Copyright © 1978, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

REPORT ON THE

DESIGN OF A SIMULATOR PROGRAM (SAMPLE)

FOR IC FABRICATION

by

Sharad Narayan Nandgaonkar

Memorandum No. UCB/ERL M79/16

June 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CONTENTS

Part 1 Design Documentation

Chap. 1 An Overview

1. Introduction
2. Organization of the Report

Chap. 2 The Simulator

1. Structure of the Simulator
2. Some more discussion on the Structure of the Simulator
    1. On the State-Variable Characterization of the System
    2. On the Input/Output of the Simulated Machines within the Lab
    3. On the Actual Program Structure
    4. On the Design of the Input Interface
        1. Syntax and Semantics
        2. Error Handling Philosophy

Chap. 3 Miscellanea

1. Validity of the Simulation
2. Usage of the Program
    1. Computer Time and Memory Requirements
    2. The Programming Language Used
3. Management of the Program Code
4. Graphical Output Option
5. Some History of the Program
    1. Contributions to the Program
    2. Documentation of the Program

Appendix 1A : Some Statistics on the Program Size

1. Source Code Size
    1. Size of Interface
    2. Size of the Core of the Program
2. Object Code Size
    1. Data
    2. Instructions

Fig. 1 : Pictorial View of the Program
Fig. 2 : Material Flow Model for Real Machines
Fig. 3 : Information Flow in the Simulated Lab
Fig. 4 : Control-Structure of the Program
Fig. 5 : Information flow structure of The Program
Fig. 6 : Control Structure + Information Flow in the Program

References
Acknowledgements

(Contents - contd.)

Part 2) Implementation Documentation for the User Interface

The documentation files forming part 2 of this report are as follows

Lexical Analyzer Definition etc.	-	docu1 (='lexdef') lexvar lexfea
Parser Definition etc.	-	pardef parvar
Prettyprinter documentation	-	prprdocu
<ex-stmt-subr>s for machine1	-	m01var m01varb m01varc errmesm01
for machines 2, 3, and 4	-	data234_org errmes234
Some features of the program	-	progfea1

{Organization of the source code and the source code are given at the end of the Implementation documentation, i.e. at the end of part 3}

(Contents - contd.)

Part 3) Implementation Documentation for the Machines and Components

The documentation files in part 3 are as follows

Documentation of Default Values for all Machines and Components	-	defaultdocu
Machine1 documentation	-	mac1doc1
The Organization of all the source code for the program (both the User Interface, and the Core) is given in	-	sc_org

The source code itself is given in the same sequence as the files appear in sc_org.

Part 4) User Documentation

The Files in part 4 are as follows

index
exdatafile
example
howto

pargram

trialdocu

pfaudit

Part 1

Design Documentation

Chapter 1 An overview

1.1. Introduction

In the manufacture of integrated circuits a major step is to etch a particular geometric pattern onto a semiconductor chip using photolithographic techniques. Traditionally this has been aided by various part by part analyses of some of the substeps involved in these processes. A project has been undertaken to create a unified user-oriented simulation program for this process as a whole.

Clearly, such a project is necessarily a part of the total manufacturing process. It has to be linked with the analytic, and experimental work done by other workers in the field, and its value determined from the validity and usefulness of the results. A first version of the simulation program, SAMPLE (Simulation and Modelling of PhotoLithography and Etching), developed at the University of California, Berkeley, has been up for the last few months and is being continuously expanded further. The results are quite encouraging and more ideas are on the agenda for incorporation into the program.

This report presents the idea behind the program, its design, specification, and other details along with the source code of the user-interface, as it stands at present (June 1978). Also included in the report is the documentation prepared for its users. The organization of this report is sketched in the following section.

1.2. Organization of this Report

This report is organized as follows.

Part 0)

Abstract
Contents

(The contents give a more detailed sectionwise outline of the report.)

Part 1) Design Documentation

In this part an overview of the program is given. The basic ideas behind its design, and its top level structure are discussed.

Part 2) Implementation Documentation for the User Interface

This gives the details of the input interface. The syntax and semantics of the input language are defined, and the User Interface code is presented.

Part 3) Implementation Documentation for the Machines and Components

This gives the details of the core of the program i.e. the machines and components in the simulated lab. The models used for the machines, the formulae used for computation, the discretization and other constraints imposed by numerical analysis/calculation considerations etc. are discussed. Also listed are the detailed documentation of the data-structure of the program core, error-messages, deficiencies, improvements, and bugs. A list of the default values of the relevant parameters describing the lab is also included.

Part 4) User Documentation

This gives an example of a typical set of input data to the program, information for using the TRIAL-statement, and other material helpful for using the program.

Chapter 2 : The Simulator

2.1. Structure of the Simulator

A simulator for a physical process should model the process as closely as possible. Preferably, it should be more convenient to use than actually running the physical process, and should provide easy ways to monitor the progress of the process by providing various types of information about the simulated process which may be difficult, or even techinally impossible, to obtain from the corresponding actual process. Also, it should allow a better control over the process so that the effects of varying individual factors should be easily discernible in the outcome of the process, and should give a better insight into how to control the actual process. In short, it should approach the ideal of a highly controlled and instrumented set-up in which to study the process. Especially, when the processes under consideration interact with each other in a multitude of ways the value of such a simulator for understanding the interactions is quite obvious.

A digital computer program is the most promising way to fulfill the above expectations from a simulator. Once the models of the physical phenomena are properly formulated in terms of equations in the various parameters involved, the desire for '<u>controllability</u>' and '<u>observability</u>' can be conveniently realized in such a program. Also a program allows one to handle not only the closed form solutions of static systems but also the flow of time in a dynamic process that can only be described by a differential or difference equation.

Such a program, SAMPLE, was designed for simulating the lithography and etching processes used in the fabrication of semiconductor wafers.

Design of SAMPLE

The processes in semiconductor lithography typically involve the image formation of a particular pattern on a photoresist coated wafer. For highest resolution work, positive photoresist is used. The image formed on the photoresist layer causes local bleaching in the layer to various degrees depending on the intensity distribution. The exposed wafer is put in a developing solution which etches out the bleached portions of the resist. In between these two main processes of bleaching and development, the resist may be deliberately modified, e.g. by baking in an oven, or by putting it in chlorobenzene solution.

To simulate such a processing sequence, the user tells SAMPLE about the <u>components</u>, like the imaging system or the

radiation source, that he wants to use, what the configuration of the system is, what layers are present on it and how thick they are, what kind of pattern is present on the mask, how much radiation dose does he want to expose the wafer with, what kind of developer does he want to use, and so on. He also tells what he wants done with the components, what processes should they be subjected to, whether the wafer should be exposed or not, whether the exposed wafer should be developed or not, and if the wafer is to be baked how long should it be baked. All these things he specifies by giving the physical dimensions of the components involved, the characteristic parameters describing the processes, and any other relevant input parameters that describe what he has in mind. Finally, he also specifies the sequence in which the the operations like bleaching (exposing the wafer to the image) and etching, or baking are to be carried out. And what resultant aspects of the resulting product he wants to monitor.

To make the above communication between the human user and SAMPLE suitable for both of them they must understand each other. In particular, the user should know what SAMPLE can do, how he does it and what should he told to him and in what way. SAMPLE takes this information and processes it according to his understanding of the actual phenomena, which is nothing but the models described in the literature [see Bibliography].

The situation can be described in a more pictorial way as shown in figure 1.

The user gives SAMPLE four types of information to control and observe the process :

control :

1) The static specification of the components like mask, imaging system, wafer etc. which just involves the line size, space size, lens parameters, and layer thicknesses etc.

2) The parameter specification for the dynamic processes as characterized by their models. e.g. the A, B, C parameters for describinng the bleaching processes, the development rate as a function of bleaching (M-parameter) for the etching process.

3) The sequence of operations performed on the components. e.g. telling the controller to proces the components in the exposing machine, etching machine etc.

observe :

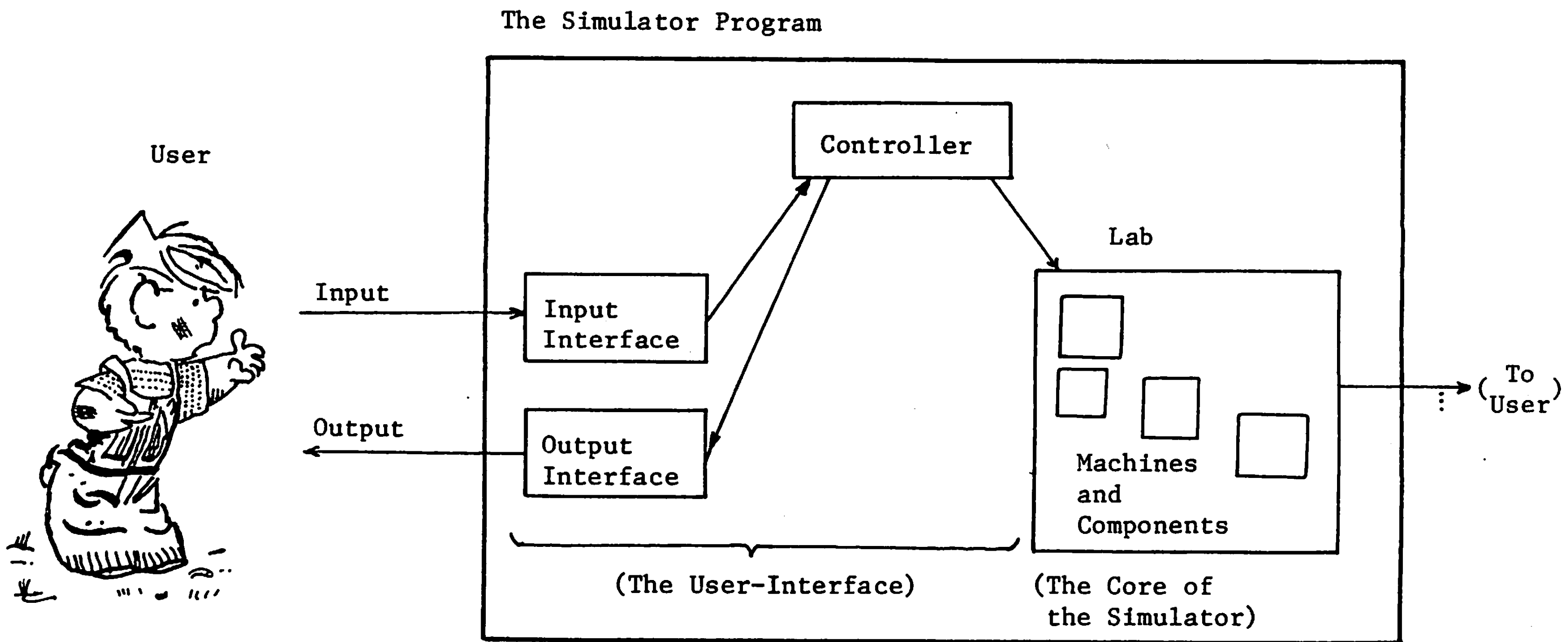


Fig. 1 Pictorial View of the Program Usage (Information-flow)

4) What parameters are to monitored.

Naturally, the controller can be viewed as a person in charge of a lab where he gets the components to the specification of the user, gets the machines to process the components (e.g. to expose the wafer in the imaging machine, or etch it in the etching machine). And finally carries out the operations that the user tells him to carry out in the sequence that he tells SAMPLE to perform them.

It is precisely the above picture that was implemented in the program SAMPLE. The core of the simulator is the lab and the machines in the lab which are represented in the program as the values of the parameters that specify the components. The processing is carried out as a numerical time-step simulation on those values by program subroutines. The controller is that subroutine in the program which gets the information from the user through the input interface subroutines and then puts it in the proper place for the components (e.g. the appropriate subroutines to carry out the individual component-process (e.g. bleaching, or etching) simulation. Also, the controller controls certain output from the program.

Once the above theoretical model was recognized for the program, it was built to fit other limitations and considerations of a practical nature, like the available memory space and cost of computer time used. To keep the input format convenient to the user the form of input statements was designed to have mnemonic keywords like CONTACT, PROJ, DOSE, ETCHRATE, RUN etc. with the corresponding numerical parameters following them. [For exact details see the implementation documentation in parts 2 and 3.] Also most of the information about the system need not be specified if it has some standard value (e.g. the standard wavelength, the standard etchrate relation for the standard developers with the standard photoresists etc.). Any system parameter that is required but not specified in the input has this standard value by default (but the number of items in a given form of input statement cannot be changed arbitrarily, except as specified in the documentation). A table of the standard values used is given in part 3 of this report.

The one important difference between the models envisioned above for the real machines (see fig. 2) and the actual models implemented in the program for the simulated machines is the decomposition of the bleaching process into three subprocesses in order not to repeat some lengthy calculations unnecessarily when a single parameter is changed. The program considers the exposure (bleaching) process as consisting of three different and somewhat 'orthogonal' (i.e. independently executable) subprocesses :

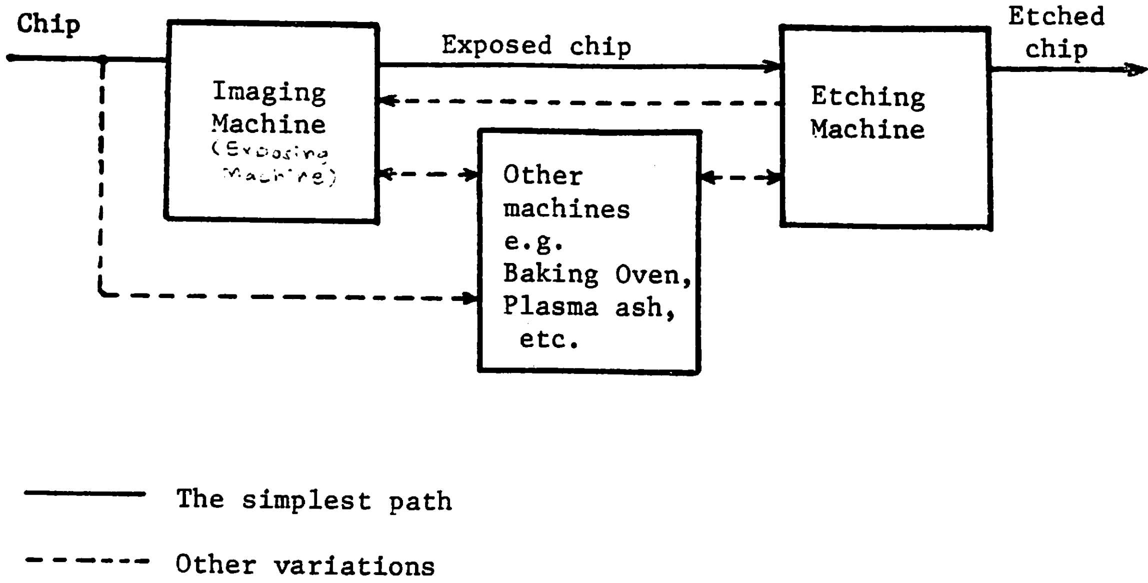


Fig. 2 Material Flow Model for Real Machines in a Lab

1) Image formation on the resist surface

2) Calculation of a table which gives a relation between the image intensity at a point on the resist surface and the bleaching produced (the M-parameter) at the points below it, in the resist layer. This calculation proceeds through the intercoupled computations (coupled differential equations) of the standing wave formed in the PR layer and the energy absorption dependent on it. The M values which represent the bleaching, as well as the rate of absorption are both dependent on the total amount of energy absorbed at that point till that time.

3) The calculation of the actual bleaching (M-values) in the resist layer from the horizontal image intensity pattern found in (1) and the table calculated in (2).

The image formation on the resist surface is indeed independent of the bleaching process inside the resist layer. The horizontal image intensity computation can be carried out independently of the bleaching produced because of it. Further, the bleaching process itself can be decomposed into two separate calculations as mentioned above. This is possible because of the approximation that inside the wafer the light travels as a plane wave normal to the surface. This implies that the light intensity at any point inside the layers, in particular in the resist layer, can be found from the image intensity at the single point above it on the surface. Thus the bleaching effect at that inside point is dependent on the image intensity at only one single point on the surface of the layer (after the horizontal image intensity calculations). This makes possible the division of the bleaching calculation as indicated above. Moreover, because of this the standing wave and bleaching calcultions in subprocess (2) above reduce from a 3-dimensional case to much simpler 1-dimensional case and the calculations in (3) can be carried out by using simple interpolation on the values obtained in (1) and (2).

Thus the individual machines available in SAMPLE are :

1) Horizontal image intensity distribution calculation

2) Standard-bleaching calculation

3) Actual-bleaching calculation

4) Etching calculation

5) Other machines for diffusion and M-degradation calculations to simulate the effect of baking or a chlorobenzene soak.

When the user tells SAMPLE to run a particular machine, say machine 1 (by by a RUN 1 statement in the input), the controller runs that machine and prints out the output for the user. This printout gives information about the many relevant parameters used in the program that represent the actual processes and machines, as well as the parameters used for discretization of time and space in the simulation for the numerical routines. Various portions of the output may be turned on or off by specifying so in the input [see the implementation documentation]. An example of the input (and comments on the output) is shown in part 4.

Also apart from the standard routines available in the system, a facility is provided in the system to carry out some user-defined processing through the TRIAL-statement [see part 2, and 4]. The TRIAL-statement allows the user to introduce temporarily his own special purpose routines in the program. The TRIAL statement is used by putting in two more simple subroutiones to integrate the user's routines with the rest of SAMPLE [see part 4].

In the case of an error in the form of an input statement the input interface provides the proper diagnostic messages and just prints out the rest of the input statements as far as it can recover from the confusion created by the error. This helps in pointing out to the user as many syntactic errors in the input format as possible in the first run itself.

2.2. Some More Discussion of the Program Structure

2.2.1. On the State-Variable Characterization of the system

The central part of the program is the (simulated) lab, and the components and machines in it. They have been represented by values of certain parameters that characterize them, and hence the conceptual and operational structure of the program revolves around them. Since the values of those parameters specify the state of the system at any given moment, they are the state-variables of the system. Once this state-variable nature of the system (real or simulated) is recognized, various known concepts from state-variable theory of control systems (e.g. for linear electrical networks) could be applied consciously and hence systematically (before that recognition there was a tendency to put things into the program on a 'newly-discovered-need' or 'trial-and-error' basis without any guideline for assessing how naturally they fit into the program structure). Also, this identification of the system as a state-variable controlled system (what else can any system be?) * clarified further the nature of variables that should be stored globally (i.e. the data-structure) (in COMMON blocks in FORTRAN) for the system. Also the user-specifiable operations (the unit-operations = the primitives) to be performed on the values of these variables, by any numerical calculations or otherwise, were clearly identified by trying to achieve the aims of good controllability and observability ** for the system. And while developing and debugging the system, these concepts indicated techniques which helped spotting out errors before they caused too much trouble.

---

* [With due respect to other types of models and characterizations of systems :] For analog systems the state-variable theory of control systems is well-known but also for digital systems it is known in the form of Huffman model for sequential circuits (in this model a set of D-Flipflops acts as the state-variable memory) and various other similar models, in operating systems it is recognized by concepts like 'a process is defined by the data which represents it', in programming languages 'the semantics itself is given by an interpreter which describes how the state-vector changes as the computation progresses' (John McCarthy, 1967), and so on.

** Informal Definitions :

Controllability - The system can be taken from any state to any other state in a finite time by a suitable input to it.
Observability - The state of the system can be inferred by observing its output for a finite time.
In the program many state variables are also the input variables, or the output variables themselves, thus simplifying the controlling or observing actions.

