
        SimExit("error finding top polygon type!");
    }
 top_polygon = Find_Polygon(top_v,Material_name);
 return(top_polygon);
}
/*********************************/
Write_SAMPLEnplnr()
/*********************************/
{
        int i,j;
        FILE *fp,*fp2;
    float rate,steps,starttime,etchmodel;
    int isteps,istarttime,ietchmodel;
    char askstring[80];

    Get_Layers();
/*********************************************/
/* write SAMPLE input file for nonplanar etch */
/*********************************************/
    if (number_layers > 5){
                GLprintf("Too many layers for this version of SAMPLE");
                return(-1);
    }
    GLprintf("creating nonplanar input file");
        fp = fopen("SAMPLE_nonplanar","w");
    fprintf(fp,"etchnumlay  %d ;\n",number_layers);
        for(i=number_layers-1;i>=0;i--){
    if(i == 0 ) {
    fprintf(fp,"etchprof\n");
    } else {
        fprintf(fp,"nonplanar %d\n",number_layers - i -1 );
    }
        for (j=0;j<layer_array[i].number_pts;j++){
      if(!(j!=0 &&(layer_array[i].x[j]==layer_array[i].x[j-1]
                    && layer_array[i].z[j]==layer_array[i].z[j-1]))){
                fprintf(fp,"%10.6f %10.6f\n",layer_array[i].x[j],
                                  -1.0*(layer_array[i].z[j] -zmax) );
      }
    }
    fprintf(fp," ; \n");
        }
   /* sprintf(askstring,"Etchmodel (see SAMPLE user guide)");
    Ask(askstring);
    Answer_Float_Proc(askstring,&etchmodel);
    ietchmodel = 1*etchmodel;*/
    ietchmodel = 1;
    fprintf(fp,"etchrates %d ",ietchmodel);
        for(i=number_layers-1;i>=0;i--){
        sprintf(askstring,"etchrate for %s, layer %d (um/sec)",
                   layer_array[number_layers-1-i].layer_type,i);
        Ask(askstring);
                Answer_Float_Proc(askstring,&rate);
                fprintf(fp," %10.6f ",rate);
    }
    fprintf(fp," ; \n");
    fprintf(fp,"etchplot 1 0 0;\n");
    fprintf(fp,"etchwindow %10.6f ;\n",xmax-xmin);
        fprintf(fp,"etchaccur 3 ;\n");
    sprintf(askstring,"timestep in seconds");
    Ask(askstring);
        Answer_Float_Proc(askstring,&starttime);
        sprintf(askstring,"number of steps");
        Ask(askstring);
        Answer_Float_Proc(askstring,&steps);
    isteps = 1*steps;
        istarttime = 1*starttime ;
           fprintf(fp,"etchtime %d %d, %d;\n",istarttime,
                  istarttime*isteps,isteps);
        fprintf(fp,"etchrun ;\n");
        fprintf(fp,"end ;\n");
           fclose(fp);
        GLprintf("SAMPLE_nonplanar written");
}
/************/
Find_xmin_and_xmax(poly_ptr,xleft,xright,zleft)
/************/

   /**********
   Routine to find the leftmost point and right most
   point of a polygon and return the x-coordinate values
   of those vertexes
*************/
struct Polygon *poly_ptr;
float *xleft,*xright,*zleft;
{
        struct Vertex *tempVertex;

        *xleft = 10000.0;
    *xright = 0.0;
    *zleft = 0.0;

        if (poly_ptr->HeadVertex != NULL) {
                        tempVertex = poly_ptr->HeadVertex;
                        do {
                                if(Move(&tempVertex, poly_ptr->Name)==0);
                                if(tempVertex->xz.x < *xleft){
                                    *xleft = tempVertex->xz.x;
                    if(tempVertex->xz.z > *zleft) {
                                *zleft = tempVertex->xz.z;
                                    }
             }
                                if(tempVertex->xz.x > *xright){
                                    *xright = tempVertex->xz.x;
             }
           } while(tempVertex != poly_ptr->HeadVertex);
     }
}
```

Appendix C

Catalogue of useful C functions in SIMPL-DIX and SIMPL-2 listed according to their use.

