ELECTRICAL PROBING OF TEST STRUCTURES FOR OPTICAL LITHOGRAPHY

by

Jay Fleischman

Memorandum No. UCB/ERL M88/60

7 September 1988

ELECTRICAL PROBING OF TEST STRUCTURES FOR OPTICAL LITHOGRAPHY

by

Jay Fleischman

Memorandum No. UCB/ERL M88/60

7 September 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Electrical Probing of Test Structures for Optical Lithography

Jay Fleischman

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ABSTRACT

The use of electrical testing to characterize Optical Projection Printing is convenient for studying effects across the field of the die, across the wafer, and over many wafers. The electrical data is not subject to operator interpretation and can be compiled into informative graphs and charts via software, making the entire process completely automated. New test structures designed to be sensitive to various processing elements can then be analyzed for sensitivity and compared to simulated results. This will provide a set of test structures that can be used to refine and monitor a fabrication process or characterize the equipment used in the process.

A flexible system for automatic measurement and display of test results for a large class of structures was implemented. It consists of the following equipment: an Electroglas 2001X Automatic Prober and the HP 4062 system (4085A SWM, 4084A SWC, and 4141A DCS) controlled by an HP9836 Pascal workstation. The input control file allows functions and keywords specific to lithography and can be generalized to accommodate other applications. This is accomplished by using an input interpreter to connect pins, force and measure voltages and currents, and to analyze those measurements. The output generated is then set to a VAX where it is parsed using the keyword and plotted using routines written in C. Example plots for cases such as linewidth vs focus and dose (SMILE), linewidth variation within the die, flare, conductivity of checkerboards, defects, and lines, and exploratory structure resistance tables have been generated.

This paper describes step by step operation of the above system. First the hardware initialization will be presented. Next, the structure of the probing software and prober input control files will be discussed. Then various parsers designed to create specific formats of output data for easy analysis are described. Finally, a step by step example and some experimental results are presented with a few comments.

September 6, 1988

Table of Contents

Chapter 1

Hardware

The components consist of three main parts: first, there is the Electroglas automatic prober itself, then the several components of the HP 4062 system, and finally the HP Pascal workstation, which serves as the controller. The 4062 measurement system contains the SWC 4084A Switching Matrix Controller, the SWM 4085A Switching Matrix, and the DCS 4141A DC Source/Monitor components for all the measurements taken and programs written at this time. The CMS 4280A 1 MHz C Meter and C-V Plotter may be added in the future, and the probing software revised as necessary.

Appendix A is an excerpt from Norman Yuen's thesis describing how to set up the Electroglas prober, mount the wafer, and align the wafer for measuring. This document should be sufficient to set up and operate the prober, but there is a correction necessary to all previous documentation on the automatic prober regarding the orientation of the wafer. Since the mask can be in any of four possible rotations, *there is no set position of the flat relative to the machine.* When the wafer is on the chuck and observable through the microscope, the orientation necessary to probe will become evident. For probing, there must be agreement with the prober file and the die orientation. Since a focus/exposure matrix is usually fixed relative to the flat, special versions of the parsers may be necessary. This is discussed further in the software section.

Care must be taken to connect the different components. First, the controller Pascal workstation (HP9836U), hard-drive system, expansion box (9888), printer, prober, and measurement components are *all on the same HP-IB bus.* Thus, a lot of "piggyback" cables will be attached together. But, there is at least one cable *that has a bad connection* to additional cables. Therefore, it must be the last in any given sequence. The cable most suspect at this time is the one to the printer. Second, any device on the HP-IB bus can take hold of the bus, denying access to all other devices and causing the system to lock up. If the system locks up for some reason, each machine may have to be reset (or turned off) and complete reinitialization may be necessary. If this situation occurs, try to close any output files and save them to the hard-drive before powering down the HP9836U. Finally, the connection between the SWC and the SWM 4085A Switching Matrix is limited. It supports all 4 SMU connections and 2

auxiliary ports. These two ports can be used for either VM's or VS's, but not both. VM1 or VS1 should be connected to AUX1, and VM2 or VS2 should be connected to AUX2. Hence, not all possible connections on the HP4045 are supported by this system. Currently, the VM's are both connected and all sources are provided through the SMU's.

To power up the computer controller, first turn on the hard disk, then the expansion box, and finally the HP9836U. It will then immediately recognize all its extensions and find the Pascal system on the hard drive. If a system floppy disk is in a drive, the Pascal system will not be automatically loaded, so to avoid confusion, power up this way unless you know what you are doing (or wish to experiment...).

Chapter 2

Prober Software

The prober software consists of the Pascal program AUTOPRB and several parsers in C which run on VAXes. AUTOPRB is currently located in the GAMES: volume on the hard disk of the HP9836U workstation while the parsers are located in "simsoft" on esvax. AUTOPRB is a revised edition of the E290 program, which in turn is a cut down version of BSIM. Much of the overall structure has been maintained, but the measurement routines have been completely rewritten. The prober file format has also been changed to simplify writting long files, and at the same time make the file more readable to the operator. Finally, the output has been improved to include two output formats: a long descriptive format, and a short specialized one. The latter format is a subset of the former and can be redirected to the printer or to an output file. The screen display during probing has been changed to reflect more pertinent data. Now all information related to the device being tested is displayed. This way, the operator can determine during the run if the wafer is being probed correctly and if the results are reasonable.

To run AUTOPRB, simply eXecute it from the main Pascal menu, and then follow the directions it displays on the screen. A listing of the program is contained in Appendix B. Due to the general structure of AUTOPRB, adding features like more user defined functions and measurement configurations is easy to do. At this time, the entire prober file is read into several data structures to describe the probing of the die. For long probing, this may exceed the memory of the HP9836U. The revised structure of the measuring routines coupled with the data structures used to describe each test makes it simple to convert the program to read in the prober file sequentially *while* performing the measurements. Thus, rho, a user macro that will be discussed later, will have more power, as the sheet resistance used can be changed for all corresponding tests to reflect its variation over the die. Although all user macros for testing that have been written so far incorporate only one measurement, this is also not necessary. Simply describe the test using the various data structures to reflect the equivalent number of devices, there location, and lower level tests to be done accordingly. This creates the possibility to probe the *whole die* in one statement from the prober file. This demonstrates the potential of

the user macro capability. Although it becomes unwieldy in this case, using a single macro to test sheet resistance and linewidth may be useful. All macros and functions are defined in the AUTOPRB program (in Pascal) at this time. Future improvements may include a script-type language where different macros could be saved in files and recalled for used by commands in the prober file, but this would be a complicated addition.

Chapter 3

Prober file

An example prober file is provided below. It is self documented feature by feature. For examples of the output of the AUTOPRB program, see Appendix C (short format) and Appendix D (long format).

Currently, there are 3 measurement macros: u4pt, uif, and uvf, and 5 functions: fvdp, fclw, fflw, fsht, and f2tr. Either lower or upper case can be used for their names. In addition to these, ; is used for a prober file comment, # is used for a comment which is also transferred to the output file, and rho is used to set sheet resistance parameters.

The program also allows a more detailed measurement control to make the full capabilities of the equipment available to the user. There are commands to connect any smu, vm, or vs (see notes in hardware) to any pin number and to specify their attributes. Then the pre-defined functions can be used, provided that the operator understands how they work in detail. In any case, the long output format will provide all information (currents and voltages) necessary to derive whatever function is desired by parsers. This interface is very similar to that of the HP4045 system and is easy to learn and understand.

The user defined measurements are built from a set of these lower level commands.

```
u4pt 42 6 46 48
```

For instance, is just:

```
smu2 42 i 0.001
smu3 6 v 0
vm1 46
vm2 48
```

The function fflw then can compute {rnom * 150E-6 * ismu3 / (vm1-vm2)}, where rnom is the nominal rho, ismu3 is the current in smu3, 150E-6 is the length of the line, and vm1-vm2 is the change in voltage. For the exact definitions of the rest of the functions and macros see the AUTOPRB program listing.

```
10885
10885
800
320
11111110
11111110
11111110
111X1110
11111110
11111110
11111110
00000000

;NO COMMENTS above the testing matrix
;Comments anywhere from then on, starting the line with a ";"
;2x10 probe, but since documentation for 2x5, use 800 instead of 1600
; for the X micron site size.  This file will only contain EVEN X
; coordinates for this reason.
;Line 1 = X die size (microns)
;Line 2 = Y die size (microns)
;Line 3 = X SITE size (microns)
;Line 4 = Y SITE size (microns)
;Lines 5 - 13 contain the wafer testing matrix, X marks the die  the
; prober is aligned to when the program starts.

rho 32.6 30.0 35.0
;Set sheet resistance for linewidth calculations
;32.6 in nominal, 30.0 is min and 35.0 is max.  Two variables are used
;to keep track of sheet resistance.  One is the forced value (nom) and the
;other is the measured one, being updated on every sheet resistance function
;call, if it is between min and max.

mx=4
my=17
;mx,my is the ABSOLUTE coordinates of the first device to be measured
;and the prober must be aligned to it when the program starts.

dv=LINEWIDTH
;dv stands for device

# 1.0u line
;This is a comment which WILL BE PRINTED in the output file (long format)
; and is a maximum of 80 characters starting with a "#"
```

```
u4pt 42 6 46 48
;u4pt is a four point probe measurement macro. It connects smu2 to the
;first pin number, smu3 to the second pin, vm1 to the third, and vm2 to the
;fourth. smu2 is a current source of 1mA, smu3 is GND, and vm1 & vm2
;measure the change in voltage.

fflw 1.0u
;fflw is Function Forced rho Line Width. It uses rho nom to calculate
;linewidth, assuming the u4pt macro was used to make the measurement
;1.0u is the KEYWORD which can be used to parse to in analyzing the output
;The output line is "Keyword diex diey sitex sitey test_result"
;See the output formats for further information.

fclw 1.0U
;fclw is Function Current rho Line Width. It uses the current MEASURED
;sheet resistance (if no van der pauw was measured, it defaults to rho nom)
;and otherwise acts like fflw. Notice several functions can be done on the
;same measurement.

dv=VanDerPauw
;the next device will, by default, use the same site location as the
;previous device unless mx and my are explicitly set. This is to accommodate
;several device measurements on the same probe site.
# 1.5u line vdp
u4pt 36 18 34 20
fvdp 1.5uvdp
;fvdp is Function Van Der Pauw. It calculates sheet resistance, rho,
;assuming the u4pt measurement macro. This is then stored as the Current
;rho, if it is between min and max rho. 1.5uvdp is the Keyword to parse to

mx=4
my=15
dv=FOCUS
# Focus A (1 D) 3 12u lines in parallel 0.4u
uif 48 6
;uif is User macro I (current) Forced. smu2 is connected to the first pin
;number and smu3 to the second. smu2 is a 1mA current source and smu3 is
;GND. Voltage difference is measured from smu2 to smu3.

f2tr foc0.4
;f2tr is Function 2 Terminal Resistance. It assumes either macro uif or uvf
;to be used for measuring. Delta V is divided by the current in smu3 to
;produce a resistance value.

fsht dfoc0.4
;fsht is Function SHorT. It detects a short or open circuit by using an
;arbitrary 3K ohm cut-off point.

;An example of the last user macro is "uvf 48 6"
;uvf is User macro Voltage Forced. It applies 3 volts to smu2 (first pin
;number) relative to smu3 (GND, assigned to the second pin number).
```

Chapter 4

Parsing

The analysis of output from each test structure and test site strategy requires a separate parser. However, they are all *very* much the same. First, identification information is read in, then the program checks the first word in every line, comparing it to the keyword passed to it. When the keyword is found, it reads the location of the die and site where the device was measured and the result of the test function. This corresponds to the information available from the short output format. Since it is also included in the long format, that file can also be used, but since the short format is many times shorter, it can be transferred more rapidly to esvax and parsed. The different parsers simply display the result of the test in different manners.

Since the focus exposure matrix is made relative to the flat, not the orientation of the mask, the coordinates of the die that was measured during probing may change for a given "position" in the focus exposure matrix. This occurs because the probing is dependent only on the orientation of the mask. Hence, all parsers that display information concerning the focus exposure matrix (probing over the whole wafer) must be made to fit the orientation so it can interpret the coordinates correctly. Appendix E contains two example parser programs. The first, lwfe, is for line-width SMILE plots in the "assumed lower left" orientation. The second, exfeul, creates the tabular resistance output for a wafer in the "upper left" orientation. These listing show how to adjust the indices to reflect changes in orientation and how to create different types of output.

The names "lwfe" and "exfeul" are strange, but describe several things. The parsers' names are made up of five parts. The first part describes what structure the parser was made for (i.e. line-width (lw), flare (fl), or the exploratory structures (ex)). The second part refers to the scope of the information to be displayed (i.e. over the wafer focus exposure matrix (fe) or within the die (in)). The first two parts combine to determine the way the test result is to be displayed. The last three parts have default values and may be missing. The first of these refers to the orientation of the wafer. The assumed orientation is where the SEM lines are in the lower left corner of the die when the flat is at the top. Different orientations are described by two letters to describe where the SEM lines are (i.e. upper left

corner (ul), etc.). The last two parts correspond to the version of the mask used. Since the site location is used in determining certain aspects of the display and to distinguish between some structures, this information is necessary. If this field is absent, it is assumed to work regardless of the version. The versions are described by the stepper used and the version for that stepper (i.e. the first version done on the Berkeley stepper is b1, and the first for the AWIS stepper is a1). At this point, only a1 and b1 versions have been made and the lack of a version in the name means that either of these two version will work on that parser.

Figure 1 is a flow chart showing how to probe a wafer and process the output from the software point of view. All the parsers are invoked in the same way: Parser_name "keyword" [input_file]. The input file is optional and, if left out, the default is used. See the parser program listing for their defaults. In general, the default is related to the function like smileplot.data, explor.data, or flare.data. In this way, many different sets of input data can be process with ease. The plotting output is made up of up to eight files: plot.in1 rewritten every time the parser is invoked. The plot.in files are simply files containing a number of points, (x,y) on each line. The __.plot file contains the script commands to the CREEP plotter module. Therefore, to make the plot on the apple laserwriter printer, invoke the plotter and pass it the name of the script file (i.e. plot ___.plot).

The other output type is a table of resistances generated by exfe. This is contained in the file explor.out. It is *approximately* 1/4 of a page, but since 66/4 is not an integer, it doesn't come out quite right. By concatenating several files together, the result can be edited to produce 4 results per page.

Probing Software Path

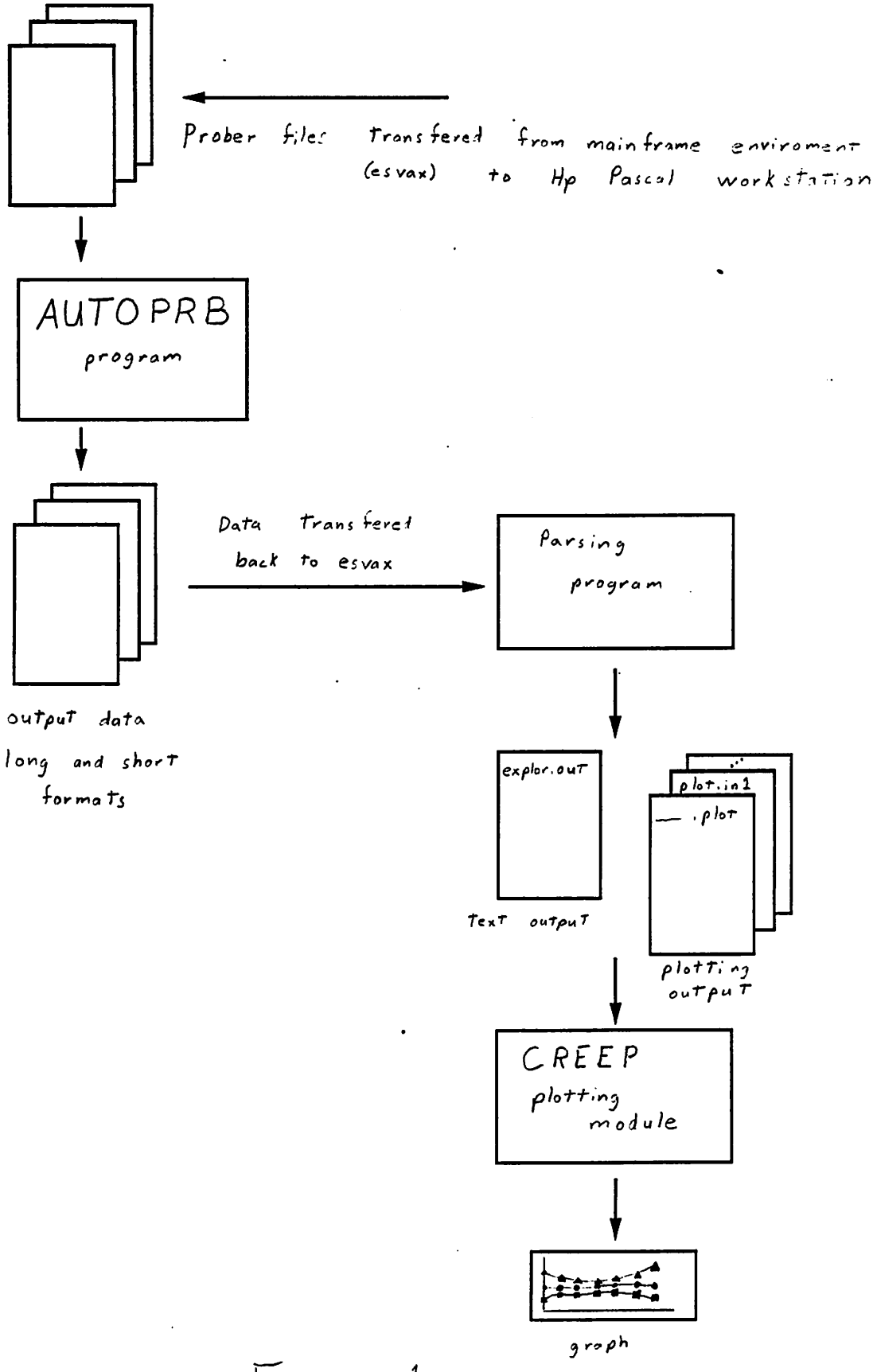


Figure 1

Chapter 5

Sample Session

First write the prober file using any text editor. This can be on a mainframe and later transferred to the HP workstation, or done on the workstation itself. To keep a copy in the mainframe environment is usually a good idea so the first method is preferred. To transfer the file to the HP, the terminal emulator must be used. Turn on the hard drive, expansion box, and HP9836U. From the main command menu, type the following:

```
p
-> Load what code file?
#11:NEWKBD
p
-> Load what code file?
#11:VT2
x
-> Execute which file?
VT2
```

The input message prompts may differ slightly. At this point you will be running the emulator, an HP2648, but the protocol must be set up first, so choose that option (option 4: create configuration) and respond to the questions accordingly For this example, connection was made to a VAX/UNIX (option 1), at 9600 BAUD and full duplex (option 3). After setting up the emulator, go into emulation mode and log in through the Cory Hall Port Selector. To transfer a file, press <ctrl> and <exec> at the same time. A menu will be display with the options to transfer files in either direction. Simply follow directions. One special note is that all volumes are referenced through numbers. In this work, files were always transferred back and forth through the RAM: volume, which is referenced as #45:.