Another decision was to give both the programmer and the user the same controllability and observability so that there are no 'under the table deals' in the system to which the user doesn't have an access. This is useful to keep the programmer from any temptations to make the system unnecessarily complicated to manage or maintain.

Being a simulated lab it can easily be made 'over-controllable' than the real system (i.e. a state "inaccessible" in the real system - like a diffractionless image - can be simulated). But the temptations for such non-real over-control were firmly resisted (though even now it can be obtained by using the TRIAL-statement of the program) and that must have saved a lot of irrelevant playing with the program (unlike extra observability which is very convenient to have if it can be switched off when not wanted).

The above concepts have not only given the guidelines for the structure of the program but they give a good formal and informal terminology to communicate various aspects of the program.

2.2.2. On the Input/Output of the machines within the lab

From the information flow of the simulated lab as shown in fig. 3, it is clear that the machines are interfaced to each other through their common data structure. Hence, to make the system modular, the machines should not have any more interconnections than those arising from a well-defined interface. But at the same time, it is convenient to have some redundant information in this interface between machines for clarity in the user output. e.g. when running machine1 its wavelength input should be copied onto its output data-structure interfacing it with machine3 so that machine3 can tell the user about the wavelength at which its input was derived, without constraining mahchine1 to keep its input wavelength unchanged till machine3 is finished with the intermediate output of machine1. This can be generalized to more machines interconnected at one point and may be useful if the state of the system is to be saved on some temporary output file (not possible at present) for continuation of the run at some future time, but it is easy to overdo it.

2.2.3. On the actual program structure

It is only fair to say that the discussion in the two subsections above presents the design guidelines which were consciously followed, rather than a strict workshop discipline that was rigidly adhered to. There are some very small deviations but still the correspondence between the program and the above structure is quite an exact one.

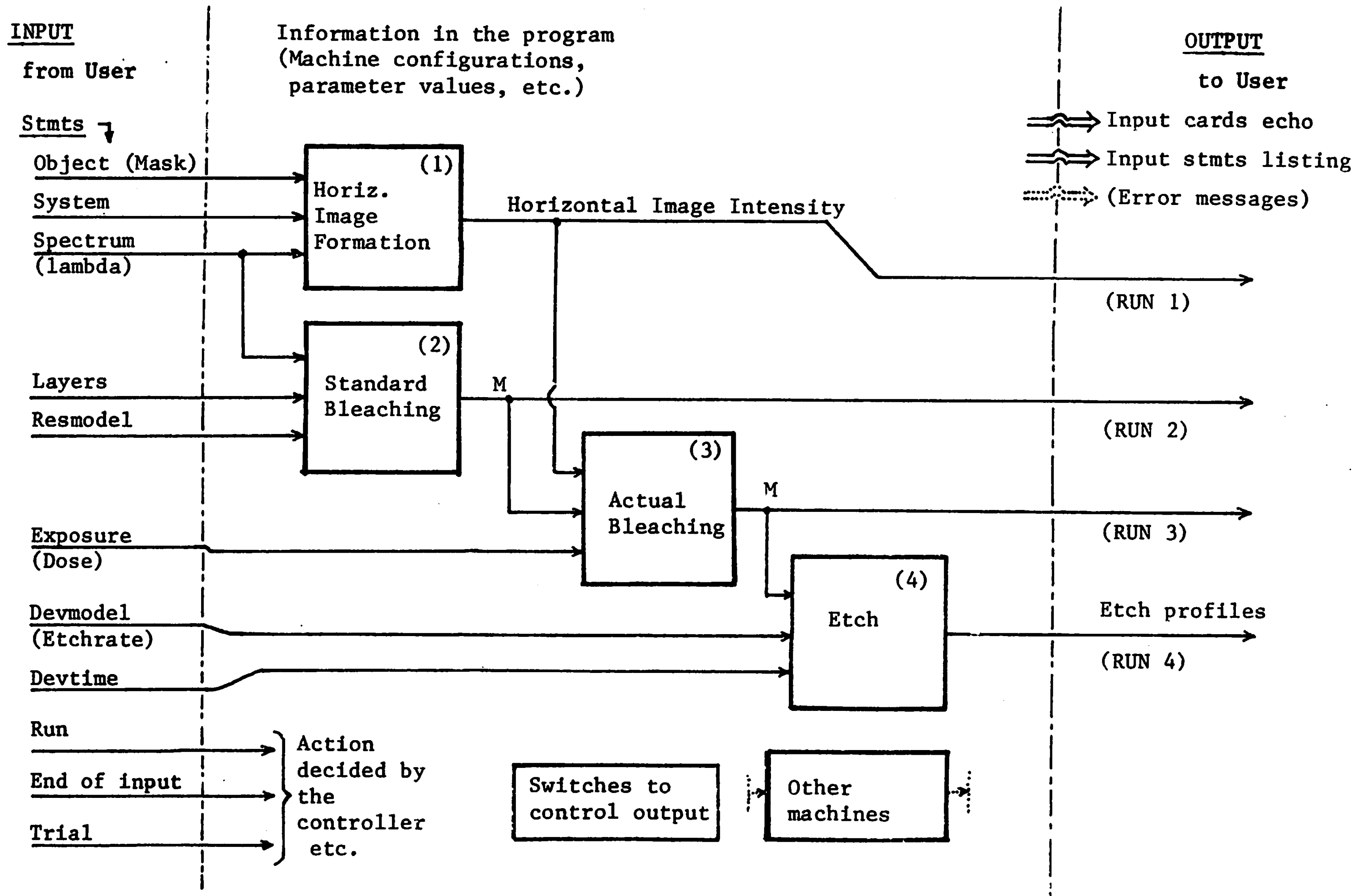


Fig. 3 Information-flow in the Simulated Lab

With that structure for the core of the simulator, the actions and properties of the user-interface follow straightaway from the concepts of controllability and observability except for the exact formal definitions of its input syntax and semantics. These are given in detail in part 2. The control structure of the input interface and the information flow in it are shown in figures 4, 5, and 6, in the context of the system as a whole. Also these figures show the structure of the output interface more clearly than the way it is shown in fig. 1. That 'output interface' in fig. 1 stands for the 'centralized' components like the prettyprinter for statements, the diagnostic error-message subroutines of the input interface, and the implicit (trivial) 'prettyprinter' to echo the cards on the output (these different outputs should either go to differnt output files and/or the user should be able to turn them on or off (except for the error messages) by an input statement to that effect). It does not consider the 'distributed' output coming from the various machines in the lab. Indeed figures 4, 5, and 6 are more precise in showing this.

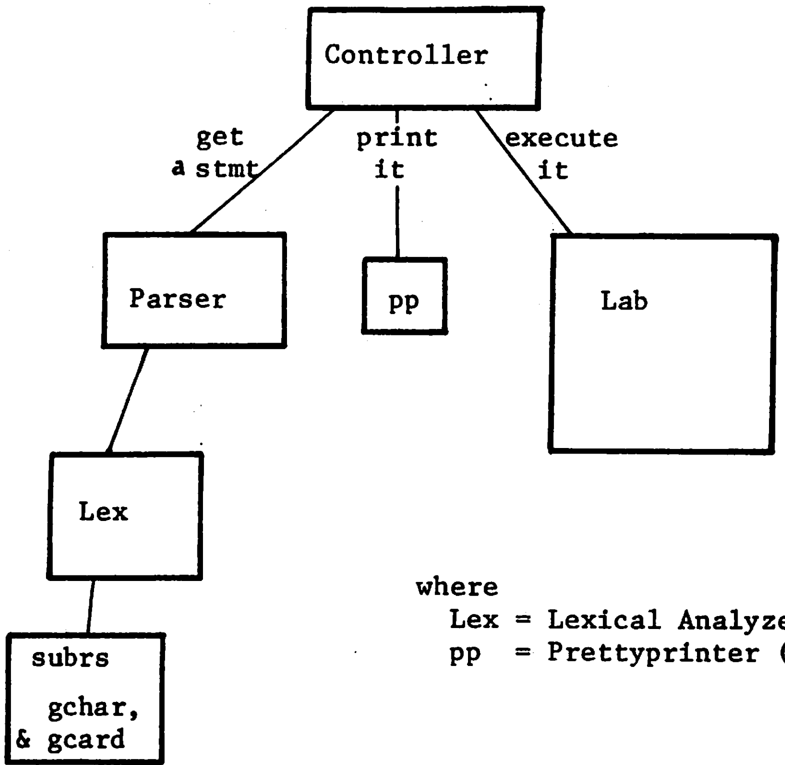
One deficiency of the figures is that they do not indicate clearly the position of the subroutines which 'execute' the input statements (by either storing the components in the lab or by setting the various (output) swithches (information flow action), or by running the machines (by calling other subroutines) according to input commands. In the figures these commands are considered to be just a part of the 'controller' because doing otherwise causes more loss in simplicity and clarity than the resultant gain in precision.

A comment : Note that the above models of the simulated processes, and hence the program, are completely "deterministic" i.e. they involve no random variables, probabilities, or stochastic processes.

2.2.4. On the Design of the Input-Interface

2.2.4.1. Syntax and Semantics

The input language is a very simple 2-level and non-procedural language designed in a standard straightforward manner [see figure 5]. After the card input is implicitly converted to a single character-stream by subroutine gcard (which also prints out the card images on the output file (-'prettyprints' them i.e. marks them to be recognized as the echo of the input cards)) the lexical analyzer can get the characters in the character stream by calling subroutine gchar (see figures 4, 5, and 6). The first level of the input grammar, for the lexical analyzer (scanner), is a simple type 3 (i.e. regular) language defined by a BNF syntax [see part 2]. The lexical analyzer forms the lexical tokens, like keywords and numbers, and passes them on with

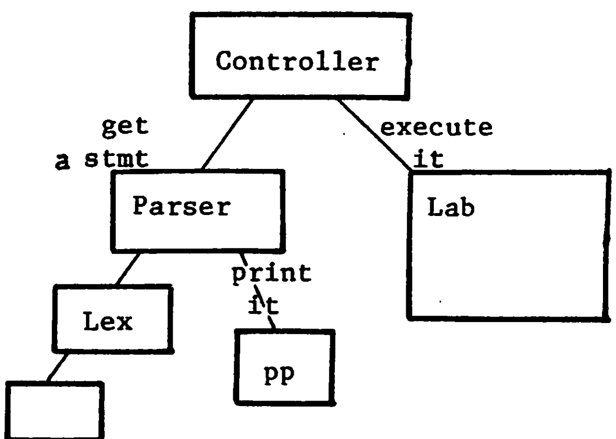


where
Lex = Lexical Analyzer
pp = Prettyprinter (for stmts)

(a) : Graphically (exactly as it is)

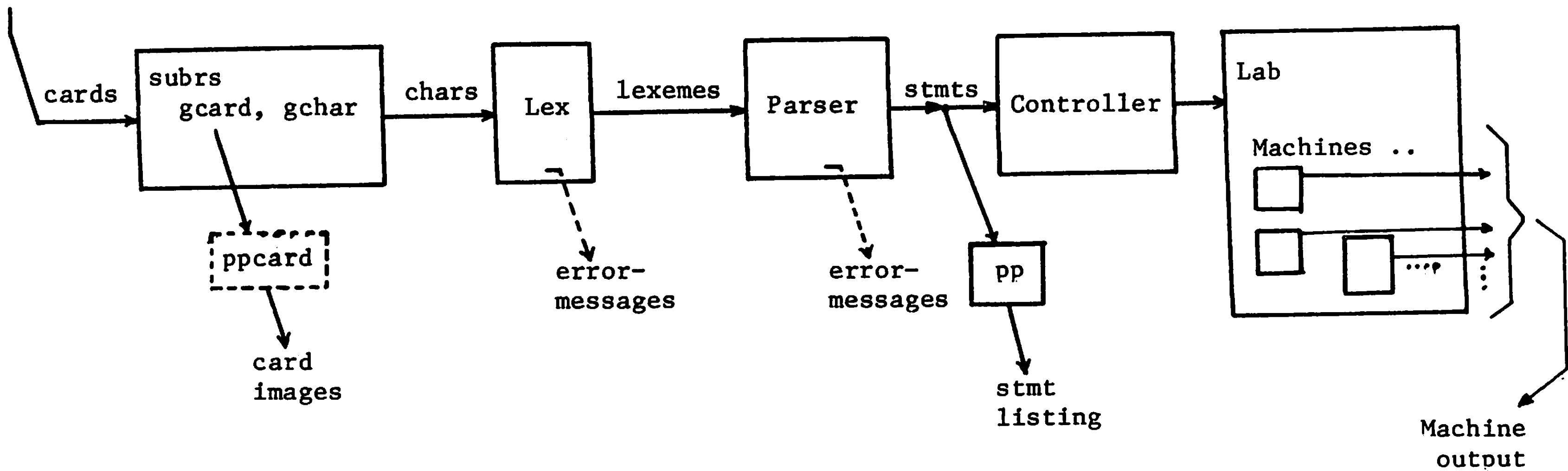
```
begin
initialize various things ;
repeat
   [get a stmt from input ;
   [print it out in an expanded form ;
   [if  ((no previous error) or
   [     ('end-of-input-stmts' stmt))
   [   then execute it ;
   until (end of input stmts)
end
```

(b) : 'Equivalently' for (a) (as a high level code)



(c) : 'Equivalently' for (a) (Graphically)

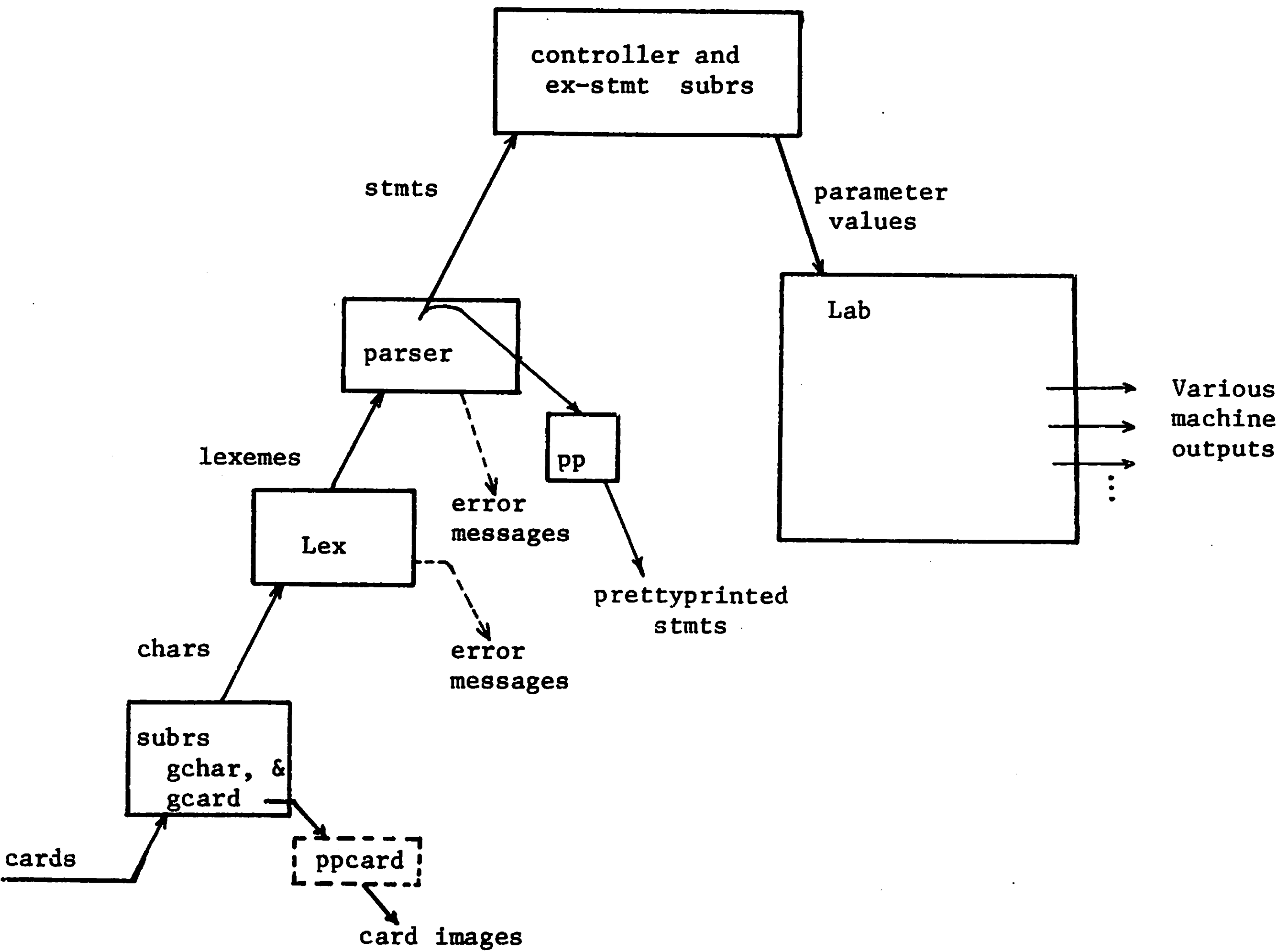
Fig. 4 Control Structure of the Program



where
  ppcard = Implicit 'prettyprinter'
           for cards ("quotes" the
           card image to distinguish
           it from other output).
  pp = The prettyprinter for stmts
       (= "ppstmt")

Output to the User

Fig. 5   The Information-flow Structure of the Program



<u>Fig.6</u> Control-structure + Information-flow of the Program
(Based on fig. 4(c), and 5)

their values (meanings), as lexemes to the parser. The parser forms the input statements from these lexemes according to another regular grammar [see part 2] and finally the controller uses these statements in an interpreter-like fashion - performing the suitable actions indicated by their meanings.

2.2.4.2. Error Handling Philosophy

The error handling philosophy of the input interface is based on a simple idea. The input grammar is 'fully defined' over all the possible combinations of the input characters. This completeness is achieved by attaching a meaning of the type "erroneous form" to any <item> which is "illegally" formed. Thus at the first three levels # there are defined <erroneous-char>s, <erroneous-lex-token>s, and <erroneous-stmt>s which are nothing else but <item>s to be formed at that level that are not correctly-constructed in the user input. When these <erroneous-item>s are detected in the input to that level, an error-handling routine (each level has its own) is called. That error-handling routine prints out a proper diagnostic message for the user, attaches the meaning of 'error-value' to that item (each level has only one 'error-value' for the <erroneous-item>s formed at that level) and then increments a private counter keeping track of how many errors were detected at that level in the current run (and if the number of detected errors at that level exceeds a certain limit then the program is stopped right there by executing a STOP-statement).

In the same spirit, the controller 'executes' an <erroneous-stmt> by setting up an error-flag in the lab which causes the processing in the lab to stop. The user interface continues its job so that the user may know how the rest of the input is going to be accepted.

This scheme of treating the errors by associating a separate "meaning" to them has resulted in a very simple, clean, and modular design for the input interface. The interface does not panic or get off-balance because of any simple input mistakes but goes on performing its job in a smooth manner.

{There are a few small deviations from the above design in the actual code because of the way it was first written.}

(see fig. 5). The first three levels are the recognition and formation levels for (1) card to characters, (2) chars to lex-tokens, (3) lex-tokens to stmts; and the forth level is the interpretation level for (4) stmts to action

Chapter 3 Miscellanea

3.1. Validity of the Simulation

The validity of the results obtained in the simulator is discussed in the papers given in References {section 'Usage'}.

3.2. Usage of the Program

At present the program is still being expanded, and hence it is mostly being used by a small group within University of California at Berkeley, on the CDC 6400 computer with the CALIDOSCOPE operating system of the University's Computer Center.

