SIMPL(SIMULATED PROFILES FROM THE LAYOUT)

by

M. A. Grimm

Memorandum No. UCB/ERL M83/79

14 December 1983

SIMPL(SIMULATED PROFILES FROM THE LAYOUT)

by

M. A. Grimm

Memorandum No. UCB/ERL M83/79

14 December 1983

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Michael A. Grimm

Author

SIMPL(SIMulated Profiles from the Layout)

Title

RESEARCH PROJECT

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, in partial satisfaction of the requirements for the degree of Master of Science, Plan II.

Approval for the Report and Comprehensive Examination:

Committee: Andrew R. Neureuther, Research Advisor

December 14, 1983 Date

W. G. Oldham

Dec 14, 1983 Date

SIMPL (Version 1.0 Dec 12, 1983)

SIMulated Profiles from the Layout-version 1.0

Developed by :
Michael Grimm
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720

[]

SIMPL
Table of Contents

[]

Abstract

Often times it is necessary to visualize an IC profile from looking at the layout. The transformation from the planar view to the cross-sectional view is difficult, especially with complex mask sets. SIMPL (SIMulated Profiles from the Layout) is a new CAD tool which automatically generates the cross-section of an integrated circuit from the layout. SIMPL has several uses. First, it provides the designer visual feedback on the structures which will be fabricated. Second, it provides information about critical design rules. Finally, since SIMPL can give visual feedback at every step in the process, it is an aid to process development.

SIMPL uses simplified models and rectangular approximation to the structures in order the run very quickly. The models are not analytic as in other rigorous two dimensional simulators such as SAMPLE or SUPRA, but this allows SIMPL to run much faster. Research is continuing on a second version of SIMPL which will automatically give more detailed topographical and electrical information from the layout. This will be accomplished by linking other process simulators such as SAMPLE, SUPRA and MINIMOS.

SIMPL is written in 1600 lines of C code. Typical CPU time for the final profile of a CMOS inverter or a bipolar transistor in approximately 5 seconds on a VAX-11/780.

[]

SIMPL Report

Simulation Program

SIMPL(SIMulated Profiles from the Layout - version 1) is a new CAD tool which automatically generates the cross-section of an IC from the layout. This gives the designer rapid feedback on the circuit topology. The layout is taken from a KIC or CIF(CalTech Intermediate Form) file. The cross-section is simulated at an arbitrary cut-plane specified by user. The graphic output is also in KIC so that it can be displayed on the same graphics editor as the layout. The input to SIMPL consists of generalized fabrication steps which can be arranged arbitrarily to describe many different processes. SIMPL is designed to be fast, and uses only rectangular shapes trading off accuracy for speed, which is desirable for on-line operation.

SIMPL takes three types of inputs, one from the user, one from the layout and one from the process file. The user specifies two points which define the cross-section cut-plane and the scale factor in the Y direction. The Y scale factor makes the output more readable, compensating for the small dimensions in the vertical plane of an IC. SIMPL takes the user defined cross-section plane and intersects it with the layout to get the mask crossings. The program then executes the commands in the process file with the proper mask edges and simulates the wafer topography.

A flexible grid structure is used to reduce the number of rectangle required to represent a cross-section. The process begins with two rectangles, the substrate and the air above, which are as large as the desired cross-section. As the wafer is processed, more rectangles are introduces to represent mask edges and layers in the profile.

SIMPL is written in 1600 lines of C code. The C language utilizes data structures and pointers to variables, which are integral parts of the grid. Each rectangle is represented by a data structure(node), which contains the physical information about each rectangle. The node stores the material type, doping concentration, and the x,y coordinates of the upper left corner and the lower right corner. In addition every node has a pointer to its four neighbors. These pointers allow searching movement in any direction from any node.

Splitting the array as required by new mask edges or process effects can be done in both the x and the y direction. The split is generated by inserting a new row of boxes and setting the pointers and x,y coordinates accordingly. The pointers are superior to a two dimensional array, since only three rows of boxes are affected in a split. If it was an array, then all of the array below or to the right of the split would be affected.

All operations on the profile are broken down into two simple operations: splitting boxes and switching node type and doping. During every step the position of the next rectangle if

calculated, the grid is split to accommodate the new box, and it's type and doping are set to their new values. Quite often, the rectangle already exists in the grid and only the type and doping need to be updated. Some of the grid lines become obsolete after some time. In order to minimize the grid and the run time the grid is checked for grid lines that are not necessary.

For speed the physical process models in SIMPL have been made very elementary. At the moment lateral etching and diffusion are not included. The doping model is a gaussian with variable peak depth, standard deviation, and implant blocking thickness. SIMPL does not simulate a redistribution of the doping profile in later high temperature steps, therefor the input values must include the drive-in component. The oxidation model input is the oxide thickness if there were no initial oxide. The new oxide thickness is the square root of the sum of the squares of the initial oxide thickness and the input oxide thickness.

A six mask bipolar process with 38 process steps and an eight mask CMOS process with 55 steps have been simulated. Execution time which occurs for generating the final profile directly requires only 5 sec. on a VAX 11-780/UNIX.

[]

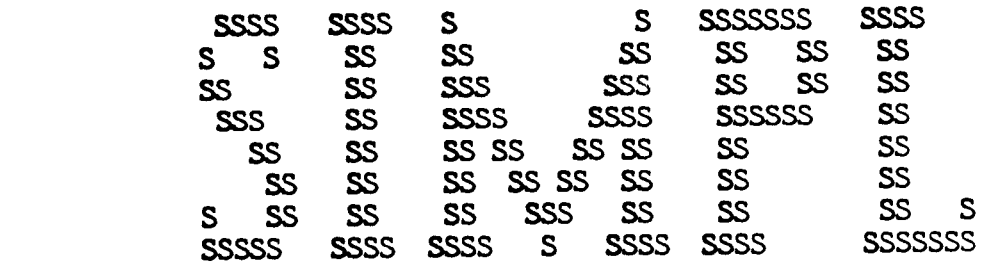
ACKNOWLEDGEMENTS

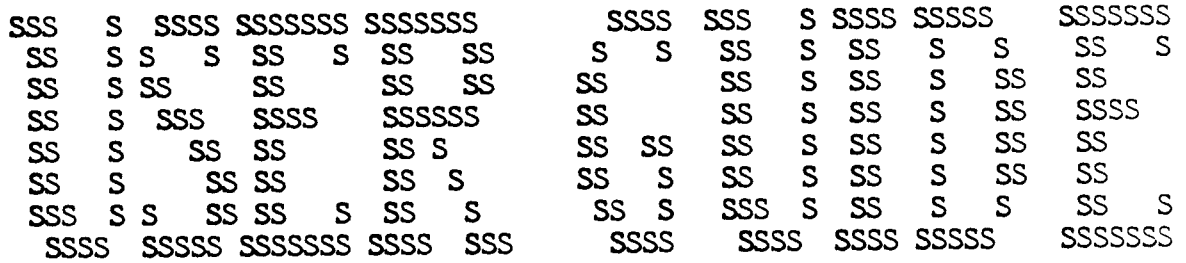
I would like to express my thanks to Prof. Andy Neureuther for his time and patience throughout this project. He has been of the utmost help in completing this work. I would also like to than my partner and friend Ken Lee for his valuable assistance. My thanks are also extended to Prof. Bill Oldham for reading the paper. I would like to acknowledge Ken Keller for his assistance with KIC. I would also like to thank my wife Brigit for her support and patience.

A very special thanks is extended to all those at the University of California, Berkeley who have made my stay here, both as a graduate and an undergraduate, very memorable and rewarding.

GO BEARS!!!

[]





SIMPL (Version 1.0 Dec 12, 1983)

SIMulated Profiles from the Layout-version 1.0

Developed by :
Michael Grimm, Keunmyung Lee, Andrew Neureuther
Electronics Research Laboratory
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720

[]

The SAMPLE Group
Room 332 Cory Hall
Electronics Research Laboratory
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720 U.S.A.

This project has been supported by DARPA.

[]

SIMPL USERS GUIDE
Table of Contents

page #

OVERVIEW

SIMPL is a CAD(Computer Aided Design) tool which simulates the topography of an Integrated Circuit from the layout. It is capable of simulating the cross-section at every step of the wafer fabrication. The basic process steps are oxidation, deposition, etching, implantation and diffusion, and masking. The profiles are modelede with rectangles using simple physical model. This allows rapid online feedback on the device features which the layout will generate.

SIMPL presently consists of about 1600 lines of C code. Typical CPU time is approximately 5 seconds on a VAX-11/780 with UNIX to simulate the final profile for a cmos inverter. The layouts and profiles can be viewed with the KIC program, while the data is stored in CIF(CalTech Intermediate Form) which is a layout standard.

The development of SIMPL has been supported by the DARPA. To encourage open exchange of information SIMPL is available on an as-is basis for a small handling fee. We appreciate your feedback on the performance of the program. However, we can not offer assistance in implementing the program on your computer system or in training users.

[]

How to tell SIMPL to simulate:
(A note on the types of input statements to SIMPL.)

The SIMPL program imitates the actual processing sequence of a wafer. The process flow closely follows the actual sequence used in the lab. Steps which do not have an effect on the topography such as rinse, prebake, and sintering are not simulated.

There are three types of inputs to SIMPL, general information, the process flow and the layout.

The user begins by entering several inputs in interactive mode. The inputs describe the files needed to run the simulation, the segment of the layout to be simulated and some scale factors.

The first input file is the layout command. The layout can be either CIF or KIC(it is assumed that the user is familiar with KIC, see Appendix A for details on the KIC program). This file is a description of the layout which will be simulated.

The other input file contains a list of the process commands. Every processing input into the program conveys the information necessary to perform one step of the process. The input contains the command and the parameters which characterize a specific action, as if it were being performed in the lab. The parameters describe the topographical effects of the action, rather than the settings on the machines which perform the actions. This means that the user specifies the oxide thickness rather than the time and temperature of the oxidation. Therefore the user must have previously characterized the process before he can use that information to visually see the profiles. The details of each input statement are given later.

The inputs commands can be used in an arbitrary sequence, except for substrate definition. This feature allows SIMPL to simulate many different technologies such as NMOS, PMOS, CMOS or bipolar. Here is a list of the current process commands which make up the process sequence:

SUBS - specify the substrate material
OXID - oxidize the silicon
DEPO - deposit a layer onto the surface
ETCH - selectively etch the surface
DOPE - dope the silicon with implant or diffusion
MASK - photo transfer the mask to the resist

[]

A Short Example

This example consists of three parts 1) interactive inputs 2) process file 3) layout file. The following examples assume a familiarity with UNIX. These examples can be found on ESVAX in ~sam3/simpl/example. Move to this directory to run this example. To start, alias simpl to ~sam3/simpl/prog/simpl. Then give the command "simpl".
The commands are,

```
alias simpl ~sam3/simpl/prog/simpl
cd ~sam3/simpl/example
simpl
```

SIMPL will ask for some more information in an interactive mode.
The responses to the interactive questions are given:

QUESTION	ANSWER
split h=horizontally v=vertically?	h
microns per lambda?	1.0
start value?	-50
stop value?	50
scale factor in y direction?	10
the y plane you want split?	0
the mask file?	ex.mask
the input file?	ex.process
the base name for the output files?	ex.out
output profile at every step y=yes n=no?	n

The meaning of the interactive inputs are,

split h=horizontally v=vertically?
: is the cut-line segment of the mask horizontal or vertical?

microns per lambda?
: the number of microns per one lambda of KIC units

start value?
: the starting coordinate of the cut-line

stop value?
: the final coordinate of the cut-line

scale factor in y direction?
: a factor which magnifies the profile in the vertical plane to make the very small dimensions visible

the (x,y) plane you want split?
: this specifies the coordinate of the plane which the cut line lies on

the mask file?
     file which contains the mask layout

the input file?
     file which contains the process inputs

the base name for the output files?
     this gives the base name for the output files

output profile at every step y=yes n=no?
     specify a profile for every step of the process

The process file contains a simple input-example illustrating the use of SIMPL. First the example input file is given and then its meaning is explained. The example covers all the simple statements that SIMPL uses.