While the files are transferring, set up the hardware. Turn on the Electroglas 2001X and respond to the prompts on the monitor. Mount the wafer, remembering to turn on the vacuum, and align it. Next, position the probes over the origin die and device as specified in the prober file to be used. Turn on the SWC and DCS. Make sure the automatic prober is ONLINE and the I/O is set to GPIB (option 2 on line 7 of MODE DISPLAY). For more detailed instruction see Appendix A.

After transferring the files, logout and terminate the emulator. This will bring you back to the

main menu. To speed up processing, you can use the Filer to transfer GAMES:AUTOPRB.CODE to AUTOPRB.CODE (default volume is RAM:), and the prober files to the RAM: volume. Quit the Filer by pressing "q" and execute AUTOPRB (type "x" followed by AUTOPRB to the prompt). AUTOPRB will then provide directions and options to which you can respond. All questions have defaults and you can always change the options before probing if a mistake is made. The long format output is automatically directed to GAMES:PRBLONG.OUT. The short format is default RAM:SHORT.OUT, it but can be renamed in the options. The default prober file is PROBER.TEXT, but it can also be renamed.

Probing is initiated by pressing any key other than "c". During probing, the current status of the equipment is displayed. This includes the information you typed in as the options, the position of the prober on the wafer, within the die, the test being done and its results, and the results of any user function, if specified, along with its keyword. When the probing is done, you have the option of exiting the program or doing more tests.

When you are finished probing, transfer whatever files you wish back for processing. Transferring the short format output saves time and space. To transfer back to esvax, press "control-execute" as in transferring from the VAX, but choose the "to host" option instead of "from host" as before. When the data is transferred, processing continues using the parsers.

An example prober file for a SMILE plot on the 290 wafers is:

```
10885
10885
800
320
11111110
11111110
11111110
111X1110
11111110
11111110
11111110
00000000

rho 32.6 30.0 35.0

mx=4
my=17
dv=LINEWIDTH
#1.0u line
u4pt 42 6 46 48
fflw 1.0u

dv=LINEWIDTH
#1.5u line
u4pt 32 16 36 38
fflw 1.5u
```

Assuming the output data is in the file 290.smile.data, the SMILE plot is generated for a 1.0 micron line using the "1.0u" keyword (assuming the standard orientation) by:

```
lwfe "1.0u" 290.smile.data
plot smile.plot
```

The SMILE plot of 1.5u lines is generated by:

```
lwfe "1.5u" 290.smile.data
plot smile.plot
```

Other structures would be parsed and plotted by similar commands.

Chapter 6

Results

Classical structures and some exploratory ones have been tested on both the AWIS and Berkeley steppers. The results shown here are for the Berkeley wafer. The OCT/VEM CAD tool was used for layout. The test patterns were converted onto the mask using the GCA 3600F pattern generator. The resist for the process was 1.2 microns of KTI 820 Micropositive photoresist at 120 degrees C prebake. The wafers were printed on a GCA 6200 10X stepper at a numerical aperture (NA) of 0.28 microns, wavelength of 0.4358 microns, and partial coherence factor of 0.7. These wafers were exposed in a standard focus-exposure matrix, with exposures ranging from 0.06 sec to 0.12 sec in 0.01 sec steps, and with focus settings ranging from 254 to 290 in steps of 6 (GCA units). They were developed on a MTI Omnichuck Photoresist Development Station with KTI 934 (50% concentration). Spin-Spray and a 60 sec development time were used. The wafers had 0.4 microns of Phosphorous doped poly, annealed for 30 min at 850 degrees C. They were etched in a LAM Plasma Etcher using carbon tetracloride.

The classical results were as expected. SMILE plots were quickly and easily generated (see Figure 2). The variation of linewidth within the die increased almost linearly with respect to the distance from the center of the die as shown in Figure 3.

Flare structures were created by a line near a large dark field. Two types were made, one with the line on the outside corner of the field, and one on the inside. There is a definite bias from outside to inside separation to as much as 0.3 microns as shown in Figures 4 and 5. This may be caused by a bias in the plasma etch. In Figure 5, the points that drop to 0um at 5um separation are due to defects in this particular wafer, but in both Figures 4 and 5, the first separated point corresponding to Middle of the die (versus the Corner and Edge) is uncharacteristically *larger* than those at the corner and edge. This was found to be a *mask* error (Figure 4 shows this best due to the lack of discontinuity).

Figure 6 contains two pictures of the mask used for the checkerboard test patterns. In Figure 7, the plasma etch step may be causing the round inside corners verses the sharper outside corners of the checkerboard pattern This checkerboard also seems to indicate an astigmatism, as the corners of the

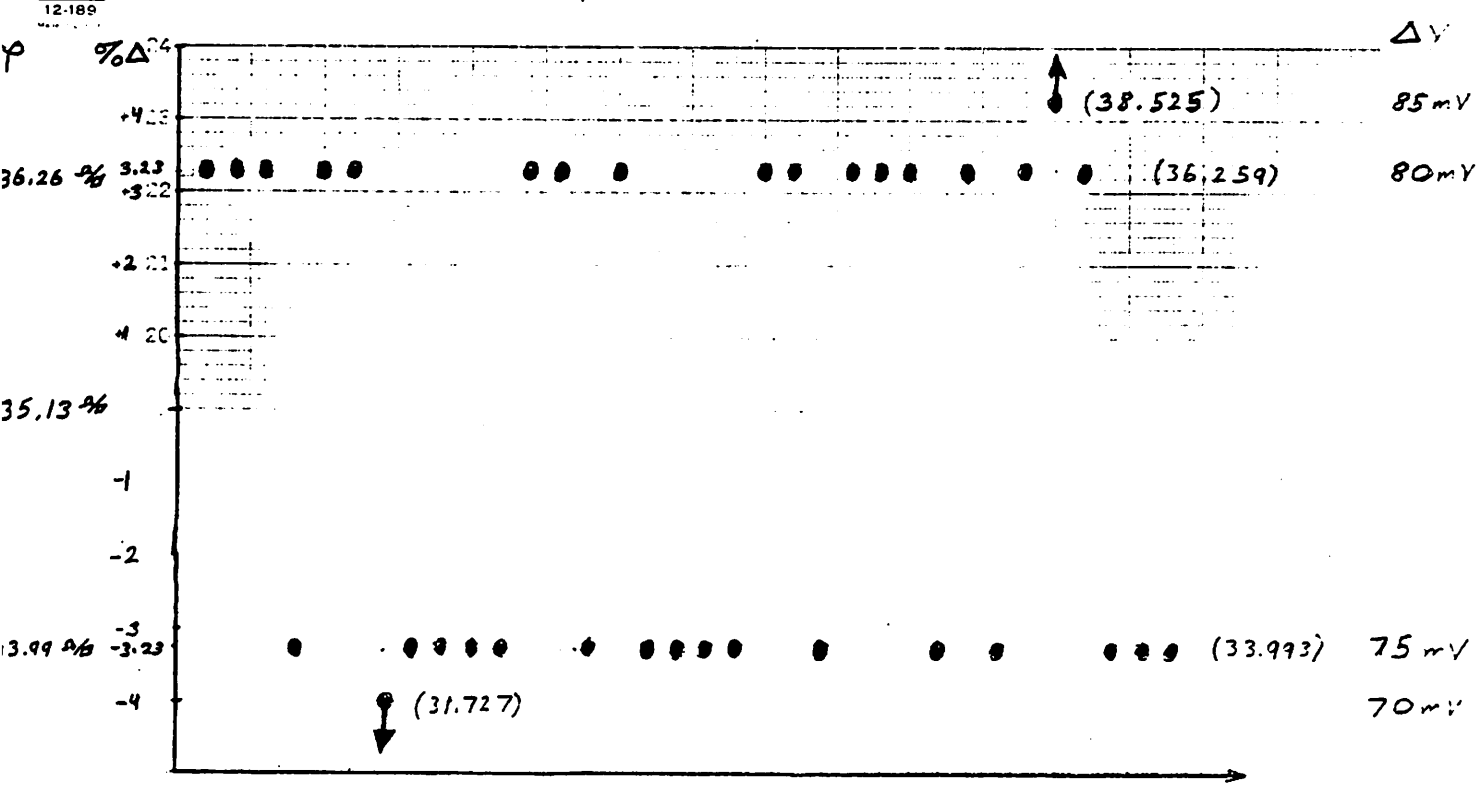
squares are connected along one diagonal more than the other. The resistance table output of these checkerboards is given on the bottom of the next page. Figure 8 is a picture of the "focus" test structure. It consists of three parallel lines. The resistance table output of this structure is also on the next page, at the top.

The test structure defect3A did not work well, as the resolution of the process was never good enough to clear the poly between the defect and contact, even during over-exposure. But, defect3B did appear to present some possibly useful structures. The resistance table output for this structure follows on page 23. Defect3B consists of two elbows with defects of different sizes aligned between the corners of those isolated elbows.

Although some results, graphs, and charts are presented here from the 290 wafer, for the complete set of data for the Berkeley and AWIS wafers, the description of the test structures, and their layout, see those documents..

One problem in electrical probing is the determination of sheet resistance. This variable can change over the wafer and even within a die (see Graph A). For this reason, Van Der Pauw structures were made next to each linewidth site, but the Van Der Pauw structures need to be redesigned. The resistance in the leads to the symmetric square is so great that the voltage drop from corner to corner is much smaller than the overall change in voltage. Hence, to get a reading which is significant to several figures, the current necessary is so great it is excessive for the small size of the leads. This may cause the leads to heat up and go up in smoke, or at least affect the local temperature, making the sheet resistance found invalid. In this work, the sheet resistance was assumed to be constant and was forced to be 32.6 ohms per square in all the calculations.

Graph A


ρ
%Δ
ΔV
(38.525)
85 mV
36.26 Ω/□
(36.259)
80 mV
35.13 Ω/□
(33.993)
75 mV
(31.727)
70 mV
Repetitions

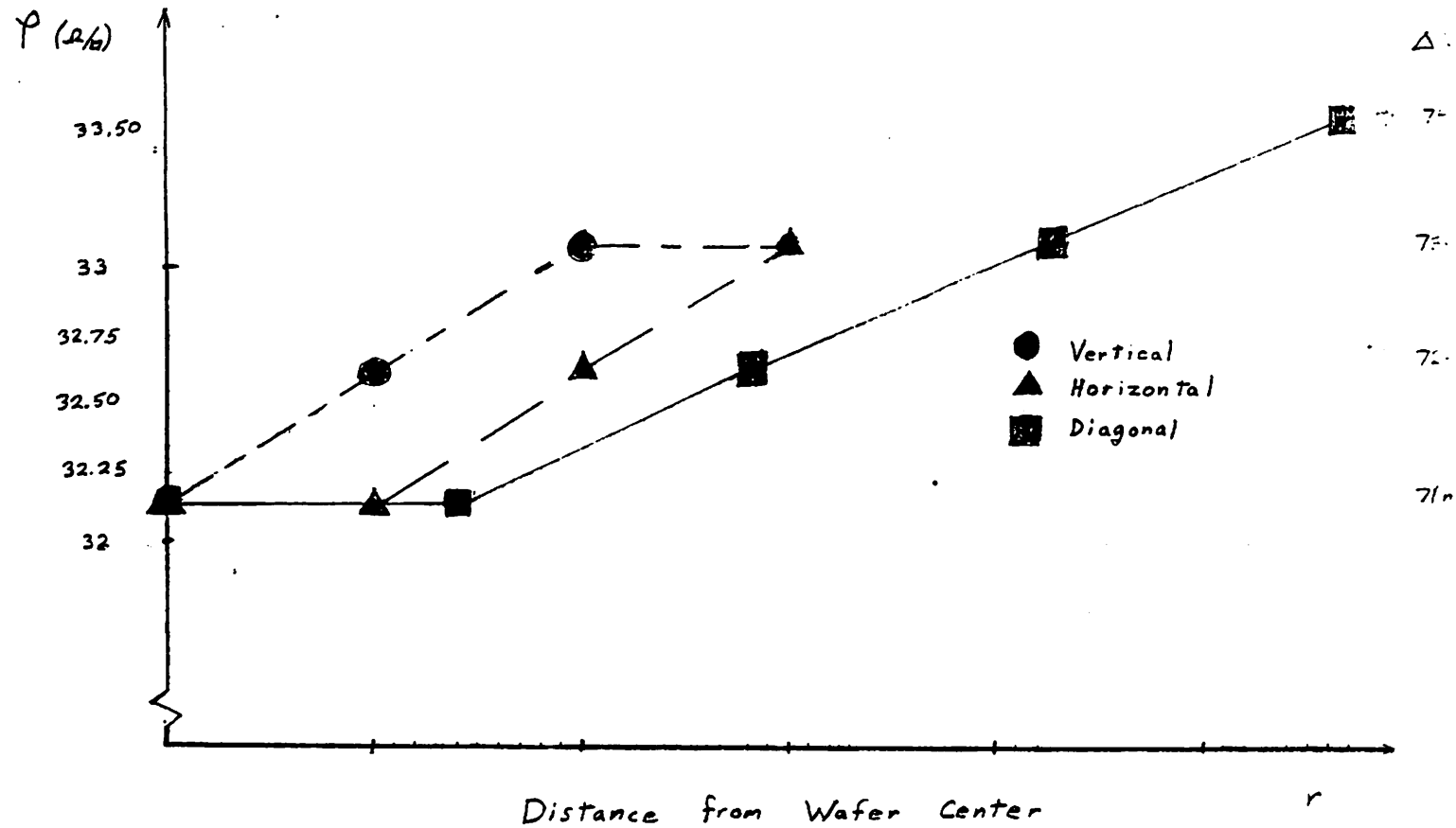

ρ (Ω/□)
33.50
33
32.75
32.50
32.25
32
Vertical
Horizontal
Diagonal
Distance from Wafer Center
r

Figure 2

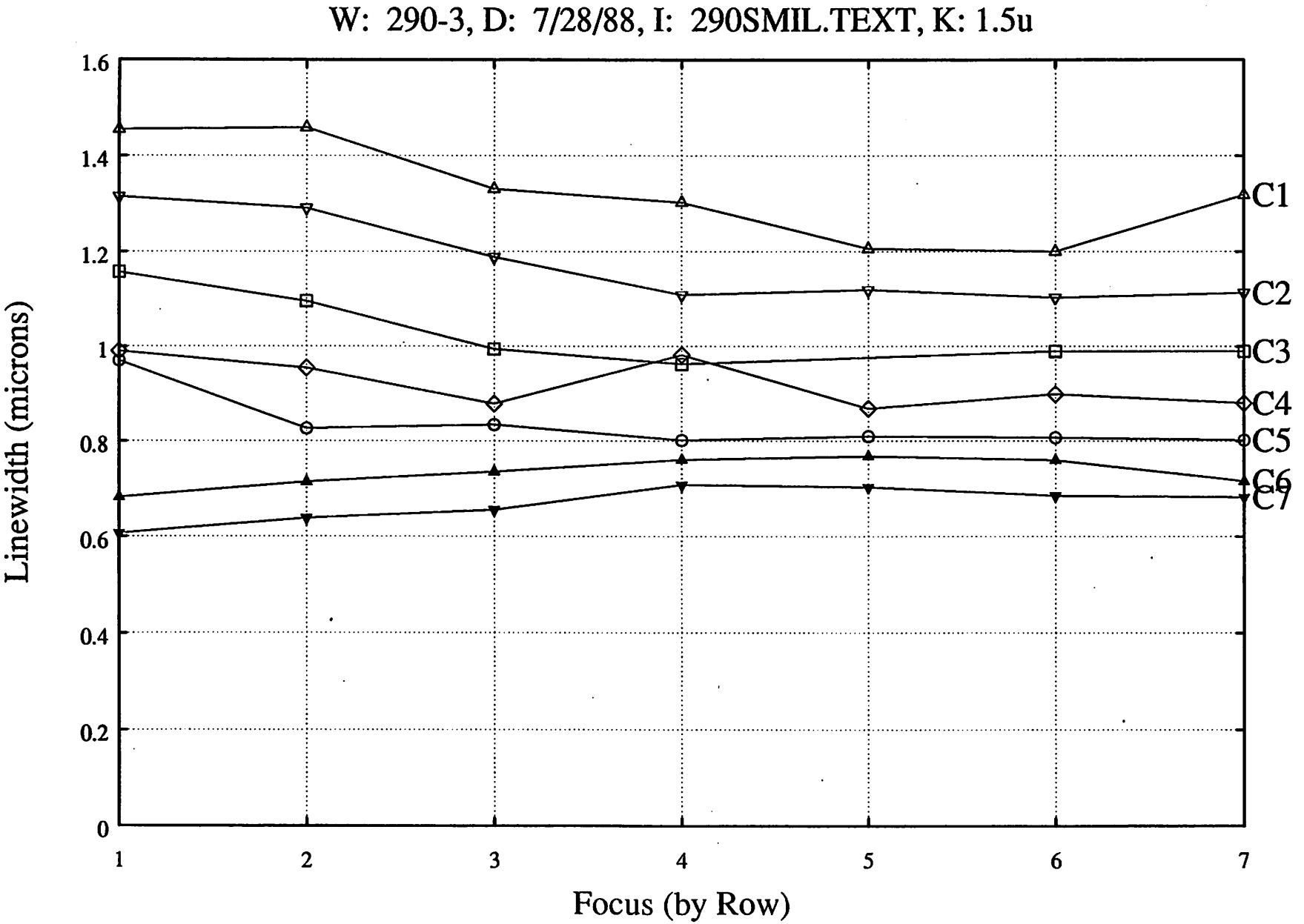

W: 290-3, D: 7/28/88, I: 290SMIL.TEXT, K: 1.5u
Linewidth (microns)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
7
Focus (by Row)
C1
C2
C3
C4
C5
C6
C7