3.2.1 Computer Time and Memory Requirements

An Average run of the exposure and development sequences for two chips costs about $ 2.0 to 2.5 at the normal ('J') priority, and approximately $ 1.0 to $ 1.50 at deferred ('D') priority (CPU time approx. 4-7 seconds). But if the diffusion machine is used then it takes about 25 to 30 seconds of the CPU time and costs about 3 dollars at D priority. The Program requires slightly less than 112000 (in octal = 37888 in decimal) words (60 bits each) of memory to load and a little less than that to execute. For a more detailed statistics of the program size see appendix 1A.

The memory and time requirements should get reduced by using some manual optimizing in the source code, and in particular the time requirements should get reduced significantly by rewriting some critical sections in the program and using an optimizing compiler like FTN, or RUNW.

3.2.2 The Programming Language Used

The program is written in ANSI Standard FORTRAN [see References] to be easily transportable to other machines. (The few, very small deviations from this standard are listed in the 'Implementation Documentation' (Parts 2, and 3), and should get removed soon.)

3.3. Management of the Program Code

The source code was entered on the UNIX interactive Operating System on the PDP 11/70 computer at the Computer Center, UCB. The hierarchical directory structure of UNIX is used to store the various modules of the source code kept

in separate files. The fort compiler on UNIX for standard Fortran was used to detect simple syntax errors in the source code when newly entered. Then the code was sent on the 'rcslink' to CDC 6400. There it was first checked by using the 'AID' compiler and then with 'RUN'. The RUN-compiled object code is kept (modularly) on the 'Permanent Files' of 6400, and loaded from them for each run.

No memory overlaying is done for loading and executing the program.

After the program code settles down sufficiently, an optimized object code kept in a single permanent file for fast loading and execution should cut down the usage cost even further.

There are plans to put this program on some of the new minicomputers for wider usage.

3.4. Graphical Output Option

For human engineering reasons, some of the output, like the etch-contours, is plotted out on the printer output, rather than just printing the co-ordinates numerically. But since the precision of a character-positioned printer-plot is not very high, an input selectable option is provided if more accurate plots are desired. Under this option, a set of cards with the co-ordinates of the points on the curve and some other information relevant for the plot is punched out on the 6400. Then a precise plot is obtained from these numbers using a digital plotter driven by a desk calculator interfaced with a card reader.

{Actually such storing of information on an intermediate output file for postprocessing by a different program is a generalization of the same simple idea (as mentioned in sec 2.2.2) of outputting the state-variables of the program for later use by the simulator program itself for a continuation run.}

3.5. Some History of the Program

3.5.1 Contributions to the Program

This program is one part of the work being done at UCB in the field of semiconductor processing. The original idea to have a such a user-oriented, unified simulator program for photolithography is due to Prof. Andrew R. Neureuther, and Prof. William G. Oldham (spring 1977). After starting in July 1977, the first complete version was ready on December 31, 1977 (10 p.m.). Before that, isolated routines to simulate the exposure and etching were written by Dill

and coworkers [See References : July 1975], Neureuther and coworkers [1976], and by Jewett and coworkers [June 1977]. Now they have been adapted for use in this program by Michael O'Toole. Mike has extensively modified the core routines (the lab), added some more to handle the optical phenomena based on models of his own, and also various plotting routines to generate a desirable graphical output. The syntax of the User Interface was designed by Prof. Oldham, and so also was the diffusion program. The overall organization of this program and its data-structure are due to Sharad N. Nandgaonkar. And also the User Interface was written by him.

3.5.2 Documentation of the Program

The documentaion of the program, in its basic details, was sketched out as the program was developed but was put in this form only when writing this report. Putting it in a final form should have been done along with the writing of the program code.

The editing and document preparation facilities available on UNIX were a great help in assembling it in this final form.

Corrections to section 3.5.1 :

Initial History :

1) Isolated routines at IBM : only results published
2) Exercised by Murti Narasimham (at IBM)
3) Further work on applications was by Neureuther et al [1976]
4) and on algorithms was by Jewett et al [1977]

Appendix 1A

Some Statistics on the Program Size

The Program is considered to be made up of the parts : interface, and core (= the lab routines to simulate the machines). Hence the size of the program is given decomposed as

```
sizeof[ program = interface {= (lex + parser + prpr
                              + <ex-stmt-subr>s}
                    + core {= (lab) }
```

where prpr = the prettyprinter for the statements, and '<ex-stmt-subr>' is used to denote the subroutines in the user interface that 'execute' statements, along with the top level controller (but not including the machines in the core of the program).

{Note : The program has not settled down fully and hence some of the numbers are not final. Also the numbers are quite accurate for the current status (June 1978) of the program, though not exact, so the results shown for addition may differ slightly from the arithmetic sum.}

1A.1. Source Code Size

The source code size is given as number of lines (= cards) for the routines. The numbers in parentheses are for the code without the comment cards, and are estimates that should be accurate to within 3 to 8 %. The source code is kept in UNIX files.

1A.1.1 For the Interface

Lex	= 356	(190)
Parser	= 860	(500)
Prpr	= 265	(200)
<ex-stmt-subr>s	=635	(350)

Hence total size for the input interface = Total(sI)
= 2116 (1240) lines

1A.1.2 For the Core of the Program

mac 1	= 500	(250)
mac 2, 3, and 4	= 1000	(925)
mac 5 (diffus)	= 150	(75)

```
Hence total size for the lab = total2
     = 1650    (1250)    lines
     = total size for the core of the program
     = total(sC)
```

1A.1.3 Total Program

```
Hence the size for the total program
     = total(sP) = total(sC) + total(sI)
     = 2116 (1240) + 1650 (1250)
     = 3766 (2490) lines
```

1A.2. Object Code Size

The size for the RUN-compiled object code is given here in terms of the memory words (60 bit each) for the CDC 6400 computer. A suffix "B" following a number tells that the number is in octal (i.e. base 8), all other numbers are represented in the usual decimal system.

Obviously, like the source code these are not the final sizes and not exact. Moreover, a couple of routines get duplicated under different names in the compiled code because of the way the program was developed. Hence the total size is slightly larger than what is really necessary.

```
object code = Data {global data (in labelled COMMON blocks)
              + The BLANK COMMON block}
              + Instructions { For the program code +
                        for the standard FORTRAN 4
                        subroutine library, and the
                        system routines, etc.}
```

1A.2.1 For the Data-Structure

1A.2.1.1 Labelled COMMON Blocks

The global data kept in labelled COMMON blocks (about 33 of them) stores the variables used from more than one routine.

```
Lex               = 420 B   = 272     words
Parser            =  41 B   =  33     words
Prpr              =   0 B   =   0     words
<ex-stmt-subr>s=      0 B   =   0   words
___________________  ________________
```

```
Hence total size of the data in the input interface is
     = 461 B = 305 words
```

Plus the "state-variables" of the machines and components in the lab take up 10177 B = 4223 words. Out of which "COMMON /EXPTBL/ EXPOS(21), RMZDOS(51,21)" for the

output of machine2 takes 2104 B = 1092 words, and "COMMON /MVSPOS/ RMZDOS(52,52)" for the output of machine3 takes 5220 B = 2704 words. So the rest take up only 653 B = 427 words.

Hence total number of words used in labelled COMMON blocks is : Interface + Lab = 461 B (=305) + 10177 B (=4223) = 10660 B (=4528) words.

1A.2.1.2 Blank COMMON Block

The Blank COMMON Block is used for temporary storage of big character arrays for plotting the various curves in the printer output. Its size is 30605 B = 12677 words.

Hence total number of storage used for data in ~~labelled~~ COMMON blocks is 41465 B = 17205 words.

1A.2.2 Instructions

1A.2.2.1 For the program code

Lex :	1214 B =	652
Parser :	2040 B =	1056
Prpr :	1504 B =	884

subtotal = 5040 B = 2592 words

Plus the <ex-stmt-subr>s take 1453 B = 811 words.

Hence the total instructions in the User Interface take 6513 B = 3403 words of memory.

The Machines in the lab take :

mac1 = horiz. image	=	1537 B =	863
mac2 = std. bleach	=	4751 B =	2537
mac3 = actual bleach	=	631 B =	409
mac4 = etching	=	4641 B =	2465
mac5 = diffusion	=	366 B =	246

subtotal for the 5 machines = 14570 B = 6520

Hence, {User interface} + {machines}
= {1453 B (=811)} + {14570 B (=6520)}
= 16243 B (= 7331) words

Further, program plpsp, and subroutine outitl take 4457 B = 2351 words (this is because of the buffer for tape21 - which was intended for use with the CALCOMP). So the total is

22722 B = 9682 words.

1A.2.2.2 For the System

The 100 B = 64 words in low core are used by the system, and the subroutines from the standard FORTRAN 4 library, and the operating system take up another 6400 B = 3328 words. Hence, together they take 6500 B = 3392 words.

Thus the total memory requirement for the instruction part is :

22722 B (= 9682) + 6500 B (= 3392)
= 31422 B (= 13074) words.

Hence the total memory requirement for running the program is

41465 B (= 17205) + 31422 B (= 13074)
= ~~72887~~ 73107 B (= 30279) words

Actually, some routines no longer used in the program but still present in the object files etc. make the memory requirement to be somewhat larger than the above, at present.

The program requires, at present, 103053 B (= 34347) words to load (the loader sits in the blank COMMON area while loading), and 111425 B (= 36653) words to execute.

References

Standard FORTRAN

1. FORTRAN vs Basic FORTRAN (FORTRAN IV vs FORTRAN II) - Communications of the ACM, v. 7, no. 10, Oct 1964, pp 591-625.
   {This was the one that was adhered to}

2. "Clarification of FORTRAN Standard - Second Report", CACM, v. 14, no. 10, Oct 1971, pp 628-642.
   {This gives references to other relevant documents}

Photoresists, Optical Models, etc.

1. F.H. Dill, A.R. Neureuther, J.A. Tuttle, and E.J. Walker, "Modelling Projection Printing of Positive Photoresists", IEEE Transc. on Electron Devices, vol. ED-22, no. 7, pp 456-466, July 1975.
   {And its references}

2. Discussions with

   Prof. William G. Oldham,
   Prof. Andrew R. Neureuther,
   and Michael O'Toole.

Etching Simulation Algorithms

1. R.E. Jewett, P.I. Hagouel, A.R. Neureuther, and T. van Duzer; "Line Profile Resist Development Simulation Techniques", Polymer Engineering and Science, June 1977, vol. 17, no. 6, pp 381-384

2. A.R. Neureuther, R.E. Jewett, P.I. Hagouel, T. van Duzer; "Surface-Etching Simulation and Applications in IC Processing", Proceedings of the Kodak Microelectronics Seminar, 1976, pp 81-91

Design Philosophy

1. Bernard P. Zeigler, "Theory of Modelling and Simulation" (A Wiley-Interscience Publication, John Wiley & Sons, 1976) Library of Congress no. QA 76.9C65Z44
   {A post facto reference. It presents many ideas, but was found after the program was already written and working.}

Usage of the Program

1. A.R. Neureuther, M. O'Toole, W.G. Oldham, S.N. Nandgaonkar; "Use of Simulation to Optimize Projection Printing Profiles", Meeting of the Electro-Chemical Society of America, at Seattle, Washington, May 21-26, 1978.

2. W.G. Oldham, S.N. Nandgaonkar, A.R. Neureuther, and M. O'Toole, "A General Simulator for VLSI Lithography and Etching Processes, Part I - Application to Projection Lithography", to be submitted to IEEE Trans. on Electron Devices, VLSI issue.

Acknowledgements

I take this opportunity to thank Prof. William G. Oldham for proposing this project to me. Apart from the financial support, he gave me a chance to design a product using the principles learned in both Electrical Engineering as well as in Computer Sciences, something I wanted to do when looking with awe at the SPICE program, and when musing, as the Teaching Assistant of EECS 133A (Power Systems Laboratory) in Winter 1977, about the design of an interactive program to simulate just one or two electrical machines (motors and generators) that a student could play with. Also, his screening out of minor issues made it possible to get this program up so fast.

Equally grateful am I to Prof. Andrew R. Neureuther for his support, guidance, and encouragement. (And above all, he liked the idea first.)

So also I appreciate the help of Michael O'Toole for patiently adapting his routines to the interface, and for various other suggestions regarding, and not regarding, the program.

Many of the concepts that went into this program suggested themselves in the various discussions that we have together.

This work was supported in parts, by grants (NSF) ENG77-14660 from the National Science Foundation, and other grants from the Hughes Aircraft Co., Intel Corp., and Signetics Corporation.

Part 2

Implementation Documentation for the User Interface

Implementation of an User_Interface

The input to the program is considered to be formed according to a two level language. The first level will be recognized by a lexical analyzer, and the second level will be recognized by a parser. Then a simple statement by statement interpretation will be done of the input statement stream.

The Lexical Analyzer for the input language is based on the following grammar.

```
        {The input characters are taken from the character set recogni-
         zed by the ANSI Standard FORTRAN, i.e.   A..Z  0..9 and the
         following special characters : " ", "+", "-", ".", ",",
         "(", ")", "$", "=", "*", and "/". }

   0    <end-of-ip_char> ::= $|{detected by the eof of input}
   1    <letter> ::= A|B| .. |Z
   2    <digit>  ::= 0|1| ..|9
   3    <sign>   ::= +|-
   4    <cont_symb> ::= /
   5    <separator> ::=  |(|)|,
   6    <start_symb>::= *
   7    <period>    ::= .

        {In this simple language identifiers and keywords are considered
         to be the same at present.  i.e. <id> = <kwd> }

        <kwd>   ::= <letter>|<kwd><letter>
        <unsigned_int>  ::= <digit>|<unsigned_int><digit>
        <fractional_part> ::= <unsigned_int>
        <unsigned_real> ::= <unsigned_int>.|.<unsigned_int>|
                                   <unsigned_int>.<fractional_part>
        <unsigned_number> ::= <unsigned_int>|<unsigned_real>|
                                   <separator><unsigned_real>
        <number> ::= <sign><unsigned_number>
        <eof_lex_token> ::= <end-of-input_char>
```

The numbers at the left of the BNF definition of the first 8 nonterminals are their associated ip_char_type (their 'meaning'). And the only three types of nonterminals that are considered to be legal <lexical_token>s are

```
   <lexical_token> {= <lex_token>}
    0    ::= <eof_lex_token>|
    1        <kwd>
    2        <number>
                 {The numbers on the left tell their type.  They
                  have some meaning associated with them, and that
                  meaning is as follows :
                         <eof_...> has no meaning assoc. with it;
                         <kwd> ~ the keyword formed;
                         <number> ~ the value of the number formed }
```

At this stage the lexical analyzer performs an additional function. It

checks whether the <kwd> belongs to a set of reserved words or not. It filters only the reserved words out to be sent to the parser. And these <reswd>s are { with their associated values {'rswdvl'}}

<reswd> ::=

1	LAMBDA
2	DOSE
3	TO
4	PROJ
5	CONTACT
6	LINE
7	SPACE
8	LINESPACE
9	ETCHRATE
10	ANALYTIC
11	CURVE
12	DEVTIME
13	RESMODEL
14	RUN
15	LAYERS
16	TRIAL

Now with these <reswd>s {which form a subset of the set of <kwd>s} the legal input is defined to be {at this lexical analyzer level, i.e. level 1} :

```
        Legal_input = stream of the legal input lexical tokens
        {as a regular expression}
        ::= <sep>*<lex_token>{<sep>+<lex_token>}*<sep>+<eof_lxtoken>
            or rather
            <sep>*{<lex_token><sep>+}*{lambda|<lex_token>}<eof_ltoken>
                {in somewhat ad hoc notation}
                {The <lex_token> in the above means a non<eof_ltoken>.}
```

Also, for brevity, a physical input card may be considered to be

```
        <start_char>{<lex_token>*}<end-of-card_char>
```

{Which has some implications on the implementation {e.g. fixed finite physical length}.}

At present no comments are allowed. But they can be easily introduced {at various levels {of input}. e.g.

1) after a special character to the end of the physical card
2) as a string of characters enclosed between special character(s)
{3) as a statement of the language }

Also a special symbol to mark the end of an input-statement (optionally) can be introduced. {The repetition of such symbol generates the empty sentence which will have to be handled properly}.

The parser will take the stream of legal input <lex_token>s and parse it to form <statements>s = <stmt>s of the language. That becomes the second level of the definition of the language.

```
List of variables in the lexical analyzer (scanner) subprogram.

Common Blocks :

   /lexsca/     {For lexical scanning (some variables)}
        icsign  = input char sign as a value in { +1, -1 }
        ricsgn  = real form of icsign
        jrswdt(<max_length>, <num_of_reswd's>)
                = jreswd(10, 16)
                = reserved words table ( 16 reswd's, each at most 10
                   characters long)
        kwdarr(<max_length>)
                = kwdarr(10)   array to form the current keyword as
                   a lexical token.
        nmrswd = the constant <num_of_reswd's>
               = 16  {Initialized by a data-initialization-stmt}

   /lexsem/     {lexical analysis semantics.  This is the only data
                 communication link between the scanner and the
                 parser.}
        kwdval  = keyword value
        rnmval  = real number value
        intval  = integer_part value
        fraval  = fractional_part value
        rintvl  = real form of intval
        lxtkty  = lexical token type

   /charac/     {character handling functions ( the lowest level)
                 i.e. the physical level.}
        iprptr  = input physical record pointer (i.e. pointer to the
                   character in the 80 column card (with 82 column
                   image) that will be the next input character.)
        ipchar  = holds the current input character
        ictype  = tells the type of the character in ipchar
                   (as associated in the input grammer)
        ipcard(80 +2) = ipcard(82) image of the physical input card

   /errfla/     {holds various values of error indices}
        ierflg  = error flag ??
        iersvi  = (int) error severity indicator
        ismesi  = (int) sum of the esi's ( error severity indices )

   /endofx/     {logical flags for end-of-x}
        noipdk  = end of input physical deck
        noipr   = end of input physical record
        noilr   = end of input logical record
        noicst  = end of input character stream
        nologi  = end of logical input

   /prtime/     {for cpu time used etc.  not implemented.
                 program day, date, time, timer etc.}


Local variables and temporary variables
```

```
tmpsgn  = temp_var for sign (multiplier +1.0, or -1.0) in subr glexem
idigvl  = input digit value in subrs fumint, and frmfra
tdigvl  = real( idigvl ) in frmfra
ntenpr  = negative power of ten (in frmfra)
itemp1  = temp index for do-loop in subr frmkwd (2)
        = temp val of funtction ( in function idgval ) (11)
        = for implied do on read and write in subr gcard
```

-----....-----.....

Range of variables and their meaning for specific values
{i.e. some of them should be declared as constants of the program and used for symbolic reference with initialization by DATA stmt.}

```
/lexsca/.ictype = input character type
        = 0     $ end of logical i/p string symbol
        = 1     letter
        = 2     digit
        = 3     sign
        = 4     period
        = 5     separator
        = 6     start_symbol (*)
        = 7     continuation_symbol (/)
                (-1 < ictype < 8        esp. -1 => not from the above
                                                  set. ( i.e. "=" ))

/lexsem/.lxtkty = lex_token type
        = 0     <end-of-input_lex_token>
        = 1     reserved word ( kwd )
        = 2     number
        = -1    <invalid lex-token> = <erroneous_lex-token>

         kwdval = kwd (reswd) value
        = 1     LAMBDA          2   DOSE                 3   TO
          4     PROJ            5   CONTACT              6   LINE
          7     SPACE           8   LINESPACE            9   ETCHRATE
         10     ANALYTIC        11  CURVE                12  DEVTIME
         13     RESMODEL        14  RUN                  15  LAYERS
         16     TRIAL           and -1 for unknown kwd (<invalid kwd>)
```

Subroutines and their arguments, call structure (as a tree (notation for writing down is by indentation , and arrows)), and naming conventions etc.