SIMPL does not put any restrictions the columns in which the parameters appear, but the parameters must be in the correct order. No extraneous characters are allowed inside the command lines except at the end of a line. Comments can be placed at the end of a command line, or on a separate line with the "C" comment command. Blank lines are not allowed.

The example process input file called ex.process:

```
C
C input process file example
C this example implants a N-WELL into a P substrate
C
SUBS P 1e14 comment at end of a line!
OXID 1.0
DEPO RST 0.5
MASK NEG NWEL 0 0
ETCH ERST 0.6
ETCH OX 1.1
ETCH RST 0.6
DOPE N 1.0e16 0.5 1.0 0.5
```

The meaning of the statements is as follows

C
  comments may be added with this command

SUBS P 1e14
  form the substrate with P type material of 1e14 doping

OXID 1.0
  oxidize the silicon to 1.0micron

```
DEPO RST 0.5
  deposit 0.5 micron of resist(RST)

MASK POS NWEL 0 0
  photomask the resist with the NWEL mask, over etch
  and misalignment are null

ETCH ERST 0.6
  etch exposed resist(ERST), this patterns the resist

ETCH OX 1.1
  etch or 1.1microns of oxide, this only occurs in the
  regions where the resist has been opened

ETCH RST 0.6
  etch off all the remaining resist

DOPE N 1.0e16 0.5 1.0 0.5
  dope the silicon with N type impurities with Nmax=1e16,
  sigma for the gaussian distribution is 0.5micron, depth
  of the peak is 1.0micron below the surface, and the
  masking thickness necessary to block the doping
  in unwanted areas is 0.5micron
```

The mask file is called ex.mask and is the layout file for the graphics system. In this example a KIC file is used, but the CIF format of the same file can be used. This is a very small file with only one box which lies in the plane that is split.

```
KIC file                  comments
--------------------      --------------------
(Symbol ex.mask);         header
9 ex.mask;
DS 0 1 1;
L NWEL;                   Layer name is NWEL
B 4400 2400 -2200 0;      box which forms the mask
DF;                       end of data
E
```

```
CIF file for comparison
-----------------------------------------------
(CIF file of symbol hierarchy rooted at ex.mask);
DS 1 1 1;
9 ex.mask;
L NWEL;
B 4400 2400 -2200 0;
DF;
C 1;
E
```

When SIMPL is run with the test example above, the following are the results to expect.
The output to the terminal

OUTPUT	COMMENT
split h=horizontally v=vertically?	interactive inputs
h	
microns per lambda?	
1.0	
start value?	
-50	
stop value?	
50	
scale factor in y direction?	
10	
the y plane you want split?	
0	
the mask file?	
ex.mask	
the input file?	
ex.process	
the base name for the output files?	
ex.out	
output profile at every step y=yes n=no?	
n	
NWEL	mask name
-44000 0	mask crossings
	commands

```
C
C comments may be added with this command
C this example implants a N-WELL into a P substrate
C
#01  SUBS  TYPE=P  DOPE=1.00e+14
#02  OXID  THICK=1.00
#03  DEPO  TYPE=RST  DOPE=0.00e+00  THICK=0.50
#04  MASK  POL=NEG  MASK=NWEL  BLOAT=0.00  SHIFT=0.00
#05  ETCH  TYPE=ERST  THICK=0.60
#06  ETCH  TYPE=OX  THICK=1.1
#07  ETCH  TYPE=RST  THICK=0.60
#08  DOPE  TYPE=N  PEAK=1.00e+16  DEPTH=1.00  DELTA=0.500  BLOCK=0.50
```

across and down 3 18	diagnostics
memory used = 2408	

The output file, ex.out, is a KIC file which can be viewed

by running KIC with this file. The output file is in ascii format and can be viewed. Note that is is in the same format as ex.mask. These two files can be viewed at the same time using KIC.

```
(Symbol ex.out);
9 ex.out;
DS 0 1 1;
L OX;
B 5000 1002 7500 10041;
B 600 1002 300 10041;
L N12;
B 4400 2 2800 8505;
L N13;
B 4400 14 2800 8513;
L N14;
B 4400 72 2800 8556;
B 4400 52 2800 9514;
L N15;
B 4400 178 2800 8681;
B 4400 178 2800 9399;
L N16;
B 4400 540 2800 9040;
L P13;
B 4400 2 2800 8503;
L P14;
B 5000 9540 7500 4770;
B 4400 8502 2800 4251;
B 600 9540 300 4770;
L LABL;
94 final_profile 0 0;
B 10000 20000 5000 10000;
DF;
E
```

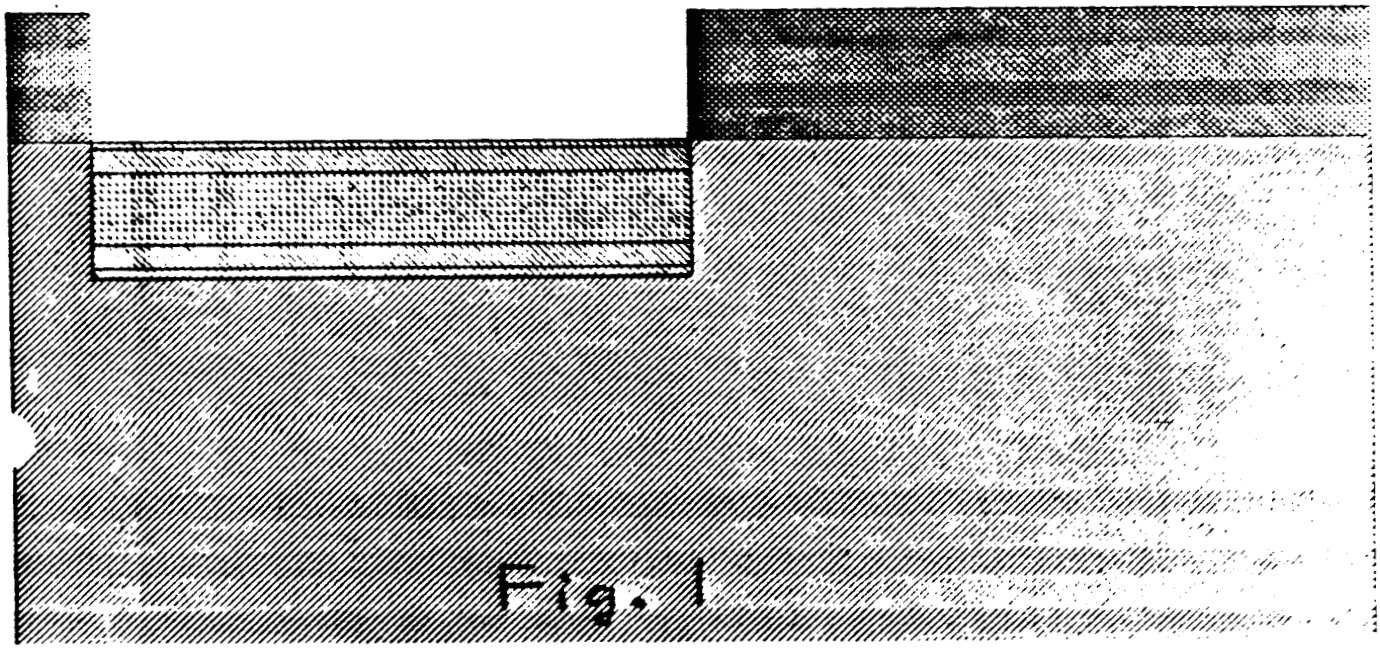


Fig. 1

SIMPL Commands

Introduction

This section of the user manual catalogs both the interactive inputs and process input command statements that are presently contained in SIMPL. The first two lists are a terse summary of the inputs and their arguments for quick reference. Detailed descriptions of each statement follow next.

[]

LIST OF INTERACTIVE INPUTS
(in sequential order)

split h=horizontally v=vertically?
microns per lambda?
start value?
stop value?
scale factor in y direction?
the y plane you want split?
the mask file?
the input file?
the base name for the output files?
output profile at every step y=yes n=no?

LIST OF PROCESS INPUTS
User process commands

C comments may be added with this command
SUBS TYPE DOPE
OXID THICK
DEPO TYPE DOPE THICK
MASK POL MASK BLOAT SHIFT
ETCH TYPE THICK
DOPE TYPE PEAK DEPTH DELTA BLOCK
ETCH TYPE THICK

[]

INTERACTIVE INPUTS
DETAILED DESCRIPTION

split h=horizontally v=vertically?

The cross-section is simulated along a line segment of the layout file. This segment can lie anywhere in the layout, but must be oriented in the horizontal or vertical direction. Horizontal means the the start and stop points have the same y coordinate, and vertical means that the start and stop points have the same x coordinate. The input is only valid if 'h' or 'v' is entered.

microns per lambda?

The layout is in KIC format with relative units called lambda which form the coordinate grid of the layout. One lambda represents a specific number(or fraction) of microns. The scale factor is a floating point number.

start value? stop value?

The start and stop values are integers which are the starting coordinates for the cut-line segment. The values are specified in lambda. These values are obtained while viewing the layout on the KIC terminal.

scale factor in the y direction?

This scale factor is an integer which makes the cross-section easier to view. Since the vertical dimensions of Integrated Circuits are very small(gate oxide ~25nm) compared to the lateral dimensions(gate length ~1micron) it is very hard to see the vertical features in a 1:1 scale. The scale factor magnifies the vertical dimension.

the (x,y) plane you want split?

The cut-line lies in the horizontal or vertical plane. If it is split horizontally then it is necessary to specify the y coordinate of the plane which is split. If the split is vertical then the x coordinate of the plane must be specified. The coordinate is an integer number in lambda.

the mask file?

The mask file is the name of the file which contains the layout. The file must be in the current directory.

the process input file?

The process input file is the name of the file which contains the process input commands This file must be in the current directory.

the base name for the output files?

The base name is the name for the final profile. If the profile for every step is output, the the files are named basename.xx where xx is the process step number.

output profile at every step y=yes n=no?

This input allows the user to specify if just the final profile or if a profile for every step is created. The names of the steps follow the convention stated in base name command.

[]

PROCESS INPUTS
DETAILED DESCRIPTION

GENERAL USAGE

The process inputs follow a few simpl conventions. These will be outlined before the commands are specified in detail.

Several commands specify a material type to be operated upon. There is a fixed list of layers which may be used. When the material is doped silicon, the doping must be added in addition to the type(P,N). The list of layer names follow

layer	comment
POLY	poly silicon
OX	oxide
METL	metal
RST	resist
ERST	exposed resist
NTRD	nitride
SI	intrinsic silicon
N	n-type silicon, doping must be specified
P	p-type silicon, doping must be specified
L1	undefined layers
L2	
L3	
L4	
L5	

The parameters for thickness, depth, delta and sigma are all floating point numbers in microns. The values for doping are floating point numbers of atoms per centimeter cubed.

DETAILED COMMAND LIST

C comments may be added with this command

The comment card allows comments to be interspersed with the process commands. No blank lines are allowed.

SUBS TYPE DOPE

This command forms the substrate material. It must be the first non-comment input line and can only be used once. TYPE specifies the material used and DOPE is the doping which is only specified it the TYPE is N or P.

OXID THICK

The oxidation is characterized by the thickness of the new

oxide. This thickness refers to the thickness which would grow on bare silicon. There is a correction factor which accounts for initial oxide, this is the square root of the sum of the squares of initial thickness plus new thickness. The model also accounts for the 46% consumption of the silicon.

DEPO TYPE DOPE THICK

The deposition command deposits a material on the surface of the wafer. The material is specified by TYPE and the optional doping. The layer thickness is specified by THICK.

MASK POL MASK BLOAT SHIFT

The MASK command exposes the resist with the specified MASK. The POL term specifies the polarity of the mask, if the polarity is positive(POS) then the mask is used to image the resist, or if the polarity is negative(NEG) then the inverse of the mask is used to expose the resist. The BLOAT command allows the user to increase the size of the mask to approximate lateral effects. If BLOAT is a positive number then the features on the layout are expanded by this amount. Bloat is a floating point number and can be negative to simulate a shrink. SHIFT is also a floating point number which moves the mask to the right by the specified distance.

ETCH TYPE THICK

The etch command is selective to the specified TYPE. The etch will proceed to a depth of THICK. If the material is thinner than THICK then the layer will be removed completely, with the lower layers unaffected.