Figure 3

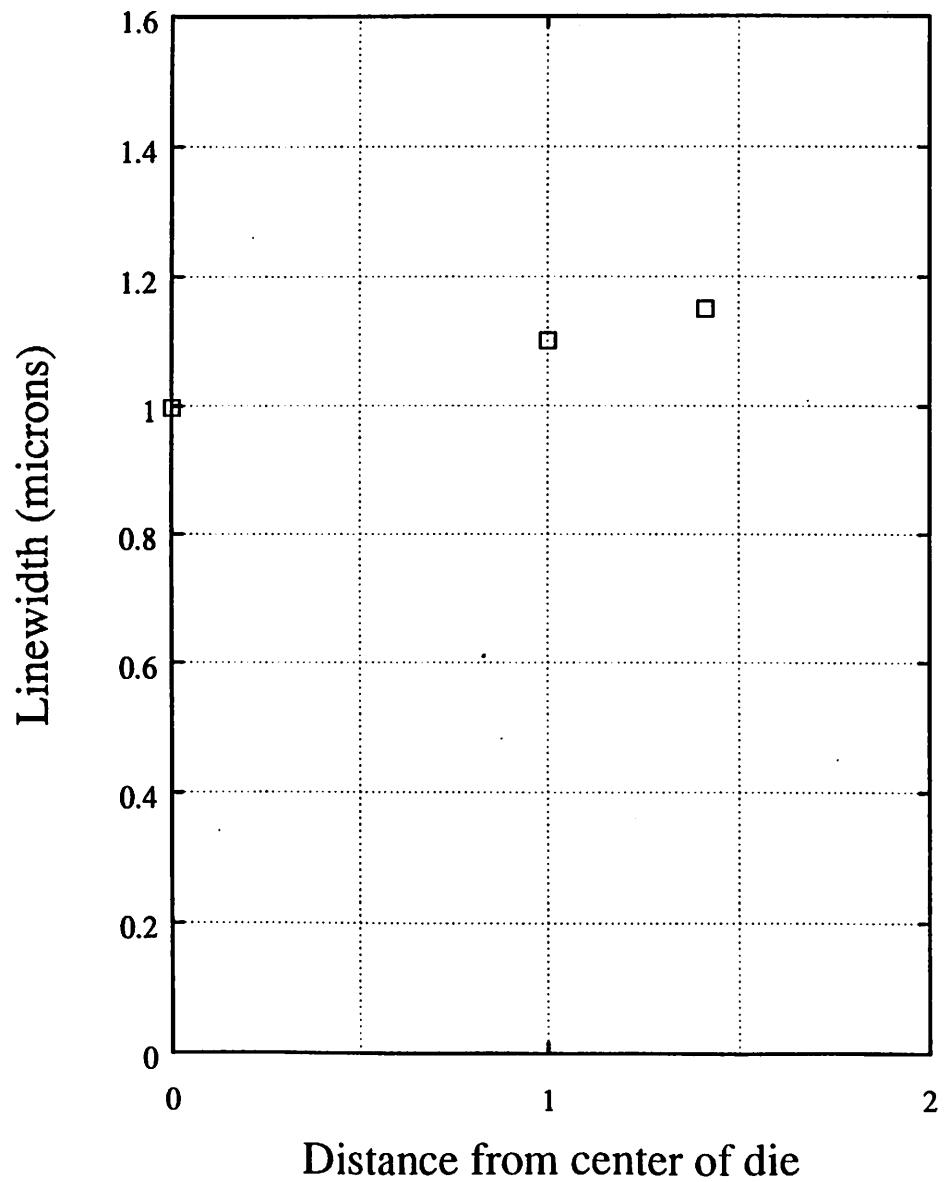

W: 290-3, C: 3 R: 6, D: 8/3/88, I: INTPRB.TEXT, K: 1.5u
Linewidth (microns)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
Distance from center of die

Figure 4

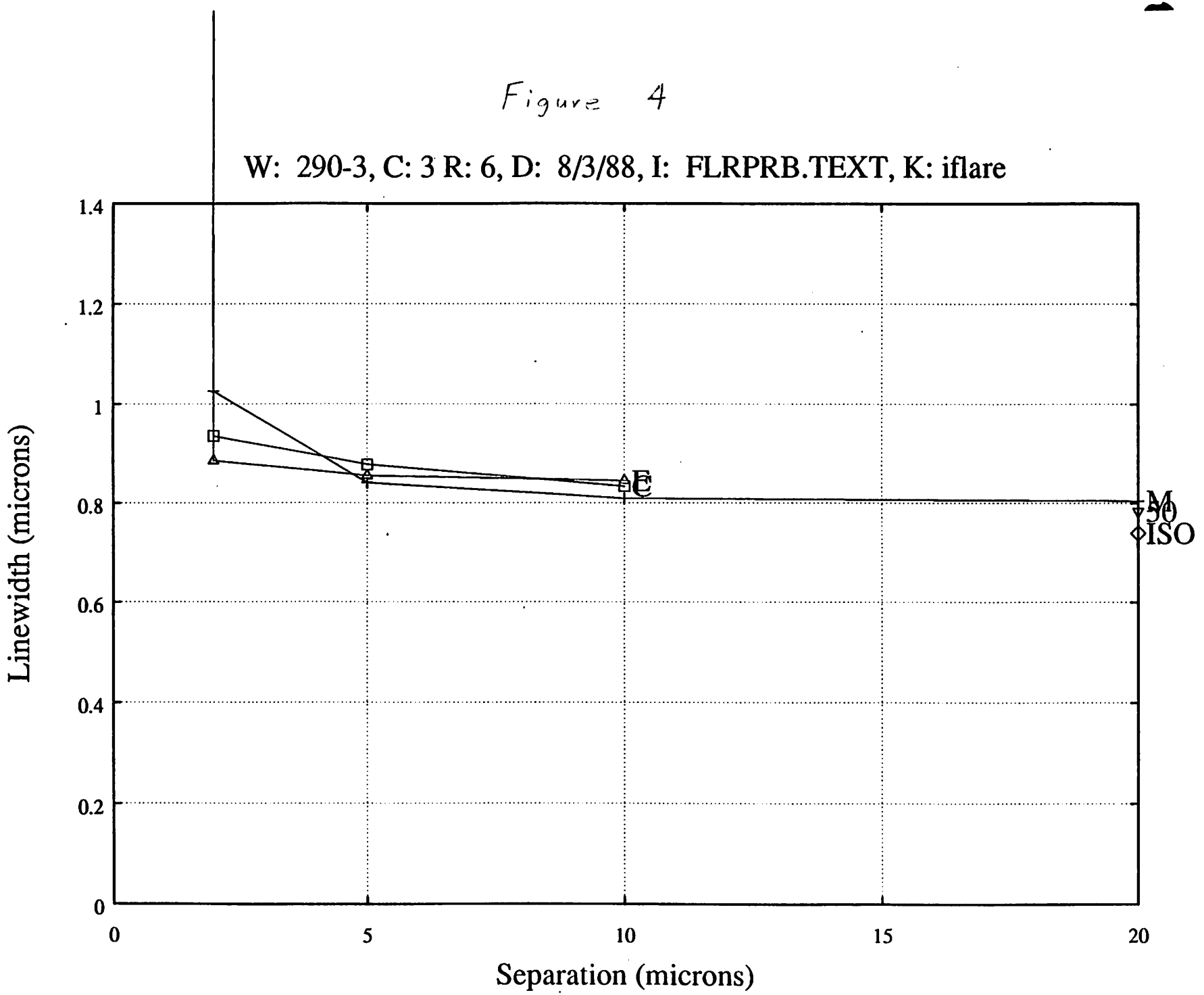
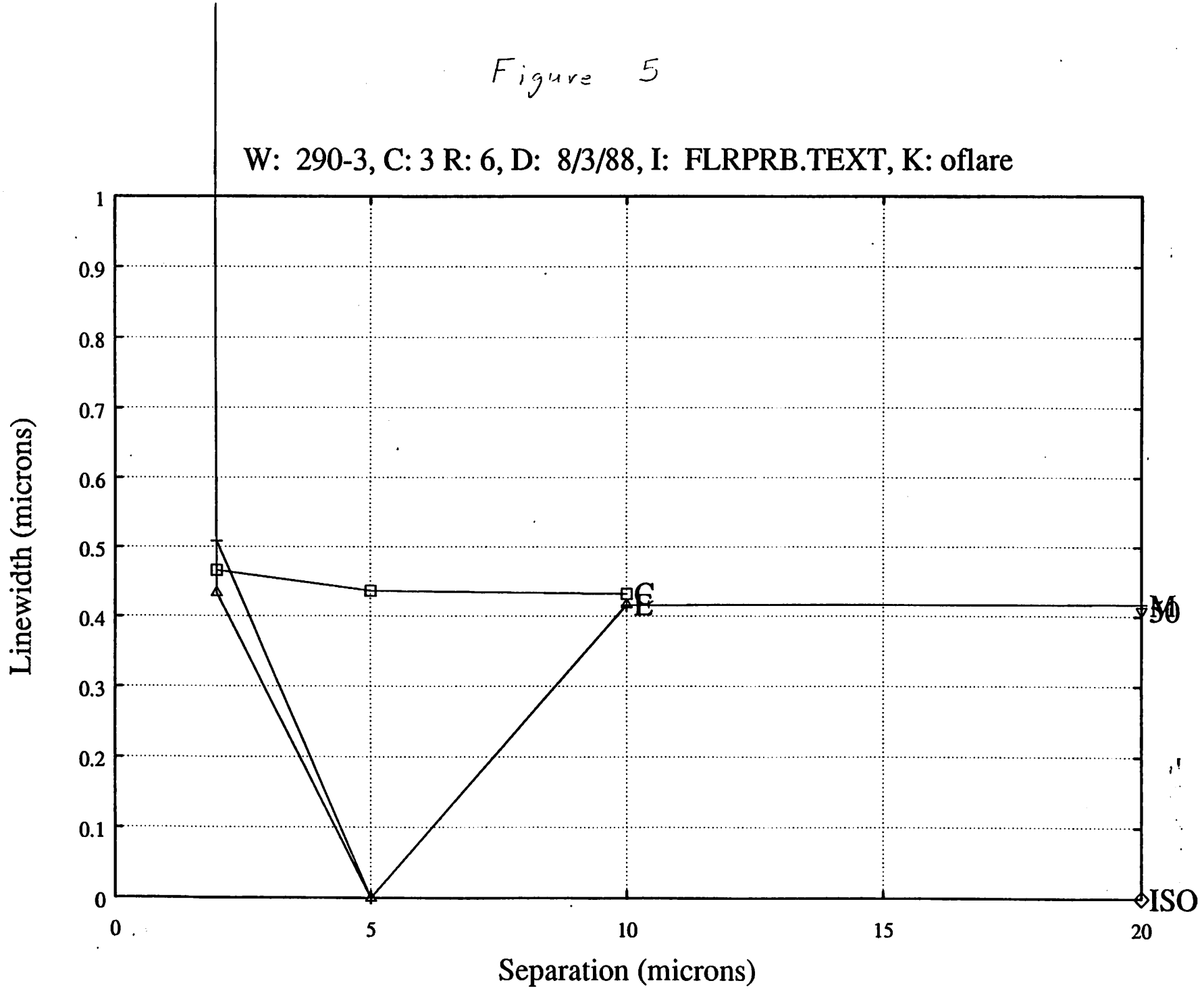
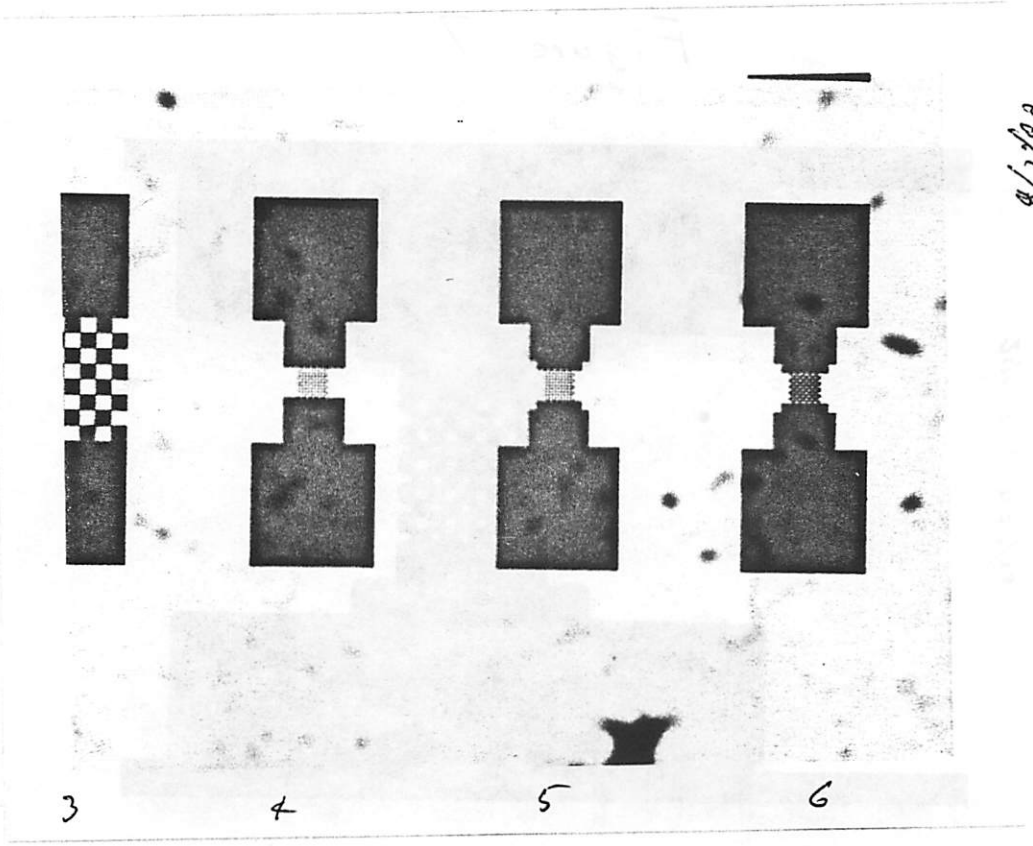

W: 290-3, C: 3 R: 6, D: 8/3/88, I: FLRPRB.TEXT, K: iflare
Linewidth (microns)
Separation (microns)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
E
C
M
50
ISO

Figure 5


W: 290-3, C: 3 R: 6, D: 8/3/88, I: FLRPRB.TEXT, K: oflare
Linewidth (microns)
Separation (microns)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
E
M
50
ISO



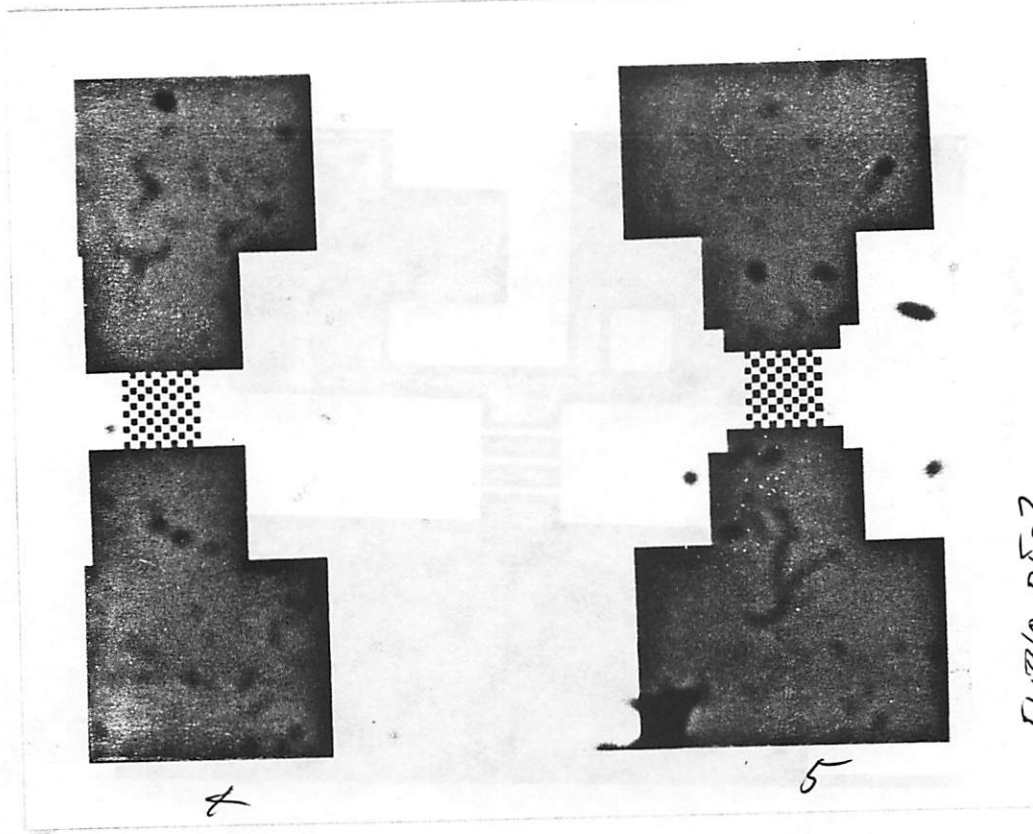
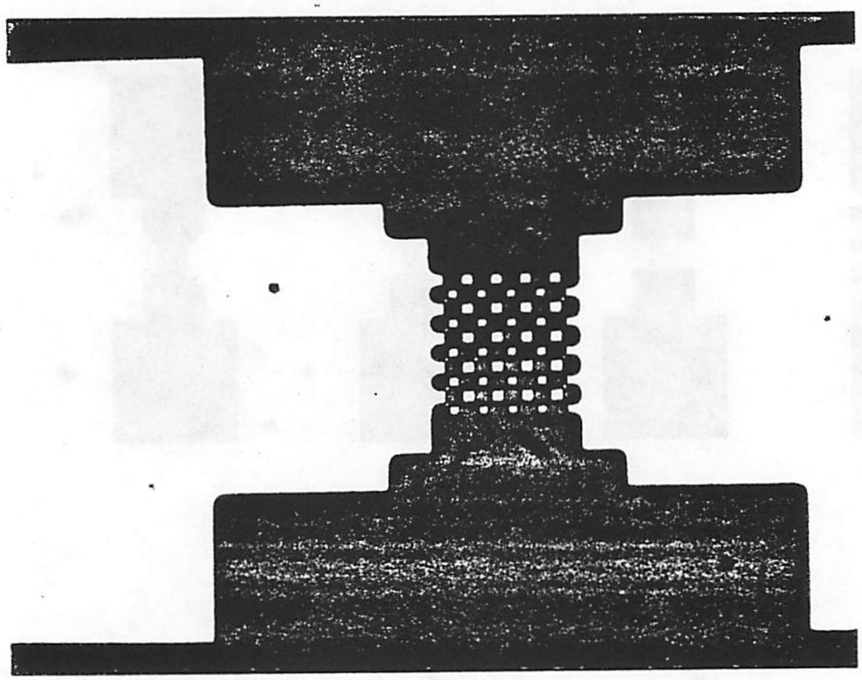


Figure 6

Figure 7



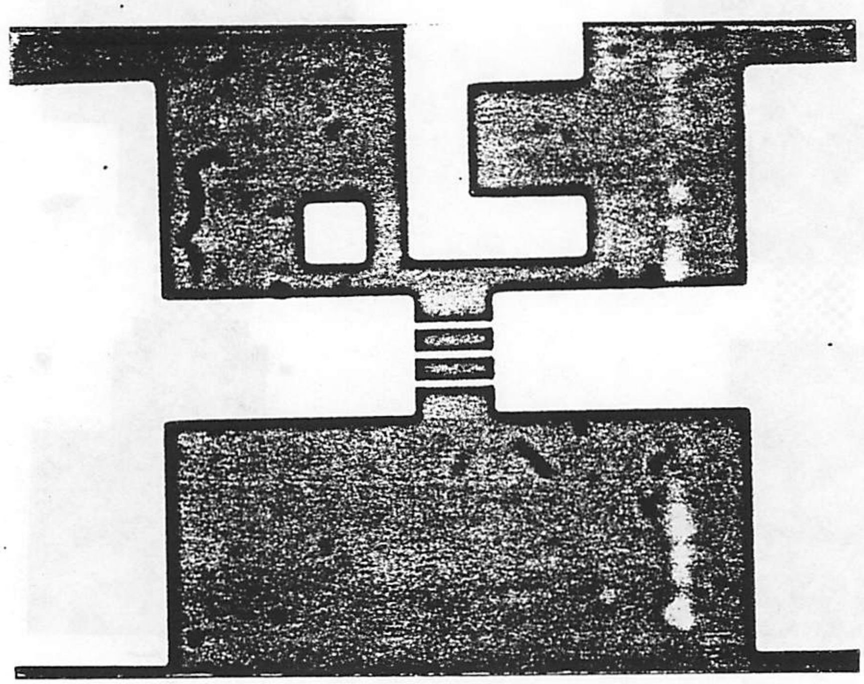


Figure 8

W: 290-3, D: 8/2/88, I: EXPRB.TEXT, K: foc0.8

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	1	0	0	0	0	0
R2	1	1	0	0	0	0	0
R3	1	1	0	0	0	0	0
R4	1	1	1	0	0	0	0
R5	1	1	1	0	0	0	0
R6	1	1	1	0	0	0	0
R7	1	1	1	0	0	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	568	846	999999	999999	999999	999999	999999
R2	1038	2218	999999	999999	999999	999999	999999
R3	556	652	999999	999999	34557	999999	999999
R4	563	677	1204	999999	999999	999999	999999
R5	527	689	672	999999	999999	999999	999999
R6	627	632	959	999999	999999	999999	999999
R7	423	660	1482	999999	999999	27034	999999