Subroutine names

```
        gxxxxx  => get<~xxxxx>
        fxxxxx  => fetch<~xxxxx>
        frmxxx  => form<~xxx>
```

Some features of the scanner

1) Machine Dependent Features
   1) common /prtime/ ~
      and related features {CAL RUN FORTRAN p11-6}
   2) program statement with tape declarations etc.
   3) eof(input)
      esp. reading a blank card at the end of input which appears in the output (but the program is not dependent on it).
   4) function ipchty comparing hollerith data will have to be reduced to a giant .or. stmt.
   5) all character data constants used in the expressions will have to be replaced by variable(constant)s initialized by a DATA stmt.
   6) seems to assume at least one input card as input.
      (see subr gcard) (verify carefully) (March 9, 1978)

2) Improvements possible
   1) In recognizing numbers
   2) Scale factors for numbers
   3) for scanning errors locating the position of the error by an up_arrow.
   4) include comment stmt in the language
   5) numbers should be recognized as real numbers and integers.
      (and used as such in the parser input grammer).

3) Redundant Features
   in handling some errors which can never arise,
   in subr glexem   ictype = 5 is not possible
   in subr frmint   first char will be a digit (always)
   etc.

4) Side-effects, bugs, etc.
   Two kwds must be separated by at least one separator
   123.456.789 will be recognized as two numbers 123.456 and .789

Defining the Syntax and Semantics for the Parser

The parser input language is defined to be a simple type 3 (regular) language (so is recognizable by a finite state machine). It's semantics is based on the intended meaning of the statement by the user. This input language is defined over all possible combinations of the lexical tokens obtained from the lexical analyzer.

The definitions of syntax and semantics are given below. In the syntax definition upper case letters are used for the keyowrds in the input and lower case letters are used for other tokens (i.e. numbers, <end of input char stream> lexical token, and the <error lex-token>).

In the definitions the <erroneous-stmt> is not explicitly shown. An <error lex-token> occurring in a statement, or any arrangement of lexical-tokens other than the ones shown below is considered to be generating the <erroneous-stmt>. Erroneous-stmts are handled by the error-handling routine in the parser and another routine which skips over the current <erroneous-stmt> till the next sensible beginning for a statement (i.e. a proper keyword or the <end of input char stream> lexical token is found.

The semantics, i.e. the meaning, of all other statements is given immediately after the syntax definition in terms of the parameters in the statement. The various different keywords that start a statement are considered to form a different type of input statement (totally there are 11 different types, numbered from 0 to 10 (both inclusive)). And within a particular type of an input statement the various different forms allowed are considered to be forming different kinds within that type of input statement.

To summarize, the <input-stmt>s are considered to have different 'type's (as shown below by the integer value to the left of the syntax definition) of input statements, and within each type are considered various different 'kind's of input statements possible within that type (indicated in their name in the definitions below).

* * *

{All numbers are considered to be real, unless explicitly mentioned to be integers}

-1) <erroneous-stmt> (= <invalid-stmt>)

0) <end-stmt> ::= <end of input char stream lex-token>

This statement informs the end of input statements to be processed, so various things to be done at the end of the run can be done.

<lambda-stmt kind 1> ::= LAMBDA number
this tells that there is only one wavelength present in the input spectrum, and it is equal to the value of number in microns.

<lambda-stmt kind 2> ::= LAMBDA number1 number2
| <lambda-stmt kind 2> number number
where the number pairs are the wavelength followed by its relative intensity. The wavelength (number1) is in microns and the relative intensities are all internally normalized by the program to get a sum of 1.0 . Upto 10 such pairs are allowed.

1) <lambda-stmt> ::= <lambda-stmt kind 1>
| <lambda-stmt kind 2>
with their meanings as defined above.

<dose-stmt kind 1> ::= DOSE number1
where number1 is the total dose at the main wavelength in units of (mJ/(sq.cm.)).

<dose-stmt kind 2> ::= DOSE number1 TO number2 number3
where number1 is as above and number2 is also another dose specification {number1 < number2}, and number3 is an integer which tells how many values of dose are to be selected in the interval [number1,number2]. These values are selected to cover the interval uniformly.

2) <exposure-stmt> ::= <dose-stmt kind 1> | <dose-stmt kind 2>
with the meanings above for the right hand side components. {This is the '<dose-stmt>'}

<proj-stmt kind 1> ::= PROJ number1
which tells that the imaging system is a projection type imaging system, with its Numerical Aperture = number1.

<proj-stmt kind 2> ::= <proj-stmt kind 1> number2
where the meaning of <.> is as above and moreover the wavelength at which this value of number1 holds is given by number2 (in microns).

<proj-stmt kind 3> ::= <proj-stmt kind 2> number3 number4
| <proj-stmt kind 3> number
where the number3 is an integer telling that that many values for the MTF weights are going to follow number4 which is the cutoff frequency of that MTF. So the values of frequencies at which those values of MTF-weights holds are the number3 values spread at equal intervals between

the zero (spatial) frequency to the cutoff freq. number4 (both inclusive).

<contact-stmt kind 1> ::= CONTACT number1
where this tells that the imaging system is a contact type imaging system with the separation between chip and mask being number1 microns.

<contact-stmt kind 2> ::= <contact-stmt kind 1> number2 number3
where this tells that the contact-type imaging system specified as before has an additional specifications that the parameters C1 and C2 (see part 3 of this report) are given by number2 and number3 resp.

<system-stmt kind 1> ::= <proj-stmt kind 1>
<system-stmt kind 2> ::= <proj-stmt kind 2>
<system-stmt kind 3> ::= <proj-stmt kind 3>
<system-stmt kind 4> ::= <contact-stmt kind 1>
<system-stmt kind 5> ::= <contact-stmt kind 1>
with meanings as above

3) <system-stmt> ::= <system-stmt kind 1>
¦ <system-stmt kind 2>
¦ <system-stmt kind 3>
¦ <system-stmt kind 4>
¦ <system-stmt kind 5>
with meanings as above.

<mask-stmt kind 1> ::= LINE number
which tells that the mask is a single line with width equal to number microns (A line is a fully opaque region).

<mask-stmt kind 2> ::= SPACE number
which tells that the mask is a single space with width equal to number microns (space = fully transparent region)

<mask-stmt kind 3> ::= LINESPACE number1 number2
which tells that the mask is a periodic pattern of lines and spaces with linewidth = number1 microns, and spacewidth = number2 microns.

<object-stmt kind 1> ::= <mask-stmt kind 1>
<object-stmt kind 2> ::= <mask-stmt kind 2>
<object-stmt kind 3> ::= <mask-stmt kind 3>
with meanings as above.

4) <object-stmt> ::= <object-stmt kind 1>
¦ <object-stmt kind 2>
¦ <object-stmt kind 3>
with their meanings as above.

<er-analytic stmt> ::= ETCHRATE ANALYTIC number1 number2 number3
where (er = etchrate), number1, number2, and number3 are the

E1, E2, and E3 parameters that specify the etchrate as an analytic function of the M-parameter (see part 3 of this report).

<er-curve stmt> ::= ETCHRATE CURVE number1
| <er-curve stmt> number

which tells that the numbers following the keyword CURVE are to be interpreted as follows : number1 = the number of number's that are going to follow in that statement. And the following number's are the values of the etchrate as a function of M-value at the equispaced number1 points (hence (number1 - 1) divisions) on the closed interval on M = [0,1]. The etchrate is specified in units of Angstroms/sec.

<devmodel-stmt kind 1> ::= <er-analytic stmt>
<devmodel-stmt kind 2> ::= <er-curve stmt>

with meaning as above.

5) <devmodel-stmt> ::= <devmodel-stmt kind 1>
| <devmodel-stmt kind 2>

with meanings as above. So a <devmodel-stmt> specifies the development model to be used in the etching machine.

<devtime-stmt kind 1> ::= DEVTIME number1

which tells that the chip is to be developed for number1 seconds with final etch-contours output at the end of the development (i.e. the etching).

<devtime-stmt kind 2> ::= DEVTIME number1 TO number2 number3

which tells that the chip is to be developed for number2 seconds, with totally number3 (an integer) outputs, at equispaced points from number1 to number2 (both inclusive). (number1 < number2)

6) <devtime-stmt> ::= <devtime-stmt kind 1>
| <devtime-stmt kinf 2>

with their meanings as above.

7) <resmodel-stmt> ::= RESMODEL number1 number2 number3 number4 number5 number6 number7

which specifies the photo-resist model to be used. The numbers are interpreted as follows :

- number1 = the wavelength at which this model holds (intended for use when multiple wavelength case can be handled, at present its value is ignored - but a number must be present at this position)
- number2 = The A parameter of the resist (see part 3)
- number3 = The B parameter of the resist (see part 3)
- number4 = The C parameter of the resist (see part 3)
- number5 = The real part of the index of refraction of the photo-resist at the wavelength of number1 microns.
- number6 = The imaginary part of the index of refraction of the photoresist to go with number5. At present it's value is ignored; the imaginary component of the index of refraction is found from the

expression k = the imag. part = -(A+B)(lambda)/(4 pi) where lambda is the wavelength at which A, and B are specified, and pi = 3.14159265... .

number7 = the thickness of the photo-resist film on the chip.

8) <run-stmt> ::= RUN
| RUN number1

which tells to run the machine whose number is equal to number1 (an integer) if number1 is in the interval [1,4]. If number1 is equal to 0 then the machines 1, 2, 3, and 4 are run in that sequence. If number1 is not present - as in the first variant of this stmt - then it is considered to be equal to 0 and hence the first variant is converted to the second variant internally.

9) <layers-stmt> ::= LAYERS number1 number2
| <layers-stmt> number3 number4 number5

where number1 tells the real part of the refractive index of the substrate of the chip, and number2 tells the imag. part of that index (The substrate is considered to be essentially infinitely thick). The following numbers tell information about the other layers present on the chip (only a maximum of 2 such layers (e.g. for oxide and/or nitride) is allowed) in the following fashion :

number3 = real part of index of refraction for that layer
number4 = imag. part of the refractive index for it
number5 = the thickness of the layer.

The layer closest to the substrate is specified first, etc. The photoresist layer is not considered to be a part of these layers (it is specified in the resmodel-stmt as shown above).

10) <trial-stmt> ::= TRIAL number1
| <trial-stmt> number

This stmt gives the numbers to the user-defined subr 'extria' to handle them as it wishes. The intent of this stmt is to be able to introduce new things in the program very conveniently on a trial basis. Its use is limited only by the user's ingenuity in writing the extria subr (etc).

The Data-Structure for the Parser

The only data communication between the lexical-analyzer and the parser occurs through the lexical analyzer COMMON block

```
      common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
```

Apart from this only the error flag that the lex sets on detecting an error can affect the parser (because of lex). The block /lexsem/ is explained in the lexical analyzer data structure (file docu1 (= 'lexdef')).

The data structure used by the parser is as follows.

```
common /parsem/ {Parser semantics.  This is similar to /lexsem/ for
                 lex.  This block is for the communication of data
                 between the parser and the interpreter (and the
                 prettyprinter).}
        istmty = (int) statement type {from the stmt-kwd}.  Its values
                 indicate
                 {-2    stmt synthesizing started (in subr gstmt)}
                  -1    <erroneous-stmt>   (= <invalid-stmt>)
                   0    <end-stmt>
                   1    <lambda-stmt>
                   2    <exposure-stmt>
                   3    <system-stmt>
                   4    <object-stmt>
                   5    <devmodel-stmt>
                   6    <devtime-stmt>
                   7    <resmodel-stmt>
                   8    <run-stmt>
                   9    <layers-stmt>
                  10    <trial-stmt>
        istknd = (int) statement kind {i.e. which flavour of the
                 above statement type}.  (If the stmt formed has no
                 different kinds i.e. only one kind then 0 )
        stnmls(25) = list of numbers in the statement
                 (stmt.number.list)
        nminst = pointer to the last number in stnmls { hence =
                 the total number of numbers in the statement }
        nmpntr = number pointer (tells which number the parser is
                 is considering at present)
```

```
common /endfl2/ {all logical flags}
        nostmt = end of stmt stream (flag for use by the controller)
                 = .TRUE. if <end-stmt> is executed, .FALSE. otherwise
        lxused = if the current lexeme was 'used up' then .TRUE.
                 else .FALSE.
```

```
common /errfl2/ {error flags}
        lparer = (logical) parsing error (syntax not according to
                 the parser-expected ('good') grammar)
        legstm = if the stmt (being) formed and (to be) returned
```

by gstmt is 'legal' then .TRUE. ,
.FALSE. otherwise

-----....-----.....

Names of subroutines, and variables etc.

subr fsxxxx = subr to form-stmt-xxxx

The prettyprinter for the statements is written to output the meaning of the input statements in an expanded form, the way it is going to be understood by the <ex-stmt-subr>s. It does not need any global data-structure for itself. It gets the input stmt from the parser in the COMMON block /parsem/.

A machine readable output can be outputted (e.g. on cards for 6400, or just the normal output for use with Unix) which can be read back as input and is nicely formatted, but the simplicity of the input does not make this worthwhile.

The variables associated with the operation of machine 1 (the horizontal imaging machine) are :

Common Blocks :

For getting the information from the input statements to the machine :

```
        {All lengths are in microns.}
        {CONSTANTS in each common block are in upper-case letters}
        {default values of all variables are as in subr inim01}

   /errlab/     {For any error in the 'lab'}
        lnoerr = (logical) no error
                 when true => no error detected upto now


   /math/       {Mathematical constants}
        PI     = pi = 3.14159265358979...
        T2BYPI = 2/pi


   /spectr/     {Illumination spectrum (in the normalized sense).  For
                 each lambda-stmt and mc1 (= m01).  Dose-stmt is a
                 separate one.}
        MXNLMD = max number of lambdas (i.e. optical wavelengths) = 5
        nmlmbd = actual no. of wavelengths
        rlambd( MXNLMD ) = rlambd( 5 ) = the wavelengths
        relint( MXNLMD ) = relint( 5 ) = relative intensity at the
                 corresponding wavelengths ( so that their sum = 1.0 )


   /objmsk/     {The object (= mask) is described here}
                {For object-stmt, and mc1 (=m01)}
        MLINE  = mask is a line = 1
        MSPACE = mask is a space = 2
        MLNSPA = mask is a periodic pattern of line and space (i.e. a
                 grating) = 3
        maskty = mask type ( from the set {MLINE, MSPACE, MLNSPA})
        rlw    = (real var) line width
        rsw    = (real var) space width


   /imgsys/     {The opto-mechanical image-forming system is described
                 here}
                {For system-stmt and mc1 (=m01)}
                {Associated with and continued into /img2pr/, and /img3co/}
        IMSPRO = 1 => projection type system
        IMSCON = 2 => contact-type system
        imsyty = (actual) imaging system type (from the set {IMSPRO, IMSCON}


   /img2pr/     {For projection type ststem}
         rna(MXNLMD, 2) = rna(5, 2) = NA for each wavelength (~,1),
                 and the wavelength (~,2).
                 {Note that this stores wavelengths}
```

```
        MXNWTF = max no. of mtf weight factors = 21
                 {From zero frequency to a max freq}
        nmwtfc(MXNLMD) = nmwtfc( 5 ) = number of weight factors for
                 that lambda
                 { 0 => none, so 0 => diffraction limited mtf }
        rmtfwt(MXNLMD, MXNWTF) = rmtfwt(5, 21) = the mtf weight factors
                 for equispaced vn/vc at a given wavelength (upto the
                 spatial frequency specified in tospfr( i {= lambda-
                 index} ).
        tospfr( MXNLMD ) = tospfr( 5 ) = to spatial frequency at that
                 wavelength ( i.e. at that lambda, there are nmwtfc
                 weight factors given upto this spatial frequency,
                 including the two end points, and at equispaced
                 intervals )
                {----- at present this is in /imgsys/~  because it does
                 not fit in a single card (line) in the common stmt
                 for /img2pr/~}


   /img3co/      {For contact type system}
          sepmtc = mask to chip separation    {in microns, as specified}
        c1      = c1       { Of the formula for intensity as I =         }
                {  = I0 * c1 * exp( -c2 * x * sqrt( 2/(lambda*sepmtc) )}
        c2      = c2       { in the formula above }
```

```
More variables for machine 1.

Common Blocks :

Variables internal to the system (lab) machines
        {At present horizontal image = sum of all horizontal images at
         all wavelengths * their relative intensities }

        Spatial frequency components of mask and image are specified
        by the parameters { relevant only for a projection type system }
           frequency = {f0=0.0, f1, f2, ..., fn} = {fi} = {f[i]}
           amplitude = {a0, a1, a2, ..., an}     = {ai} = {a[i]}
           { phase is zero, (even symmetry of the mask)
             i.e. all are cosine functions}
           so that i = component number
                     = {0, 1, 2, ..., n}
        'Normalized' means
           frequencies converted to vn/vc = fn/fc = fn/vc
                where vc is the cutoff frequency (spatial) at that
                lambda for the lens.
           amplitudes (, and phases ) same as before.


   /fouser/     {Fourier series parameters of the mask and image, and
                 associated parameters of tha lens.}
                {continued into /fouse2/}
        MXNMFR = max number of spatial frequency components (of the
                 mask, and image) = cardinality of i = 11  (so that
                 i = 0, 1, 2, ..., 10)
        nmfrcm = number of spatial freq components (of mask, and image)
                 hence max value = MXNMFR - 1
                 {For each wavelength the calculations go on independ-
                  -ently.  Hence this need not be an array indexed on
                  the wavelength}
        fsfrqa( MXNMFR ) = fsfrqa( 11 ) = Fourier series frequencies
                 actual  {for both mask and image}
        fsfrqn( MXNMFR ) = fsfrqn( 11 ) = Fourier series frequencies
                 normalized  {for both mask and image}


   /fouse2/     {continuation of /fouser/}
        fsamsk( MXNMFR ) = fsamsk( 11 ) = fs amplitudes for the mask
        fsaimg( MXNMFR ) = fsaimg( 11 ) = fs amplitudes for the image
                 {fsaimg(.) is not used at all in this version.  It is
                  implicitly calculated as local scalar variable}


The computation for each wavelength goes on separately and the results
are additively accumulated in the output (horint(.)).  That gives the
proper output (for this version).


   /lenspa/     {Lens parameters (for projection type systems)}
        vc     = the cutoff frequency of the lens (mtf) equation at
                 that lambda
```

```
        dlmtfn( MXNMFR ) = dlmtfn( 11 ) = diffraction limited mtf
                at the (normalized) frequencies (vn/vc)
        actmtf( MXNMFR ) = actmtf(11) = actual mtf (at the normalized,
                or actual frequency components)


--------------------


Output of machine 1


   /horimg/      {Horizontal image}
        deltax = (xgriun =) x grid unit
        MXNGDV = max number of grid divisions = 50
        MXNGPT = max number of grid points = (MXNGDV + 1) = 51
        nmgrdv = actual number of grid divisions (= 40 at present)
                 {should be made user specifiable}
        nmhpts = actual number of horizontal grid points = nmgrdv + 1
                 {changed----- nmhpts = nmgrdv = 40 (because of the new
                  convention that the grid points are at the midpoints
                  of the grid divisions rather than the endpoints of
                  the divisions)}
        horint( MXNGPT ) = horint( 51 ) = horizontal intensity at the
                grid points {i.e. at deltax * 'i'}


   /ipstm1/      {input state (relevant parts) information of machine 1
                  preserved at its output for keeping track of what
                  input does the output correspond to.}
                 {Hence information from lambda-stmt, system-stmt, and
                  object-stmt should be stored here}
        [--- not implemented at present ---]
        {From lambda-stmt}
                 {copy of ~  =  c~ }
        ncmlmd <- /spectr/.nmlmbd
        crlmbd( 10 ) <- /      /.rlambd( MXNLMD )
        crelin( 10 ) <- /      /.relint( MXNLMD )
        ----------
        {From object-stmt}
        ncmskt <- /objmsk/.maskty
        crlw   <- /      /.rlw
        crsw   <- /      /.rsw
        ----------
        {From system-stmt}
        {What other things are wanted here?}
```

Jun 19 02:50 1978   File m01varc   Page 1

Local variables (temporary variables) for the horizontal image machine
i.e. machine 1 (= mc1 = m01)

```
subr exlmbd
   sumtmp = ...
```

Jun 19 02:55 1978   File errmesm01   Page 1

Error Message Summary for machine(1)

In case of an error detected by machine 1 the error message will be

```
***_+_***            ERROR IN MACHINE(1) =nnnnn
```

where the nnnnn is a decimal number. Its meaning is as follows :

10   (in function dmtfna(vnbyvc))   vnbyvc > 1.0

20   (in function dmtfna(vnbyvc))   vnbyvc < -0.00001

30   (in subroutine exrun) parameter to the run-stmt was out of the legal range for it.