DOPE TYPE PEAK DEPTH DELTA BLOCK

The DOPE command simulates either the implant or diffusion process. This command does not calculate a redistribution of the impurities, so the input parameters must also take the drive-in into consideration. The impurity profile for each DOPE command is approximated by a gaussian curve. The gaussian has a peak doping(PEAK), depth below the surface(DEPTH), sigma of the curve(DELTA), and a critical thickness for implants (BLOCK). BLOCK is the masking thickness necessary to keep the impurities from entering the silicon. PEAK is a floating point number of the doping concentration peak in atoms per centimeter cubed. DEPTH, DELTA and BLOCK are both floating point numbers in microns. The impurity concentration of a region is calculated as the linear sum of the gaussians plus substrate concentration.

[]

The SIMPL program's structure

This section of the SIMPL user's guide contains information for those who want to modify, customize or become more familiar with the code. The casual user may not wish to read the contents following. The following sections will outline the internal structure of the program and give some ideas on modifying the code.

[]

Subroutine	file
alloc(n)	alloc.c
boxes(line,node)	boxes.c
checkacross()	check.c
dep(layer,doping,t)	dep.c
diff(nmax,alpha,offset,ycrit)	diff.c
diffdown(nmax,alpha,node,ypeak)	diff.c
diffup(nmax,alpha,node,ypeak)	diff.c
dopinglevel(npeak)	dop.c
doptolayer(doping)	dop.c
edge()	edge.c
error(message)	inputs.c
etch(layer,t)	etch.c
exinit()	exinit.c
expose(start,stop)	expose.c
gauss(nmax,alpha,x)	gauss.c
gettype(line,type,typenum,doping)	inputs.c
init()	init.c
inputs()	inputs.c
interact()	interact.c
kalloc()	alloc.c
kicfile(num,label)	kicfile.c
kiclayers(fp)	kiclayers.c
label_(label,line)	inputs.c
main()	main.c
maskname(line)	maskname.c
mem()	alloc.c
nalloc()	alloc.c
newboxx(start,new,north,xval)	newbox.c
newboxy(start,new,west,yval)	newbox.c
nextsplit(npeak,nmax)	next.c
nodelayer(node)	node.c
numacross()	num.c
numdown()	num.c
numtotype(i)	exinit.c
oxide(depth)	ox.c
pairs(node,high,low)	pairs.c
palloc()	alloc.c
photo(polarity,name,bloat,shift)	photo.c
printnode(high,low,fp)	printnode.c
printpairs()	printpairs.c
rectangle(x1,x2,y1,y2,node)	rect.c
searcheaxt(startnode,xval)	search.c
searchsouth(startnode,yval)	search.c
sgn(n)	sgn.c
splitx(xval)	split.c
splity(yval)	split.c
sub(n,d)	sub.c
talloc()	alloc.c
typetonum(type)	exinit.c
validox(node)	valid.c
wires(line,node)	wires.c

alloc:

boxes:
  rectangle

check:
  nodelayer

dep:
  splity

diff:
  diffdown
  diffup

diffdown:
  dopinglevel
  gauss
  nextsplit
  sgn
  splity

diffup:
  dopinglevel
  gauss
  nextsplit
  sgn
  splity

dopinglevel:

doptolayer:

edge:
  boxes
  maskhome
  maskname
  nalloc
  printpairs
  wires

error:

etch:
  splity

exinit:

expose:
  splitx

gauss:

inputs:
  checkacross
  dep
  diff
  error
  etch
  gettype
  kicfile
  label_
  oxide
  photo
  sub

interact:

kalloc:
  alloc

kicfile:
  kiclayers

kiclayers:
  kalloc
  nodelayer
  numtotype
  printnode

label_:

main:
  edge
  exinit
  init
  inputs
  interact
  mem
  numacross
  numdown

maskname:
  nalloc

mem:

nalloc:
  alloc

newboxx:
  talloc

newboxy:
  talloc

numacross:

numdown:

numtotype:

ox:
  splity
  validox

pairs:
  palloc

palloc:
  alloc

photo:
  expose

printnode:

printpairs:

rect:
  pairs

searcheast:

searchsouth:

sgn:

splitx:
  newboxx
  searcheast
  talloc

splity:
  newboxy
  searchsouth
  talloc

sub:
  splity

talloc:
  alloc

typetonum:

valid:

wires:

gettype:
  error
  typetonum

init:
  talloc

next:
  sgn

node:
  doptolayer

rectangle

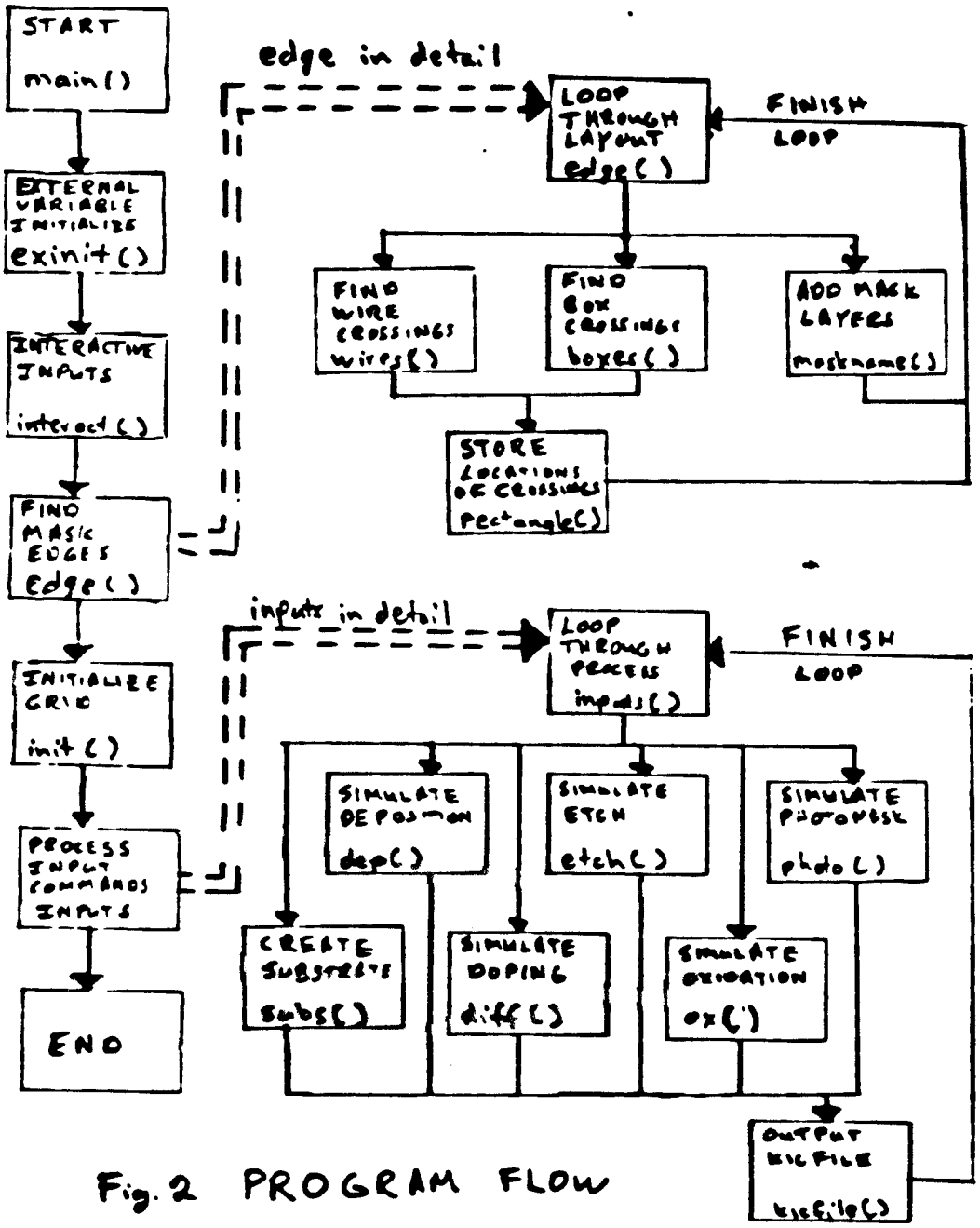


Fig. 2 PROGRAM FLOW

Internal Structure of SIMPL

A flexible grid structure is used to reduce the number of rectangles required to represent a cross-section. The process begins with two rectangles, the substrate and the air above, which are as large as the desired cross-section. As the wafer is processed, more rectangles are introduces to represent mask edges and layers in the profile as shown in Fig.[3]. Note that the newly introduced grid lines cross the entire cross-section. A CMOS inverter with 4 steps requires about 3000 rectangles.

SIMPL-1 is written in 1500 lines of C code. The C language utilizes data structures and pointers to variables, which are integral parts of the grid. Each rectangle is represented by a data structure(node), which contains the physical information about each rectangle, shown in Fig.[4]. The node stores the material type, doping concentration, and the x,y coordinates of the upper left corner and the lower right corner. In addition every node has a pointer to its four neighbors. These pointers allow searching movement in any direction from any node.

Splitting the array as required by new mask edges or process effects can be done in both the x and the y direction. The split is generated by inserting a new row of boxes and setting the pointers and x,y coordinates accordingly as demonstrated if Fig.[5].

All operations on the profile are broken down into two simple operations: splitting boxes and switching node type and doping. During every step the position of the next rectangle if calculated, the grid is split to accommodate the new box, and it's type and doping are set to their new values. Quite often, the rectangle already exists in the grid and only the type and doping need to be updated.

[]

Program data structure and grid

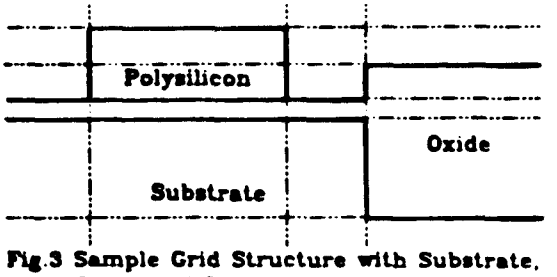


Fig.3 Sample Grid Structure with Substrate, Oxide and Polysilicon

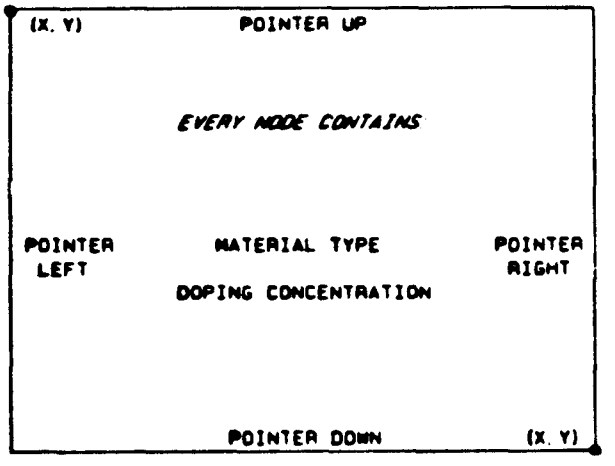


Fig.4 Contents of Data Node for Rectangles

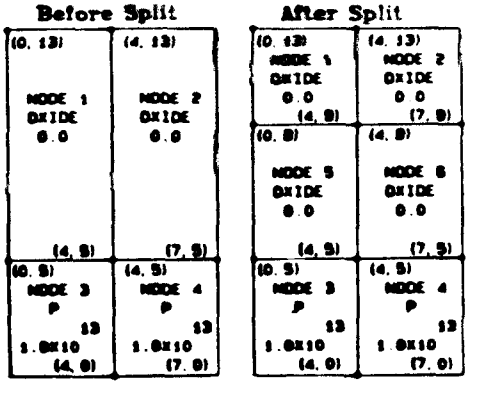


Fig.5 Horizontal Splitting Example on Y=9

HOW TO COMPILE SIMPL

SIMPL is compiled with the aid of a makefile. The MAKE program keeps track of which files which have been updated and will only recompile those. The source code for simpl is located on ESVAX in ~sam3/simpl/prog. When in this directory it is only necessary to type "make" and the make program will compile all new files. The executable code is then put into a file called "simpl". If it is necessary to recompile the whole program after a change in the operating system, then you must first remove all the files which end in ".o". This is done by entering "rm *.o". Then recompile using the "make" program. The updated executable is always put in "simpl".