W: 290-3, D: 8/2/88, I: EXPRB.TEXT, K: foc1.0

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	1	1	1	0	0	0
R2	1	1	1	1	0	0	0
R3	1	1	1	1	1	0	0
R4	1	1	1	1	1	1	0
R5	1	1	1	1	1	1	0
R6	1	1	1	1	1	1	0
R7	1	1	1	1	1	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	868	1207	2234	1516	999999	999999	999999
R2	1783	2151	1632	728	999999	999999	999999
R3	519	1246	866	1367	1284	999999	999999
R4	496	574	808	840	1506	2155	999999
R5	670	900	848	1174	670	932	999999
R6	1055	689	1079	1492	771	2189	999999
R7	621	2070	1040	2022	1641	999999	999999

W: 290-3, D: 8/2/88, I: EXPRB.TEXT, K: exp6

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	1	1	1	1	1	1
R2	1	1	1	0	0	0	0
R3	1	1	1	0	0	0	0
R4	1	1	0	0	0	0	0
R5	1	1	0	0	0	0	0
R6	1	1	1	0	0	0	0
R7	1	1	1	1	0	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	820	589	579	1323	488	1219	1738
R2	370	1015	1295	999999	999999	999999	999999
R3	377	516	892	999999	999999	999999	999999
R4	466	656	999999	999999	999999	999999	999999
R5	389	459	999999	999999	999999	999999	999999
R6	520	481	830	999999	999999	999999	999999
R7	501	932	1266	1401	999999	999999	999999

W: 290-3, D: 8/2/88, I: EXPRB.TEXT, K: exp7

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	1	0	0	0	0	0
R2	1	0	0	0	0	0	0
R3	0	0	0	0	0	0	0
R4	0	0	0	0	0	0	0
R5	0	0	0	0	0	0	0
R6	0	0	0	0	0	0	0
R7	0	0	0	0	0	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	588	822	999999	999999	999999	999999	999999
R2	1124	999999	999999	999999	999999	999999	999999
R3	999999	999999	999999	999999	999999	999999	999999
R4	999999	999999	999999	999999	999999	999999	999999
R5	999999	999999	999999	999999	999999	999999	999999
R6	999999	999999	999999	999999	999999	999999	999999
R7	999999	999999	999999	999999	999999	999999	999999

W: 290-3, D: 8/9/88, I: DEF3PRB.TEXT, K: 3B0.3u

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	0	0	0	0	0	0
R2	1	0	0	0	0	0	0
R3	1	0	1	0	0	0	0
R4	1	1	0	0	0	0	0
R5	1	0	0	0	0	0	0
R6	1	1	0	0	0	0	0
R7	1	1	1	0	0	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	2325	5441	4179	9962	7788	31639	45157
R2	2395	5855	17207	3610	3300	4847	20548
R3	2175	3128	2567	4755	999999	999999	5665
R4	2347	2914	999999	999999	999999	999999	999999
R5	2145	3420	999999	999999	999999	999999	999999
R6	1691	2368	999999	999999	999999	999999	999999
R7	1199	1572	1926	999999	999999	999999	999999

W: 290-3, D: 8/9/88, I: DEF3PRB.TEXT, K: 3B0.5u

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	0	0	0	0	0	0	0
R2	0	0	0	0	0	0	0
R3	1	0	0	0	0	0	0
R4	1	0	0	0	0	0	0
R5	1	1	0	0	0	0	0
R6	1	1	1	0	0	0	0
R7	1	1	0	1	0	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	5768	3791	43014	13724	16411	78586	665705
R2	3142	4351	3494	70897	5838	5129	41531
R3	1529	3499	3564	3525	999999	999999	20311
R4	2661	3472	999999	999999	999999	999999	999999
R5	2001	2351	999999	999999	999999	999999	999999
R6	1784	2593	2822	999999	999999	999999	999999
R7	1718	1944	3219	2134	4694	999999	999999

W: 290-3, D: 8/9/88, I: DEF3PRB.TEXT, K: 3B0.6u

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	1	0	0	0	0	0
R2	1	0	0	0	0	0	0
R3	1	1	0	0	0	0	0
R4	1	0	0	0	0	0	0
R5	1	1	1	0	0	0	0
R6	1	1	1	0	0	0	0
R7	1	1	1	1	1	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	2783	2863	3245	5621	3040	52868	47322
R2	2726	13004	165746	4547	6173	4980	35055
R3	1598	2510	3850	4841	5459	7651	10409
R4	1656	3755	4525	10220	999999	999999	999999
R5	2255	2095	2765	8793	999999	999999	999999
R6	1779	1719	2085	3116	3274	999999	999999
R7	1293	1291	2266	2409	2911	4111	999999

W: 290-3, D: 8/9/88, I: DEF3PRB.TEXT, K: 3B0.8u

Continuity

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	1	1	1	1	0	0	0
R2	1	1	1	1	0	0	0
R3	1	1	1	0	0	0	0
R4	1	1	0	0	0	0	0
R5	1	1	0	0	0	0	0
R6	1	1	1	0	0	0	0
R7	1	1	1	1	0	0	0

Resistance

	Column						
	C1	C2	C3	C4	C5	C6	C7
R1	2592	1750	2667	2639	4233	13640	291545
R2	1599	2243	2856	2869	3318	4785	19538
R3	2052	2374	2475	3967	7323	4635	10825
R4	1869	2704	4040	4554	5093	4523	4822
R5	1602	2373	3077	3664	4015	6723	8459
R6	1412	2174	2283	4012	3446	5964	9349
R7	1383	2066	2683	2896	6050	4010	3893

Acknowledgement

This work was supported by the SRC-SEMATECH Industrial Consortium on Deep UV Lithography and the California State MICRO program.

Appendix A

Automatic Prober Hardware

Equipment and set-up Procedure
From
Norman Yuen's Thesis

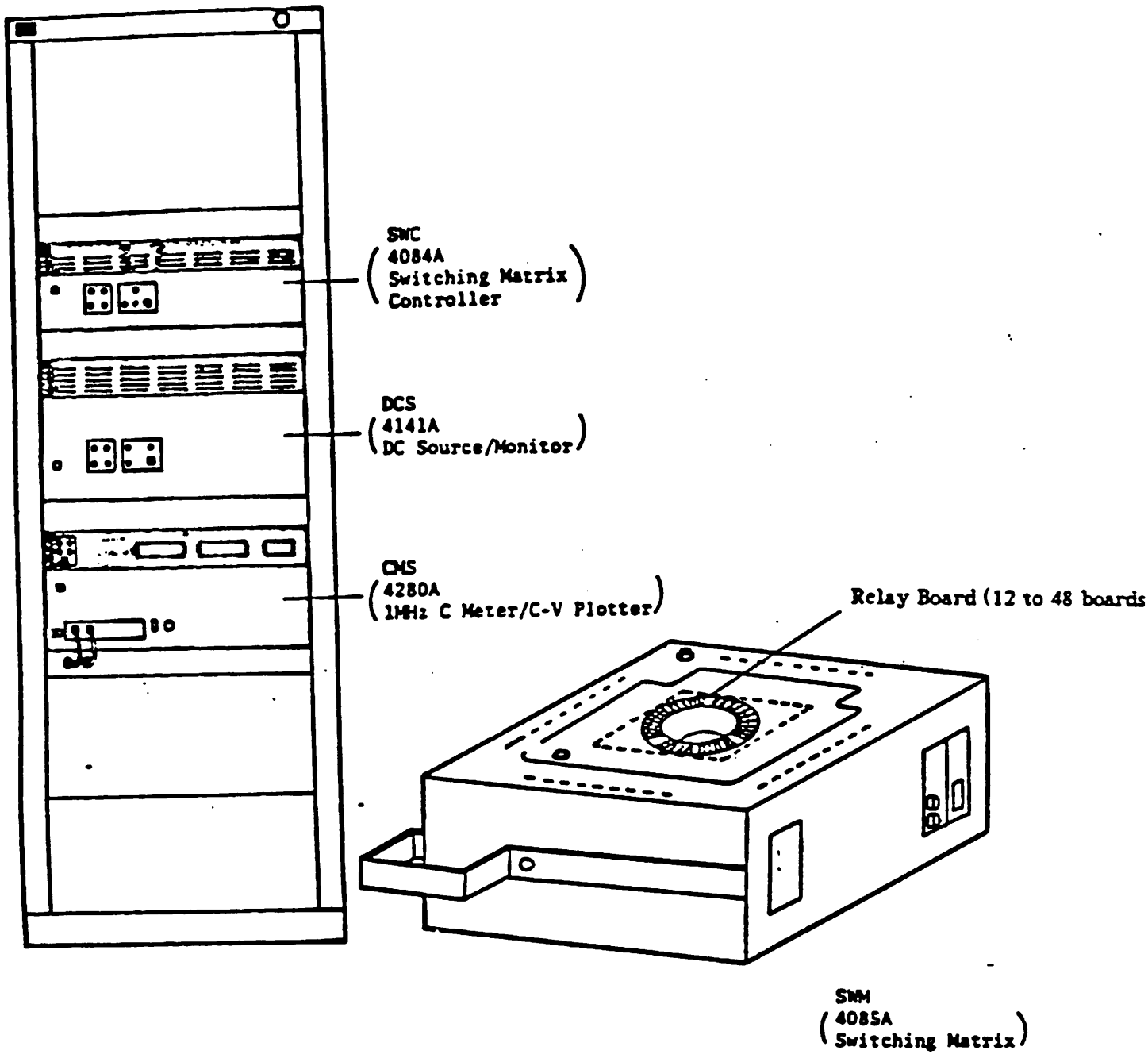
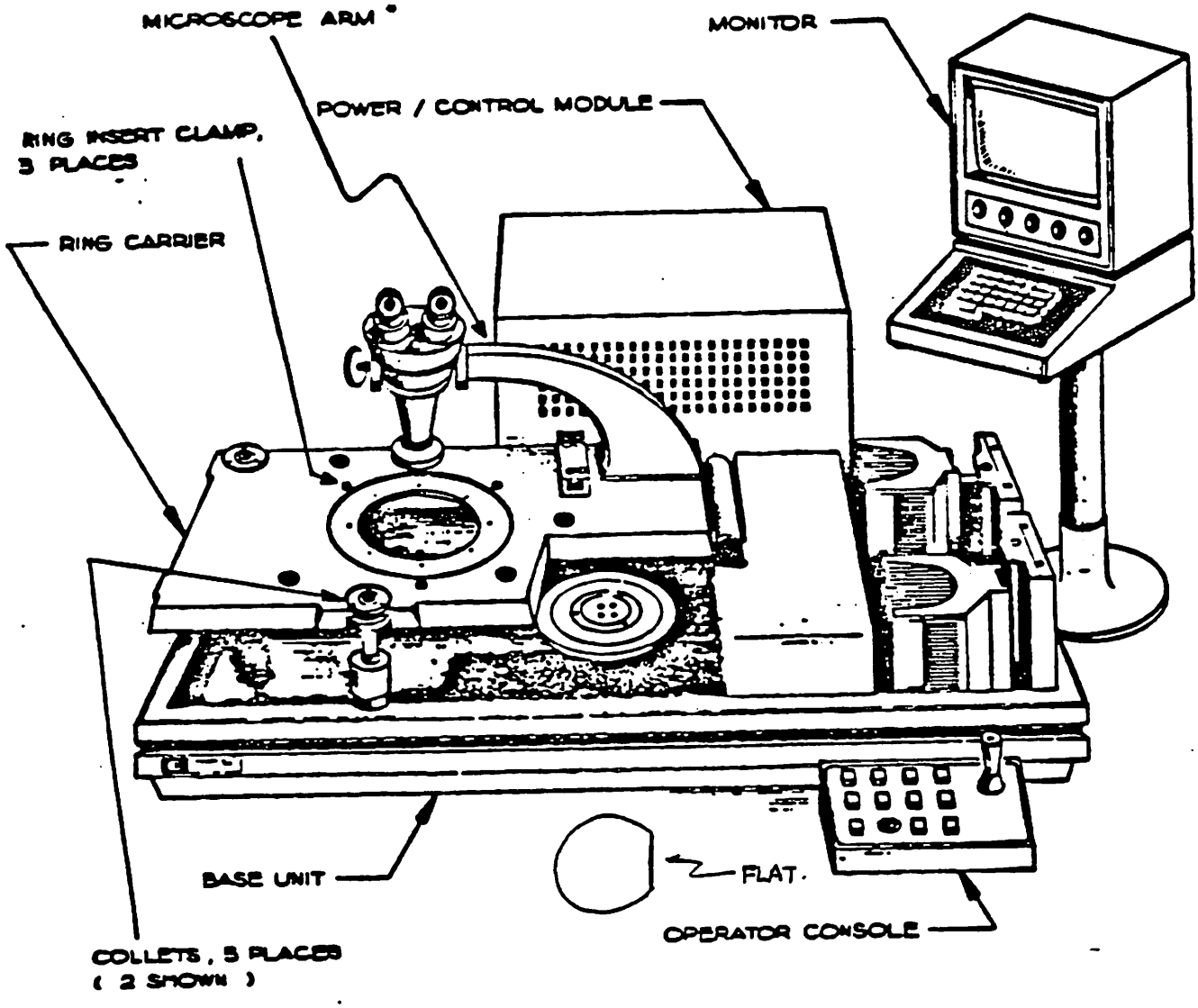


Figure 3-1. HP 4062 system (not to scale)

** NOTE WAFER FLAT ORIENTATION WHEN MOUNTING THE WAFER



*The microscope arm shown will not work with the 4062 system.. The arm must have a rectangular shape so that the microscope can reach over and into the matrix.