40   (in subroutine runmc1) imsyty != imgpro, or imgcon
i.e. neither a projection type system nor a contact type system
(this error is not user-generatable)

50   (in subroutine runmc1) mask is neither line, nor space, nor linespace

60   (in subroutine runmc1) like 10 (for contact type system)

70   (in subroutine exsyst) number of weight factors (= itemp2) is outside the legal range of 0 < itemp2 <= mxnwtf

Documentation of the data-structure for interfacing machines 2, 3, and 4 to the controller (through the <ex-stmt-subr>s).

All the relevant data is kept in labelled COMMON blocks as follows.

```
Exposure-stmt  ->  Exposure information

Machine(3)

/exposu/          {Holds exposure information provided by the
                   user in the exposure stmt}
    mxnmex = max number of exposure steps = 11
    nmexpo = number of exposures (exposure steps)
    ncouex = number count of exposures finished by now
    dose1  = initial dose(for the first step)
    dose2  = final dose (for the last step)
        {dose1, and dose2 are 'dial-setting's,  dose1 < dose2}
    dose   = actual dose 'already' given at present {i.e. the
             current reading on the dial}
    dosest = dose step = exposure step = (dose2 - dose1)/(nmexpo - 1)

subr exexpo makes it possible to treat both kinds of <exposure-stmt>s
uniformly.
local vars for subr exexpo
    temp = real(nmexpo)
           and real(nmexpo) - 1.0

subr inim
    {Contains all the common blocks for machines 2, 3, and 4 required
     to communicate with the controller}
local vars
    temp - for conversion from integer to real


Variables for subr exdvtm
devtime (etchtime) stmt  ->  dial setting in machine(4)
COMMON Blocks

/etchtm/          {etchtime (= devtime) information}
    mxneht = max number of etch-time steps = 11
    nmehtm = number of etching steps ( < mxneht )
    ncoeht = (number) counter for etch steps
             { 0 <= ncoeht < (=) nmehtm }
    ehtm1  = initial etchtime (for the first step)
    ehtm2  = final etchtime (for the last step)
    ehtm   = actual etchtime 'already' given at present
    ehtmst = etchtime step = (ehtm2 - ehtm1)/(nmehtm - 1)

(Like subr exexpo) subr exdvtm makes it possible to treat both the
kinds of the etch-time-stmts (=devtime-stmts) uniformly (in terms
a physical machine dial-setting).

/ethpar/ ~  {From Mike}

local var
```

```
    temp = real(nmehtm), and real(nmehtm) - 1.0

Handling the information coming from the resmodel-stmt.

/resmod/          {contains resist model information for bleaching
                   (and not for etching)}
    rslmbd = the wavelength at which the A, B, C parameters, and
             refractive index have been specified.
    prthic = PR layer thickness
    prpara = Photoresist parameter A
    prparb = PR parameter B
    prparc = PR parameter C

/resmo2/          {Continuation of /resmod/ ~ }
    prparn = PR refractive index parameter n (the real part)
    prpark = PR refractive index parameter k (the imag. part)

No local (temp) variables in subr exrsmo.
/chipar/ ~  } {From Mike}
/respar/ ~  }
subr exlaye

/laystm/          {should put mxnlay, nmlaye etc.}
                  {not put in.  see /chipar/ }

    local vars
    nlayet = number of layers (temporary)
    ilayer = index on layers, and also a DO-loop index
    itemp1 = do-loop index (counting variable for number of layers)
    iwvlen = index on wavelength (at present iwvlen = 1)
    iparse = parser index (on parsem(25))

    nmstra = number of strata (i.e. substrate, layer1, layer2, and
             PRlayer make 3 strata)
           = nmfilm = number of films
             {at present a local variable}
/layers/ ~   {From Mike.}
For subr exdvmo

    devmodel stmt = etchrate stmt

/erpara/          {Devmodel information = etchrate information}
                  {Etch rate parameters}
    kerspe = etchrate specification type = {kerfun, or kercur}
    kerfun = (constant) etchrate is specified as an analytic
             function = 1
    kercur = (constant) etchrate is specified by equidistant
             points on a curve
    etche1, etche2, etche3 should be in this common block but
        at present they are also in Mike's /ethpar/ block.
        These are the E1, E2, and E3 parameters for etchrate
        analytic specification

/erpar2/          {continuation of /erpara/ }
    mxnerd = max num of etchrate curve specifying divisions = 20
```

```
    mxnerp = max num of etchrate points = (maxnerd + 1) = 21
    nerdiv = (actual) number of etchrate divisions
    nerpts = (actual) number of etchrate points
    etchra(mxnerp) = etchra(21)
           = etchrate curve points specified by the user

/ethpar/         {From Mike}

local var        itmep1 = parsem(1)
```

The common block /simpar/ (from Mike) has the 'simulation parameters'
i.e. the discretization parameters
nprlyr, nprpts, nendiv, deltm, deltz
These are the discretization step sizes (variables) (or related to them) necessary for the purpose of simulating a continuous process on a digital computer.

The numerical accuracy of simulation (w.r.t. the continuous model) is dependent on the vaules of these variables, so also the total computing time (and hence the cost) of the simulation.

```
Error message numbers from the <ex-stmt-subr>s to indicate
errors in stmts for machines 2, 3, and 4.
(macnum = machine number)

      subroutine errm(macnum, ierrnm)
produces the output.


macnum = 2

ierrnm = 10 the number of layers (=nmlayet) (including the
                substrate, but excluding the photoresist layer
                does not satisfy the condition
                    ((1 .le. nlayet) .and. (nlayet .le. 3))
                        {The 3 is mxnlyr - 1}
                called from subr exlaye


macnum = 3

 ierrnm = 10 number of exposure steps specified in the input
                (= nmexpo) is outside the legal range of
                2 <= nmexpo <= mxnmex
                where mxnmex = 11
          20 tells that the condition (0.0 <= dose1 < dose2) is
                not valid in the exposure-stmt.


macnum = 4

 ierrnm = 10, 20 are similar to machine3 case but for the etching
                machine.
        30 error in the input from the parser.
                total number of numbers in the stmt is not as said
                to be in the input first numerical parameter.
        40 number of etchrate curve specification points (nerpts)
                is outside the range
                    ((2 .le. nerpts) .and. (nerpts .le. mxnerp))
                where mxnerp = 21
```

Some features of the program

Change the RESMODEL stmt (in the next version) to two different statements so that the geometric information (PR thickness) is specified in one of them, and the A, B, C, parameters, the refractive index (only the real part), and the wavelength at which these values hold are specified in another.

Part 3

Implementation Documentation for the Machines and Components

Default Values :

The simulation requires a specification of values for various parameters used in the computation. These values may be explicitly specified by the user or they may be implicitly assumed by the program. SAMPLE has a relatively complete set of default values stored for all the components used in the computation. If the user respecifies a parameter-value in the input then the new specified value supersedes the previous value of that parameter. The default-values were chosen to be a typical set of values found in actual processes.

Apart from the above parameter values, SAMPLE also needs to know various other control options for some actions to be performed (e.g. output of certain variables, or punching cards for the plotter etc.). These control switches (flags) can be set by the user or may remain at the default value (setting) assumed by SAMPLE. The default values for these were chosen to satisfy the typical minimal need of the user.

Also the default sequence of operations to be simulated when the user tells to run the machines by a 'RUN' or 'RUN 0' statement in the input is fixed to be the usual sequence of machines 1, 2, 3, and 4. i.e. (1) the horizontal image computation, (2) standard bleaching calculation, (3) actual bleaching calculation, and then (4) etching computation.

Since all the parameters and control switches are represented by variables in the program, this default values specification is done in the initializing subroutines (or may be done with DATA-statements (in a BLOCK DATA subprogram)). The following is a paraphrase of that initialization.

0) Miscellaneous :

The mathematical constant pi = 3.14159265358979 .
The no-error flag /errlab/.lnoerr is set to .true.

1) The default system configuration, component sizes, and other values :

1.1) The Horizontal Image Calculation :

The illumination spectrum :-
Number of wavelengths : maximum = 5, default = 1 .
The wavelength = 0.4047 microns,
with relative intensity = 1.00 (obviously).

The object (mask) :
a linespace pattern with linewidth = spacewidth = 1.0 microns.

The imaging System :
A projection type system with Numerical Aperture (NA) = 0.125 and maximum number of weight factors for the MTF = 21, default = 0 (i.e. diffraction limited MTF).

For a contact type system : C1 = 0.25, and C2 = 2.00 .

Discretization for Numerical Calculations :
The maximum number of frequencies to be used in the computation of Fourier components of mask and image = 11 (= default).

For the horizontal image representation, the x-axis grid unit = 1.0 microns (this gets superseded in the computations), maximum number of grid divisions on the x-axis = 50, and the maximum number of grid points = max. no. of grid divisions + 1 . Default of the number of grid divisions and the grid points is 40 (for both).

1.2) The Standard-Bleaching Calculation :

Positive Photoresist (PR) Model :
At wavelength of 0.4047 microns, the parameters of the resist are A = 1.055 (1/microns), B = 0.094 (1/microns), C = 0.02 (sq.cm./mJ). The refractive index = (n,k) a complex number = (1.70, -0.02) and the value of k gets superseded by the value calculated from the A and B parameters. $\{k = \frac{-(A+B)\lambda}{4\pi}\}$

The Chip Configuration :
Maximum number of layers in the chip, including the PR layer and the substrate, is 4, default = 2 (for the PR and the substrate only). The PR layer is 0.8 microns thick. The substrate has (n,k) = (5.613, -0.19).

Discretization parameters :
The number of PR grid divisions in the vertical direction, z-axis, is = 50 . The no. of grid points = no. of PR divisions + 1. The number of Energy divisions = 15, the relative inhibitor concentration M is divided on the basis of (0.4/(no.of energy divisions -1)) units to have a uniform coverage of the etch-rates. The grid unit in the z-direction = PR layer thickness/(no. of PR divisions).

1.3) Actual Exposure and Bleaching Calculation :
A single exposure (max = 11), with a dose of 50 milliJoules/(sq.cm.) at the mask.

1.4) Etching :
The default etchrate is an analytic fuction of M, with the parameters E1 = 5.67, E2 = 8.19, E3 = -12.5 . If curve of rate vs. M is to be specified then the max no. of etch rate divisions = 20 (for 21 points of M in [0,1]), default = 0 . The profiles are output 4 times from 20 seconds to 80 seconds.

2) <u>Control switches (for output)</u> :

Most of the relevant intermediate values are always printed out. But some lengthy output (for arrays) is not printed out by default. Also, the default option for the punching of cards for a plotter for the etch profiles is no.

Description of Machine1 : (Horizontal Image Formation)

Machine1 simulates the horizontal image formation on the resist surface from a given simple geometric pattern on the mask. The models used to characterize the process of image formation (which is not a time sequential process) are based on wave optics, and can be found in various textbooks and journals (see the References). Only the relevant assumptions, and final formulae as used in the program are given here. Also noted, where significant, are the modifications and approximations of the analytic formulation because of numerical analysis/calculation considerations and program time/memory constraints. Finally, the actual FORTRAN source code for the machine follows. Thus, in a sense, this is the definition of the models used for the real machine and the simulated machine.

Model for the process :

Modulation Transfer Function (MTF). The incident radiation is assumed to be fully incoherent. The topmost surface on the chip (i.e. te upper surface of the photoresist layer) is assumed to be at the image plane.

The information processing ocurring in the machine is as follows.

The illumination spectrum is specified by the number of discrete wavelengths in it and their relative intensities (sum total of relative intensities = 1.0).

The parameters of the illumination spectrum are :
Wavelengths in microns,
Relative intensity (as fraction of total intensity) at that wavelength.
The total radiation dose (mJ/(sq.cm.)) at the mask is not relevant here.

The mask is specified by its linewidth and/or spacewidth in microns.

Imaging system : Projection Type or Contact Type

For the Projection type imaging system

1) Numerical Aperture, NA
2) MTF for diffraction limited, incoherent, well-focussed case :
   Vc = 2(NA)/(wavelength)
   V = a spatial frequency (1/microns) of mask or image

s = V/Vc = normalized spatial frequency
MTF(s) = (2/pi)(arccos(s) - s*sqrt(1 - s*s))

3) If the assumptions of incoherence, and well-focussing do not hold then the MTF can be specified as weight factors by which the above MTF(s) should be multiplied. These weights are specified at some equispaced points in the range of s = [0,1], and the program calculates the weights at intermediate points by linear interpolation of weight between two neighbouring points, and then multiplies the MTF value calculated from the above formula by this weight to get (an approximation of) the actual MTF at that frequency s.

The 1D (spatial) Fourier Transform of the mask multiplied by the MTF gives the Fourier transform of the image. For the periodic pattern the Fourier transform becomes the Forier series, and calculations get simplified. But even for the single line or single space case, because of the cutoff frequency of the MTF, the pattern can be approximated by a periodic pattern of a reasonably long spatial period since at the higher frequencies the components diminish very fast in amplitude.

The Fourier series amplitudes and frequencies are found by the following formulae : (Because of the symmetry of the pattern around the axes chosen the phase is identically zero i.e. only cosine components are present)

L = Line width (microns)
S = Space width (microns)
(See Figure 1.)

$$A_0 = \left(\frac{L}{L + S}\right)$$

$$A_n = \left(\frac{1}{n}\right)\left(\frac{2}{\pi}\right)\sin\left(\frac{n\pi L}{L + S}\right)$$

So V_n = Fourier Series Frequencies are

$$v_0 = 0, \quad v_n = \frac{n}{L + S} \qquad n = 1, 2, \ldots$$

Where the last term is the largest term less than the cutoff frequency Vc of the MTF. The image intensity distribution along the x-axis is found by

$$I(x) = A_0 + \Sigma A_n \cos\left(\frac{2\pi n x}{L + S}\right)$$

Thus the horizontal intensity distribution for the projection type printers can be calculated.

For the <u>contact type</u> imaging system :

Mask to chip separation = h (microns)
(See Figure 2)

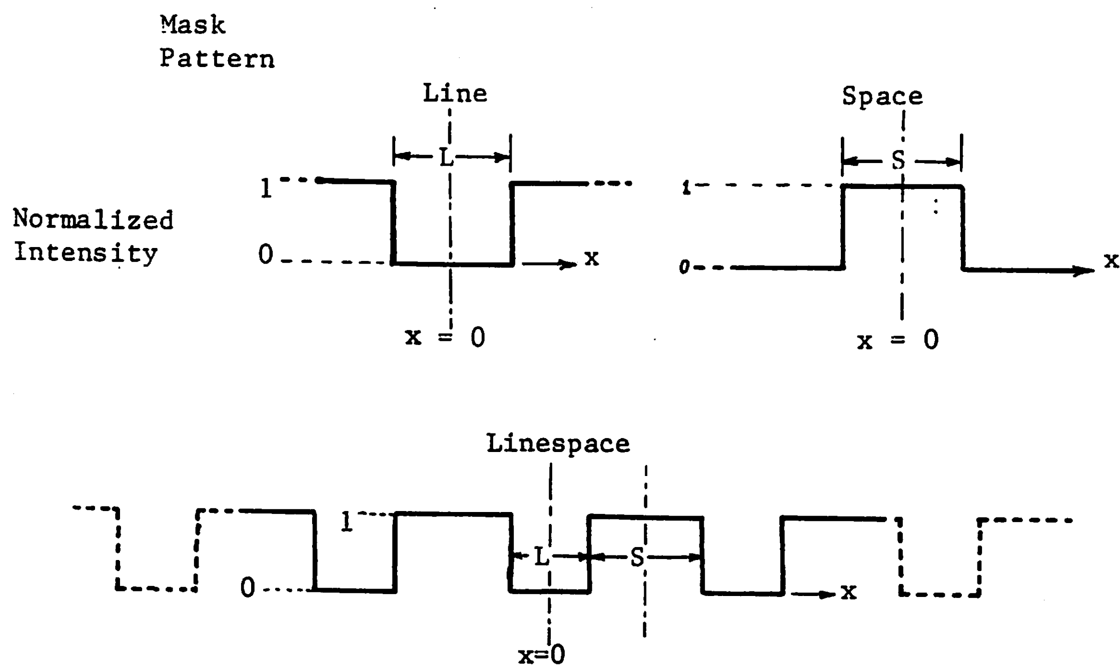


Fig. 1 Specification of a Mask

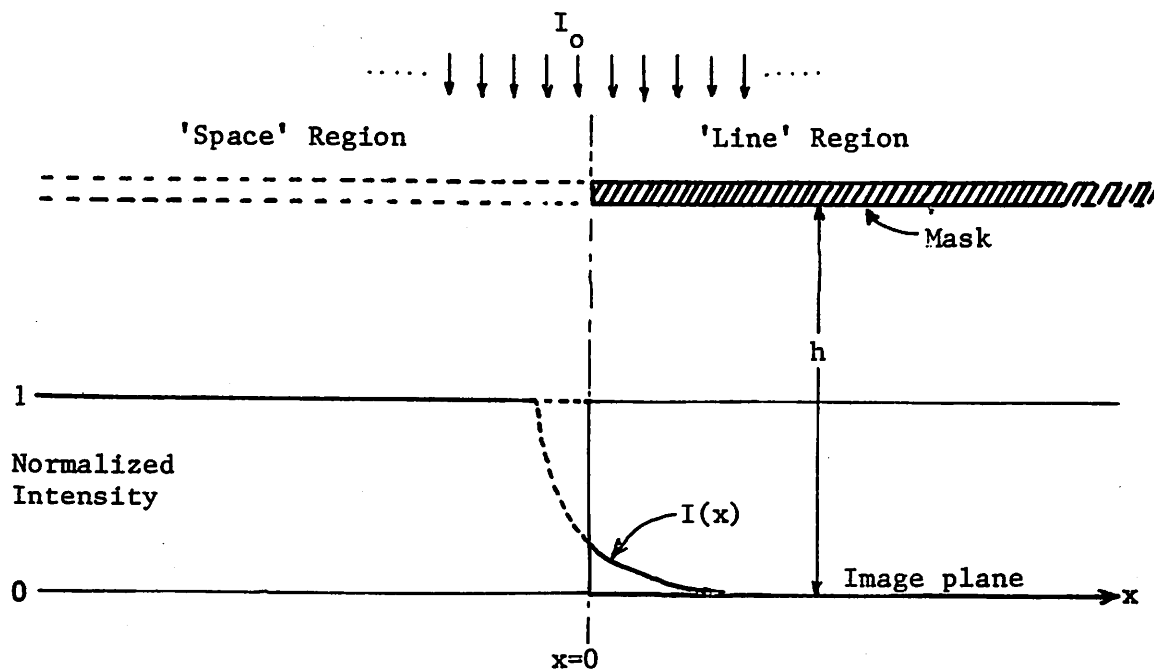


Fig. 2 Horizontal Image Intensity Distribution For a Contact Type System

The Intensity I(x) is approximated by

$$I(x) = C_1 I_0 \exp(C_2 \mathrm{sqrt}(\frac{2}{\lambda h}))$$

and is extrapolated backwards from the shadow region till it becomes equal to Io (units of mJ/(sq.cm.)).