[]

HOW TO DEFINE NEW LAYERS

SIMPL has five layers which do not have a material name attached to them. They are called L1,L2,L3,L4 and L5. The user is free to use these at his disposal. These layers do not have any special properties such as SI, POLY, OX, RST or ERST. These layers can be used as masks for doping or etching, they can not be oxidized or doped, and they can not be exposed to ERST in a masking step. They can be used as interconnect, insulator or a nitride like material for local oxidation.

The names of these layers can also be changed to reflect the name of the material which they represent. The name convention is one to four characters, starting with at least one capital letter and optionally followed by numbers. This convention is not enforced in SIMPL, but it is enforced on the KIC program. The code which contains the list of layer names is located in the file ~sam3/simpl/prog/exinit.c. In this file, the names can be changed. The program then should be recompiled as in the instructions on the previous page.

If more materials are needed they can be added to the end of the list of materials in the code. It is imperative that the value of NLAYER(number of layers) be updated to reflect the additions. NLAYER is found in the same file as the layer list, but inside the subroutine exinit(). The program must then be compiled.

Layers with special properties can not be added simply. If the layer has new properties, then the new routines which simulate the the property must be added. If the layer has the same properties as one of the other special layers, then the code must be updated to make checks for the new layer.

[]

HOW TO INCREASE MEMORY ALLOCATION

The memory allocation of SIMPL is fixed in size. It is currently set at 500k bytes. The buffer is used to store the nodes of the grid and to produce output files. When all of the buffer is used, SIMPL gives an error message telling you to reduce the complexity of the simulation. If all the steps are being output it is possible to to reduce the buffer requirements by only outputting the final profile. The buffer size can be changed to reflect the needs of the users and the machine.

The location of the allocation is in the file ~sam3/simpl/prog/alloc.c. The buffer size is set by ALLOCSIZE in the definitions. This value can be changed to reflect the desired size. The program must be recompiled after any change. Compiling instructions are given earlier in this manual.

[]

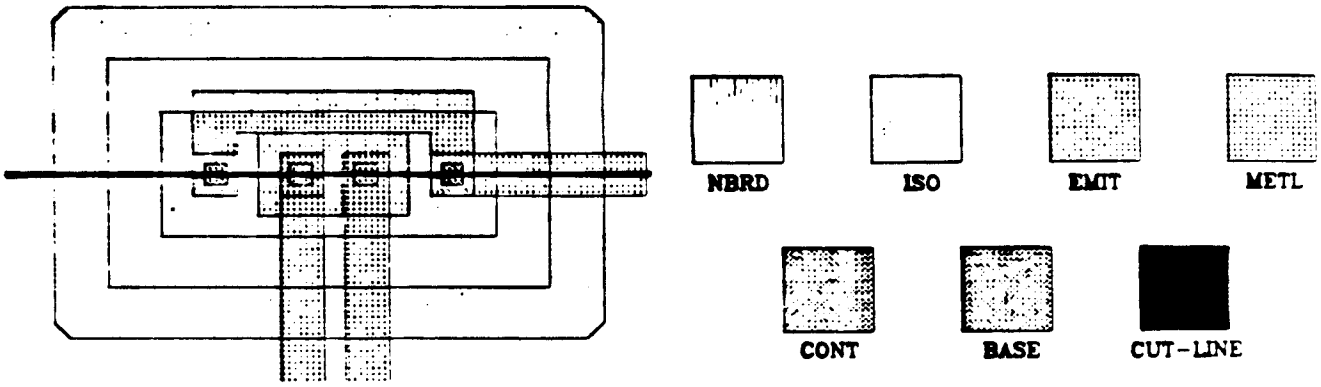


Fig.6 Bipolar Transistor Layout with Cut-line for SIMPL-1

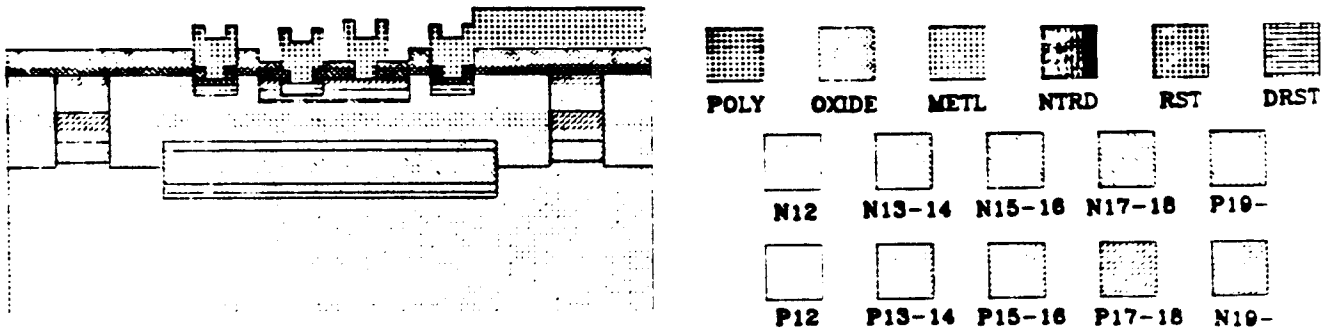


Fig.7 Cross-section of Bipolar along the Cut-Line in Fig.6
5X Scale for Y

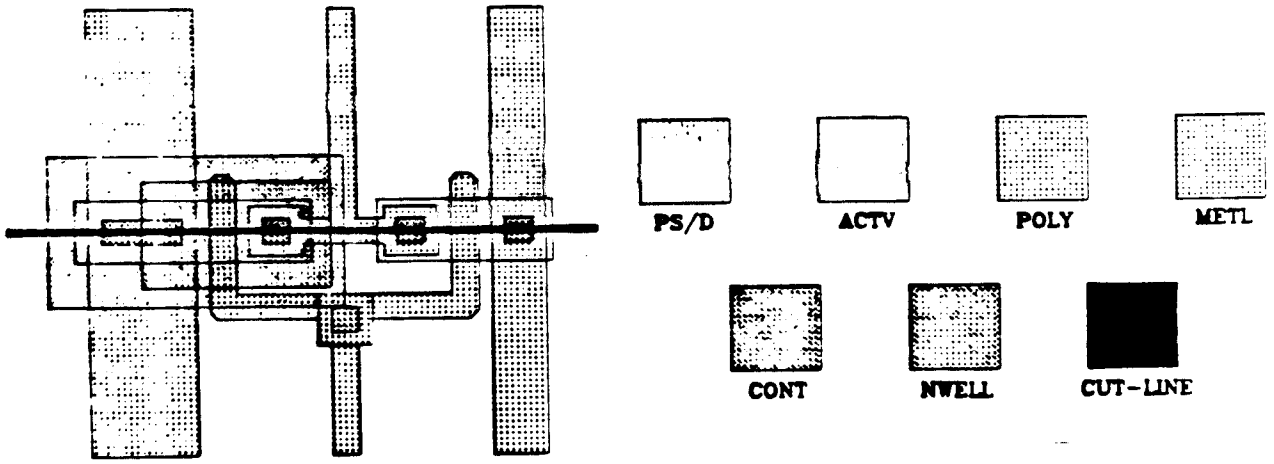


Fig.8 Layout of an Berkeley CMOS Inverter with Cut-line for SIMPL-1

```
1     SUBS P 1e13
2     OXID 0.1
3     DEPO NTRD 0.1
4     DEPO RST 0.5
5     MASK POS NWEL 0 0
6     ETCH ERST 0.6
7     ETCH NTRD 0.2
8     ETCH RST 0.6
9     OXID 0.5
10    ETCH NTRD 0.2
11    DOPE N 1.1e15 1.5 0 0.2
12    ETCH OX 0.7
13    OXID 0.1
14    DEPO NTRD 0.1
15    DEPO RST 0.5
16    MASK POS ACTV 0 0
17    ETCH ERST 0.6
18    ETCH NTRD 0.2
19    ETCH RST 0.6
20    DEPO RST 1.0
21    MASK POS NWEL 0 0
22    ETCH ERST 1.1
23    DOPE P 1e21 0.15 0.05 0.2
24    ETCH RST 1.1
25    OXID 0.7
26    ETCH NTRD 0.2
27    DOPE P 1e20 0.5 0 0.2
28    DEPO POLY 0.25
29    DEPO RST 0.5
30    MASK POS POLY 0 0
31    ETCH ERST 0.6
32    ETCH POLY 0.6
33    ETCH RST 0.6
34    DEPO RST 1.0
35    MASK POS PSD 0 0
36    ETCH ERST 1.1
37    DOPE N 1e21 0.2 0.1 0.2
38    ETCH RST 1.1
39    DEPO RST 1.0
40    MASK NEG PSD 0 0
41    ETCH ERST 1.1
42    DOPE P 1e21 0.2 0.1 0.2
43    ETCH RST 1.1
44    DEPO OX 0.5
45    DEPO RST 1.0
46    MASK NEG CONT 0 0
47    ETCH ERST 1.1
48    ETCH OX 1.0
49    ETCH RST 1.1
50    DEPO METL 1.0
51    DEPO RST 1.0
52    MASK POS MTL 0 0
53    ETCH ERST 1.1
54    ETCH METL 1.1
55    ETCH RST 1.1
```

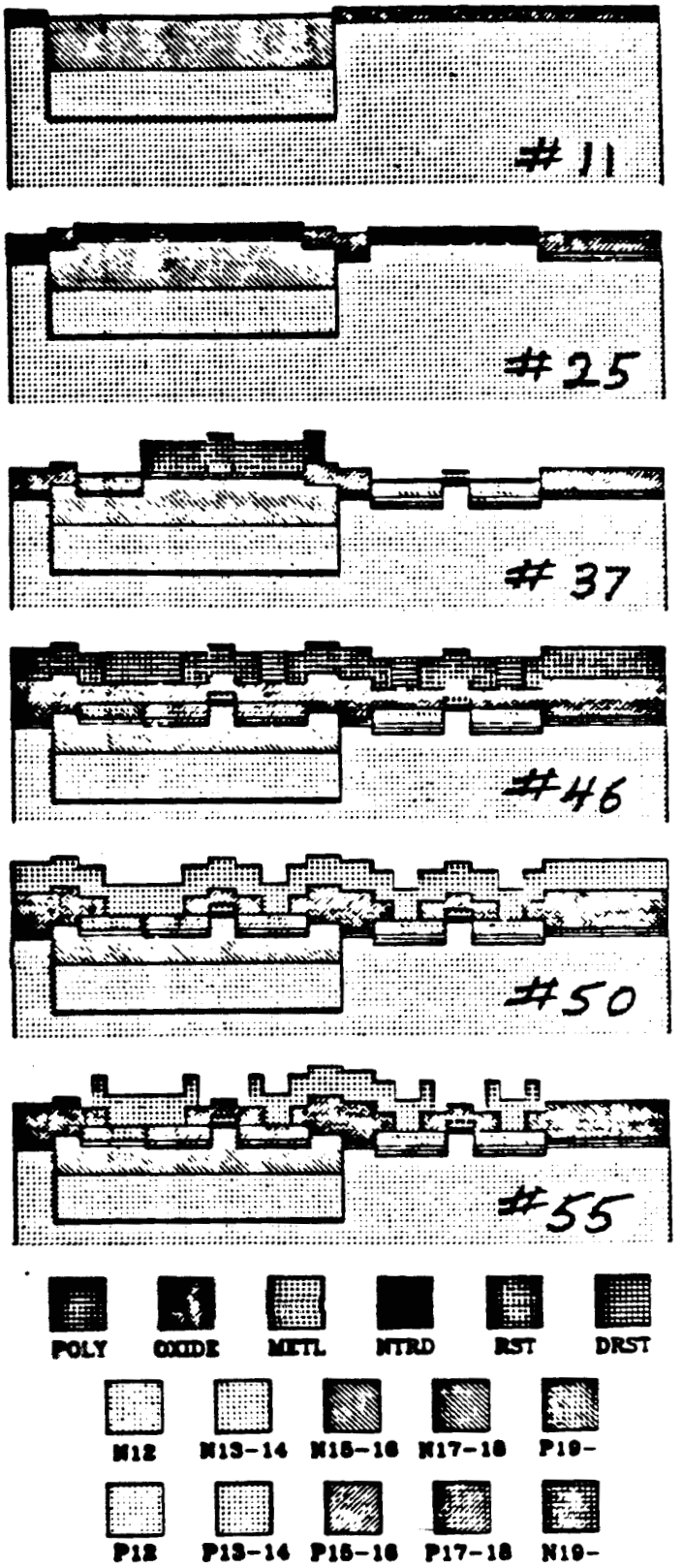


Fig.9 Process Input and Profiles of Berkeley CMOS Inverter along the Cut-line in Fig.8. 5X Scale for Y.