Figure 3-2. Electroglas 2001X Automatic Prober

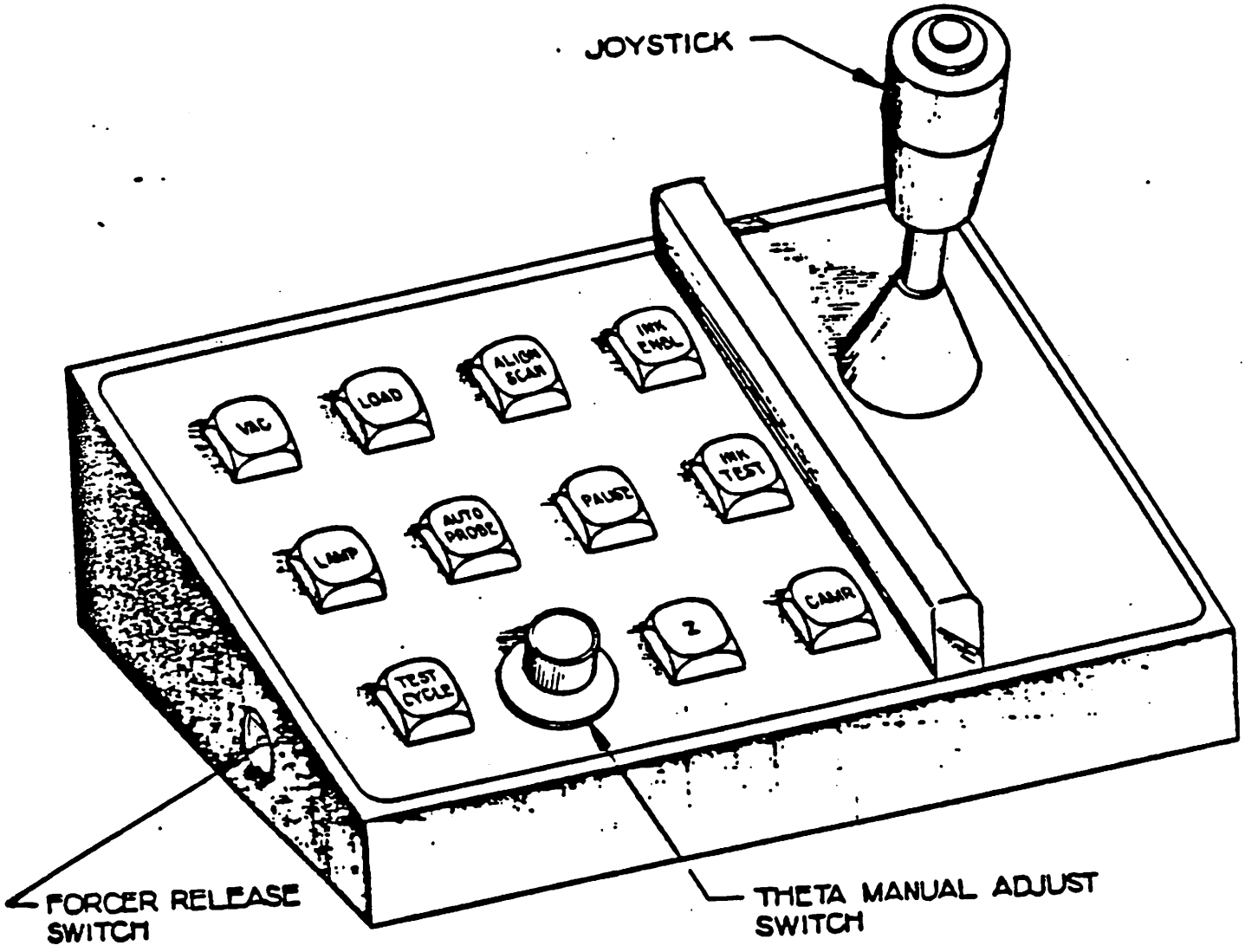


Figure 4-2. Operator Control Console

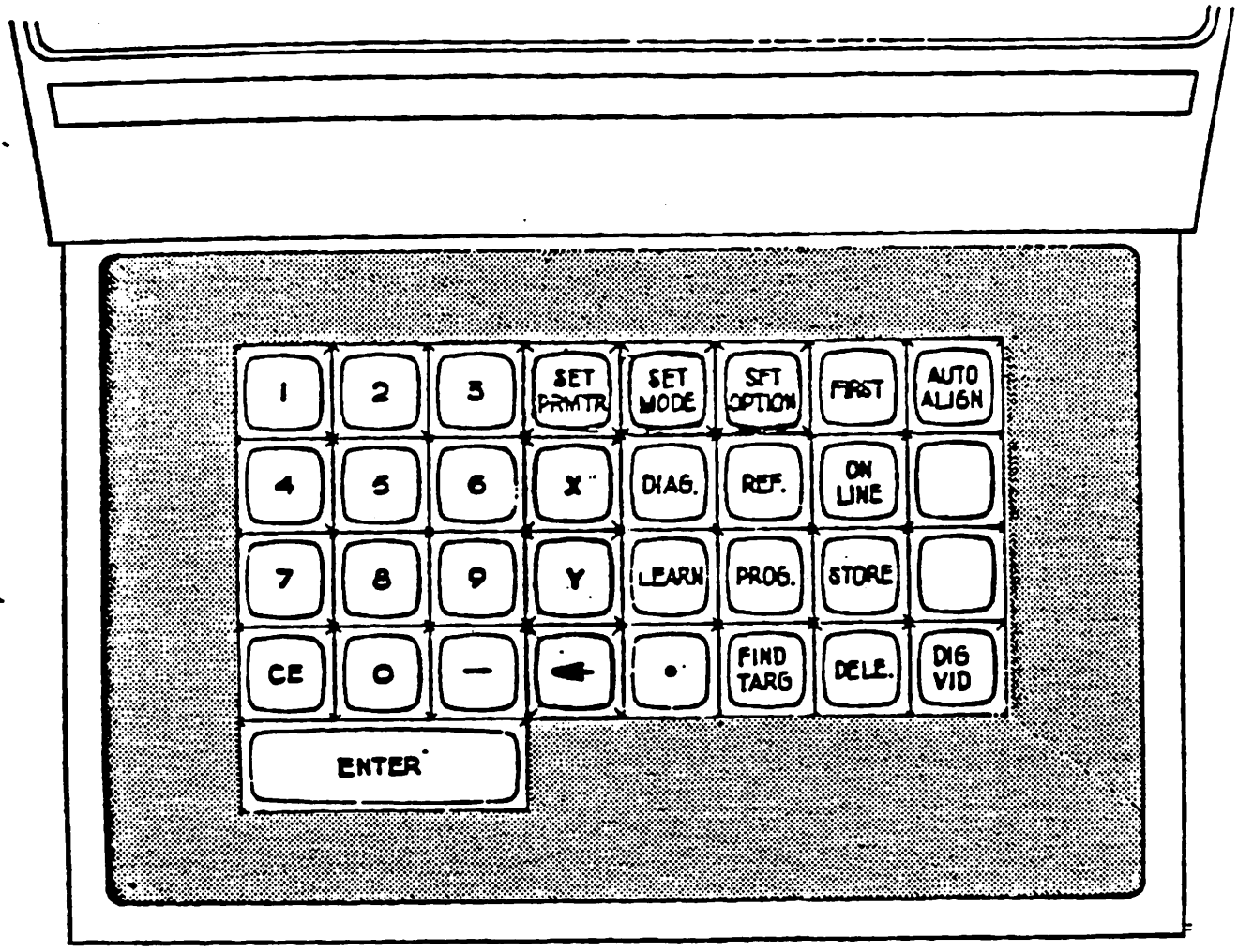


Figure 4-3. Monitor Console Keyboard

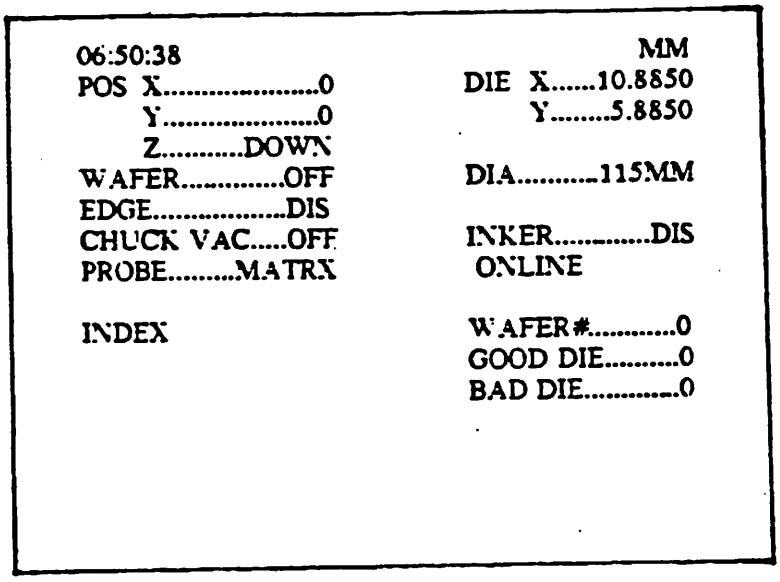


figure 4-4a. Default Display

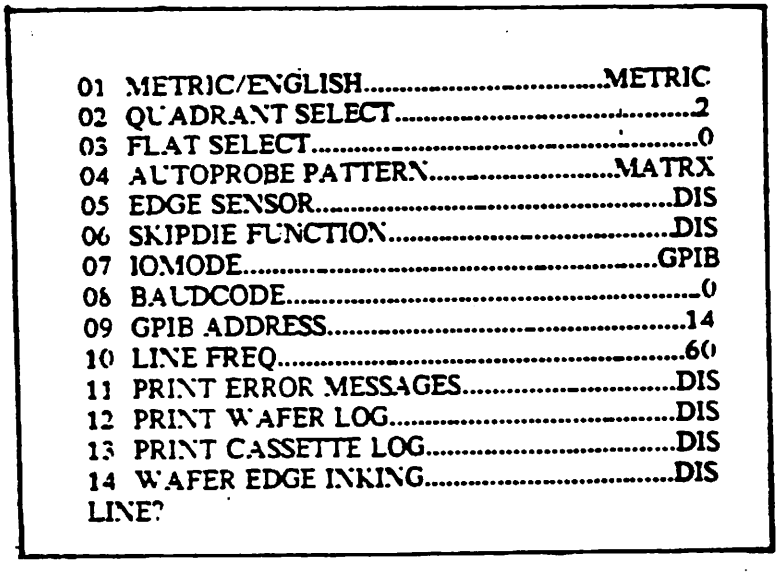


figure 4-4b. SET MODE Display

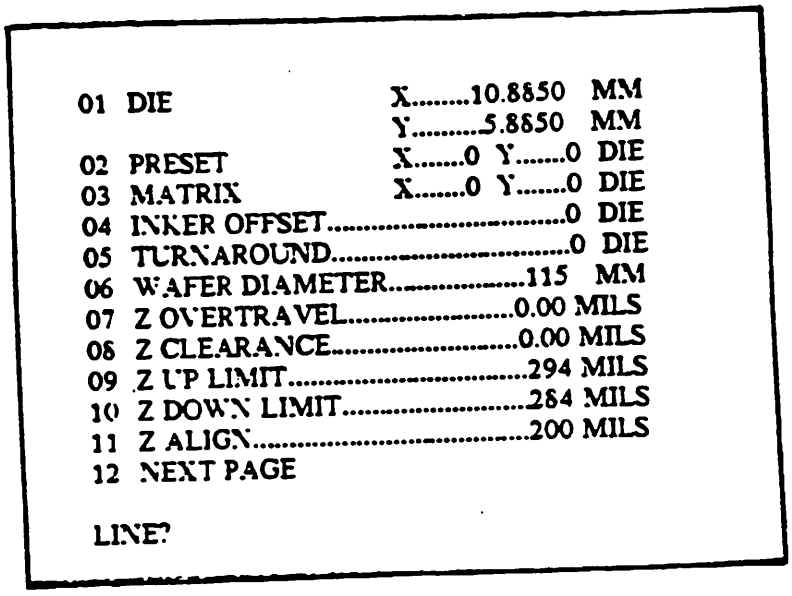


figure 4-4c. SET PRMTR Display

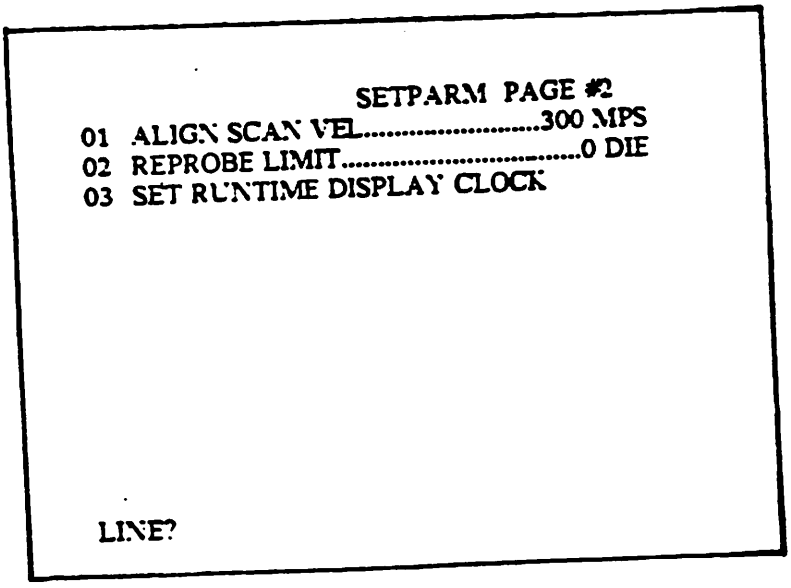


figure 4-4d. SET PRMTR Page 2 Display

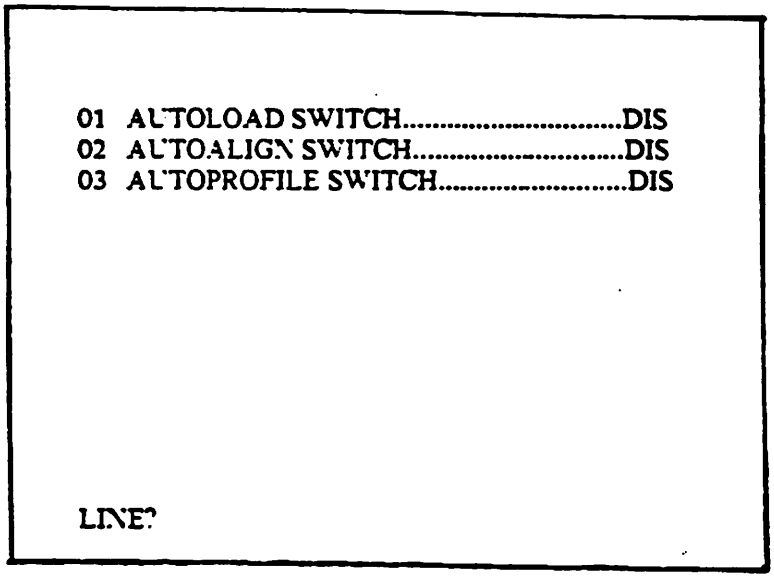


figure 4-4e. SET OPTION Display

so that all these steps will be done automatically upon power-on. An example of an AUTOSTART file is shown in figure 4-1.

4.3. Setting Up The Electroglas 2001X

1) Plug in the appropriate probe card.

2) Turn ON the prober.

3) If the monitor is not already ON, turn it ON.

4) Answer the questions on the monitor or else wait for default response from prober. It takes about 30 seconds for a default response.

Type Message Plus Enter=> Enter key
Wait for Pattern Rec I/O Test... Wait about 30 seconds
Rom Test? Y
Repeat Test? Enter key

5) Now you will see **XY MOTOR BLANK** at the bottom of the screen. The stage is now floating on the platform. Pull it to the front right corner against the 2 edges.

6) Press the button inside the left side of the joystick control panel (figure 4-2). THE PROBER WILL NOT RESPOND UNTIL YOU PRESS THIS BUTTON.

7) Pull out the vacuum level at the front panel. A hissing noise indicates compressed air coming into the probe station.

** 8) Using the monitor console keyboard (figure 4-3), set up the prober as shown in figures 4-4b to 4-4e. Of course, the parameters *DIE, WAFER DIAMETER, ZUP LIMIT*, and *Z DOWN LIMIT* will depend on your particular wafer. To get any of the screens, press the appropriate key on the monitor console keyboard.

9) Place the wafer on the chuck and press [VAC] and [LAMP]. The device pads must be roughly parallel to the probes.

10) Align the wafer (i.e., rotate the chuck until the device pads are perfectly parallel to the probes). The following section describes more fully how to align the wafer.

11) After the wafer is aligned, use the joystick to move the wafer so that the probes are directly over the pads of the devices to be tested. The die should be the origin die and the devices should be the origin row of devices within that die, as specified in the prober file.

12) Press [Z] to raise the wafer to the probes. You may have to reset the *Z UP LIMIT* parameter (see figure 4-4c) in the [SET PRMTR] mode. If the probes are not touching the pads, slightly increase *Z UP LIMIT* and then press [Z] twice (once to lower wafer, once to raise it). When you see that all the probes are barely touching all the pads, increase *Z UP LIMIT* by 0.5 to 1.0 MILS to provide some overdrive. Too much overdrive will wear out the probes and the pads very quickly.

4.3.1. Aligning the Wafer

Aligning the wafer will take some practice. Alignment is done when the wafer (actually the stage holding the wafer) is moving from side to side under the probes.

The idea is to rotate the wafer until the device pads are parallel to the probes. When they are not parallel, then even if the probes are directly over the pads at one end of the wafer, the probes will not be directly over the corresponding pads at the other end of the wafer (figure 4-5).

Initially, the prober should be setup similar to the one shown in figures 4-4b to 4-4e. Some prober parameters will depend on the particular wafer being used. The *WAFER DIAMETER* parameter tells the prober how far the stage should move before reversing the direction of movement. The *ALIGN SCAN VEL* parameter ([SET PRMTR] , page 2) sets how fast the stage will move. Three hundred MPS is a reasonable speed for wafer alignment.

4.3.1.1. Alignment Instructions

Refer to the operator control console (figure 4-2) while reading the following instructions. The rectangularly enclosed words below represent keys on the joystick panel.

1) Follow steps 1 to 9 of section 4.3.

2) Press [ALIGN SCAN].
This causes the stage to move from the right front corner to the right of the probes.

3) Press [PAUSE].
The stage moves to the left of the probes.

4) Turn the joystick until you see "scan" at the lower left column of the monitor. Use the joystick to move the wafer until you see some linear pattern below the probes. You will use this pattern in step 6) to judge whether the probes are parallel to the pads or not. The pattern can be the pads of a row of devices or the edge of the dies.

5) Press [PAUSE].
The stage now moves back and forth. Successively pressing [PAUSE] causes the stage alternately to stop and start moving.

6) As the wafer is moving, look under the microscope and compare the probes to the pattern. If they are not parallel from one end of the wafer to the other, then turn the theta knob, causing the wafer to rotate. Play with the knob until the probes are parallel to the pattern.

7) When you are confident that they are parallel, use the joystick to move the pads to directly underneath the probes. You can use the joystick to move it fast (scan), slow (jog), or fixed (index). How far the stage moves in index mode depends on what the X and Y die sizes are set to (see figure 4-4c).