Thus from the input illumination spectrum, mask geometry, and characteristics of the imaging system the horizontal image intensity distribution is found.

Note :

1) The ability of machine1 subroutines to handle multiple wavelength case is not well matched with the input interface yet,

2) Michael O'Toole has written a new version of this machine which treats partial coherence and defocus analytically rather than using weight factors. It will soon supersede the above version.

References : (for machine1)

1) Levi, Leo, "Applied Optics : A Guide to Optical System Design / vol. 1", John Wiley & Sons, 1968, (pp. xviii + 620)

2) See references in part 1 of this report.

Source Code

The Source Code Storage

The source code is arranged in the Unix login directory in various (sub)directories.

The directories holding the source code are as follows :

```
    lex-an       - Has 1) The subrs gcard, and gchar
                       2) The code for the Lexical Analyzer
    parser       - The Parser code
    prpr         - The Prettyprinter code
    mac1         - Has 1) The <ex-stmt-subr>s that deal with machine1,
                       2) The code for initializing the default values
                          for machine1,
                       3) The code for machine1 itself
    ifm234       - Has 1) The <ex-stmt-subr>s that deal with machines
                          2, 3, and 4.  And a default version of the
                          'extria' subr to execute the TRIAL-stmt by
                          doing 'nothing'.
                       2) The code for initializing machines 2, 3, & 4.
    diffusion    - Has 1) The current 'extria' subr
                       2) The diffusion machine (= machine5)
```

In these directories the source code is arranged as follows (some files need rearrangement).

Some of the routines that were useful in getting the program up but which are not in the final program code are also given below {in curly brackets}. Similarly for some other similarly useful files. The files holding CALIDOSCOPE control cards for CDC 6400, and the files holding shell commands to unix (for quickly putting together the relevant files for that module) are just a matter of convenience in using the system and hence not given here.
(prog = program, subr = subroutine, func = function)

```
1) Top level controller
    Directory ifm234
        Files           subprograms

        pplpsp  -       Prog plpsp  ( the main program )
        subrtitle -     subr outitl (for printing the title)
        srunlab -            runlab (the controller (see part 1, fig.4))
```

Now, the routines for the input interface.

```
2) The Parser routines
    Directory parser

        subr_1  -       subr initpa
                             gstmt
                             gflexem
```

```
                        skpstm
                        errpar(iarg1, iernum, messap)
        subr_2   -   subr fslmbd
                        fsexpo
                        fssyst
        subr_3   -   subr fsobje
                        fsdvmo
                        fsdvtm
                        fsrsmo
                        fsrun
        subr_4   -   subr fslaye
                        fstria
        {ststpar -   subr tstpar}
        {decl    -   declarations of data-structure for the parser}
        {pardrive -  program main (to test parser - its driver)}
```

3) The Lexical Analyzer routines
    Directory lex_an

```
        {decldata -  declaration of data-structure for lex_an}
        lex_1    -   subr errlex(iernum, iargsi, messag)
                        fkwdvl
                        frmfra
                        ~~frmkwd~~
                        ~~frmfra~~
                        frmint
                        frmkwd
                        gcard
                        gchar
                        glexem
                        gnsep
                     func idgval(iargch)
                     subr initla
                     func ipchty(jargch)
        {ststlex -   subr tstlex}
        {               procrd}
        {lexdrive -  prog main  }
```

4) The Prettyprinter (for stmts) routines
    Directory prpr

```
        pp       -   subr prprst
        {tpp     -   subr testpp}
```

The following are the files having the <ex-stmt-subr>s, and the error-handling routines, etc. They 'interface' the machines to the controller.

5) For interfacing machine1
    Directory mac1

```
        serrm01 -    subr errm01(n)
        subrinter2 - subr exlmbd
                        exsyst
                        exobje
```

6) For interfacing machines 2, 3, and 4
Directory ifm234

```
    sifc1    -     subr exstmt
                        exrun
    sifc2    -     subr errm(macnum, ierrnm)
                        exrsmo
    sifc3    -     subr exlaye
                        extria    (The default version. Does nothing)
                        exexpo
    sifc4    -     subr exdvmo
                        exdvtm
```

7) For interfacing other things on a trial basis
Directory diffusion

```
    trial1   -     subr extria    (The currently used version)
```

In part 3 of this report the machines in the core of the program are given. Their organization is as follows.

The initializing routines (should be made BLOCK DATA subprograms, and hence an integral part of the machines).

1) Initializing default values for the machines.
Directory mac1

```
    sinim01 -      subr inim01
```

Directory ifm234

```
    sinim234 -     subr inim
```

The machines themselves :

2) machine1
Directory mac1

```
    subrimage1 -   subr runmc1
    subrmtf -      func dmtfna(vnbyvc)
                   subr setmtf(ncomp, iwlen)
                   func (icompo, iwl)
    subrimage2 -   subr outmc1
```

3) The machines 2, 3, and 4 are in Mike's directory.

4) Machine5
Directory diffusion

```
    subr1batpt -   func df(r, sigma)
                   subr diffus(sigma)
```

-----....-----.....

```
1           program plpsp(input, output, punch,
2          1              tape5=input, tape6=output, tape7=punch)
3     c
4     c   the photolithographic processes simulation program.
5     c   authors - sharad n. nandgaonkar
6     c             michael o-toole
7     c              and others
8     c            (erl, eecs, ucb)
9     c   may 6, 1978
10    c
11          call outitl
12          call runlab
13          stop
14          end
```

Jun 20 05:18 1978 File ifm234/subrtitle Page 1

```
1           subroutine outitl
2     c   outputs the heading title to the output of the program
3     c
4           dimension kdash(80)
5           data kdash /1h1, 78*1h-, 1h0/
6     c
7     c   output the lines in the output heading
8     c   the output is centered for an 80 column printout
9           write (6, 9001) kdash
10          write (6, 9002)
11          write (6, 9003)
12          write (6, 9004)
13          write (6, 9006)
14          write (6, 9001) kdash
15    c
16     9001 format( 1x, 80a1)
17     9002 format( 1x, 4x, 5h*****, 28x, 5hplpsp, 29x, 5h***** )
18     9003 format( 1x, 5h*****, 12x,
19         1             46hphotolithographic processes simulation program,
20         2              12x, 5h***** )
21     9004 format( 1x, 32x, 16h(erl, eecs, ucb) )
22     9006 format( 1h0, 23x, 34h(version  0.0.0.0   may  6, 1978) )
23    c
24          return
25          end
```