KIC notes

This section is designed to provide some help getting started with KIC. KIC is a computer graphics package to aid in circuit layout which has been developed at U. C. Berkeley. The program uses a data base very similar to CIF(CalTech Intermediate Form) which is a layout standard. The layout files which are created using KIC are interpreted by SIMPL to find the mask crossings. The output files which are created bye SIMPL are compatible with KIC.

SIMPL can not interpret all of the KIC extensions. The layout must contain only rectangles and wires, no circles or polygons are interpreted. Also there can not be any heirarchy in the layout. If the circuit has been laid out with instances, it must be flattened to be fully interpreted.

When KIC is run, it requires a file in the present directory which gives the characteristics of all the layers. This file is the ".KIC" file. There is an example in the directory ~sam3/simpl/example.

The KIC files can be converted to CIF format by running the KICTOCIF program. Once in CIF format they can be plotted on the Versatec printer using the CIFPLOT command. CIFPLOT uses a pattern file which contains the pattern for each layer, an example is also in ~sam3/simpl/example.

[]

SOURCE CODE FOR SIMPL

[]

```
CFLAGS = -cvgp
LFLAGS = -gp

OBJECTS = alloc.o boxes.o check.o dep.o\
    diff.o dop.o edge.o etch.o exinit.o\
    expose.o gauss.o init.o inputs.o interact.o\
    kicfile.o kiclayers.o main.o maskname.o\
    newbox.o next.o node.o num.o ox.o\
    pairs.o photo.o printnode.o printpairs.o rect.o\
    search.o sgn.o split.o\
    sub.o valid.o wires.o

simpl : $(OBJECTS)
    cc $(LFLAGS) $(OBJECTS) -o $@ -lm

.c.o: ; cc $(CFLAGS) $*.c
```

```
struct tnode {
/* node format */
/* north west x coordinate, north west y coordinate, south east x */
/* coordinate, south east y coordinate, material type, pointer up, */
/* pointer down, pointer left, pointer right */
/* doping is positive for n and neg for p  and 0 for non Si */
/* when a pointer is NULL, there are no more boxes in that direction */
/* the limit for x,y coordinates is set to LIMIT in microns */
    int nwx;
    int nwy;
    int sex;
    int sey;
    int type;
    float doping;
    struct tnode *north;
    struct tnode *south;
    struct tnode *west;
    struct tnode *east;
};

struct knode {
/* node format */
/* node  points to the tnode */
/* prev  points to the previous node */
    struct tnode *high;
    struct tnode *low;
    struct knode *prev;
};

struct nnode {
/* name holds the mask name, first holds the first node */
/* nextmask points to the next mask in the list */
    char name[10];
    struct pnode *first;
    struct nnode *nextmask;
};

struct pnode {
/* this is the node which nnode points to */
/* start contains the start of a box and stop is the stop */
/* next is the next pair */
    int start;
    int stop;
    struct pnode *nextpair;
};
```

```
#include "defs"

#define ALLOCSIZE 500000

static char allocbuf[ALLOCSIZE];
static char *allocp = allocbuf;

mem()
{
printf("memory used = %d\n",allocp -allocbuf);
}

char *alloc(n)
/* this routine allocates space for n characters */
/* it returns a pointer to available space or 0 when not enough space */
int n;
{
if(allocp + n <= allocbuf + ALLOCSIZE){
    allocp += n;
    return(allocp - n);
    }
else{
    printf("not enough memory-decrease size of profile\n");
    exit(1);
    }
}

struct tnode *talloc()
/* this routine allocates space for the structure tnode */
{
char *alloc();
return((struct tnode *) alloc(sizeof(struct tnode)));
}

struct knode *kalloc()
/* this routine allocates space for the structure knode */
{
char *alloc();
return((struct knode *) alloc(sizeof(struct knode)));
}

struct pnode *palloc()
/* this routine allocates space for the structure pnode */
{
char *alloc();
return((struct pnode *) alloc(sizeof(struct pnode)));
}

struct nnode *nalloc()
/* this routine allocates space for the structure nnode */
{
char *alloc();
return((struct nnode *) alloc(sizeof(struct nnode)));
}
```

```
#include <stdio.h>
#include "defs"

boxes(line,node)
/* this routine takes a cif line and determines the coordinates */
/* of the edge of the box */
/* the line is in the form of "B delta_x delta_y x_avg y_avg;" */
char line[];
struct nnode *node;
{
int x,y,delx,dely,x1,y1,x2,y2;
char box[10];

/* get arguements */
if(sscanf(line,"%s %d %d %d %d",box,&delx,&dely,&x,&y) != 5)
    printf("wrong num of arguments\n");
else{
    /* calculate edges */
    x1 = x - delx/2;  /* left edge */
    x2 = x1 + delx;  /* right edge */
    y1 = y - dely/2;  /* bottom edge */
    y2 = y1 + dely;  /* top edge */
    /* check to see if this box is in the cross-section */
    rectangle(x1,x2,y1,y2,node);
    }
}
```

```
#include "defs"

checkacross()
/* this routine checks to see if there are any horizontal splits */
/* that are not needed i.e. each box above the split is the same */
/* the box below the split */
{
extern struct tnode *home;
struct tnode *box, *left, *north;

left = home; /* start in the upper left corner */
/* loop throught the rows */
while((left = left->south) != 0){
    box = left;
    /* loop through the columns as long as there is a match */
    while(nodelayer(box) == nodelayer(box->north)){
        box = box->east;
        /* if we reached the end or the row and all matched then */
        /* remove the split */
        if(box == 0){
            /* removing a row! */
            box = left;
            /* loop coulmns to eliminate split */
            while(box != 0){
                /* adjust the pointers */
                north = box->north;
                north->sey = box->sey;
                if(box->south == 0)
                    north->south = 0;
                else{
                    north->south = box->south;
                    box->south->north = north;
                    }
                box = box->east;
                }
            break;
            }
        }
    }
}
```

```
#include "defs"

dep(layer,doping,t)
/* deposit a material characterized by layer, doping and thickness */
int layer,t;
double doping;

{
extern struct tnode *home;
struct tnode *top,*box;
int surf;

/* start in upper left */
top = home;
/* loop through the coulmns */
while(top != 0){
    /* move through the air boxes */
    box = top;
    while(box->type == 0){
        box = box->south;
        if(box == 0){
            printf("never found the surface in dep()\n");
            exit(1);
            }
        }
    surf = box->nwy + t; /* new surface = old surface + thickness */
    splity(surf); /* split at the new surface */
    box = box->north;
    /* loop upwards and change the air boxes to the new material */
    while((box->nwy) <= surf){
        box->type = layer;
        box->doping = doping;
        box = box->north;
        }
    top = top->east;
    }
}
```

```
#include <math.h>
#include "defs"

diff(nmax,alpha,offset,ycrit)
/* nmax >0 if n type and <0 if p type */
/* diff includes both diffusion and ion implants */
/* the doping profile is specified by a gaussian */
/* with a characeristic width(alpha),peak, and offset below the surf */
/* ycrit is the critical thickness which the implant can not pass */
/* since the impurities do not diffuse in later steps */
/* the characteristic values must be adjusted to reflect the profile */
/* after all high temp steps */
double alpha,nmax;
int ycrit,offset;
{

int ysurf,thick,ypeak;
extern struct tnode *home;
struct tnode *surf,*top;
extern int SI;

top = home; /* start in the upper left corner */
/* loop across the columns */
while(top != 0){
    surf = top;
    /* loop through air boxes */
    while(surf->type == 0)
        surf = surf->south;
    ysurf = surf->nwy; /* surface of the wafer */
    /* determine thickness on non silicon */
    while(surf->type != SI){
        surf = surf->south;
        if(surf == 0)
            break;
        }
    /* if surf = 0 then there was no silicon in that column */
    if(surf == 0)
        break;
    thick = ysurf - surf->nwy; /* calculate thickness on non SI */
    /* if thick is > ycrit then no implant gets into the SI so exit */
    if(thick <= ycrit){
        if(thick >= offset) /* if thick>offset then only diffuse down */
            diffdown(nmax,alpha,surf,ysurf-offset);
        else{
            /* diffuse both up and down around ypeak */
            ypeak = ysurf - offset;
            splity(ypeak);
            while(surf->sey >= ypeak)
                surf = surf->south;
            diffdown(nmax,alpha,surf,ypeak);
            surf = surf->north;
            diffup(nmax,alpha,surf,ypeak);
            }
        }
    top = top->east;
    }
```

```
}

diffdown(nmax,alpha,node,ypeak)
/* diffusion of impurities into SI */
/* this is a half gaussian with the peak towards the top of the SI */
double alpha,nmax;
int ypeak;
struct tnode *node;

{
double nmin,npeak,nsplit,nsub,ngauss,sgn();
double fabs(),log(),sqrt(),dopinglevel(),nextsplit(),gauss();
int ysplit,ygauss;
extern int SI;

nmin = 1e10; /* minimum discrete doping */
/* loop as long as in SI */
while(node->type == SI){
    nsub = node->doping; /* initial doping */
    ygauss = ypeak - node->nwy; /* gaussian distance */
    npeak = gauss(nmax,alpha,ygauss) + nsub; /* doping at ygauss */
    nsplit = dopinglevel(npeak); /* calc the discrete doping level */
    ngauss = nsplit - nsub; /* calc the new gaussian value */
    /* if the gaussian value is small then exit */
    if(fabs(ngauss) < nmin)
        break;
    if(fabs(ngauss) <= fabs(nmax)){
        ygauss = sqrt(alpha*alpha*log(fabs(nmax/ngauss)));
        while((ysplit = ypeak - ygauss) >= node->sey){
            if(ysplit != node->nwy){
                splity(ysplit);
                node->doping = 3*nsplit;
                node = node->south;
            }
            nsplit = nextsplit(nsplit,nmax);
            ngauss = nsplit - nsub;
            if(fabs(ngauss) < nmin || sgn(ngauss) != sgn(nmax))
                break;
            ygauss = sqrt(alpha*alpha*log(fabs(nmax/ngauss)));
            }
        if(fabs(ngauss) < nmin || sgn(ngauss) != sgn(nmax))
            break;
        if(ysplit < node->sey)
            node->doping = 3*nsplit;
        }
    node = node->south;
    if(node == 0)
        break;
    }
}

diffup(nmax,alpha,node,ypeak)
/* diffusion of impurities into SI */
/* this is a half gaussian with the peak towards the bottom of the SI */
double alpha,nmax;
int ypeak;
```

```
struct tnode *node;

{
double nmin,npeak,nsplit,nsub,ngauss,sgn();
double fabs(),log(),sqrt(),dopinglevel(),nextsplit(),gauss();
int ysplit,ygauss;
extern int SI;

nmin = 1e10; /* minimum discrete doping */
while(node->type == SI){
    nsub = node->doping;
    ygauss = node->sey - ypeak;
    npeak = gauss(nmax,alpha,ygauss) + nsub;
    nsplit = dopinglevel(npeak);
    ngauss = nsplit - nsub;
    if(fabs(ngauss) < nmin)
        break;
    if(fabs(ngauss) <= fabs(nmax)){
        ygauss = sqrt(alpha*alpha*log(fabs(nmax/ngauss)));
        while((ysplit = ypeak + ygauss) <= node->nwy){
            if(ysplit != node->sey){
                splity(ysplit);
                node = node->south;
                node->doping = 3*nsplit;
                node = node->north;
            }
            nsplit = nextsplit(nsplit,nmax);
            ngauss = nsplit - nsub;
            if(fabs(ngauss) < nmin || sgn(ngauss) != sgn(nmax))
                break;
            ygauss = sqrt(alpha*alpha*log(fabs(nmax/ngauss)));
            }