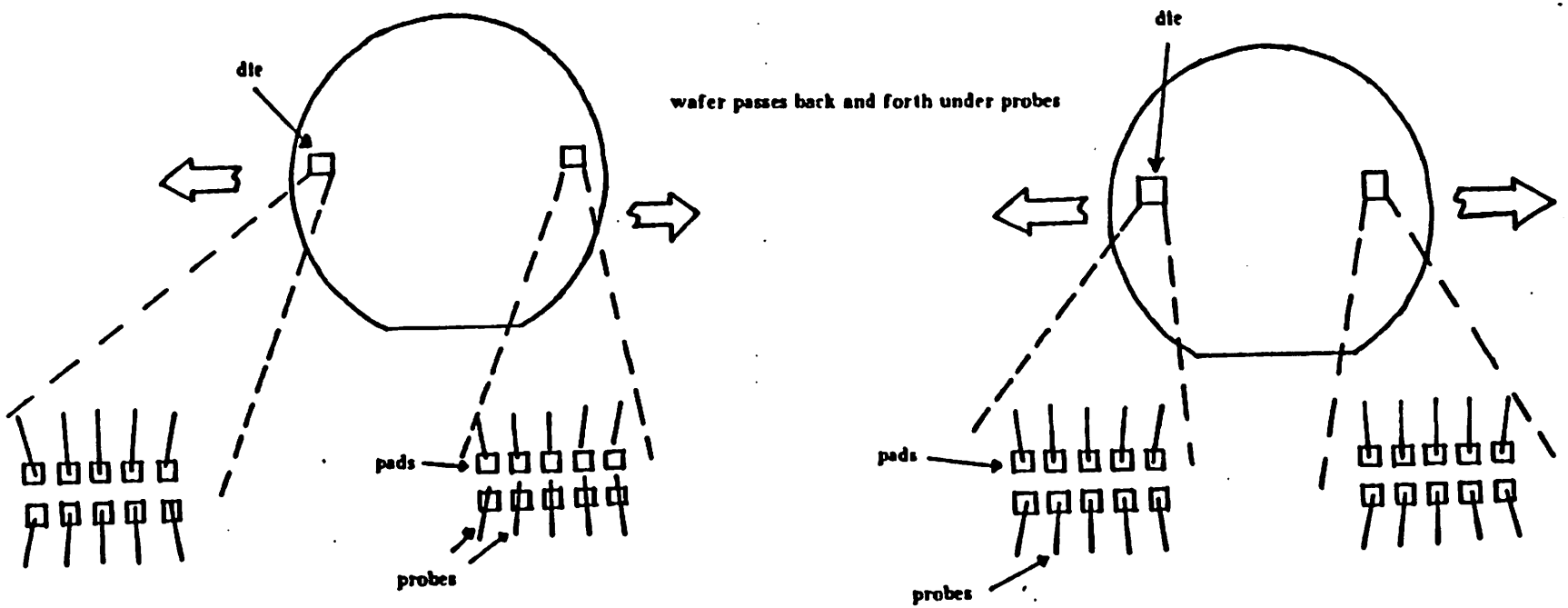


Figure 4-5. Wafer Alignment

Appendix B

AUTOPRB Code

```
$REF 420$
$UCSD$
program bsim (input,output);

import hpib_0, hpib_1, hpib_2, hpib_3,
       general_0, general_1, general_2,
       iodeclarations, iocomasm, dgl_lib;

type shortstring=string[30];
type chuck_pos_type=(Zup, Zdown);
type linestring=string[80];
type source= record
                pin : integer;
                value : real;
            end;
type ivsm= record
                pin : integer;
                kind : char;
                value : real;
            end;
type funtp= record
                fnum : integer;
                keyword : shortstring;
            end;

const
       max_num_devices=100;  {max number of devices per die}


var  device: string[20];
     mode: char;
     process, wafer, lot, date, operator: string[80];
     output_file, prober_file: string[15];
     new_die, end_of_die, initial_jump, end_of_wafer, new_wafer: boolean;
     end_of_program: boolean;
     time_count: integer;

     printer: boolean;
     {PROBER ORIENTED PARAMETERS}
     step_array :array[1..8, 1..8] of char;
                                        {this holds the automatic array}
     number_devices, present_device, number_die, present_die: integer;
     present_diex, present_diey: integer;  {x y die location}
     xdie_size, ydie_size, origin_diex, origin_diey: integer;
     xmult, ymult : integer;

     rnom, rvdp, rmin, rmax : real;

     comment: array[1..max_num_devices] of string[80];

     funcs: array[1..2,1..max_num_devices] of funtp;

     vm1: array[1..max_num_devices] of integer;
     vm2: array[1..max_num_devices] of integer;

     vs1: array[1..max_num_devices] of source;
     vs2: array[1..max_num_devices] of source;

     smu1: array[1..max_num_devices] of ivsm;
     smu2: array[1..max_num_devices] of ivsm;
     smu3: array[1..max_num_devices] of ivsm;
     smu4: array[1..max_num_devices] of ivsm;

     devices: array[1..max_num_devices] of string[10];
     mx, my: array[1..max_num_devices] of integer;
                                     {device types and internal die steps}
```

```
     temporary, userout, userout2: text;


     hpib_controller_isc: integer;  {interface select code of controller}
     hpib_controller_isc_prober: integer;
     hpib_dev_addr: integer;
     hpib_addr_DCS, hpib_addr_SWC, hpib_addr_CMS: integer;
     hpib_addr_prober: integer;
     PROBER: integer;

     continue: char; {move probes to new location, then enter CR to continue}
                    {also user inputs a 'y' or 'n' or CR to tell what to do
                     in prepare_for_iv_graphics procedure}
     colon_position: integer; {colon position in output_file name}
     period_position: integer; {period position in '.TEXT' in output_file name}
     first_time_thru_program: boolean; {allows 'graceful' ending of program
                     if user decides to quit before actually enterring program}
     measure_error: integer; {error flag for any errors encounterred while
                              measuring the device before extraction of
                              parameters}

    minutes_per_device, minutes_per_device_above_thresh,
    minutes_per_device_subth: real;
    number_of_increments, number_of_increments_subth: integer;

    TRACEMD, TRACEselftest, TRACE_linear_ext, TRACE_sat_ext, TRACEiv,
    TRACEpf: boolean;

    DCS, SWC: integer;

procedure init_global_variables;

begin
     hpib_controller_isc:=700;  {isc is 7.  have to X100}
     hpib_addr_SWC:=22;  {4084A matrix controller}
     hpib_addr_DCS:=23;  {4141A DC source/monitor}
     hpib_addr_CMS:=24;  {4280A C meter/C-V plotter}
     hpib_controller_isc_prober:=700;
     hpib_addr_prober:=14;
     PROBER:=hpib_controller_isc_prober + hpib_addr_prober;
end;  {proc init_global_variables}

procedure bsim_timer(var present_increment:integer);

{const minutes_per_device=1.6;
      number_of_increments=37;}

var  minutes_to_die_completion,minutes_to_wafer_completion:real;
     i:integer;

begin
     minutes_to_die_completion:=minutes_per_device*(number_devices-
                                present_device+(number_of_increments-
                                present_increment)/number_of_increments);
     minutes_to_wafer_completion:=(number_die-present_die)*number_devices*
                             minutes_per_device+minutes_to_die_completion;

     {here is where the speedometer is written}
     if present_increment=1 then
          begin
               gotoxy(27,11);
               write('                                          ');
          end;
     gotoxy(27,11);
     for i:=1 to present_increment do
```

```
          write('X');
     present_increment:=present_increment+1;
     {here is where the time values are written}
     gotoxy(26,10);
     write('      ');
     gotoxy(68,10);
     write('      ');
     gotoxy(26,10);
     write(minutes_to_die_completion:1:1);
     gotoxy(68,10);
     write(minutes_to_wafer_completion:1:1);
     gotoxy(21,23);
end;  {end of bsim timer}

procedure pause;

var i: integer;

begin
     for i:=1 to 300000 do;  {takes about 5 seconds}
end;

{************************************************************************}
{from miscfunc.TEXT}

procedure device_self_test (hpib_dev_addr: integer);

const endbit=2;  {end bit position in status byte}
      failbit=5; {fail bit position in status byte}

var result:   string[80];
    bitset:   boolean;
    sbyte, device:   integer;

begin {proc device_self_test}
     bitset:=false;
     device:=700+hpib_dev_addr;
     if hpib_dev_addr=hpib_addr_DCS then begin
          writestring (device, 'ID;');
          readstring (device, result);
          writeln ('the device ID is ', result);
          writeln ('.....4141A self testing.....');
          writestring (device, 'TS;'); {DCS start self test}
     end
     else if hpib_dev_addr=hpib_addr_SWC then begin
               writestring (device, 'IDEX');
               readstring (device, result);
               writeln ('the device ID is ',result);
               write ('.....install the relay test ');
               write ('adapter (16075A) and hit ');
               write ('enter.....>');
               readln;
               writeln ('.....4084A self testing.....');
               writestring (device, 'TSEX');   {SWC start self test}
          end;
     while not bitset do begin
          sbyte:=spoll(device);
          bitset:=(bit_set(sbyte, endbit) or bit_set(sbyte, failbit));
     end;
     writeln ('status byte is ', sbyte:1);
     if hpib_dev_addr=hpib_addr_DCS then writestring (device, 'TR;')
          {get self test result for DCS}
     else if hpib_dev_addr=hpib_addr_SWC then writestring (device, 'TREX');
                    {get self test results for SWC}
     readstring (device, result);
     writeln ('the self test result is ', result);
```

```
     write ('the status byte after the next spoll is ');
     writeln (spoll(device):1);
end; {proc device_self_test}


{*************************************************************************
*************************STRING CONVERSION ROUTINES**********************
*************************************************************************}


function realtostr:linestring;
     {this function will read a real number string, reading only legitimate
      characters from the user}

var  buffer:string[80];
     value:char;
     i,j,max:integer;
     number:real;

begin
    realtostr:='0.0';
    i:=1;
    j:=1;
    readln(buffer);
    max:=strlen(buffer);
    if max <> 0 then
        realtostr:='                                                    ';
        for j:=1 to max do begin
             value:=buffer[j];
             if ((value>='0') and  (value<='9')) or (value='.') or (value='E')
                       or (value='e')or (value='+') or (value='-') then begin
                  realtostr[i]:=value;
                  i:=i+1;
             end;
        end;
end;


function digit(number:char):integer;
      {this function returns an integer given a character}
begin
     if (number>'9') or (number<'0') then begin
          gotoxy(0,22);
          writeln('ERROR in function(digit).  Erroneous number = ',number);
     end;
     digit:=ord(number)-ord('0');   {this is the new function}
end;  {end of function digit}


function strtoreal(realstring:linestring):real;
     {this function takes a real string and converts it into the
      corresponding number}

var  filestring : linestring; {string that will be treated as a file}
     pos : integer; {of no earthly value, required by I/O call}
     realnumber : real; {intermediate storage place before put into strtoreal}

begin
     if (strlen(realstring)<>0) then begin
          setstrlen(filestring,0); {clear out the 'file' filestring}
          strwrite(filestring,1,pos,realstring);
                                          {write realstring into filestring}
          strread(filestring,1,pos,realnumber);
                                    {read filestring into realnumber as a real}
          strtoreal:=realnumber;
                        {put the real number into our function name- strtoreal}
```

```
     end
     else
        strtoreal:=0.0;  {in case only a CR & LF are enterred}
end; {function strtoreal}

{************************************************************}
{***********HPIB TALK, LISTEN & WAIT PROCEDURES**************}
{************************************************************}


procedure talk_to_hpib(var hpib_address:integer);
      {this procedure sets up the hp9836 to talk to the hpib.  hpib_address
       is the current hpib address.}
begin
    untalk (7);
    unlisten (7);
    talk (7, my_address(7));
       {my_address returns the HP-IB address of this (i.e., 7)
        HP-IB interface.  It returns the talker address for
        proc talk.  The first 7 is the controller's isc.}
    listen (7, hpib_address);
       {The 7 is the controller's isc; the 2nd parameter is
        the listener address.}
end; {procedure talk_to_hpib}


procedure listen_to_hpib(var hpib_address:integer);
     {this procedure sets up the hp9836 to listen to the hpib.
      hpib_address is the current hpib address.}
begin
    untalk (7);
    unlisten (7);
    talk (7,hpib_address);
    listen (7, my_address(7));
end; {procedure listen_to_hpib}


procedure wait_till_hpib_ready(var hpib_address:integer);
    {this procedure tells the hp9836 to wait until hpib is
     ready. hpib_address is the current hpib address.}
    {last change made: JUNE 21, 1984  by: JRP}

var statusbyte:integer; {status of serial poll of hpib}

begin
    statusbyte := spoll (700 + hpib_address);
    while statusbyte <> 1 do {keep polling until statusbyte=1}
        statusbyte := spoll (700 + hpib_address);
end; {procedure wait_till_hpib_ready}


procedure wait_till_bit12_IOSTATUS_set;
    {this procedure polls bit 12 of the IOSTATUS register
     until it is set to '1'}
var bitset:boolean;

begin
         {not(bit_set(IOSTATUS(7,7),12)) returns the not-ready-for-data-bit}
    bitset:=false;
    while not(bitset) do
        bitset:=not(bit_set(IOSTATUS(7,7),12));
end;
```

```
{*************************************************************************
*****************************USER INPUT FUNCTIONS*************************
*************************************************************************}


procedure selection_input (xpos, ypos: integer; char1, char2: char;
                           var selection:char);
begin
    gotoxy(xpos,ypos);
    read (selection);
    while (selection<char1) or (selection>char2) do
        begin
            gotoxy(xpos,ypos);
            write(' ');
            gotoxy(xpos,ypos);
            read(selection);
        end;
    gotoxy(xpos-3,ypos);
    write('    ');
    gotoxy(xpos-3,ypos);
    write(selection);
end;

{*************************************************************************
*****************************MAIN MENU ROUTINES***************************
*************************************************************************}


procedure initial_bsim_page;
     {this procedure shows the initial bsim information and takes the user
      inputs}
begin
     writeln(#12);  {clears the screen}
     gotoxy(12,0);
     write('AUTOMATIC PROBER CONTROL PROGRAM');
     gotoxy(20,1);
     write('UC BERKELEY 1988 VERSION 1.0');
     gotoxy(0,2);
     writeln('This Program can be used in either of the following modes:');
     writeln('[1] Fully Automatic or [4] quit! ');
     writeln;
     write('FULLY AUTOMATIC OPERATION requires a prober file, and');
     writeln(' tests all devices ');
     write('in the file without interuption. This mode requires an');
     writeln(' automatic prober.');
     writeln;
     gotoxy(0,14);
     write('Select a Mode of Operation  >');
     gotoxy(32,14);
     write('[1]:FULLY AUTOMATIC');
     gotoxy(32,17);
     write('[4]:EXIT PROGRAM');
     selection_input(29,14,'1','4',mode);
end;


procedure clear_output_file;
     {this procdure simply clears out the output file.  if the user has chosen
      the same name output file as he did on the previous run, the old one will
      get erased.  this is done so that only the present devices under test
      will get recorded in the output file.  if an output file of the same
      name does not exist, it will be created first (so as not to get an
      operating system error)}
begin
         {suppose the user chooses 'bsimout.TEXT', for example, for his
          output file, quits the program, and then runs it again
```

```
          and chooses 'bsimout.TEXT' again.  now if the original
          bsimout.TEXT was not destroyed by the user, all information
          to be put into bsimout.TEXT in the 2nd run will simply be
          added to what is already in bsimout.TEXT from the first run!
          that's why we must destroy the output file before it gets
          created, and in case it doesn't exist, we will create it
          first, and then destroy it....all to be followed by opening
          the file up and making it writeable!!  are you confused yet?}

    rewrite(userout,'GAMES:PRBLONG.OUT');{create output_file if it doesn't}
    close(userout,'save');      {exist, or open and save if it does}
    reset(userout,'GAMES:PRBLONG.OUT');{now make output_file readable and}
    close(userout,'purge');     {then destroy it}
    rewrite(userout,'GAMES:PRBLONG.OUT'); {now create new output_file and}
    {close(userout,'save');         make it writeable}
end;  {of procedure clear_output_file}


procedure standard_input_display;
     {this procedure prompts the user for all inputs common to
      modes of operation}
begin
     gotoxy(0,2);
     writeln('Process Name=?  >');
     writeln('Lot=?  >');
     writeln('Wafer=?  >');
     writeln('Date=?  >');
     writeln('Operator=?  >');
     writeln('Output File=(short format)?  >');
     writeln('Print Output? (default n) >');
end;


procedure input_standard_values;
     {this procedure handles inputing values common to all modes of operation}
const blanking='                                                              ';

var i:integer;
    pr_option: string[5];

begin
     gotoxy(17,2);  {process name input}
     readln(process);
     if strlen(process)>25 then
          strdelete(process,25,strlen(process)-24);  {deletes extra char.}
     gotoxy(13,2);
     write(blanking);
     gotoxy(13,2);
     write(process);
     gotoxy(8,3);  {lot input}
     readln(lot);
     if strlen(lot)>25 then
          strdelete(lot,25,strlen(lot)-24);  {deletes extra char.}
     gotoxy(4,3);
     write(blanking);
     gotoxy(4,3);
     write(lot);
     gotoxy(10,4);  {wafer input}
     readln(wafer);
     if strlen(wafer)>25 then
          strdelete(wafer,25,strlen(wafer)-24);  {deletes extra char.}
     gotoxy(6,4);
     write('               ');
     gotoxy(6,4);
     write(wafer);
     gotoxy(9,5);  {date input}
```

```
     readln(date);
     if strlen(date)>25 then
          strdelete(date,25,strlen(date)-24);  {deletes extra char.}
     gotoxy(5,5);
     write(blanking);
     gotoxy(5,5);
     write(date);
     gotoxy(13,6);  {operator input}
     readln(operator);
     if strlen(operator)>25 then
          strdelete(operator,25,strlen(operator)-24);  {deletes extra char.}
     if strlen(operator)=0 then
          operator:='Egor';
     gotoxy(9,6);
     write(blanking);
     gotoxy(9,6);
     write(operator);
     gotoxy(30,7)  {output_file input};
     readln(output_file);
     colon_position:=strpos(':',output_file);
     period_position:=strpos('.TEXT',output_file);
     if (period_position<>0) then     {'.TEXT' has been included in file name}
         strdelete(output_file,colon_position+11,period_position
                                                    -colon_position-11)
     else      {'.TEXT' has not been included in file name}
         strdelete(output_file,colon_position+11,strlen(output_file)
                                                    -colon_position-10);
     if strlen(output_file)=0 then
          output_file:='SHORT.OUT';
     gotoxy(12,7);
     write(blanking);
     gotoxy(26,7);
     write(output_file);
     gotoxy(30,8);
     readln(pr_option);
     if (pr_option = 'y') or (pr_option = 'Y') then printer := true
        else printer := false;
     gotoxy(28,8);
     write(blanking);
     gotoxy(28,8);
     if printer then write('Y') else write('N');
end;

procedure automatic_mode_inputs;
     {this procedure handles making the automatic mode display}
var i:integer;

begin
     gotoxy(14,0);
     write('***AUTOMATIC OPERATION***');
     standard_input_display;   {this displays the standard prompts}
     gotoxy(0,12);
     writeln('Prober File=?  >');
     input_standard_values;
     gotoxy(16,12);
     readln(prober_file);
     if strlen(prober_file)>15 then
          strdelete(prober_file,16,strlen(prober_file)-15);
     if strlen(prober_file)=0 then
          prober_file:='TEST.TEXT';
     gotoxy(12,12);
     write('                                                    ');
     gotoxy(12,12);
     write(prober_file);
     gotoxy (0,19);
     writeln('Probing Instructions');
```

```
     writeln('     The prober should be on, and the probes should be down');
     write('on the starting die, starting position.');
     writeln('  (see prober instructions)');
     writeln('HIT a "C" for changes, or any other key to start.  >');
     gotoxy(52,22);
end;



procedure initial_status_inputs;
    {this procedure handles getting the first set of inputs required by one
     of the two modes of operation for measurement}
var change:char;

begin
     change:='c';
     while (change='c') or (change='C') do begin
          writeln(#12);
          automatic_mode_inputs;
          read(change);
     end;  {end of change loop}
     gotoxy(52,22);
     write('READING PROBER FILE');
     clear_output_file;
 end;   {end of procedure to get initial inputs}


 procedure initial_status_display;
      {this procedure handles making the initial status chart which is
       displayed during the progress of an extraction}
 begin
      {writeln(#12); }  {clears the display}
      gotoxy(23,0);
      writeln('***MEASURING STATUS***');
      gotoxy(0,2);
      write('PROCESS=',process);
      gotoxy(0,3);
      write('LOT=',lot);
      gotoxy(0,4);
      write('WAFER=',wafer);
      gotoxy(0,5);
      write('DATE=',date);
      gotoxy(40,5);
      write('XPOS=',present_diex:2,' YPOS=',present_diey:2);
      gotoxy(0,6);
      write('OPERATOR=',operator);
      gotoxy(0,7);
      write('OUTPUT FILE=',output_file);
      gotoxy(0,8);
      write('PROBER FILE=',prober_file);
      gotoxy(0,10);
      write('DEVICE=',devices[present_device],'            ');
      gotoxy(0,11);
      write(comment[present_device],'                                    ');
      gotoxy(0,12);
      write('DEVICE LOCATION ',trunc(mx[present_device]/xmult):4,',',
                               trunc(my[present_device]/ymult):4);
      gotoxy(40,13);
      write('VS1                ');
      gotoxy(40,14);
      write('VS2                ');
      gotoxy(40,15);
      write('VM1                ');
      gotoxy(40,16);
      write('VM2                ');
      gotoxy(40,17);
```

```
      write('SMU1                                    ');
      gotoxy(40,18);
      write('SMU2                                    ');
      gotoxy(40,19);
      write('SMU3                                    ');
      gotoxy(40,20);
      write('SMU4                                    ');
 end;

{*************************** 4062 ROUTINES  ************************}


procedure self_test_4062;

var equipment_to_test: char;

procedure matrix_self_test;

const endbit = 2;
      failbit = 5;

var self_test_result: string[200];
    failure_code, i, nextpos, sbyte: integer;
    bitset: boolean;

begin
     bitset:=false;
     gotoxy (0, 4);
     write ('install the relay test adapter (HP16075A) and hit enter  >');
     readln;
     writestring (DCS, 'CL;');  {make sure the sources are off}
     writestring (SWC, 'TSEX');  {matrix self test code}
     gotoxy (10, 10);
     write ('.......... SWITCHING MATRIX TEST IN PROGRESS ..........');
     gotoxy (20, 12);
     write ('(it takes approximately 45 seconds)');
     writestring (SWC, 'TREX');
     while not bitset do begin
          sbyte:=spoll(SWC);
          bitset:=(bit_set(sbyte, endbit) or
                   bit_set(sbyte, failbit));
     end;
     readstring (SWC, self_test_result);
     if TRACEselftest=true then begin
          gotoxy(0, 15);
          write ('The result of test is');
          gotoxy (10, 16);
          for i:=1 to strlen (self_test_result) do
               if (i mod 50) = 49  then begin
                    writeln (self_test_result[i]);
                    write ('           ');
               end
               else
                    write (self_test_result[i]);
     end;  {if TRACEselftest}
     strread (self_test_result, 3, nextpos, failure_code);
     gotoxy (0, 18);
     if (failure_code = 0) then
          writeln ('switching matrix PASSED self test!')
     else begin
          writeln ('switching matrix FAILED self test!');
          writeln ('The result is');
          gotoxy (10, 4);
          for i:=1 to strlen (self_test_result) do
               if (i mod 50) = 49  then begin
                    writeln (self_test_result[i]);
```

```
                    write ('            ');
               end
               else
                    write (self_test_result[i]);
          gotoxy (0, 21);
          writeln ('SEE USER''S GUIDE FOR EXPLANATION OF MESSAGE');
     end;  {if failure_code=0 .... else begin}
     gotoxy (0, 23);
     write ('Press "ENTER" to return to main menu  >');
     readln;
end;  {proc DCS_self_test}

procedure DCS_self_test;

var self_test_result: string[200];
    failure_code, i, nextpos: integer;

begin
     writestring (DCS, 'TS;');  {start DCS self test}
     gotoxy (6, 10);
     write ('.......... DC SOURCE / MONITOR SELF TEST IN PROGRESS .........');
     writestring (DCS, 'TR;');  {get result}
     readstring (DCS, self_test_result);
     gotoxy (0, 15);
     for i:=1 to 4 do begin
          strread (self_test_result, 3*i, nextpos, failure_code);
          write ('                        SMU ', i:1);
          if failure_code = 0 then
               writeln (' PASSED')
          else
               writeln (' FAILED');
     end;  {for i:=1 to 4}
     gotoxy (0, 23);
     write ('press enter to return to main menu  >');
     readln;
end;  {proc matrix_self_test}

begin   {proc self_test_4062}
     writeln (#12);  {clear screen}
     gotoxy (22, 0);
     write ('*** 4062 SELF TEST ***');
     gotoxy (0,3);
     writeln ('The available diagnostics are for the DC source/monitor ',
              '(HP4141) and');
     writeln ('the switching matrix (HP4085).');
     writeln ('In order to test the switching matrix, place the matrix so ',
              'that the');
     writeln ('matrix pins are facing up.  Then screw the relay test ',
              'adapter (HP16075)');
     writeln ('onto the matrix facing the pins and then you are ready.');
     gotoxy (0, 15);
     write ('Select an option  >');
     gotoxy (24, 15);
     write ('[1]:DC SOURCE / MONITOR');
     gotoxy (24, 16);
     write ('[2]:SWITCHING MATRIX');
     gotoxy (24, 17);
     write ('[3]:EXIT TO MAIN MENU');
     selection_input (19, 15, '1', '3', equipment_to_test);
     writeln (#12);
     case equipment_to_test of
          '1':  DCS_self_test;
          '2':  matrix_self_test;
          '3':  {do nothing, exit to main menu}
     end;
end;  {proc self_test_4062}
```

```
{************************ PROBER ROUTINES ************************}

procedure prober_move(deltax,deltay:integer; Zpos: chuck_pos_type);
     {this routine handles all movement of the prober}
     {the step array size must be previously set}
var  ready:boolean;
     i,spoll_byte,nextposition:integer;
     message:array[1..10] of char;
     deltaxstr,deltaystr:string[6];

begin
       {here is where deltax, and deltay are converted to strings}
  deltaxstr:='      ';
  deltaystr:='      ';
  strwrite(deltaxstr,1,nextposition,deltax:1);
  strwrite(deltaystr,1,nextposition,deltay:1);
       {here is where the trailing blanks are trimmed}
  deltaxstr:=strrtrim(deltaxstr);
  deltaystr:=strrtrim(deltaystr);
  talk_to_hpib(hpib_addr_prober);
       {Z down before move}
  writestringln(PROBER,'ZD');
  wait_till_bit12_IOSTATUS_set;
  writestring(PROBER,'MOX');
  writestring(PROBER,deltaxstr);
  writestring(PROBER,'Y');
  writestringln(PROBER,deltaystr);
  wait_till_bit12_IOSTATUS_set;
       {check for the signal that message is to be sent from prober}
  ready:=false;
  while not(ready) do
       ready:=requested(7);
  spoll_byte:=spoll(PROBER);
             {must do serial poll of prober before it will accept
              any more commands.  If move is successful, prober
              will send back "MC" followed by CR/LF.  See the
              Berkeley CMOS Process - A User's Guide, the paper
              "General purpose Test System for 290 Wafer" p.23}
       {here is where the prober message is read}
  listen_to_hpib(hpib_addr_prober);
  i:=1;
  repeat
       begin
            readchar(7,message[i]);
            i:=i+1;
       end;
  until end_set(7);
       {here is where the ZUP is issued}
  if Zpos=Zup then begin
       talk_to_hpib(hpib_addr_prober);
       writestringln(PROBER,'ZU');
       wait_till_bit12_IOSTATUS_set;
  end;
  untalk(7);
  unlisten(7);
end;   {end of procedure to move prober}


procedure set_die_size(xmicron,ymicron:integer);
     {this procedure is called to set the die size of the wafer.  For internal