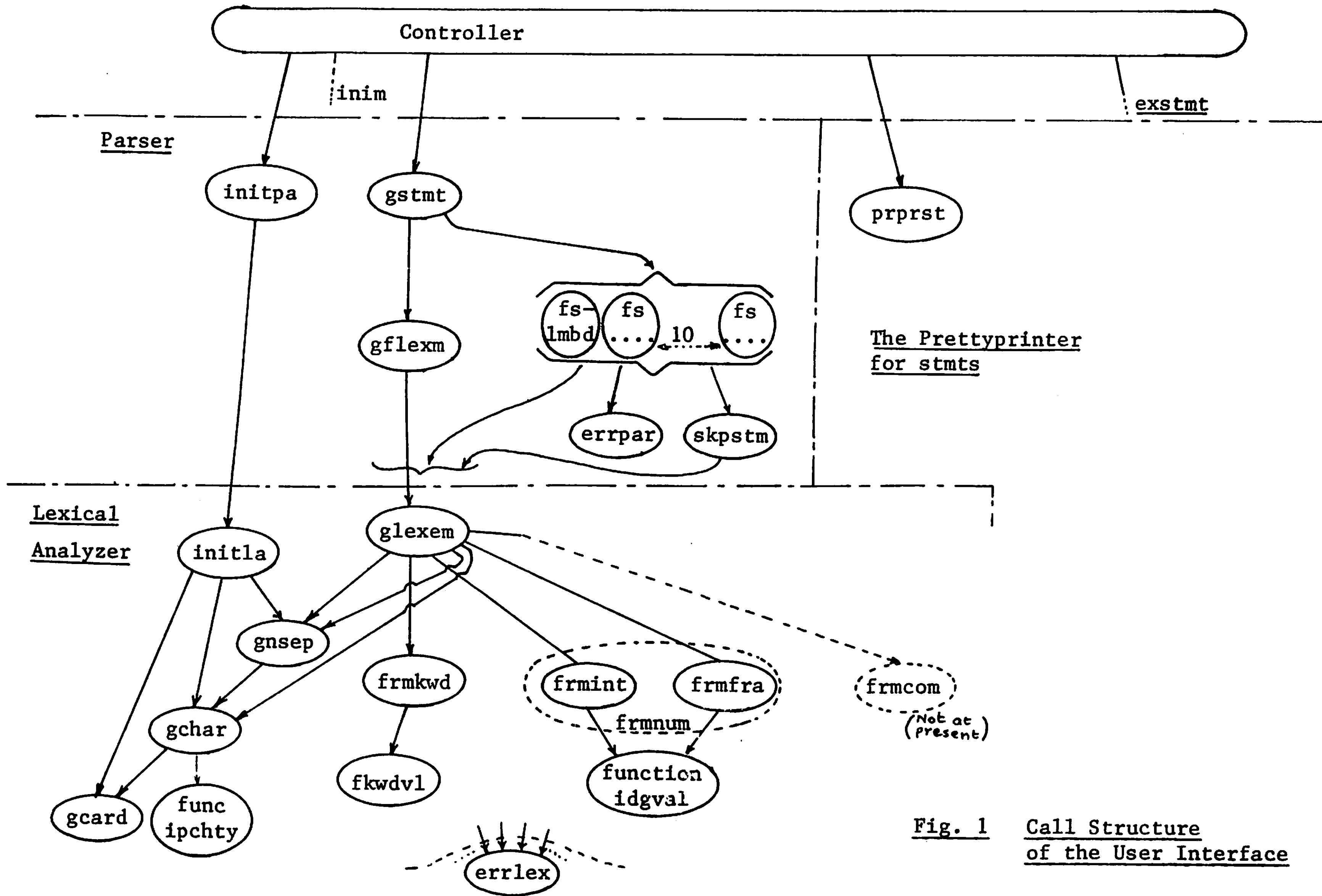


Fig. 1 Call Structure of the User Interface

```
1            subroutine runlab
2       c   driver subroutine for running the whole lab (i.e. machines 1,2,3,4)
3       c   of the photolithographic-processes-simulation-program.
4       c   -         (snn,  erl, eecs, ucb)        may 6, 1978
5       c
6              common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
7              common /charac/ iprptr, ipchar, ictype, ipcard(82)
8              common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
9              common /errfla/ ierflg, iersvi, ismesi
10             common /endofx/ noipdk, noipr, noilr, noicst, nologi
11      c
12             common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
13             common /endfl2/ nostmt, lxused
14             common /errfl2/ lparer, legstm
15      c
16             common /errlab/ lnoerr
17      c
18      c   data-structure for machine 1  (horizontal image machine)
19             common /math   / pi, t2bypi
20      c
21             common /spectr/ mxnlmd, nmlmbd, rlambd(5), relint(5)
22             common /objmsk/ mline, mspace, mlnspa, maskty, rlw, rsw
23             common /imgsys/ imspro, imscon, imsyty,          tospfr(5)
24             common /img2pr/  rna(5,2), mxnwtf, nmwtfc(5), rmtfwt(5,21)
25             common /img3co/   sepmtc, c1, c2
26      c
27             common /fouser/ mxnmfr, nmfrcm, fsfrqa(11), fsfrqn(11)
28             common /fouse2/  fsamsk(11), fsaimg(11)
29             common /lenspa/ vc, dlmtfn(11), actmtf(11)
30      c
31             common /horimg/ deltax, mxngdv, mxngpt, nmgrdv, nmhpts,horint(51)
32      c
33             common /resmod/ rslmbd, prthic, prpara, prparb, prparc
34             common /resmo2/  prparn, prpark
35             common /chipar/ mxnlyr, nmlyrs, rindex(5,4,2), thick(4)
36             common /respar/ wlabc(5,4)
37             common /simpar/ nprlyr, nprpts, nendiv, deltm, deltz
38             common /exposu/ mxnmex, nmexpo, ncouex, dose1, dose2, dose,dosest
39             common /dospar/ dos
40      c
41             common /erpara/ kerspe, kerfun, kercur, etche1, etche2, etche3
42             common /erpar2/  mxnerd, mxnerp, nerdiv, nerpts, etchra(21)
43             common /ethpar/ label(20), tout, nout, e1, e2, e3
44             common /etchtm/ mxneht, nmehtm, ncoeht, ehtm1, ehtm2, ehtm,ehtmst
45      c
46             common /exptbl/ expos(21), rmzdos(51,21)
47             common /mvspos/ rmxz(52,52)
48      c
49      c
50             logical         lnoerr
51      c
52             logical         noipdk, noipr, noilr, noicst, nologi
53      c
54             logical         nostmt, lxused
55             logical         lparer, legstm
56      c
```

```
57      c   initialize the keywords array ( with lambda, dose, to, proj etc. )
58            data jrswdt
59          1      /1hl, 1ha, 1hm, 1hb, 1hd, 1ha,  4*1h ,
60          2        1hd, 1ho, 1hs, 1he,           6*1h ,
61          3        1ht, 1ho,                     8*1h ,
62          4        1hp, 1hr, 1ho, 1hj,           6*1h ,
63          5        1hc, 1ho, 1hn, 1ht, 1ha, 1hc, 1ht,  3*1h ,
64          6        1hl, 1hi, 1hn, 1he,           6*1h ,
65          7        1hs, 1hp, 1ha, 1hc, 1he,      5*1h ,
66          8        1hl, 1hi, 1hn, 1he, 1hs, 1hp, 1ha, 1hc, 1he,  1h ,
67          9        1he, 1ht, 1hc, 1hh, 1hr, 1ha, 1ht, 1he,  2*1h ,
68          a        1ha, 1hn, 1ha, 1hl, 1hy, 1ht, 1hi, 1hc,  2*1h ,
69          b        1hc, 1hu, 1hr, 1hv, 1he,      5*1h ,
70          c        1hd, 1he, 1hv, 1ht, 1hi, 1hm, 1he,  3*1h ,
71          d        1hr, 1he, 1hs, 1hm, 1ho, 1hd, 1he, 1hl,  2*1h ,
72          e        1hr, 1hu, 1hn,                7*1h ,
73          f        1hl, 1ha, 1hy, 1he, 1hr, 1hs,  4*1h ,
74          g        1ht, 1hr, 1hi, 1ha, 1hl,      5*1h   /, nmrswd / 16 /
75      c
76      c   (the calling program should print the fancy title to the output)
77      c
78      c   initialize the program processes, variables, etc., and also the
79      c   defaultable values of the photolithographic system to be simulated
80      c
81      c   first initialize the parser
82            call initpa
83      c
84      c   initialize the horizontal image machine (= machine(1))
85      c
86            call inim01
87      c
88      c   also initialize the other three machines
89            call inim
90      c   ----- temporarily only -----
91      c
92      c   now process the input statements in an interpreter like fashion.
93      c
94      c   while the end of input statement stream has not been reached,
95
96      c   get an input statement ...
97         10  call gstmt
98      c
99      c   prettyprint it ...
100           call prprst
101     c
102     c   ...and execute it (only if no previous errors occurred, or if this
103     c   --- is the end-of-stmts stmt)
104           if (lnoerr .or. ( istmty .eq. 0 )) call exstmt
105     c
106     c
107           if (.not. nostmt) goto 10
108     c
109     c
110           return
111           end
```

```
1            subroutine initpa
2        c   inits (the lexical analyzer,) the parser, and also the system
3        c   (i.e. the two physical machines) to be simulated.
4        c
5            common /endfl2/ nostmt, lxused
6            common /errfl2/ lparer, legstm
7        c
8            logical         nostmt, lxused
9            logical         lparer, legstm
10       c
11       c
12       c   first initialize the lexical-analyzer
13           call initla
14       c
15       c   now initialize the parser
16           nostmt = .false.
17       c   now to cause the first lexeme to be read in ( 0-th lexeme is used )
18           lxused = .true.
19           lparer = .false.
20       c   - legstm need not be initialized
21       c
22       c   no parser processes have to be spawned (i.e. started).
23       c
24       c   initialize the photolithographic system to be analysed (i.e.
25       c    the defaultable variables ( and their flags etc.) ).
26       c
27           return
28           end
29           subroutine gstmt
30       c   this gets a statement from the input statement stream.  that is
31       c   done by forming a sensible statement out of the lexemes in the
32       c   input stream.
33       c   in short this is the parser.
34       c
35           common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
36           common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
37           common /endfl2/ nostmt, lxused
38           common /errfl2/ lparer, legstm
39       c
40           logical         nostmt, lxused
41           logical         lparer, legstm
42       c
43       c
44           legstm = .true.
45       c   preparation to form a new statement
46       c
47           istmty = -2
48           istknd = 0
49           nminst = 0
50           nmpntr = 0
51       c
52       c   get a fresh (i.e. first in a stmt and not used up by the previous
53       c   stmt) lexeme
54           call gflexm
55       c
56       c
```

```
57            if (lxtkty .eq. -1) goto 2
58            if (lxtkty .eq. 0) goto 1
59            if (lxtkty .eq. 1) goto 1000
60            if (lxtkty .eq. 2) goto 2000
61      c
62      c   if none of the above then a program error
63            call errpar(0, 1, 20hunknown lt in gstmt   )
64            return
65      c
66         2  call errpar( 0, 2, 20hinvalid lxtkty*gstmt )
67            call skpstm
68            return
69      c
70         1  istmty = 0
71      c   - call fsend
72            return
73      c
74      c   this branch means that a kwd heads the stmt.  hence see what it is.
75      c   --- (nmrswd in /lexsca/ = 16)
76       1000 if ((kwdval .lt. 1) .or. (16 .lt. kwdval)) goto 1999
77      c
78      c   a giant case-stmt ( with the help of a computed goto )
79            goto (1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100,
80           1      1110, 1120, 1130, 1140, 1150, 1160), kwdval
81      c
82      c   kwd = lambda   hence a lambda-stmt   (flavour found later)
83       1010 istmty = 1
84            call fslmbd
85            return
86      c   kwd = dose   hence an exposure stmt   (flavour found later)
87       1020 istmty = 2
88            call fsexpo
89            return
90      c   kwd = to   cannot occur as a header kwd
91       1030 istmty = -1
92            call errpar( 1, 1030, 20hilegl stmt structure )
93      c   skip the present illegal stmt
94            call skpstm
95            return
96      c   kwd = proj   hence system-stmt   (kind 1, 2, or 3)
97       1040 istmty = 3
98            istknd = 1
99            call fssyst
100           return
101     c   kwd = contact   hence system-stmt   (kind 4, or 5)
102      1050 istmty = 3
103           istknd = 4
104           call fssyst
105           return
106     c   kwd = line   hence object-stmt   (kind = 1)
107      1060 istmty = 4
108           istknd = 1
109           call fsobje
110           return
111     c   kwd = space   hence object-stmt (kind = 2)
112      1070 istmty = 4
```

```
113           istknd = 2
114           call fsobje
115           return
116       c   kwd = linespace   hence object-stmt   (kind = 3)
117        1080 istmty = 4
118           istknd = 3
119           call fsobje
120           return
121       c   kwd = etchrate   hence devmodel-stmt
122        1090 istmty = 5
123           call fsdvmo
124           return
125       c   kwd = analytic   hence error in the input from the scanner
126        1100 istmty = -1
127           call errpar(1, 1100, 20hilegl stmt structure )
128           call skpstm
129           return
130       c   kwd = curve   hence error (as above)
131        1110 istmty = -1
132           call errpar( 1, 1110, 20hilegl stmt structure )
133           call skpstm
134           return
135       c   kwd = devtime   hence devtime-stmt
136        1120 istmty = 6
137           call fsdvtm
138           return
139       c   kwd = resmodel   hence resmodel-stmt
140        1130 istmty = 7
141           call fsrsmo
142           return
143       c   kwd = run   hence run-stmt
144        1140 istmty = 8
145           call fsrun
146           return
147       c   kwd = layers   hence layers-stmt
148        1150 istmty = 9
149           call fslaye
150           return
151       c   kwd = trial   hence trial-stmt
152        1160 istmty = 10
153           call fstria
154           return
155       c
156       c   unexpected kwdval (not possible, ...yet)
157       c   the value of -1 for unknown kwds will not be encountered here
158       c   because at that time the value of the lexical token will be = -1
159       c   indicating an error in the lexical token.
160        1999 call errpar( 1, 1999, 20hunkwn kwdval-gstmt-- )
161           call skpstm
162           return
163       c
164       c   a number is not expected as a first lexical token of a stmt
165       c   hence -
166        2000 call errpar( 1, 2000, 20hkwd expt in gstmt    )
167           lxused = .false.
168       c   so that the number gets printed in gstmt
```

```
169             call skpstm
170             return
171             end
172             subroutine gflexm
173       c   gets a fresh lexeme (i.e. a lexeme not used by the previous stmt
174       c   esp. after a stmt which has a variable length).  this is used in
175       c   the normal course of actions by gstmt when no errors have been
176       c   detected.  and if errors have been detected then this is used to
177       c   get the next lexeme (as if it is glexem).
178       c
179             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
180             common /endfl2/ nostmt, lxused
181       c
182             logical         nostmt, lxused
183       c
184             if (lxused) goto 10
185       c   the current lexeme was not used in the construction of the previous
186       c   stmt.  hence do not get a new lexeme.
187             lxused = .true.
188             return
189       c
190       c
191          10 call glexem
192             lxused = .true.
193             if ((lxtkty .lt. 0) .or. (lxtkty .gt. 2))
194            1            call errpar(1, 10, 20hilegl lxtkty -glexe )
195             return
196             end
197             subroutine skpstm
198       c   this subroutine skips the present statement.  called when an error
199       c   is detected by the parser in the stmt being formed.  this skips to
200       c   a kwd which can head a stmt.
201       c
202             common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
203             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
204             common /endfl2/ nostmt, lxused
205             common /errfl2/ lparer, legstm
206       c
207             logical         nostmt, lxused
208             logical         lparer, legstm
209       c
210             write (6, 9000)
211        9000 format(1x, 50h--stmt to be skipped because of error (at level 2))
212       c
213             legstm = .false.
214       c   now skip the stmt by skipping over the lexical tokens which cannot
215       c   form the header of a legal stmt
216       c   -- should print out the previous lexical tokens to be skipped over
217       c   -  without forming the current stmt.  then print out the message
218       c   -  saying that the error was detected here.  then tell that the
219       c   -  rest of the lex-tokens to be skipped are =   (etc.)
220       c
221           3 if (lxtkty .eq. -1) goto 2
222             if (lxtkty .eq. 0) goto 1
223             if (lxtkty .eq. 1) goto 10
224             if (lxtkty .eq. 2) goto 20
```

```
225     c   this is not possible.   ...yet
226           call errpar(1, 2, 20hunknon lxtkty-skpstm )
227           lxused = .true.
228           call gflexm
229           goto 3
230     c
231     c   the invalid-lexical-token is encountered
232         2  write (6, 9005)
233      9005 format( 1x, 33h--the invalid-lexical-token found )
234           lxused = .true.
235           call gflexm
236           goto 3
237     c
238     c   the =end of input stmts= token
239         1  lxused = .false.
240           goto 10000
241     c
242     c   the current lexical token is a keyword
243        10  if ( (kwdval .eq. 1) .or. (kwdval .eq. 2) .or.
244          1    ((4 .le. kwdval) .and. (kwdval .le. 9)) .or.
245          2    ((12 .le. kwdval) .and. (kwdval .le. 16))    ) goto 15
246           write (6, 9010) kwdarr
247      9010 format(1x, 5x, 10a1)
248           lxused = .true.
249           call gflexm
250           goto 3
251     c
252     c   a proper header kwd for a stmt
253        15  lxused = .false.
254           goto 10000
255     c
256     c   a number has been encountered.  it cannot start a stmt.
257        20  write (6, 9020) rnmval
258      9020 format(1x, 10x, f15.5)
259     c   when this is the number at which error was detected then this
260     c   should not get thrown out twice.  hence
261           lxused = .true.
262           call gflexm
263           goto 3
264     c
265     10000 write (6, 10001)
266     10001 format( 1x, 2x, 29h=-=-=-= stmt skipping stopped )
267           return
268     c
269           end
270           subroutine errpar(iarg1, iernum, messap)
271     c   called if an error in the statement structure is encountered.
272     c   (i.e. an error at level 2)
273     c
274           common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
275           common /errfl2/ lparer, legstm
276     c
277           logical         lparer, legstm
278     c   set the input stmt type to the invalid-stmt-type
279           istmty = -1
280     c
```

```
281          legstm = .false.
282     c
283     c   write an error message
284          write (6, 9000) iarg1, iernum, messap
285      9000 format(//,1x,2h--,40herror in the current statement structure,
286         1              1x, 24h(detected by the parser), /,
287         2        1x,5x, 8hlevel = ,i6, 14h, error line*=, i6, 2h, ,a20,/)
288     c
289          return
290          end
```

```
 1          subroutine fslmbd
 2      c   forms a lambda-stmt
 3      c
 4          common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
 5          common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
 6          common /endfl2/ nostmt, lxused
 7          common /errfl2/ lparer, legstm
 8      c
 9          logical        nostmt, lxused
10          logical        lparer, legstm
11      c
12      c   istmty = 1   (because of the context of the call)   ...yet
13      c
14          if (istmty .eq. 1) goto 1
15          call errpar( 0, 0, 20herror in lambda stmt )
16          lxused = .false.
17          call skpstm
18          return
19      c
20      c   a legal lambda-stmt header
21        1 call glexem
22      c   a number is expected
23          if (lxtkty .eq. 2) goto 3
24          call errpar( 1, 2, 20hnumr expt in fslmbd  )
25          lxused = .false.
26          call skpstm
27          return
28      c   nminst and nmpntr were set to 0 in subr gstmt
29        3 nmpntr = nmpntr + 1
30          stnmls( nmpntr ) = rnmval
31      c
32          call glexem
33          if (lxtkty .eq. 2) goto 5
34      c   so this is a lambda-stmt of the first kind
35          istknd = 1
36          lxused = .false.
37      c   nminst = nmpntr = 1
38          nminst = nmpntr
39          return
40      c
41      c   now we have a lambda-stmt of the second kind
42        5 istknd = 2
43          nmpntr = nmpntr + 1
44          stnmls( nmpntr ) = rnmval
45      c
46      c   now look for more number pairs (lambdas and weights)
47       10 call glexem
48          if (lxtkty .ne. 2) goto 30
49      c   a number hence a number pair should follow
50          nmpntr = nmpntr + 1
51      c   only a max of 10 number pairs is allowed
52          if (nmpntr .le. 19) goto 20
53          call errpar( 2, 15, 20htoo many nums-fslmbd )
54          call skpstm
55          return
56      c
```

```
57          20  stnmls( nmpntr ) = rnmval
58        c
59        c   now one more number is expected to form a number pair
60                call glexem
61                if (lxtkty .eq. 2) goto 27
62        c   not a number, hence error
63                call errpar( 1, 27, 20hnum expt by fslmbd    )
64                lxused = .false.
65                call skpstm
66                return
67        c
68        c   put this second number in the stmt-num-list
69          27  nmpntr = nmpntr + 1
70                stnmls( nmpntr ) = rnmval
71        c
72                goto 10
73        c
74        c   not a number.  hence
75          30  lxused = .false.
76                nminst = nmpntr
77                return
78                end
79                subroutine fsexpo
80        c   forms the exposure-stmt
81        c
82                common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
83                common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
84                common /endfl2/ nostmt, lxused
85                common /errfl2/ lparer, legstm
86        c
87                logical         nostmt, lxused
88                logical         lparer, legstm
89        c
90        c   istmty = 2   (because of the context of the call)   ...yet
91                if (istmty .eq. 2) goto 1
92                call errpar( 0, 0, 20herror in exposu-stmt )
93                lxused = .false.
94                call skpstm
95                return
96        c
97        c   a legal exposure-stmt header
98            1   call glexem
99                if (lxtkty .eq. 2) goto 3
100               call errpar( 1, 1, 20hnum expt in fsexpo    )
101               lxused = .false.
102               call skpstm
103               return
104       c
105           3   nmpntr = nmpntr + 1
106               stnmls(nmpntr) = rnmval
107       c
108               call glexem
109               if (lxtkty .eq. 1) goto 5
110       c   if end of data reached (eof lex-token) then return
111               if (lxtkty .ne. 0) goto 4
112       c   (end-the-run lex-token  (hence stmt is complete, too))
```

```
113            lxused = .false.
114            nminst = nmpntr
115            return
116      c
117      c   so a number is present here (an error).  hence
118          4  call errpar( 1, 2, 20hkwd *to* exptd--fsex )
119            lxused = .false.
120            call skpstm
121            return
122      c
123      c   either the kwd =to= is present or a new stmt starts here
124          5  if (kwdval .eq. 3) goto 7
125            istknd = 1
126            lxused = .false.
127            nminst = nmpntr
128            return
129      c
130          7  istknd = 2
131      c
132            do 10 itemp1 = 1, 2
133            call glexem
134            if (lxtkty .eq. 2) goto 9
135            call errpar( 1, 7, 20hnum expt in glexem    )
136            lxused = .false.
137            call skpstm
138            return
139      c
140          9  nmpntr = nmpntr + 1
141            stnmls( nmpntr ) = rnmval
142      c
143         10  continue
144      c
145      c   so the exposure-stmt has been properly formed
146            nminst = nmpntr
147            return
148            end
149            subroutine fssyst
150      c   forms the ststem-stmt   (header = proj, or contact)
151      c
152            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
153            common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
154            common /endfl2/ nostmt, lxused
155            common /errfl2/ lparer, legstm
156      c
157            logical         nostmt, lxused
158            logical         lparer, legstm
159      c
160      c   istmty = 3   (from the context of the call)   ...yet
161            if (istmty .eq. 3) goto 1
162            call errpar( 0, 0, 20herror in ststem-stmt )
163            lxused = .false.
164            call skpstm
165            return
166      c
167      c   a legitimate ststem-stmt header (proj or contact) ..
168      c   istknd = 1, or 4
```

```
169          1  if (istknd .eq. 4) goto 100
170       c   the header kwd is proj   (and istknd = 1 (, or 2, or 3))
171             call glexem
172             if (lxtkty .eq. 2) goto 3
173             call errpar( 1, 2, 20hnum expt in fssyst    )
174             lxused = .false.
175             call skpstm
176             return
177       c   .. followed by the first number
178          3  nmpntr = nmpntr + 1
179             stnmls(nmpntr) = rnmval
180       c
181             call glexem
182             if (lxtkty .eq. 2) goto 5
183             lxused = .false.
184             nminst = nmpntr
185             return
186       c   .. followed by second number
187          5  istknd = 2
188             nmpntr = nmpntr + 1
189             stnmls(nmpntr) = rnmval
190       c
191             call glexem
192             if (lxtkty .eq. 2) goto 7
193             lxused = .false.
194             nminst = nmpntr
195             return
196       c
197       c   .. followed by third and fourth numbers
198          7  istknd = 3
199             nmpntr = nmpntr + 1
200             stnmls(nmpntr) = rnmval
201       c
202             itemp1 = stnmls( 3)
203             if ((0 .le. itemp1) .and. (itemp1 .le. 20)) goto 8
204             call errpar( 1, 8, 20hnum of nums out of r )
205             lxused = .false.
206             call skpstm
207             return
208       c   totally  1 number for freq-limit, and itemp1 number of numbers as
209       c   mtf-weights  are expected.  hence
210          8  itemp1 = itemp1 + 1
211       c
212             do 10 itemp2 = 1, itemp1
213             call glexem
214             if (lxtkty .eq. 2) goto 9
215             call errpar( 1, 8, 20hnum expt in fssyst    )
216             lxused = .false.
217             call skpstm
218             return
219       c
220          9  nmpntr = nmpntr + 1
221             stnmls( nmpntr ) = rnmval
222         10  continue
223       c
224       c   so all the numbers have been obtained
```

```
225             nminst = nmpntr
226             return
227       c
228       c   the header is =contact=   (istknd = 4 (, or 5))
229        100  call glexem
230             if (lxtkty .eq. 2) goto 103
231             call errpar( 1, 101, 20hnum expt in fssyst    )
232             lxused = .false.
233             call skpstm
234             return
235       c
236        103  nmpntr = nmpntr + 1
237             stnmls(nmpntr) = rnmval
238       c
239             call glexem
240             if (lxtkty .eq. 2) goto 105
241       c
242             lxused = .false.
243             nminst = nmpntr
244             return
245       c
246       c   two more numbers expected   (istknd = 5)
247        105  istknd = 5
248             nmpntr = nmpntr + 1
249             stnmls( nmpntr ) = rnmval
250       c
251             call glexem
252             if (lxtkty .eq. 2) goto 107
253             call errpar( 1, 106, 20hnum expt in fssyst    )
254             lxused = .false.
255             call skpstm
256             return
257       c
258       c   so this is the third number
259        107  nmpntr = nmpntr + 1
260             stnmls( nmpntr ) = rnmval
261       c
262             nminst = nmpntr
263             return
264             end
```

```
1            subroutine fsobje
2       c   forms the object-stmt
3       c
4            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
5            common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
6            common /endfl2/ nostmt, lxused
7            common /errfl2/ lparer, legstm
8       c
9            logical         nostmt, lxused
10           logical         lparer, legstm
11      c
12      c   this time assume that the context of call is correct, and
13      c   istmty = 3   , and istknd = 1, or 2, or 3
14      c
15           call glexem
16           if (lxtkty .eq. 2) goto 1
17           call errpar( 1, 11, 20hnum expt in fsobje   )
18           lxused = .false.
19           call skpstm
20           return
21      c
22         1 nmpntr = nmpntr + 1
23           stnmls( nmpntr ) = rnmval
24      c
25      c   now depending on whether  (line, space), or (linespace) is the
26      c   header,  no more or one more number expected
27      c
28           goto (10, 10, 20), istknd
29        10 nminst = nmpntr
30           return
31      c
32        20 call glexem
33           if (lxtkty .eq. 2) goto 25
34           call errpar( 1, 21, 20hnum expt in fsobje   )
35           lxused = .false.
36           call skpstm
37           return
38      c
39        25 nmpntr = nmpntr + 1
40           stnmls( nmpntr ) = rnmval
41      c
42           nminst = nmpntr
43           return
44           end
45           subroutine fsdvmo
46      c   forms the devmodel-stmt
47      c
48           common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
49           common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
50           common /endfl2/ nostmt, lxused
51           common /errfl2/ lparer, legstm
52      c
53           logical         nostmt, lxused
54           logical         lparer, legstm
55      c
56      c   (istmty = 4) assumed to be correct because of the context of the
```

```
 57      c   call.  hence no need to check for an error
 58      c
 59          call glexem
 60          if (lxtkty .eq. 1) goto 1
 61          call errpar( 1, 0, 20hkwd expt in fsdvmo    )
 62          lxused = .false.
 63          call skpstm
 64          return
 65      c
 66        1 if (kwdval .eq. 10) goto 10
 67          if (kwdval .eq. 11) goto 20
 68      c
 69          call errpar( 1, 1, 20himpro kwd in fsdvmo  )
 70          lxused = .false.
 71          call skpstm
 72          return
 73      c
 74      c   etchrate analytic ... stmt
 75       10 istknd = 1
 76      c
 77          do 15 itemp1 = 1, 3
 78          call glexem
 79          if (lxtkty .eq. 2) goto 12
 80          call errpar( 1, 11, 20hnum expt in fsdvmo    )
 81          lxused = .false.
 82          call skpstm
 83          return
 84      c
 85       12 nmpntr = nmpntr + 1
 86          stnmls(nmpntr) = rnmval
 87       15 continue
 88      c
 89          nminst = nmpntr
 90          return
 91      c
 92      c   etchrate curve ... stmt
 93       20 istknd = 2
 94      c
 95      c   now the first number tells how many numbers follow.  so take its
 96      c   integer value (= /lexsem/.intval (it should be an integer ... ))
 97      c   and try to read that many numbers after that.  this is slightly
 98      c   different from a general parser design but let us allow it as an
 99      c   exception.
100          call glexem
101          if (lxtkty .eq. 2) goto 25
102          call errpar( 1, 21, 20hint-num expt*fsdvmo  )
103          lxused = .false.
104          call skpstm
105          return
106      c
107       25 nmpntr = nmpntr + 1
108          stnmls(nmpntr) = rnmval
109      c
110          itemp2 = rnmval
111      c
112          if ((0 .le. itemp2) .and. (itemp2 .le. 21)) goto 27
```

```
113            call errpar( 1, 26, 20hnum out of range*dvm )
114            lxused = .false.
115            call skpstm
116            return
117      c
118        27  continue
119      c
120            do 29 itemp1 = 1, itemp2
121            call glexem
122            if (lxtkty .eq. 2) goto 28
123            call errpar( 1, 27, 20hnum expt in fsdvmo    )
124            lxused = .false.
125            call skpstm
126            return
127      c
128        28  nmpntr = nmpntr + 1
129            stnmls(nmpntr) = rnmval
130      c
131        29  continue
132      c
133            nminst = nmpntr
134            return
135            end
136            subroutine fsdvtm
137      c   forms the devtime-stmt
138      c
139            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
140            common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
141            common /endfl2/ nostmt, lxused
142            common /errfl2/ lparer, legstm
143      c
144            logical         nostmt, lxused
145            logical         lparer, legstm
146      c
147      c   (istmty = 6).  but this has the same structure as the exposure-stmt
148      c   hence let us use that subroutine (fsexpo) pretending this to be a
149      c   stmt of that type...
150            istmty = 2
151      c   ... and call fsexpo ...
152      c   ... (a bad technique -- if an error is detected by fsexpo) -----
153            call fsexpo
154      c   ... and then say that it is a devtime-stmt
155            istmty = 6
156      c
157            return
158            end
159            subroutine fsrsmo
160      c   forms the resmodel-stmt
161      c
162            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
163            common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
164            common /endfl2/ nostmt, lxused
165            common /errfl2/ lparer, legstm
166      c
167            logical         nostmt, lxused
168            logical         lparer, legstm
```

```
169      c
170      c   istmty = 7   because of the context of the call
171      c   expects seven numbers to follow the kwd (resmodel)
172      c
173            do 10 itemp1 = 1, 7
174            call glexem
175            if (lxtkty .eq. 2) goto 5
176            call errpar( 1, 4, 20hnum expt in fsrsmo    )
177            lxused = .false.
178            call skpstm
179            return
180      c
181         5  nmpntr = nmpntr + 1
182            stnmls( nmpntr ) = rnmval
183      c
184        10  continue
185      c
186            nminst = nmpntr
187            return
188            end
189            subroutine fsrun
190      c   forms the run-stmt
191      c
192            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
193            common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
194            common /endfl2/ nostmt, lxused
195            common /errfl2/ lparer, legstm
196      c
197            logical         nostmt, lxused
198            logical         lparer, legstm
199      c   the common /errfl2/ and its logical declaration are not needed
200      c
201      c   istmty = 8   ((from the context of the call) assumed correct)
202      c
203      c   for the user two different kinds of this stmt are present.
204      c   but for the parser both get converted to the same internal form.
205      c
206            call glexem
207            if (lxtkty .eq. 2) goto 100
208      c   no number following the kwd =run= hence
209            lxused = .false.
210            stnmls(1) = 0.0
211            nminst = 1
212            return
213      c
214      c   a number follows hence
215       100  nmpntr = nmpntr + 1
216            stnmls(nmpntr) = rnmval
217      c
218            nminst = nmpntr
219            return
220      c
221            end
```

```
1            subroutine fslaye
2       c   forms the layers-stmt
3       c
4             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
5             common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
6             common /endfl2/ nostmt, lxused
7             common /errfl2/ lparer, legstm
8       c
9             logical         nostmt, lxused
10            logical         lparer, legstm
11      c
12      c
13      c   (istmty = 9) assumed correct.  hence no error-checking on that.
14      c
15            call glexem
16      c
17            do 10 itemp1 = 1, 2
18            if (lxtkty .eq. 2) goto 8
19      c
20      c   a number was expected but not obtained hence
21            call errpar( 1, 7, 20hnum expt by fslaye   )
22            nminst = nmpntr
23            lxused = .false.
24            call skpstm
25            return
26      c
27         8  nmpntr = nmpntr + 1
28            stnmls( nmpntr ) = rnmval
29      c
30            call glexem
31        10  continue
32      c
33            if (lxtkty .eq. 2) goto 15
34            lxused = .false.
35            nminst = nmpntr
36            return
37      c
38      c   now a series of triples of numbers is expected
39      c
40        15  do 20 itemp1 = 1, 3
41            if (lxtkty .eq. 2) goto 18
42      c
43      c   a number was expected but not obtained hence
44            call errpar( 1, 17, 20hnum expt by fslaye   )
45            nminst = nmpntr
46            lxused = .false.
47            call skpstm
48            return
49      c
50        18  nmpntr = nmpntr + 1
51            stnmls( nmpntr ) = rnmval
52      c
53            call glexem
54        20  continue
55      c
56      c   if the present lexical token is a number then one more triple of
```

```
57      c   numbers (including that number) is expected, so go back to get the
58      c   triple
59              if (lxtkty .eq. 2) goto 15
60      c
61              lxused = .false.
62              nminst = nmpntr
63              return
64              end
65              subroutine fstria
66      c   forms the trial-stmt
67      c
68              common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
69              common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
70              common /endfl2/ nostmt, lxused
71              common /errfl2/ lparer, legstm
72      c
73              logical         nostmt, lxused
74              logical         lparer, legstm
75      c
76      c   (istmty = 10) assumed correct.  hence no error-checking done here.
77      c
78              call glexem
79              if (lxtkty .eq. 2) goto 10
80              call errpar( 1, 0, 20hnum expt by fstria    )
81              nminst = nmpntr
82              lxused = .false.
83              call skpstm
84              return
85      c
86           10 nmpntr = nmpntr + 1
87              stnmls( nmpntr ) = rnmval
88              call glexem
89              if (lxtkty .eq. 2) goto 10
90      c
91              lxused = .false.
92              nminst = nmpntr
93              return
94              end
```