Catalogue of useful C functions in SIMPL-2

This compilation lists several functions in SIMPL-2 which are useful for designing interfaces with other programs. A brief summary of the purpose of each function is listed along with its location in the SIMPL-2 source code. The version heading indicates whether a function is include in the release version (R), or was developed subsequently by Alex Wong (ASW) or by Edward Scheckler (EWS). For a complete description of the function consult the comments in the source listing for SIMPL-2. The functions listed here are grouped in the following categories:

Traverse a string, locate something in the data base
Alter the data base
Test for a condition
Utility functions
Prompts and communication with user
Load/Save

Function Name	**Location**	**Version**
Traverse, locate		
Get_LeftTop() Returns a pointer to the vertex at the top left of the profile.	String.c	R
Get_RightTop() Returns a pointer to a vertex at the top right of the profile.	String.c	R
Find_Polygon(Vertex,Name) Returns a pointer to a polygon of a given name which contains the specified vertex.	String.c	R
Move(Vertex,Material) Moves to the next vertex on the polygon with the given material name.(Moves clockwise).	String.c	R
Move_Backward(Vertex,Material) Moves counterclockwise around the polygon to next vertex with given material name.	String.c	R
Get_LeftPoly() Returns pointer to the top left polygon.	String2.c	ASW
Get_RightPoly() Returns pointer to the top right polygon.	String2.c	ASW
Find_Last_Polygon() Returns pointer too the last polygon in linked list data structure.	String3.c	ASW
Find_top_poly(Vertex) Returns pointer to the polygon with the given vertex as long as one of the material names stored in that vertex is AIR	F77Layers.c	EWS

Get_Layers()	Extracts profile and stores it in an array as a series of layers.	F77Layers.c	EWS
Find_xmin_and_xmax(Polygon,xl,xr,zl)	-Finds x and z coordinate of leftmost vertex and x coordinate of rightmost vertex	F77Layers.c	EWS

Alter Data Base

Delete_Polygon(Polygon)	Removes polygon from data base.	String.c	R
Separate_Polygon(Polygon)	Removes polygon pointer without changing the linked vertexes.	String.c	R
Insert_Polygon(Polygon1,Polygon2)	Inserts a polygon node Polygon1 following the node Polygon2	String.c	R
Append_Polygon(Polygon)	Append Polygon to last polygon in data structure.	String.c	R
Polyize_Air()	Makes AIR into a polygon	String3.c	ASW
Rename_Polygon(old_name,name,Vertex)	Renames a polygon	String3.c	ASW
Merge_Poly()	Merges all polygons which touch and are of same material type.	String3.c	ASW
Del_Dup_Polygon()	Deletes duplicate polygons.	String3.c	ASW
One_D_Search_or_Insert(Vertex,name,position)	Search for a vertex with given x position if not found, insert a vertex.	String.c	R
Two_D_Search_or_Insert(Vertex,x1,z1,x2,z2,p,q,name)	This function scans through the edges between vertex p and q following 'name' and finds the crossing point with the line segment (x1,z1),(x2,z2).	String.c	R
Insert(Vertex1,Vertex2,Vertex3,name)	Inserts Vertex1 between Vertex2 and Vertex3 following name.	String.c	R
Link(Vertex1,name,Vertex2)	link Vertex pointer 1 with 'name' to Vertex pointer 2.	String.c	R

Delete_Link(Vertex,name)	String2.c	ASW
Deletes a link to a vertex of given 'name'		
Delete_LinktoV(Vertex1,Vertex2)	String3.c	ASW
Deletes a link between vertex 1 and 2.		
AIRize_String(Vertex1,Vertex2,name)	String.c	R
assigns AIR to vertices along a string.		
Write_a_AIR(vertex)	String.c	R
Fills vertex material names with "AIR"		
Delete_Mtrl(name,Vertex)	String.c	R
Set pointer at a give name in Vertex to NULL		
Add_Mtrl(Vertex1,Vertex2,name)	String2.c	ASW
Set pointer in Vertex 1 to point to Vertex 2 with given name.		