        if(fabs(ngauss) < nmin || sgn(ngauss) != sgn(nmax))
            break;
        if(ysplit > node->nwy){
            node->doping = 3*nsplit;
            }
        }
    node = node->north;
    if(node == 0)
        break;
    }
}
```

```
#include <math.h>

doptolayer(doping)
/* converts a float doping into a layer number for kic output */
/* the doping levels are divide into groups by the log of doping */
double doping;
{
double t,log10(),fabs();
int layer;
extern int SI;

/* take the log of the doping pluss 0.5 round off factor*/
if(doping == 0)
    t = 0;
else
    t = log10(fabs(doping)) + 0.5;
if(t > 20.0)
    t = 20;
/* if the log doping is less than 12 then call it intrinsic */
/* otherwise calculate the layer number for that doping */
if(t < 12.0)
    layer = SI;
else if(doping > 0)
    layer = SI - 11 + t;
else if(doping < 0)
    layer = SI - 2 + t;
return(layer);
}

double dopinglevel(npeak)
/* calc. the discrete doping level for any doping level */
/* the doping of 10ex includes doping levels from 3e(x-1) to 3ex */
/* min. doping is 1e12 */
double npeak;

{
double abpeak,sgn(),fabs();
double nhigh = 3e19;
double nlow = 3e11;

abpeak = fabs(npeak);
/* if doping is too low then set it to intrinsic */
/* otherwise find discrete level */
if(abpeak < nlow)
    nhigh = 0;
else {
    while(nhigh > abpeak)
        nhigh /= 10.0;
    }
if(npeak < 0)
    nhigh *= -1;
return(nhigh);
}
```

```
#include "defs"
#include <stdio.h>

#define MAXLINE 200

edge()
/* find the mask edges in the appropriate area and return inputfile */
{
extern struct nnode *maskhome;
extern char *MASKFILE[];
struct nnode *maskname(), *nalloc(), *node;
char line[MAXLINE];
FILE *fp,*fopen();

node = 0;
/* the mask crossings are stored in a pointer structure */
/* every mask layer has its own pointer called node */
maskhome = nalloc();
maskhome->nextmask = 0;
fp = fopen(MASKFILE,"r"); /* look in maskfile for cif layout */
if(fp == 0){
    printf("maskfile(%s) doesn't exist!\n",MASKFILE);
    exit(1);/* exit if the mask file doesn't exist */
    }
/* only boxes and rectangles of flattened CIF */
while(fgets(line,MAXLINE,fp) != 0){
    if(line[0] == 'B')
        boxes(line,node);
    if(line[0] == 'L')
        node = maskname(line);
    if(line[0] == 'W')
        wires(line,node);
    }
fclose(fp);
printpairs();
}
```

```
#include "defs"

etch(layer,t)
/* etch a specific layer to a max thickness t */
int layer;
int t;
{
extern struct tnode *home;
struct tnode *top,*box;
int depth;

top = home; /* start in upper left */
/* loop columns */
while(top !=0){
    box = top;
        /* loop through air */
        while(box->type == 0)
            box=box->south;
        depth = box->nwy - t; /* final depth after max etch */
        splity(depth); /* split at depth */
        /* loop boxes while the type is right and not too deep */
        while(box->sey >= depth && box->type == layer){
            /* switch type to air */
            box->type = 0;
            box->doping = 0;
            box = box->south;
            if(box == 0)
                printf("etch all the way to the bottom\n");
            }
    top = top->east;
    }
}
```

```
#include <stdio.h>

struct nnode *maskhome; /* mask is the pointer to the first mask */
struct tnode *home;

char PRINTALL;/* n-only final step output  y-all steps output */
char DIR;/* h-horizontal split of layout v-vertical */
/* filenames */
char *MASKFILE[20],*INPUTFILE[20],OUTPUTFILE[20];
int PLANE; /* vlaue of the plane in the layout being split */
int LOWER,UPPER; /* values for the crossection window */
int YLIMIT; /* height of profiles */
int YSCALE;/* scale factor for y dimensions */
int NLAYER;/* total number of layers */
int SI; /* layer number */
int POLY; /* layer number */
int OX; /* layer number */
int RST; /* layer number */
int ERST; /* layer number */
float MICPERLAM;/* microns per lambda */


static char *layers[] = {
    "AIR",
    "POLY",
    "OX",
    "METL",
    "RST",
    "ERST",
    "NTRD",
    "SI",
    "N12",
    "N13",
    "N14",
    "N15",
    "N16",
    "N17",
    "N18",
    "N19",
    "N20",
    "P12",
    "P13",
    "P14",
    "P15",
    "P16",
    "P17",
    "P18",
    "P19",
    "P20",
    "L1", /* user defined layers, do not change the layers */
    "L2", /* above this point, you may charge the names of */
    "L3", /* these layers L1 - L5 and recompile */
    "L4", /* note any changes in the number of layers in NLAYER */
    "L5", /* below in exinit() */
    };
```

```
exinit()
{
extern int YLIMIT,NLAYER,SI,OX,RST,ERST;

YLIMIT = 10000; /* limit hieght to +- 10microns */
NLAYER = 30;/* total number of layers */
SI = 7; /* SI layer is #7 */
OX = 2; /* OX layer is #2 */
RST = 4; /* RST layer is #4 */
ERST = 5; /* ERST layer is #5 */
POLY = 1; /* POLY layer is #1 */
}

typetonum(type)
/* convert a text layer to its number */
char type[];
{
extern int NLAYER;

int n = 0;
while(strcmp(type,layers[n]) != 0){
    n++;
    if(n == NLAYER){
        printf("***** invalid layer(%s) *****\n",type);
        exit(1);
        }
    }
return(n);
}

char *numtotype(i)
/* convert type number to string */
int i;
{
return(layers[i]);
}
```

```
#include "defs"

expose(start,stop)
/* expose the resist in the region between start and stop */
int start,stop;
{
extern int RST,ERST;
extern struct tnode *home;
struct tnode *top,*box;

/* split vertically at start and stop */
splitx(start);
splitx(stop);
top = home;
/* move over to the first coulmn */
while(top->sex <= start )
    top = top->east;
/* loop coulmns till we move past stop */
while(top != 0){
    if(top->sex <= stop){
        box = top;
        /* loop through air */
        while(box->type == 0)
            box = box->south;
        /* loop through resist and convert it to exposed resist */
        while(box->type == RST){
            box->type = ERST;
            box = box->south;
            }
        }
    top = top->east;
    }
}
```

```
#include <math.h>

double gauss(nmax,alpha,x)
/* perform the function n = nmax(-(x*x)/(alpha*alpha)) */
double nmax,alpha;
int x;
{
double n,n1;

n1 = x/alpha;
n = nmax*exp(- n1*n1);
return(n);
}
```



```
#include "defs"

init()
/* initialize the array to one rectangle of maximum dimensions */
{
extern struct tnode *home;
extern int LOWER,UPPER,YLIMIT;
struct tnode *talloc();

home = talloc(); /* create home - upper left corner of grid */
/* the empty rectangle is set to the maximum dimension of the grid */
/* LOWER & UPPER are the min and max lateral dimensions of the grid */
/* YLIMIT is the max vertical dimension */
home->nwx = LOWER;
home->nwy = YLIMIT;
home->sex = UPPER;
home->sey = - YLIMIT;
home->type = 0; /* type 0 is air */
home->doping = 0;
/* pointer are null - no boxes in those directions */
home->north = 0;
home->south = 0;
home->west = 0;
home->east = 0;
}
```

```
#include <stdio.h>
#define MAXLINE 200

inputs()
/* read the process inputs from the process input file */
{
extern int YSCALE,SI;
extern char PRINTALL,*INPUTFILE[];
int typenum,thick,d,ycrit,offset,num,bloat,shift,valid;
char comm[20],type[20],*labl[MAXLINE],*line[MAXLINE],pol[10],name[10];
double doping,t,y1,off1,bloat1,shift1,fabs();
FILE *fp,*fopen();

num = 1;/* number for kic file */
if((fp = fopen(INPUTFILE,"r")) == 0){
    printf("inputfile(%s) doesn't exist!\n",INPUTFILE);
    exit(1);/* exit if input file doesn't exist */
    }
while(fgets(line,MAXLINE,fp) != 0){
    valid = 1;/* assume that the command is valid */
    strcpy(labl,line);
    if(sscanf(line,"%s%[^\0]",comm,line) < 1){ /* get command */
        printf("%s",labl);
        error("no command given in this line ");
        }
    if(num == 1 && strcmp(comm,"SUBS") != 0 && strcmp(comm,"C") != 0)
        error("SUBS must be the first non comment command");
    if(strcmp(comm,"SUBS") == 0){
        /* set the substrate type,(doping) */
        if(num != 1)
            error("SUBS can only be the first command");
        gettype(line,type,&typenum,&doping);
        sprintf(labl,"#%02d  SUBS  TYPE=%s  DOPE=%.2e\n",
                                            num,type,fabs(doping));
        printf("%s",labl);
        label_(labl);
        sub(typenum,doping);
        }
    else if(strcmp(comm,"DEPO") == 0){
        /* deposit type,(doping),thickness */
        gettype(line,type,&typenum,&doping);
        if(sscanf(line,"%f",&t) < 1)
            error("thickness not found");
        thick = 1000*t;
        sprintf(labl,"#%02d  DEPO  TYPE=%s  DOPE=%.2e  THICK=%.2f\n",
                                            num,type,fabs(doping),t);
        printf("%s",labl);
        label_(labl);
        dep(typenum,doping,thick);
        }
    else if(strcmp(comm,"DOPE") == 0){
        /* diffuse doping,charcteristic length,offset,block depth */
        gettype(line,type,&typenum,&doping);
        ycrit = 0;
        if(sscanf(line,"%f %f %f",&t,&off1,&y1) < 3)
            error("not enough inputs on this line");
```

```
        sprintf(labl,
"#%02d  DOPE  TYPE=%s  PEAK=%.2e  DEPTH=%.2f  DELTA=%.3f  BLOCK=%.2f\n",
                                        num,type,fabs(doping),off1,t,y1);
        printf("%s",labl);
        t = 1000*t;
        offset = 1000*off1;
        ycrit = 1000*y1;
        label_(labl);
        diff(doping,t,offset,ycrit);
        }
    else if(strcmp(comm,"ETCH") == 0){
        /* etch - type,thickness */
        if(sscanf(line,"%s %f",type,&t) < 2 )
            error("not enough inputs on this line");
        typenum = typetonum(type);
        thick = 1000*t;
        sprintf(labl,"#%02d  ETCH  TYPE=%s  THICK=%.2f\n",num,type,t);
        printf("%s",labl);
        label_(labl);
        etch(typenum,thick);
        }
    else if(strcmp(comm,"OXID") == 0){
        /* oxidation - thickness for bare silicon */
        if(sscanf(line,"%f",&t) < 1)
            error("thickness not found");
        thick = 1000*t;
        sprintf(labl,"#%02d  OXID  THICK=%.2f\n",num,t);
        printf("%s",labl);
        label_(labl);
        oxide(thick);
        checkacross();
        }
    else if(strcmp(comm,"MASK") == 0){
        /* photolithography - polarity, maskname */
        if(sscanf(line,"%s %s %f %f",pol,name,&bloat1,&shift1) < 4)
            error("not enough inputs on this line");
        bloat = bloat1 * 1000;
        shift = shift1 * 1000;
        sprintf(labl,"#%02d  MASK  POL=%s  MASK=%s  BLOAT=%.2f  SHIFT=%.2f\n",
                                        num,pol,name,bloat1,shift1);
        printf("%s",labl);
        label_(labl);
        photo(pol,name,bloat,shift);
        }
    else if(strcmp(comm,"C") == 0){
        num--;
        printf("%s",labl);
        valid = 0;
        }
    else{
        printf("illegal input command --- %s\n",line);
        valid = 0;
        }
    if(PRINTALL == 'y' && valid == 1)
        kicfile(num,labl);
    num++;
```

```
    }
sprintf(labl,"final_profile");
num = 0;
kicfile(num,labl);/* always output final profile */
}

gettype(line,type,typenum,doping)
/* get the type and doping of the input string */
char line[],type[20];
int *typenum;
double *doping;
{

*doping = 0;
if(sscanf(line,"%s%[^\0]",type,line) < 1)
    error("type not found in this line");
if(strcmp(type,"N") == 0 || strcmp(type,"P") == 0){
    if(sscanf(line,"%e%[^\0]",doping,line) < 2)
        error("doping not found in this line");
    if(type[0] == 'P')
        *doping = -*doping;
    *typenum = SI;
    }
else
    *typenum = typetonum(type);
}

error(message)
/* print out an error message and abor */
char message[40];
{
printf("%s\n",message);