      die stepping, the procedure sets the die size at 1 micron}
var  xmicronstr,ymicronstr:string[6];
```

```
    nextposition:integer;

begin
      {here is where the die size strings are setup}
  xmicronstr:='      ';
  ymicronstr:='      ';
  strwrite(xmicronstr,1,nextposition,xmicron:1);
  strwrite(ymicronstr,1,nextposition,ymicron:1);
      {trim the trailing blanks}
  xmicronstr:=strrtrim(xmicronstr);
  ymicronstr:=strrtrim(ymicronstr);
  talk_to_hpib(hpib_addr_prober);
  writestring(PROBER,'SP1X');  {this does die size preset}
  writestring(PROBER,xmicronstr);
  writestring(PROBER,'Y');
  writestringln(PROBER,ymicronstr);
  wait_till_bit12_IOSTATUS_set;
  writestringln(PROBER,'SP2X0Y0');  {this does origin preset}
  wait_till_bit12_IOSTATUS_set;
  untalk(7);
  unlisten(7);
end;   {end of procedure to set die size}



{*********************************************************************}
{******************** MEASUREMENT ROUTINES ***************************}
{*********************************************************************}

procedure clear_and_connect_matrix_pins (device: integer);

 procedure connect(what: shortstring; where: integer);

 var  pinstr: string[15];
      nextpos, startpos: integer;

 begin
   pinstr:= '0';
   if where < 10
      then startpos := 2
      else startpos := 1;
   strwrite(pinstr,startpos,nextpos,where);
   writestring(SWC,what);
   writestring(SWC,pinstr);
 end;


begin
     writestring (SWC, 'CL EX');  {clears current matrix connections}
     if (vm1[device] > 0) then connect('PC5 ON ',vm1[device]);
     if (vm2[device].> 0) then connect('PC6 ON ',vm2[device]);
     if (vs1[device].pin > 0) then connect('PC5 ON ',vs1[device].pin);
     if (vs2[device].pin > 0) then connect('PC6 ON ',vs2[device].pin);
     if (smu1[device].pin > 0) then connect('PC1 ON ',smu1[device].pin);
     if (smu2[device].pin > 0) then connect('PC2 ON ',smu2[device].pin);
     if (smu3[device].pin > 0) then connect('PC3 ON ',smu3[device].pin);
     if (smu4[device].pin > 0) then connect('PC4 ON ',smu4[device].pin);
     writestring (SWC, 'EX');         {terminator}
end;  {proc clear_and _connect_matrix_pins}


procedure measure_device(var error_code: integer; device: integer);

 var measurement, v1, v2, vsmu2, ismu3, tres : real;
     i : integer;
```

```
  procedure smu_source(port: integer; info: ivsm);

  var portstr: shortstring;
      endpos: integer;

  begin
    case info.kind of
      'i','I': begin
                 writestring (DCS, 'DI');
                 portstr := ' ';
                 strwrite(portstr,1,endpos,chr(port+ord('0')),', 1, ');
                 writestring (DCS, portstr);
                 writenumber (DCS, info.value);
                 writestring (DCS, ', 20;');
               end;
      'v','V': begin
                 writestring (DCS, 'DV');
                 portstr := ' ';
                 strwrite(portstr,1,endpos,chr(port+ord('0')),', 0, ');
                 writestring (DCS, portstr);
                 writenumber (DCS, info.value);
                 writestring (DCS, ', 0.020;');
               end;
      otherwise;
    end;
  end;

  procedure smu_measure(port: integer; info: ivsm);

  var measurement: real;
      portstr: shortstring;
      endpos: integer;

  begin
    case info.kind of
      'i','I': begin
                 writestring (DCS, 'TV');
                 portstr := ' ';
                 strwrite(portstr,1,endpos,chr(port+ord('0')));
                 writestring (DCS, portstr);
                 writestring (DCS, ';');
                 readnumber  (DCS, measurement);
                 writeln(userout, info.pin:4, measurement:21:5,
                                              info.value:23:7);
                 write(info.pin:4,measurement:10:5,info.value:13:7);
                 if port = 2 then vsmu2:= measurement;
                 if port = 3 then ismu3:= info.value;
               end;
      'v','V': begin
                 writestring (DCS, 'TI');
                 portstr := ' ';
                 strwrite(portstr,1,endpos,chr(port+ord('0')));
                 writestring (DCS, portstr);
                 writestring (DCS, ';');
                 readnumber  (DCS, measurement);
                 writeln(userout, info.pin:4, info.value:21:5,
                                              measurement:23:7);
                 write(info.pin:4,info.value:10:5,measurement:13:7);
                 if port = 2 then vsmu2:= info.value;
                 if port = 3 then ismu3:= measurement;
               end;
      otherwise;
    end;
  end;

begin
```

```
clear_and_connect_matrix_pins(device);
DCS := 723;
writeln(userout);
writeln(userout,'Pin #              Voltage                  Current');
writestring (DCS, 'CL;');
writestring (DCS, 'IT2;');
if (vs1[device].pin > 0) then
   begin
     writestring (DCS, 'DV5, 1, ');
     writenumber (DCS, vs1[device].value);
     writestring (DCS, ', 0.020;');
     writeln(userout, vs1[device].pin:4, vs1[device].value:21:5);
     gotoxy(44,13);
     write(vs1[device].pin:4,vs1[device].value:10:5);
   end;
if (vs2[device].pin > 0) then
   begin
     writestring (DCS, 'DV6, 1, ');
     writenumber (DCS, vs2[device].value);
     writestring (DCS, ', 0.020;');
     writeln(userout, vs2[device].pin:4, vs2[device].value:21:5);
     gotoxy(44,14);
     write(vs2[device].pin:4,vs2[device].value:10:5);
   end;
if (smu1[device].pin > 0) then smu_source (1, smu1[device]);
if (smu2[device].pin > 0) then smu_source (2, smu2[device]);
if (smu3[device].pin > 0) then smu_source (3, smu3[device]);
if (smu4[device].pin > 0) then smu_source (4, smu4[device]);
if (vm1[device] > 0) then
   begin
     writestring (DCS, 'TV5;');
     readnumber  (DCS, measurement);
     v1 := measurement;
     writeln(userout, vm1[device]:4, measurement:21:5);
     gotoxy(44,15);
     write(vm1[device]:4,measurement:10:5);
   end;
if (vm2[device] > 0) then
   begin
     writestring (DCS, 'TV6;');
     readnumber  (DCS, measurement);
     v2 := measurement;
     writeln(userout, vm2[device]:4, measurement:21:5);
     gotoxy(44,16);
     write(vm2[device]:4,measurement:10:5);
   end;
gotoxy(44,17);
if (smu1[device].pin > 0) then smu_measure (1, smu1[device]);
gotoxy(44,18);
if (smu2[device].pin > 0) then smu_measure (2, smu2[device]);
gotoxy(44,19);
if (smu3[device].pin > 0) then smu_measure (3, smu3[device]);
gotoxy(44,20);
if (smu4[device].pin > 0) then smu_measure (4, smu4[device]);
if ismu3 = 0.0 then ismu3 := 1E-18;
if v1 = v2 then v2 := v1*(1 - 1E-6);
if ((v2 = 0) and (v1 = 0)) then v1 := 0.000001;
for i := 1 to 2 do
  begin
    case funcs[i,device].fnum of
      1 : begin
            tres := abs(4.53236014183 * (v1 - v2) / ismu3);
            if ((tres > rmin) and (tres < rmax)) then rvdp := tres;
          end;
      2 : begin
            tres := abs(rvdp * 150E-06 * ismu3 / (v1 - v2));
```

```
                if vsmu2 > 17.5 then tres := 0.0;
              end;
            3 : begin
                  tres := abs(rnom * 150E-06 * ismu3 / (v1 - v2));
                  if vsmu2 > 17.5 then tres := 0.0;
                end;
            4 : if (abs(vsmu2/ismu3) > 3E3) then tres := 0.0
                   else tres := 1.0;
            5 : tres := abs(vsmu2/ismu3);
            otherwise;
          end;
          if funcs[i,device].fnum > 0 then
            begin
              writeln(userout2,funcs[i,device].keyword,
                 present_diex:6,present_diey:2,
                 trunc(mx[device]/xmult):6,trunc(my[device]/ymult):6,
                 '       ',tres);
              writeln(userout,funcs[i,device].keyword,
                 present_diex:6,present_diey:2,
                 trunc(mx[device]/xmult):6,trunc(my[device]/ymult):6,
                 '       ',tres);
              gotoxy(1,17);
              write('                                    ');
              gotoxy(1,17);
              write(funcs[i,device].keyword);
              gotoxy(5,18);
              write('                                   ');
              gotoxy(5,18);
              write(tres);
            end;
        end;
end; {end measure_device}



{*************************************************************************}
{************************AUTOMATIC PROBER ROUTINES************************}
{*************************************************************************}


procedure read_prober_file (var xdie_size, ydie_size,origin_diex, origin_diey,
                            number_die, number_devices:integer);

{read_prober_file performs the following:
   -counts number of dice to be probed
   -counts number of devices per die
   -fills in the arrays mx, my, devices, pd, ps, pg, pb, and
    step_array.
        mx and my contain coordinates of the device; all
          devices in the same section of the die will have
          the same mx and my.  in the prober file, mx and
          my are given only for the first device in that
          section.
        devices is the device type (10 characters)
        step_array contains the 8X8 map of the dice on
          the wafer that are to be probed.  X=origin die,
          1=to be probed, 0=not to be probed.}

var xdie_sizestr, ydie_sizestr: linestring;
    command: string[80];
    command2, command3, kind: char; {place to keep original command[2]}
    probertext: text;
    i, j, pin, npos, fnum: integer;
    value: real;
    mxmap: array[1..max_num_devices] of integer;
                {bit map to signal a new section of die, hence prober
```

```
                movement will be needed.}

begin {proc read_prober_file}
  reset (probertext, prober_file);
  number_die:=0;
  readln (probertext, xdie_sizestr);
  readln (probertext, ydie_sizestr);
  readln (probertext, xmult);
  readln (probertext, ymult);
  xdie_size:=trunc(strtoreal(xdie_sizestr));
  ydie_size:=trunc(strtoreal(ydie_sizestr));
  for j:=1 to 8  do begin   {scan rows, the x pos}
      for i:=1 to 7         {scan colums, y pos}
          do read (probertext, step_array[i,j]);
      readln (probertext, step_array[8,j]); {finish row j}
  end;   {filled in the map of the dice to be probed}
  for j:=1 to 8  do   {scan rows for dice to be probed}
       for i:=1 to 8  do begin   {scan columns}
          if step_array[i,j]<>'0'
               then number_die:=number_die+1;
                     {count the number of dice to be probed}
          if (step_array[i,j]='x') or (step_array[i,j]='X') then begin
              origin_diex:=i; {record origin of die}
              origin_diey:=j;
          end;
       end;   {for i:=1 to 8  do begin, columns}
  number_devices:=0;
  rnom := 33.0;
  rmin := 30.0;
  rmax := 38.0;
  rvdp := 33.0;

  for i:=1 to max_num_devices do
      begin
        mxmap[i]:=0;
        funcs[1,i].fnum:= 0;
        funcs[2,i].fnum:= 0;
        vm1[i]:= 0;
        vm2[i]:= 0;
        vs1[i].pin := 0;
        vs2[i].pin := 0;
        smu1[i].pin := 0;
        smu2[i].pin := 0;
        smu3[i].pin := 0;
        smu4[i].pin := 0;
      end;
       {wherever mx[i]<>0, then the probes need to be moved
        to a new section of the die indicated by mx[i] and
        my[i].  mx and my will be filled while reading
        prober file.  The 0's will be filled in so that all
        devices in a section will have the the same mx[i]
        and my[i].}
  while not(eof(probertext)) do
    begin
      readln (probertext, command);
           {if read only CR/LF on a line, then there is no
            command[1] or command[2]. the length of command
            is then 0.}
      if (strlen(command)<3) {comment or invalid com}
        then command:='zz' {just a filler}
        else command2:=command[2]; {retain command2}
      case command[1] of
        'm','M': begin
                      strdelete (command, 1, 3);
                      if (command2='x') or (command2='X') then begin
                        mx[number_devices+1]:=xmult*trunc(strtoreal(command));
```

```
                mxmap[number_devices+1]:=1;  {a 1 in mxmap denotes a
                                                new section of a die}
              end;
              if (command2='y') or (command2='Y') then
                my[number_devices+1]:=ymult*trunc(strtoreal(command));
            end; {'m','M'}
'd','D': begin
              if (command2='v') or (command2='V') then begin
                   number_devices:=number_devices+1;
                        {number_devices incremented only on dv.}
                  strdelete (command, 1,3);
                  devices[number_devices]:=command;
              end;
            end; {'d','D'}
'#':      comment[number_devices]:= command;
'v','V': begin
           command3 := command[3];
           strdelete(command,1,4);
           case command2 of
             'm','M': begin
                        pin := trunc(strtoreal(command));
                        case command3 of
                          '1': vm1[number_devices]:=pin;
                          '2': vm2[number_devices]:=pin;
                          otherwise;
                        end;
                      end;
             's','S': begin
                        pin := trunc(strtoreal(command));
                        while ((command[1] >= '0') and
                               (command[1] <= '9')) do
                                   strdelete(command,1,1);
                        strdelete(command,1,1);
                        value := strtoreal(command);
                        case command3 of
                          '1': begin
                                 vs1[number_devices].pin := pin;
                                 vs1[number_devices].value := value;
                               end;
                          '2': begin
                                 vs2[number_devices].pin := pin;
                                 vs2[number_devices].value := value;
                               end;
                          otherwise;
                        end;
                      end;
             otherwise;
           end;
         end;
's','S': begin
           command3:= command[4];
           strdelete(command,1,5);
           pin := trunc(strtoreal(command));
           while ((command[1] >= '0') and
                  (command[1] <= '9'))    do
              strdelete(command,1,1);
           kind:= command[2];
           strdelete(command,1,3);
           value := strtoreal(command);
           case command3 of
             '1': begin
                    smu1[number_devices].pin := pin;
                    smu1[number_devices].kind := kind;
                    smu1[number_devices].value := value;
                  end;
             '2': begin
```

```
                    smu2[number_devices].pin := pin;
                    smu2[number_devices].kind := kind;
                    smu2[number_devices].value := value;
                  end;
            '3': begin
                    smu3[number_devices].pin := pin;
                    smu3[number_devices].kind := kind;
                    smu3[number_devices].value := value;
                  end;
            '4': begin
                    smu4[number_devices].pin := pin;
                    smu4[number_devices].kind := kind;
                    smu4[number_devices].value := value;
                  end;
            otherwise;
          end;
        end;
'u','U': begin
          case command2 of
            '4':     begin
                      strread(command,5,npos,
                              smu2[number_devices].pin,
                              smu3[number_devices].pin,
                              vm1[number_devices],
                              vm2[number_devices]);
                      smu2[number_devices].kind := 'i';
                      smu2[number_devices].value := 0.001;
                      smu3[number_devices].kind := 'v';
                      smu3[number_devices].value := 0.0;
                     end;
            'i','I': begin
                      strread(command,4,npos,
                              smu2[number_devices].pin,
                              smu3[number_devices].pin);
                      smu2[number_devices].kind := 'i';
                      smu2[number_devices].value := 0.001;
                      smu3[number_devices].kind := 'v';
                      smu3[number_devices].value := 0.0;
                     end;
            'v','V': begin
                      strread(command,4,npos,
                              smu2[number_devices].pin,
                              smu3[number_devices].pin);
                      smu2[number_devices].kind := 'v';
                      smu2[number_devices].value := 3.0;
                      smu3[number_devices].kind := 'v';
                      smu3[number_devices].value := 0.0;
                     end;
            otherwise;
          end;
        end;
'r','R': begin
          strread(command,4,npos,rnom,rmin,rmax);
          rvdp := rnom;
        end;
'f','F': begin
          strdelete(command,1,5);
          case command2 of
            'v','V': fnum := 1;
            'c','C': fnum := 2;
            'f','F': fnum := 3;
            's','S': fnum := 4;
            '2':     fnum := 5;
            otherwise;
          end;
          if funcs[1,number_devices].fnum = 0
```

```
                    then begin
                           funcs[1,number_devices].fnum := fnum;
                           funcs[1,number_devices].keyword := command;
                         end
                    else begin
                           funcs[2,number_devices].fnum := fnum;
                           funcs[2,number_devices].keyword := command;
                         end;
                end;
        otherwise ; {nop}
      end; {case command[1] of}
    end; {while not eof(probertext)}
  close (probertext, 'SAVE');
  for i:=1 to number_devices do
       if mxmap[i]=0 then begin
             mx[i]:=mx[i-1];
             my[i]:=my[i-1];
       end;   {all devices in same section of die have the same coordinates
          mx and my.  mx[] had been filled with 0's earlier in this procedure.}
end; {proc read_prober_file}


procedure set_up_prober_initial_conditions;
     {this procedure sets up the prober to use microns instead of mils,
      and it sets up the proper coordinate system quadrant (quadrant 2)}

begin
     talk_to_hpib(hpib_addr_prober);
     writestringln(PROBER, 'SM1U1');
     wait_till_bit12_IOSTATUS_set;
     writestringln(PROBER,'SM2Q2');  {sets up quadrant=2}
     wait_till_bit12_IOSTATUS_set;
     untalk(7);
     unlisten(7);
end; {end of procedure to set micron units}


procedure step_to_next_die (var present_diex,present_diey:integer;
                            var initial_jump,end_of_wafer:boolean);

var  i,j:integer;
     next_diex,next_diey:integer;

begin
         {if initial jump then find the first die}
     next_diex:=0;
     next_diey:=0;
     end_of_wafer:=false;
     if initial_jump=true then begin
          for j:=1 to 8  do
               for i:=1 to 8  do
                    if (step_array[i,j]<>'0') and (next_diex=0) then begin
                         next_diex:=i;
                         next_diey:=j;
                    end;
          initial_jump:=false;
     end  {end of handling for initial jump}
     else begin
               {check present row}
          for i:=present_diex+1 to 8  do
               if (step_array[i,present_diey]<>'0') and
                  (next_diex=0) then begin
                             next_diex:=i;
                             next_diey:=present_diey;
               end;
          if next_diex=0 then  {next die not in the same row}
```

```
                for j:=present_diey+1 to 8  do
                     for i:=1 to 8 do
                          if (step_array[i,j]<>'0') and (next_diex=0) then begin
                               next_diex:=i;
                               next_diey:=j;
                          end;
     end;  {if initial_jump ... else begin}
     if next_diex=0 then
          end_of_wafer:=true
     else begin
               {move to next die}
          prober_move(next_diex-present_diex, next_diey-present_diey,Zup);
          present_diex:=next_diex;
          present_diey:=next_diey;
     end;
end;  {end of procedure to step to next die}


procedure unload_wafer;
       {this procedure brings the wafer back to the unload position}
begin
     talk_to_hpib(hpib_addr_prober);
     writestringln(PROBER,'HO');
     wait_till_bit12_IOSTATUS_set;
     writestring(PROBER,'MEOVER 1 BILLION BSIM EXTRACTION  ');
     writestringln(PROBER,'        CUSTOMERS SERVED');
     wait_till_bit12_IOSTATUS_set;
     untalk(7);
     unlisten(7);
end;  {end of procedure to unload wafer}


{*************************************************************************}
{**********************MAIN BSIM PROGRAM**********************************}
{*************************************************************************}


 begin  {this is the main anne3 program}
     TRACEMD:=false;          {for measure_device trace}
     TRACE_linear_ext:=false;
     TRACE_sat_ext:=false;
     TRACEselftest:=true;
     TRACEiv:=false;
     TRACEpf:=false;


{*************************************************************************}
{HERE IS WHERE HPIB ANALYZER AND PROBER ADDRESSES CAN BE CHANGED GLOBALLY }
{*************************************************************************}

      minutes_per_device_above_thresh:=1.8;
      number_of_increments:=37;
      minutes_per_device_subth:=1.9;

      init_global_variables;  {set hpib addresses for 4062, prober}

      DCS:=723;
      SWC:=722;
      first_time_thru_program:=true;
      end_of_program:=false;
      ioinitialize;  {initializes all of the I/O devices and interfaces}
      repeat    {until end_of_program}
        initial_bsim_page;  {provides menu, and determines operation mode}
        minutes_per_device:=minutes_per_device_above_thresh;
        if (mode='1') then  {if a measurement mode}
             begin
```

```
        initial_status_inputs;  {this requests all necessary inputs}
        first_time_thru_program:=false;
        if printer then
           rewrite(userout2,'#6:')
         else
          begin
            rewrite(userout2,output_file);
            close(userout2,'save');      {create first}
            reset(userout2,output_file);
            close(userout2,'purge');     {then destroy it}
            rewrite(userout2,output_file); {now create again}
          end;
    end;
case mode of   {this calls sequence of routines to service mode}
  '1':begin   {this is for automatic prober mode}
      read_prober_file(xdie_size,ydie_size,origin_diex,
                       origin_diey,number_die,number_devices);
      initial_jump:=true;                {go from origin to first die}
      end_of_wafer:=false;
      set_up_prober_initial_conditions;
      set_die_size(xdie_size,ydie_size);
      present_diex:=origin_diex;
      present_diey:=origin_diey;
      step_to_next_die(present_diex,present_diey,
                       initial_jump,end_of_wafer);
      {now the prober is at the first die to be tested}
      present_die:=1;
      writeln(userout2,'Process: ',process);
      writeln(userout2,'Wafer: ',wafer);
      writeln(userout2,'Lot: ',lot);
      writeln(userout2,'Date: ',date);
      writeln(userout2,'Operator: ',operator);
      writeln(userout2,'Input File: ',prober_file);
      writeln(userout2);
      writeln(userout,'Process: ',process);
      writeln(userout,'Wafer: ',wafer);
      writeln(userout,'Lot: ',lot);
      writeln(userout,'Date: ',date);
      writeln(userout,'Operator: ',operator);
      writeln(userout,'Input File: ',prober_file);
      writeln(userout);
      writeln(#12);
      while not(end_of_wafer) do
        begin      {here is where a die is tested}
          writeln(userout);
          writeln(userout,'Die Location ',present_diex,
                                          present_diey);
          writeln(userout);
          set_die_size(1,1);  {micron units}
          for present_device:=1 to number_devices do
             begin
                measure_error:=0;   {just to be sure}
                if not(present_device=1) then begin
                     if not ((mx[present_device] =
                                       mx[present_device-1]) and
                             (my[present_device] =
                                       my[present_device-1])) then
                          prober_move(mx[present_device]-
                                            mx[1],
                                       my[present_device]-
                                            my[1],
                                       Zup);
                          {if mx's and my's are equal, don't move}
                end;  {if not(present_device=1) ... }
                initial_status_display;
                writeln(userout,' ');
```

```
                        writeln(userout,'Device ',devices[present_device]);
                        writeln(userout,'Test Coordinates: ',
                                        trunc(mx[present_device]/xmult),
                                        trunc(my[present_device]/ymult));
                        writeln(userout,comment[present_device]);
                        time_count:=1; {for bsim_timer routine}
                                       {now at present device}
                        measure_device ( measure_error, present_device);
                        new_die:=false;
                        end_of_die:=false;
                        if present_device=number_devices then
                             end_of_die:=true;
                      end; {for present_device:=1 to number_devices do begin}
                           {end of testing all devices on the die}
{NY}                present_device:=number_devices;  {prober at last device}
                    prober_move(0, 0, Zdown);
                    set_die_size(xdie_size,ydie_size);
                    step_to_next_die(present_diex,present_diey,
                                     initial_jump,end_of_wafer);
                    present_die:=present_die+1;
                  end;  {while not end_of_wafer, which is set in procedure
                         step_to_next_die}
                        {now prober is at next die or wafer is done}
                        {the wafer is now complete}
              close(userout,'SAVE');
              if not(printer) then close(userout2,'SAVE');
            end;  {end of case mode '1' - automatic probing mode}

        '4': end_of_program:=true;
      end;  {end of case statement}
    until end_of_program=true;
    writeln(#12);  {clears the display}
    iouninitialize;  {uninitializes all of the I/O interfaces}
end.
```