Substitute(Vertex,name1,name2)	String.c	R
Change name of pointer in Vertex.		
Rename(Vertex1,Vertex2,name,name)	String.c	R
Rename the string between Vertex 1 and Vertex 2.		
New_Vertex()	String2.c	ASW
Returns a new vertex with null pointers .		
New_Polygon()	String2.c	ASW
Returns a new polygon with null pointers		
Copy_Vertex(Vertex1,Vertex2)	String2.c	ASW
Copies a vertex.		
Del_BadHeadVertex(array_of_vertices,integer)	String3.c	ASW
Cleans up a list of vertices with a bad head vertex.		
Redefine_Profile()	F77Layers.c	EWS
Takes last contour in SAMPLE_netchf77, and uses it to update the profile after a nonplanar etch simulation. Not yet fully implemented.		

Test for information

Count_Vertices(Polygon)	String.c	R
Counts the number of vertices in a polygon		
Count_Polygons(Root_Polygon)	String.c	R
Counts the number of polygons in the data base		
Which_Node(Vertex1,Vertex2,name)	String.c	R
Find pointer in vertex containing 'name'		
Inside_Polygon(flag,Polygon,x,z)	String.c	R

Determines if point is inside given polygon		
Is_There(Name,Vertex) Checks if there is 'name' in vertex.	String.c	R
Material(Vertex,name) Checks if there is an adjacent material which is not AIR or BNDR	String.c	R
Other(name1,name2) Checks if material name is other then nil, BNDR or AIR	String.c	R
Verify(Polygon,name,error) Traces through the polygon to test if the polygon is well formed. Error string set.	String.c	R
Full(Vertex) Returns TRUE if vertex pointers are all assigned	String.c	R
Thickness(Vertex,name) Returns floating point thickness of a polygon	String.c	R
Test_Segment(Vertex1,Vertex2,name)	String2.c	ASW
Test_Connect(Vertex1,Vertex2)	String2.c	ASW
Test_Connect_Mtrl(Vertex1,Vertex2,name)	String2.c	ASW
Two_Same(Vertex) Test if there are two materials of the same name in a vertex.	String3.c	ASW
Empty_Vertex(Vertex) Tests if vertex is empty	String3.c	ASW
Test_Cyclic(Vertex1,Vertex2,name)	String3.c	ASW
Utility		
Free_Polygon(Polygon)	String.c	R
Free_Vertex(Vertex)	String.c	R
Free_Mask(Mask)	String.c	R
Free_Block(Block)	String.c	R
Prompts		
GLprintf(char_string) Prints a character string to the display	Graphics_Utils.c	R
Ask(char_string)	Graphics_Utils.c	R

Used to generate a prompt to the user.

Answer_Float(request,answer) Get answer from user, convert to floating point	Misc.c	R
Answer_Up_Proc(request,answer) - Get answer string from user, convert to upper case	Misc.c	R
Answer_Integerrequest,answer) Get answer from user, convert to integer	Misc.c	R

Load/Save

Write_Savefile() Saves data base in a file	Write_Data.c	R
Read_Data() Loads data base from a file	Read_Data.c	R
Write_Savefile(fp) Save data in file indicated by file pointer fp.	Write_Data.c	EWS
Read_Data2(fp) Load data from file indeicated by file pointer fp.	Read_Data.c	EWS

A Few Useful Routines and Global Variables in SIMPL-DIX

GetYesOrNo(message) Prompts the user to click Yes or No with the mouse.	prompt_control.c	R
GetProfileViewX(viewport,profile,x) Used to translate x coordinate to an integer describing position in the viewport	view_control.c	R
GetProfileViewY(viewport,profile,y) Similar to above	view_control.c	R
PrintText(variable list)	graphics_control.c	R
SIMPL_PolygonRt Global variable for start of linked list containing profile data in SIMPL-DIX		

SIMPL-2 source files which were altered for this project:

Deposition.c
Do_Process.c
Etching.c
Read_Data.c
Write_Data.c
F77Layers.c (new file)

SIMPL-DIX source files which were altered for this project:

command_control.c
dix_main.c
dix_action4.c (new file)