exit(1);
}

label_(label,line)
/* convert the spaces in label to _ for kic files */
char *label,*line;
{
while(*label != '\n'){
    if(*label == ' ') /* replace ' ' with '_' */
        *label = '_';
    label++;
    }
*label = '\0';
}
```

```
#include <stdio.h>
interact()
/* get interactive inputs */
{
extern int LOWER,UPPER,YSCALE,PLANE;
extern char DIR,PRINTALL,MASKFILE[],INPUTFILE[],OUTPUTFILE[];
extern double MICPERLAM;
int start,stop;
int temp;
/* check for horizontal or vertical split of layout */
printf("split h=horizontally  v=vertically?\n");
while((DIR= getchar()) != 'h' && DIR != 'v')
    printf("invalid! input h or v!\n");
printf("microns per lambda?\n");
while(scanf("%f",&MICPERLAM) < 1)
    printf("invalid! MICPERLAM must be positive!\n");
printf("start value?\n"); /* start value for layout split */
while(scanf("%d",&start) < 1)
    printf("invalid! input an integer!\n");
printf("stop value?\n"); /* stop value for layout split */
while(scanf("%d",&stop) < 1)
    printf("invalid! input an integer!\n");
if(start > stop){ /* make sure stop > start */
    temp = start;
    start = stop;
    stop = temp;
    }
LOWER = start * 1000 * MICPERLAM;
UPPER = stop * 1000 * MICPERLAM;
/* scale factor for cross-section for better visibility */
printf("scale factor in y direction?\n");
while(scanf("%d",&YSCALE) < 1)
    printf("invalid! YSCALE must be a positive integer!\n");
if(DIR == 'h')
    printf("the y plane you want split?\n");
else
    printf("the x plane you want split?\n");
while(scanf("%d",&PLANE) < 1)
    printf("invalid! input an integer!\n");
PLANE = PLANE * 1000 * MICPERLAM;
printf("the mask file?\n");
while(scanf("%s",MASKFILE) < 1)
    printf("invalid! input file name!\n");
printf("the input file?\n");
while(scanf("%s",INPUTFILE) < 1)
    printf("invalid! input file name!\n");
printf("the base name for the output files?\n");
while(scanf("%s",OUTPUTFILE) < 1)
    printf("invalid! input file name!\n");
/* set flag for output of every step */
printf("output profile at every step y=yes n=no?\n");
PRINTALL = getchar();/* clear \n from buffer */
while((PRINTALL = getchar()) != 'y' && PRINTALL != 'n')
    printf("invalid!\n");
}
```

```
#include <stdio.h>

kicfile(num,label)
/* output header and bottom of kic file numbered (num) */
int num;
char label[];

{
int n = 0;
int height,width;
char *file[20],*header[20];
FILE *fp,*fopen();
extern int UPPER,LOWER,YLIMIT,YSCALE;
extern double MICPERLAM;
extern char *OUTPUTFILE[];

if(num == 0)
    sprintf(file,"%s",OUTPUTFILE);
else
    sprintf(file,"%s.%02d",OUTPUTFILE,num);
fp = fopen(file,"w");
sprintf(header,"(Symbol %s);\n",file);
fputs(header,fp);
sprintf(header,"9 %s;\n",file);
fputs(header,fp);
sprintf(header,"DS 0 1 1;\n");
fputs(header,fp);
kiclayers(fp);/* calculate and put the cross-section into the file */
sprintf(header,"L LABL;\n");
fputs(header,fp);
sprintf(header,"94 %s %d %d;\n",label,0,0);
fputs(header,fp);
width = (UPPER-LOWER)/(10.0*MICPERLAM);
height = YLIMIT*YSCALE/(10.0*MICPERLAM);
sprintf(header,"B %d %d %d %d;\n",width,2*height,width/2,height);
fputs(header,fp);
sprintf(header,"DF;\n");
fputs(header,fp);
sprintf(header,"E\n");
fputs(header,fp);
fclose(fp);
}
```

```
#include "defs"
#include <stdio.h>

kiclayers(fp)
/* transform internal format to kic format */
FILE *fp;
{
extern struct tnode *home;
extern int NLAYER;
struct tnode *top,*high,*low,*low1;
struct knode *new,*kalloc(),*x[40];
char w[40],*numtotype();
int i = 0;
int hightype,lowtype;

/* set up an empty array to point to each layer */
while(i <= NLAYER)
    x[i++] = 0;
top  = home;/* begin in upper left corner */
/* sort the grid into an array of each layer */
/* loop through the columns */
while(top != 0){
    low = top;
    lowtype = nodelayer(low);/* nodelayer finds the type of a node */
    /* loop down the rows */
    while(low != 0){
        high = low;
        hightype = lowtype;
        /* check the next box down to see if it is the same layer */
        /* if they are the same then move down one more */
        while((low1 = low->south) != 0){
            if((lowtype =nodelayer(low1)) == hightype){
                low = low1;
                }
            else
                break;
            }
        if(hightype > 0){
            new = kalloc();/* create a new node */
            new->high = high;/* point to the highest box of the type */
            new->low = low;/* point to the lowest box of the type */
            new->prev = x[hightype];/* point to the previous node */
            x[hightype] = new;/*update the pointer array to last element*/
            }
        low = low->south;
        }
    top = top->east;
    }
/* convert the layer list into a kic file */
i = 0;
/* loop through the layers */
while(++i <= NLAYER){
    /* check to see if the layer is empty */
    if(x[i] != 0){
        sprintf(w,"L %s;\n",numtotype(i));  /* put layer label */
        fputs(w,fp);
```

```
        /* put out boxes */
        while(x[i] != 0){
            printnode(x[i]->high,x[i]->low,fp);/* output the kic line */
            x[i] = x[i]->prev;/* move backwards through the pointers */
            }
        }
    }
}
```



```
#include "defs"

main()
/* main routine to run SIMPL */

{
exinit(); /* external variables initialized */
interact();/* get interactive inputs */
edge(); /* find the mask edges */
init(); /* initialized the grid */
inputs(); /* read inputfile and simulate steps */
/* diagnostic-print number of rectangles across and down */
printf("\n\nacross and down %d %d\n",numacross(),numdown());
mem();
}
```

```
#include <stdio.h>
#include "defs"

struct nnode *maskname(line)
/* this take a kic input line and puts the mask name into a node */
char *line;
{
extern struct nnode *maskhome;
struct nnode *node,*nalloc();
char mname[6],*index();

/* create a pointer list of the masks and maskcrossings */
node = maskhome; /*maskhame points to the first mask */
/* find the mask name from input line */
if(sscanf(line,"L %[^;]",mname) < 1){
    printf("mask name not found in line %s",line);
    exit(1);
    }
/* check to see if we already saw this mask */
while(node->nextmask != 0){
    if(strcmp(mname,node->name) == 0)
        break;
    node = node->nextmask;
    }
/* if it is a new mask then set up a pointer for it */
if(node->nextmask == 0){
    strcpy(node->name,mname);
    node->first = 0;
    node->nextmask = nalloc();
    node->nextmask->nextmask = 0;
    }
/* return the node which points to that mask */
return(node);
}
```

```
#include "defs"

newboxy(start,new,west,yval)
/* insert a new row horizontally */
struct tnode *start,*new,*west;
int yval;/* height of new row */

{
struct tnode *starteast,*newsouth,*talloc();

/* set the values for the new box */
new->nwx = start->nwx;
new->nwy = yval;
new->sex = start->sex;
new->sey = start->sey;
new->type = start->type;
new->doping = start->doping;
new->north = start;
new->south = start->south;
new->west = west;
if((starteast = start->east) == 0)
    new->east = 0;
else
    new->east = talloc();
/* update the values of the origional */
start->south = new;
start->sey = yval;
if((newsouth = new->south) != 0)
    (newsouth)->north = new;
/* recursive call till the row is finished */
if(starteast != 0)
    newboxy(starteast,new->east,new,yval);
}

newboxx(start,new,north,xval)
/* insert a new column vertically */
struct tnode *start,*new,*north;
int xval;

{
struct tnode *neweast,*startsouth,*talloc();

/* set the values for the new box */
new->nwx = xval;
new->nwy = start->nwy;
new->sex = start->sex;
new->sey = start->sey;
new->type = start->type;
new->doping = start->doping;
new->north = north;
if((startsouth = start->south) == 0)
    new->south = 0;
else
    new->south = talloc();
new->west = start;
new->east = start->east;
```

```
/* update the values of the origional */
start->east = new;
start->sex = xval;
if((neweast = new->east) != 0)
    (neweast)->west = new;
/* recursive call till the column is finished */
if(startsouth != 0)
    newboxx(startsouth,new->south,new,xval);
}
```



```
double nextsplit(npeak,nmax)
/*calculate the doping value for the next box of the diffusion profile*/
double npeak,nmax;

{
double abpeak,sgn(),fabs();
double nlow = 3e11;
double nlow1 = 1e11;

abpeak = fabs(npeak);
/* if the peak doping is of opposite sign to the start doping then the
next doping is 10x higher else it is 10x lower */
if(sgn(npeak)  == sgn(nmax))
    npeak /= 10.0;
else
    npeak *= 10.0;
abpeak = fabs(npeak);
/* if the doping is about intrinsic then set it to opposite type
at the minimum level */
if((abpeak) < 1.0)
    npeak = -sgn(nmax)*nlow;
/* if the doping is below the minimum then set it to intrinsic */
else if(abpeak < nlow1)
    npeak = 0;
return(npeak);
}
```

```
#include "defs"

nodelayer(node)
/* return the layer name given the node */
struct tnode *node;
{
int layer;
extern int SI;

/* if it is silicon the we need to convert the doping to layer */
/* otherwise just return layer */
if((layer = node->type) == SI)
    layer = doptolayer(node->doping);
return(layer);
}
```



```
#include "defs"

numacross()
/* count the number of boxes across for diagnostics */
{
extern struct tnode *home;
struct tnode *box;
int i = 1;

box = home;
while((box = box->east) != 0)
    i++;
return(i);
}

numdown()
/* count the number of boxes down for diagnostics */
{
extern struct tnode *home;
struct tnode *box;
int i = 1;

box = home;
while((box = box->south) != 0)
    i++;
return(i);
}
```

```
#include <math.h>
#include "defs"

oxide(depth)
/* oxidise SI and POLY - oxide thickness is depth if bare surface */
int depth;
{
extern struct tnode *home;
struct tnode *top,*box;
int topox,bottomox,newox,newbottom,bottom,surf;
extern int OX;
double tox,ddepth,hypot();

top = home;
/* loop through columns */
while(top != 0){
    box = top;
    /* loop through air to get to surface */
    while(box->type == 0)
        box = box->south;
    topox = box->nwy; /* locate top of oxide */
    /* find initial oxide thickness */
    if(box->type == OX){
        while(box->type == OX)
            box = box->south;
        bottomox = box->nwy;
        tox = topox - bottomox;/* init. oxide thickness */
        ddepth = depth;
        /* calculate new oxide thickness */
        newox = hypot(tox,ddepth) - tox;/*hypot-sqrt of sum of squares*/
        }
    else{
        /* no initial oxide case */
        bottomox = topox;
        tox = 0;
        newox = depth;
        }
    newbottom = bottomox - newox*0.46;/*calc. location of oxide bottom*/
    bottom = newbottom;
    /* if the material is SI or POLY then oxidise it */
    if(validox(box) != 0){
        /* oxidese down */
        while(box->sey > newbottom){
            box->type = OX;
            bottom = box->sey;
            box = box->south;
            }
        /* make sure the last box is split if necessary */
        if(box->nwy != newbottom && validox(box) != 0){
            splity(newbottom);
            box->type = OX;
            bottom = box->sey;
            }
        /* calculate the ammount to oxidise up */
        surf = topox + (bottomox - bottom)*1.18;
        splity(surf);
```

```
        /* move up to top of oxide */
        while(box->type == OX){
            box = box->north;
            }
        /* oxidise upwards */
        while(box->nwy <= surf){
            box->type = OX;
            box = box->north;
            }
        }
    top = top->east;
    }
}
```

```
#include "defs"

pairs(node,high,low)
/* pairs stores pairs of mask crossings */