Appendix C

AUTOPRB short ouput format

The short format is ~ 1/10 as long as the long format, but criptic.

Process: AWIS
Wafer: 21 (POS)
Lot: 1
Date: 8/5/88
Operator: JAY FLEISCHMAN
Input File: SMLPRB.TEXT

0.8uvi	1 1	4	19	8.58066E-004
0.8uva	1 1	4	19	5.96461E-004
1.0uvi	1 1	4	19	9.05827E-004
1.0uva	1 1	4	19	5.82259E-004
0.8uvi	2 1	4	19	8.01800E-004
0.8uva	2 1	4	19	6.43550E-004
1.0uvi	2 1	4	19	1.04074E-003
1.0uva	2 1	4	19	5.96461E-004
0.8uvi	3 1	4	19	5.51310E-007
0.8uva	3 1	4	19	2.06350E-006
1.0uvi	3 1	4	19	6.66238E-007
1.0uva	3 1	4	19	1.65414E-006
0.8uvi	4 1	4	19	4.21173E-007
0.8uva	4 1	4	19	4.35256E-007
1.0uvi	4 1	4	19	5.25089E-007
1.0uva	4 1	4	19	5.31671E-007
0.8uvi	5 1	4	19	3.69529E-007
0.8uva	5 1	4	19	3.64079E-007
1.0uvi	5 1	4	19	4.77818E-007
1.0uva	5 1	4	19	4.78333E-007
0.8uvi	6 1	4	19	3.36190E-007
0.8uva	6 1	4	19	3.32428E-007
1.0uvi	6 1	4	19	4.50842E-007
1.0uva	6 1	4	19	4.43242E-007
0.8uvi	7 1	4	19	0.00000E+000
0.8uva	7 1	4	19	0.00000E+000
1.0uvi	7 1	4	19	3.97800E-007
1.0uva	7 1	4	19	3.87704E-007
0.8uvi	1 2	4	19	8.28979E-004
0.8uva	1 2	4	19	6.43550E-004
1.0uvi	1 2	4	19	9.98159E-004
1.0uva	1 2	4	19	6.03824E-004

Appendix D

AUTOPRB long ouput format

The long format is unwieldy, but descriptive.

```
Process: ELITHO
Wafer: 290-3
Lot: 1
Date: 7/28/88
Operator: Jay Fleischman
Input File: 290SMIL.TEXT

Die Location          1          1


Device LINEWIDTH
Test Coordinates:           4          17
# 1.0u line

Pin #              Voltage              Current
  46               6.81600
  48               0.44070
  42               8.87400            0.0010000
   6               0.00000           -0.0009930
1.0u     1 1      4    17        7.61654E-007

Device LINEWIDTH
Test Coordinates:           4          17
# 1.5u line

Pin #              Voltage              Current
  36               4.15400
  38               0.46160
  32               5.67500            0.0010000
  16               0.00000           -0.0009956
1.5u     1 1      4    17        1.31851E-006

Die Location          2          1


Device LINEWIDTH
Test Coordinates:           4          17
# 1.0u line

Pin #              Voltage              Current
  46               7.50900
  48               0.49180
  42               9.48300            0.0010000
   6               0.00000           -0.0009922
1.0u     2 1      4    17        6.91424E-007
```

Appendix E

Parser Code

```
/*  This program parses the input from the automatic prober into a file that
 *  can be plotted.  It acts on keywords from the user so the input file
 *  must be compatible.  Input is from smileplot.data */


#include <stdio.h>


main(argc, argv)
  int argc;
  char *argv[];
{

  FILE *input,*outplot,*output;

  static int dotype[] ={0,2,3,4,5,6,11,10,9};

  static char *outnames[] = {
      "plot.in1",
      "plot.in2",
      "plot.in3",
      "plot.in4",
      "plot.in5",
      "plot.in6",
      "plot.in7"
      };


  int diex,diey,i,j;
  float lw[8][8],y,tres,dumj,range;
  char wafer[20],date[20],infil[20],word1[30],keywrd[40],line[80];
  char inputfile[30];

/* diex is the x co-ord of the die and corresponds to a given exposure
   diey is the y co-ord of the die and corresponds to a given focus    */

  if (argc > 2) strcpy(inputfile,argv[2]);
     else strcpy(inputfile,"smileplot.data");
  if ((input = fopen(inputfile,"r")) == NULL)
     {
       printf("input data (%s) not found\n",inputfile);
       return(1);
     }

  if (argc > 1)
     strcpy(keywrd,argv[1]);
     else { printf("no keywrd to parse to\n");
            return(2);
          }
  range = 0.0;
  while (fgets(line, 80, input) != NULL)  {
     sscanf(line,"%s",word1);
     if (strcmp("Wafer:",word1) == 0){
         strcpy(wafer,line+6);
         wafer[strlen(wafer)-1] = '\0';
         }
     if (strcmp("Input",word1) == 0){
         strcpy(infil,line+11);
         infil[strlen(infil)-1] = '\0';
         }
     if (strcmp("Date:",word1) == 0){
         strcpy(date,line+5);
         date[strlen(date)-1] = '\0';
         }
     if (strcmp(keywrd,word1) == 0) {
         sscanf(line,"%*s %d %d %*d %*d %e",&diex,&diey,&tres);
```

```
        lw[diex][diey] = tres * 1E6;
        if (lw[diex][diey] > range) range = lw[diex][diey];
        }
  }
  if (range > 2.0) range = 2.0;
  fclose(input);

  output = fopen("smile.plot","w");

  fprintf(output,"plotter;\nappend = on;\nwindow 600 400;\n");
  fprintf(output,"autor = off;\n");
  fprintf(output,"set uxlabel = \"W: %s, D: %s",wafer,date);
  fprintf(output,", I: %s, K: %s\"\n",infil,keywrd);
  fprintf(output,"set lxlabel = \"Focus (by Row)\"\n");
  fprintf(output,"set lylabel = \"Linewidth (microns)\"\n");
  fprintf(output,"r plot.in1;\nget_range;\ny_range 0 %-f;\n",range);

  for (i = 1; i < 8; i++){
    fprintf(output,"set lmark = \" C%-d\"\n",i);
    fprintf(output,"pl = 1;\n");
    fprintf(output,"psplot plotout;\n");
    fprintf(output,"pl = 11;\n");
    fprintf(output,"pt = %-d;\n",dotype[i]);
    fprintf(output,"psplot plotout;\n");
    if (i < 7) fprintf(output,"r plot.in%-d;\n",i+1);

    outplot = fopen(outnames[i-1],"w");
    for (j = 7; j > 0; j--){
      dumj = 8 - j;
      if (lw[j][i] > range) y = range;
        else y = lw[j][i];
      if ((y > 0) || (j == 1) || (j == 7))
          fprintf(outplot,"%e   %e\n",dumj,y);
    }
    fclose(outplot);
  }

  fprintf(output,"!/od/sample/jef/bin/psplt plotout;\n");
  fprintf(output,"!rm plotout;\nkill_p;\n");
  fclose(output);
}
```

```
/*  This program parses the input from the automatic prober into a file that
 *  can be plotted.  It acts on keywords from the user so the input file
 *  must be compatible.  Input is from explor.data.  Output is simply
 *  explor.out.  This version is for the "lower left" orientation of the
 *  wafer.  The only change necessary for this is to rearrange the order of
 *  the indecies and counting up/down (see below) */


#include <stdio.h>


main(argc, argv)
  int argc;
  char *argv[];
{

  FILE *input,*output;

  int diex,diey,i,j;
  float r[8][8],tres;
  char wafer[20],date[20],infil[20],word1[30],keywrd[40],line[80];
  char outputfile[30],inputfile[30];

/* diex is the x co-ord of the die and corresponds to a given exposure
   diey is the y co-ord of the die and corresponds to a given focus    */

  if (argc > 3) strcpy(outputfile,argv[3]);
     else strcpy(outputfile,"explor.out");
  if (argc > 2) strcpy(inputfile,argv[2]);
     else strcpy(inputfile,"explor.data");
  if ((input = fopen(inputfile,"r")) == NULL)
     {
       printf("input data (%s) not found",inputfile);
       return(1);
     }
  if (argc > 1)
     strcpy(keywrd,argv[1]);
     else { printf("no keywrd to parse to\n");
            return(2);
          }
  while (fgets(line, 80, input) != NULL)  {
     sscanf(line,"%s",word1);
     if (strcmp("Wafer:",word1) == 0){
         strcpy(wafer,line+6);
         wafer[strlen(wafer)-1] = '\0';
         }
     if (strcmp("Input",word1) == 0){
         strcpy(infil,line+11);
         infil[strlen(infil)-1] = '\0';
         }
     if (strcmp("Date:",word1) == 0){
         strcpy(date,line+5);
         date[strlen(date)-1] = '\0';
         }
     if (strcmp(keywrd,word1) == 0) {
         sscanf(line,"%*s %d %d %*d %*d %e",&diex,&diey,&tres);
         r[diex][diey] = tres;
         }
  }
  fclose(input);

  output = fopen(outputfile,"w");

  fprintf(output,"\nW: %s, D: %s",wafer,date);
  fprintf(output,", I: %s, K: %s\n\n",infil,keywrd);
  fprintf(output,"%16s%10s%30s\n","Column","|","Column");
```

```
  fprintf(output,"    C1 C2 C3 C4 C5 C6 C7 |      C1      C2      C3      C4");
  fprintf(output,"     C5      C6      C7\n");

/* Simply adjust the indecies i and j to reflect the coordinate transfer
 * from the orientation of the wafer when probing to the orientation when
 * the flat is at the top (so focus/exposure correctly corresponds to "Rows"
 * and "Columns" in the plot (or table as in this case) */

  for (i = 1; i < 8; i++){
    fprintf(output," R%-d",i);
    for (j = 1; j < 8; j++){
      if (r[j][i] > 3E3)
          fprintf(output,"  0");
        else
          fprintf(output,"  1");
    }
    fprintf(output," | R%-d",i);
    for (j = 1; j < 8; j++){
      if (r[j][i] > 999999) r[j][i] = 999999;
      fprintf(output,"%7.0f",r[j][i]);
    }
    fprintf(output,"\n");
  }
  fprintf(output,"%26s\n%18s%8s%32s\n\n\n","|","Continuity","|","Resistance");
  fclose(output);
}
```