/* nnode pointes to the list of pairs for each mask */
/* pnodes are the crossings */
/* low and high are the coordinates of the mask crossings */
/* the pairs are stored in numerical order */
/* this also combines overlapping rectangles */
struct nnode *node;
int high,low;
{
struct pnode *prev, *pres, *next, *ins, *palloc();

if(node->first == 0){
    /* case for the first pair of that mask */
    pres = palloc();/* create a node */
    pres->start = low;
    pres->stop = high;
    pres->nextpair = 0;/* set the next pair to null */
    node->first = pres;/* start the link list with this node */
    }
else{
    /* not the first pair case */
    /* must insert the new crossings into the link list */
    pres = node->first;
    prev = 0;
    /* loop through the list */
    while(pres != 0){
        /* if the pair is greater than the present pair */
        if(low > pres->stop){
            prev = pres;
            pres = pres->nextpair;
            /* if we are at the end then add the node to the end */
            if(pres == 0){
                pres = palloc();
                pres->nextpair = 0;
                pres->start = low;
                pres->stop = high;
                prev->nextpair = pres;
                break;
                }
            }
        else if(high < pres->start){
            /*if pair is less than present pair then insert the pair*/
            ins = palloc();
            ins->nextpair = pres;
            ins->start = low;
            ins->stop = high;
            if(prev == 0)
                node->first = ins;
            else
                prev->nextpair = ins;
            break;
            }
        else{
```

```
                /* pair overlaps present pair - need only combine pairs */
                /* pair over extends on the low end - reset low value */
                if(low < pres->start)
                    pres->start = low;
                /*pair overextends on the high end-may cover several pairs*/
                if(high > pres->stop){
                    pres->stop = high;
                    next = pres->nextpair;
                    pres->nextpair = 0;
                    while(next != 0){
                        if(next->start > high){
                            pres->nextpair = next;
                            break;
                            }
                        else if(next->start <= high && next->stop >= high){
                            pres->stop = next->stop;
                            pres->nextpair = next->nextpair;
                            break;
                            }
                        }
                    }
                break;
                }
            }
        }
    }
```

```
#include "defs"

photo(polarity,name,bloat,shift)
/* expose resist with mask(name) and of set polarity */
/* bloat makes the mask wider and shift is a shift to the right */
char *polarity,*name;
int bloat,shift;
{
extern struct nnode *maskhome;
extern int LOWER,UPPER;
struct nnode *mask;
struct pnode *node;
int start,stop,pol;
mask = maskhome;
if(strcmp("NEG",polarity) == 0)
    pol = 1;
/* if pol = 1 then the mask crossings are the light spots */
else if(strcmp("POS",polarity) == 0)
    pol = -1;
/* if pol = -1 then the mask crossings are the dark spots */
/* then the pairs to expose are the spaces between the crossings */
else{
    printf("type not known");
    exit(1);
    }
while(strcmp(mask->name,name) != 0){ /* find the right mask */
    mask = mask->nextmask;
    if(mask == 0){
        printf("mask (%s) not found!\n",name);
        exit(1);
        }
    }
node = mask->first;
if(pol == -1) /*if neg polarity, start is the absolute LOWER bound*/
    start = LOWER;
while(node != 0){ /* loop through the pairs */
    if(pol == 1){
        start = node->start - bloat + shift;
        stop = node->stop + bloat + shift;
        expose(start,stop);/* expose the region */
        }
    if(pol == -1){
        /* start is set by the stop of the previous pair */
        /* stop is the begining of the present pair */
        stop = node->start + bloat + shift;
        expose(start,stop);
        start = node->stop - bloat + shift;/* update start */
        }
    node = node->nextpair;
    }
if(pol == -1){ /* expose the final are to the far right edge */
    stop = UPPER;
    expose(start,stop);
    }
}
```

```
#include "defs"
#include <stdio.h>

printnode(high,low,fp)
/* print out rectangles for kic files */
struct tnode *high;/* high points to the top of the box */
struct tnode *low;/* low points to the bottom of the box */
FILE *fp;
{
extern int YSCALE,YLIMIT;
extern int UPPER,LOWER;
extern double MICPERLAM;
int delx,dely,xavg,yavg;
int nwx,nwy,sex,sey;
char w[40];

/* check edges against the limits */
if((nwx = high->nwx) > UPPER)
    nwx = UPPER;
if(nwx < LOWER)
    nwx = LOWER;
if((nwy = high->nwy) > YLIMIT)
    nwy = YLIMIT;
if(nwy < -YLIMIT)
    nwy = -YLIMIT;
if((sex = high->sex) > UPPER)
    sex = UPPER;
if(sex < LOWER)
    sex = LOWER;
if((sey = low->sey) > YLIMIT)
    sey = YLIMIT;
if(sey < -YLIMIT)
    sey = -YLIMIT;
/* calculate delta x and delta y for the box */
delx = sex - nwx;
dely = nwy - sey;
/* if either delta x or y is 0 then the box has no area */
if(dely != 0 && delx != 0){
    /* calculate the average values for the rectangle */
    xavg = (nwx + sex)/2 - LOWER;
    yavg = (nwy + sey)/2 + YLIMIT;
    /* scale from .001u internal to .01lambda in kic */
    dely = dely * YSCALE * 0.1 / MICPERLAM;
    yavg = yavg * YSCALE * 0.1 / MICPERLAM;
    delx = delx * 0.1 / MICPERLAM;
    xavg = xavg * 0.1 / MICPERLAM;
    /* output one box if not null */
    if(dely != 0 && delx != 0){
        sprintf(w,"B %d %d %d %d;\n",delx,dely,xavg,yavg);
        fputs(w,fp);
        }
    }
}
```

```
#include "defs"

printpairs()
/* print out list of the mask crossings */
{
extern struct nnode *maskhome;
struct pnode *node;
struct nnode *mask;

mask = maskhome;
/* loop through the masks */
while(mask != 0){
    printf("\n%s\n",mask->name);/* print mask name */
    node = mask->first;
    /* loop through the pairs */
    while(node != 0){
        printf("%d %d\n",node->start,node->stop);/* print pair */
        node = node->nextpair;
        }
    mask = mask->nextmask;
    }
}
```

```
#include "defs"

rectangle(x1,x2,y1,y2,node)
/* calculate if the rectangle is cut by the crossection plane */
/* send the crossing points to pairs */
/* x1,x2 are the lower,higher x coordinates of the rectangle */
/* y1,y2 are the lower,higher y coordinates of the rectangle */
int x1,x2,y1,y2;
struct nnode *node;
{
extern int UPPER,LOWER,PLANE;
extern char DIR;
extern double MICPERLAM;
int low,high;

/* low and high are the coordinates of the crossing */
low = 0;
high = 0;

/* convert kic .01u to internal 0.001u */
x1 = 10*x1*MICPERLAM;
x2 = 10*x2*MICPERLAM;
y1 = 10*y1*MICPERLAM;
y2 = 10*y2*MICPERLAM;
if(DIR == 'h'){  /* horizontal splits */
    /* check the bounds of the rectangle and the split plane */
    if(y1 <= PLANE && y2 >= PLANE){
        /* set high and low */
        low = x1;
        high = x2;
        /* adjust to min and max bounds */
        if(x1 < LOWER)
            low = LOWER;
        if(x2 > UPPER)
            high = UPPER;
        }
    }
else if(DIR == 'v'){  /* vertical splits */
    if(x1 <= PLANE && x2 >= PLANE){
        low = y1;
        high = y2;
        if(y1 < LOWER)
            low = LOWER;
        if(y2 > UPPER)
            high = UPPER;
        }
    }
/* if low is less than high then the rectangle is split by the plane
within the min and max bounds- otherwise they are = or > */
if(low < high)
    pairs(node,high,low);
}
```

```
#include "defs"

struct tnode *searchsouth(startnode,yval)
/* searches south for a box that contains the coordinate yval */
struct tnode *startnode;
int yval;
{

while(startnode->sey > yval)
    startnode = startnode->south;
return(startnode);
}

struct tnode *searcheast(startnode,xval)
/* searches east for the box containing the coordinate xval */
struct tnode *startnode;
int xval;
{

while(startnode->sex < xval)
    startnode = startnode->east;
return(startnode);
}
```



```
double sgn(n)
/* return 1 if n>0   0 if n=0   -1 if n<0 */
double n;
{
double n1 = 0;

if(n < 0)
    n1 = -1;
else if(n > 0)
    n1 = 1;
return(n1);
}
```

```
#include "defs"

splity(yval)
/* split the grid horizontally */
int yval;/* height of split */

{
extern struct tnode *home;/* upper left corner */
struct tnode *startnode,*newnode,*west,*talloc(),*searchsouth();

startnode = searchsouth(home,yval);/*find (columm)node containing yval*/
/* if the south east y value = yval then no split is necessary */
if(startnode->sey != yval){
    newnode = talloc();/* create a new node */
    west = 0;/* the node to the west is null */
    newboxy(startnode,newnode,west,yval);/* insert new boxes */
    }
}

splitx(xval)
/* split the grid vertically */
int xval;/* x location of the split */

{
extern struct tnode *home;/* upper left corner */
struct tnode *startnode,*newnode,*north,*talloc(),*searcheast();

startnode = searcheast(home,xval); /*find (row)node containing xval*/
/* if the south east x value = xval then no split is necessary */
if(startnode->sex != xval){
    newnode = talloc();/* create a new node */
    north = 0;/* the node to the north is null */
    newboxx(startnode,newnode,north,xval);/* insert new boxes */
    }
}
```

```
#include "defs"

sub(n,d)
/* create a substrate by making the south rectangle proper type and */
/* doping --- only makes sense immediately after initialization */
int n;
double d;

{
extern struct tnode *home;
struct tnode *node;

splity(0); /* split at zero */
node = home->south; /* move to bottom half */
node->type = n; /* switch type */
node->doping = d; /* switch doping */
}
```



```
#include "defs"

validox(node)
/* return 0 if material can't be oxidised or n>0 if it is SI or POLY */
struct tnode *node;
{
extern int SI,POLY;
int t;

t = node->type;
if(t == POLY || t == SI)
    return(t);
else
    return(0);
}
```

```
#include "defs"
#include <stdio.h>

wires(line,node)
/* get the rectangle coordinates of the wire segments */
char line[];
struct nnode *node;
{
int width,px1,px2,py1,py2,x1,y1,x2,y2;
char wire[10],*shorten();

/* get width and first x,y coordinates of the rectangle */
if(sscanf(line,"%s%d%d%d%[^\0]",wire,&width,&px1,&py1,line) < 5)
    printf("wrong num of arguments in line %s",line);
width /= 2;/* set width to the half width of the rectangle */
/* loop through the string pulling out the subsequent pairs */
while(sscanf(line,"%d%d%[^\0]",&px2,&py2,line) > 1){
    /* does the pair lie horizontal */
    if(py1 == py2){
        /* set ymin and ymax */
        y1 = py1 - width;
        y2 = py2 + width;
        /* set xmin and xmax */
        if(px1 <= px2){
            x1 = px1 - width;
            x2 = px2 + width;
            }
        if(px2 < px1){
            x1 = px2 - width;
            x2 = px1 + width;
            }
        }
    /* does the pair lie vertical */
    if(px1 == px2){
        /* set xmin and xmax */
        x1 = px1 - width;
        x2 = px2 + width;
        /* set ymin and ymax */
        if(py1 <= py2){
            y1 = py1 - width;
            y2 = py2 + width;
            }
        if(py2 < py1){
            y1 = py2 - width;
            y2 = py1 + width;
            }
        }
    rectangle(x1,x2,y1,y2,node);
    /* move the second pair to the first spot and loop around */
    px1 = px2;
    py1 = py2;
    }
}
```