%

```
1            subroutine tstpar
2       c  driver subroutine for testing the parser of the
3       c  photolithographic-processes-simulation-program.
4       c  -        (snn,  erl, eecs, ucb)       february 26, 1978
5       c
6            common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
7            common /charac/ iprptr, ipchar, ictype, ipcard(82)
8            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
9            common /errfla/ ierflg, iersvi, ismesi
10           common /endofx/ noipdk, noipr, noilr, noicst, nologi
11      c
12           common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
13           common /endfl2/ nostmt, lxused
14           common /errfl2/ lparer, legstm
15      c
16           logical         noipdk, noipr, noilr, noicst, nologi
17      c
18           logical         nostmt, lxused
19           logical         lparer, legstm
20      c
21      c  initialize the keywords array ( with lambda, dose, to, proj etc. )
22           data jrswdt
23          1      /1hl, 1ha, 1hm, 1hb, 1hd, 1ha,  4*1h ,
24          2       1hd, 1ho, 1hs, 1he,           6*1h ,
25          3       1ht, 1ho,                     8*1h ,
26          4       1hp, 1hr, 1ho, 1hj,           6*1h ,
27          5       1hc, 1ho, 1hn, 1ht, 1ha, 1hc, 1ht,  3*1h ,
28          6       1hl, 1hi, 1hn, 1he,           6*1h ,
29          7       1hs, 1hp, 1ha, 1hc, 1he,      5*1h ,
30          8       1hl, 1hi, 1hn, 1he, 1hs, 1hp, 1ha, 1hc, 1he,  1h ,
31          9       1he, 1ht, 1hc, 1hh, 1hr, 1ha, 1ht, 1he,  2*1h ,
32          a       1ha, 1hn, 1ha, 1hl, 1hy, 1ht, 1hi, 1hc,  2*1h ,
33          b       1hc, 1hu, 1hr, 1hv, 1he,      5*1h ,
34          c       1hd, 1he, 1hv, 1ht, 1hi, 1hm, 1he,  3*1h ,
35          d       1hr, 1he, 1hs, 1hm, 1ho, 1hd, 1he, 1hl,  2*1h ,
36          e       1hr, 1hu, 1hn,                7*1h ,
37          f       1hl, 1ha, 1hy, 1he, 1hr, 1hs,  4*1h ,
38          g       1ht, 1hr, 1hi, 1ha, 1hl,      5h*1h  /, nmrswd / 16 /
39      c
40      c  (the calling program should print the fancy title to the output)
41      c  this routine should print out the fact that it is only testing the
42      c  parser
43           write (6, 9000)
44       9000 format( 1x, 5h*-*-*, /, 1x, 5h-*-*-, 2x,
45          1              24hdriver subroutine tstpar, /, 1x, 5h*-*-*, 2x,
46          2              18hto test the parser, /, 1x, 5h-*-*- )
47      c
48      c  initialize the program processes, variables, etc., and also the
49      c  defaultable values of the photolithographic system to be simulated
50      c
51           call initpa
52      c
53      c  now process the input cards in an interpreter like fashion.
54      c
55      c  while the end of input statement stream has not been reached,
56
```

```
57      c   get an input statement ...
58        10  call gstmt
59      c
60      c   prettyprint it ...
61            call prprst
62      c
63      c   ...and execute it.
64      c     call exstmt
65      c
66      c
67            if (.not. nostmt) goto 10
68      c
69      c   end of the input statement stream was reached, hence pack up.
70      c       call endjob
71      c
72            return
73            end
```

Jun 21 02:55 1978 File parser/decl Page 1

```
1      c
2             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
3             common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
4             common /endfl2/ nostmt, lxused
5             common /errfl2/ lparer, legstm
6      c
7             logical         nostmt, lxused
8             logical         lparer, legstm
9      c
```

Jun 21 02:58 1978 File parser/pardrive Page 1

```
1             program main(input, output, tape5=input, tape6=output)
2             call outitl
3             call tstpar
4             stop
5             end
```

```
1       c   (snn,  erl, eecs, ucb)          march 6, 1978
2       c
3             common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
4             common /charac/ iprptr, ipchar, ictype, ipcard(82)
5             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
6             common /errfla/ ierflg, iersvi, ismesi
7             common /endofx/ noipdk, noipr, noilr, noicst, nologi
8       c
9             logical         noipdk, noipr, noilr, noicst, nologi
10      c
11      c   initialize the keywords array ( with lambda, dose, to, proj etc. )
12            data jrswdt
13           1      /1hl, 1ha, 1hm, 1hb, 1hd, 1ha,  4*1h ,
14           2        1hd, 1ho, 1hs, 1he,            6*1h ,
15           3        1ht, 1ho,                      8*1h ,
16           4        1hp, 1hr, 1ho, 1hj,            6*1h ,
17           5        1hc, 1ho, 1hn, 1ht, 1ha, 1hc, 1ht,  3*1h ,
18           6        1hl, 1hi, 1hn, 1he,            6*1h ,
19           7        1hs, 1hp, 1ha, 1hc, 1he,       5*1h ,
20           8        1hl, 1hi, 1hn, 1he, 1hs, 1hp, 1ha, 1hc, 1he,  1h ,
21           9        1he, 1ht, 1hc, 1hh, 1hr, 1ha, 1ht, 1he,  2*1h ,
22           a        1ha, 1hn, 1ha, 1hl, 1hy, 1ht, 1hi, 1hc,  2*1h ,
23           b        1hc, 1hu, 1hr, 1hv, 1he,       5*1h ,
24           c        1hd, 1he, 1hv, 1ht, 1hi, 1hm, 1he,  3*1h ,
25           d        1hr, 1he, 1hs, 1hm, 1ho, 1hd, 1he, 1hl,  2*1h ,
26           e        1hr, 1hu, 1hn,                 7*1h ,
27           f        1hl, 1ha, 1hy, 1he, 1hr, 1hs,  4*1h ,
28           g        1ht, 1hr, 1hi, 1ha, 1hl,       5*1h   /, nmrswd / 16 /
29      c
```

```
1             subroutine errlex( iernum, iargsi, messag )
2        c   depending on error severity and accumulated error-severity-index
3        c    this routine aborts, or continues.
4        c   iargsi = int argument (for) severity index
5        c
6             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
7             common /errfla/ ierflg, iersvi, ismesi
8        c
9        c   set up the lexical token type to invalid-lexical-token type
10            lxtkty = -1
11       c
12            ismesi = ismesi + iargsi
13            write (6, 100) messag, iargsi, iersvi, ismesi, iernum
14        100 format( //, 1x, 19h-error in the input, /, 2x,
15           1                38h(detected by the lexical analyzer) -- ,a20,/,
16           2           2x, 8hiargsi =, i5, 10h, iersvi =, i5,
17           3                10h, ismesi =, i5, 10h, iernum =, i5      )
18            if (iernum .ge. 100) stop
19            if (ismesi .ge. 10 ) stop
20            return
21            end
22            subroutine fkwdvl
23       c   fetches the keyword value ( and puts it in /lexsca/.kwdval )
24       c
25            common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
26            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
27       c
28            do 200 itemp1 = 1, nmrswd
29       c
30            do 100 itemp2 = 1, 10
31            if ( kwdarr(itemp2) .ne. jrswdt( itemp2, itemp1 ) ) goto 200
32        100 continue
33       c   normal exit => keyword found in table
34       c
35            itemp3 = itemp1
36            goto 300
37        200 continue
38       c   normal exit => word not found in the table
39       c
40            kwdval = -1
41            write (6, 9000) kwdarr
42       9000 format( /, 1x, 33h-the input has an unknown word = , 10a1 )
43            call errlex( -1, 1, 20hunknown kwd found*sf)
44            return
45       c
46        300 kwdval = itemp3
47            return
48            end
49            subroutine frmfra
50       c   forms the fractional part of a real number
51       c   no error-checking is expected ( or required ).
52       c
53            common /charac/ iprptr, ipchar, ictype, ipcard(82)
54            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
55       c
56            fraval = 0.0
```

```
57            ntenpr = 0
58         10 if (ictype .ne. 2) goto 100
59            ntenpr = ntenpr + 1
60            idigvl = idgval( ipchar )
61            tdigvl = idigvl
62      c
63            do 20 itemp1 = 1, ntenpr
64            tdigvl = tdigvl* 0.1
65         20 continue
66      c
67            fraval = fraval + tdigvl
68            call gchar
69            goto 10
70        100 return
71            end
72            subroutine frmint
73      c   forms the integer part of a real number
74      c
75            common /charac/ iprptr, ipchar, ictype, ipcard(82)
76            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
77      c
78            intval = 0
79            if (ictype .eq. 2) goto 10
80            call errlex( -1, +1, 20hdigit-exptd - frmint )
81            return
82         10 idigvl = idgval( ipchar )
83            intval = 10 * intval + idigvl
84            call gchar
85            if (ictype .eq. 2) goto 10
86            return
87            end
88            subroutine frmkwd
89      c   forms keywords from the input character stream
90      c
91            common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
92            common /charac/ iprptr, ipchar, ictype, ipcard(82)
93            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
94      c
95      c   keywords are not allowed to be more than 10 characters long.
96      c   no error-checking expected ( or required )
97      c
98            do 50 itemp1 = 1, 10
99            kwdarr( itemp1 ) = 1h
100        50 continue
101     c
102     c
103           do 100 itemp1 = 1, 10
104           kwdarr( itemp1 ) = ipchar
105           call gchar
106           if ( ictype .ne. 1 ) goto 200
107       100 continue
108     c
109     c   normal exit means that the letter-sequence is more than 10
110     c    characters long.  hence error
111     c
112           call errlex( -1, 1, 20hkwd-longr,10 letters )
```

```
113            return
114       200  call fkwdvl
115            return
116            end
117            list
118            subroutine gcard
119       c   gets a card from the input deck
120       c
121            common /charac/ iprptr, ipchar, ictype, ipcard(82)
122            common /endofx/ noipdk, noipr, noilr, noicst, nologi
123       c
124            logical         noipdk, noipr, noilr, noicst, nologi
125            logical           eof
126       c
127            if (noipdk) call errlex( -1, +1, 20hmore-cards-expected-  )
128       c
129            read (5, 10) (ipcard(itemp1), itemp1 = 1, 80)
130        10  format( 80a1 )
131       c
132            write (6, 20) (ipcard(itemp1), itemp1 = 1, 80)
133        20  format( /, 1x, 12hinput card =, 80a1, 1h$ )
134       c
135            ipcard(81) = 1h
136            ipcard(82) = 1h
137            iprptr = 1
138            if ( eof(5) ) goto 100
139            return
140       c
141       100  noipdk = .true.
142            noicst = .true.
143            return
144            end
145            nolist
146            subroutine gchar
147       c   get a character from the input deck
148       c
149            common /charac/ iprptr, ipchar, ictype, ipcard(82)
150            common /endofx/ noipdk, noipr, noilr, noicst, nologi
151       c
152            logical         noipdk, noipr, noilr, noicst, nologi
153       c
154            if ( nologi ) goto 300
155            if ( noicst ) goto 200
156            if ( iprptr .lt. 82 ) goto 100
157       c
158            call gcard
159       100  ipchar = ipcard( iprptr )
160            iprptr = iprptr + 1
161            ictype = ipchty( ipchar )
162            return
163       c
164       200  ipchar = 1h$
165            ictype = ipchty( ipchar )
166            nologi = .true.
167            return
168       300  call errlex( -1, +1, 20htried to get xtra ip )
```

```
169            return
170            end
171            subroutine glexem
172      c   gets the next lexeme and puts its type in lxtkty (=lexical token
173      c    type), and its =value= in kwdval, or rnmval.
174      c   the pair ( lxtkty, lxmval(= kwdval, or rnmval) ) is a lexeme.
175      c
176            common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
177            common /charac/ iprptr, ipchar, ictype, ipcard(82)
178            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
179      c   get a non-separator character from the input char-stream
180            call gnsep
181      c
182            icsign = +1
183            tmpsgn = icsign
184      c   a giant case-stmt
185          3 if ( ictype .eq. 3 ) goto 30
186            if ( ictype .eq. 0 ) goto 1
187            if ( ictype .eq. 1 ) goto 10
188          2 if ( ictype .eq. 2 ) goto 20
189            if ( ictype .eq. 4 ) goto 40
190      c   ictype = 5 is not possible here ( non-separator ),  ...yet
191            if ( ictype .eq. 5 ) goto 50
192            if ( ictype .eq. 6 ) goto 60
193            if ( ictype .eq. 7 ) goto 70
194            call errlex( -1, +1, 20hunxpd-ictype-in glxm )     <----[call gchar
195      c                                                              [return
196      c   right pad lexical token
197          1 lxtkty = 0
198            return
199      c
200      c   a keyword is to be formed as a lexical token
201         10 lxtkty = 1
202            call frmkwd
203            return
204      c
205      c   a real number is to be formed as a lexical token
206         20 lxtkty = 2
207            call frmint
208            rnmval = intval
209            if (ictype .eq. 4) goto 25
210            rnmval = rnmval*tmpsgn
211            return
212         25 call gchar
213         26 call frmfra
214            rnmval = rnmval + fraval
215            rnmval = rnmval * tmpsgn
216            return
217      c
218         30 lxtkty = 2
219            if (ipchar .eq. 1h+) icsign = +1
220            if (ipchar .eq. 1h-) icsign = -1
221            tmpsgn = icsign
222            call gchar
223            call gnsep
224            if ( (ictype .eq. 2) .or. (ictype .eq. 4) ) goto 35
```

```
225           call errlex( -1, +1, 20hnumbr exptd aftr sgn )
226           return
227     c
228        35 goto 2
229        40 lxtkty = 2
230           rnmval = 0
231           call gchar
232           if ( ictype .eq. 2 ) goto 45
233           call errlex( -1, +1, 20hdigit exptd in glxm )
234           return
235     c
236     c   now it is similar to an ordinary number case, hence --
237        45 goto 26
238     c
239        50 call errlex( -1, +1, 20hseprtr prohib-glexem )
240           return
241     c
242     c   at present start-symbol (*), and continuation-symbol (/) for cards
243     c     have no meaning
244        60 call gnsep  <-          <-[ 60  call gchar
                                          call grsep
                                          goto 3
245           goto 3
246        70 call gnsep  <-          <-[ 70 { same as for '60' }
247           goto 3
248           end
249           subroutine gnsep
250     c   get a non-separator character from the input char stream
251     c
252           common /charac/ iprptr, ipchar, ictype, ipcard(82)
253     c
254        10 if ( ictype .ne. 5 ) goto 20
255           call gchar
256           goto 10
257        20 return
258           end
259           function idgval( iargch )
260     c   the argument is not used directly.
261     c   gives the value of a digit-character
262     c   assumes that ipchar (= iargch) is a digit-character
263     c    (because of the calling context )
264     c
265           common /charac/ iprptr, ipchar, ictype, ipcard(82)
266     c
267     c   a giant case-stmt
268     c
269           itemp1 = -10000
270           if (ipchar .eq. 1h0) itemp1 = 0
271           if (ipchar .eq. 1h1) itemp1 = 1
272           if (ipchar .eq. 1h2) itemp1 = 2
273           if (ipchar .eq. 1h3) itemp1 = 3
274           if (ipchar .eq. 1h4) itemp1 = 4
275           if (ipchar .eq. 1h5) itemp1 = 5
276           if (ipchar .eq. 1h6) itemp1 = 6
277           if (ipchar .eq. 1h7) itemp1 = 7
278           if (ipchar .eq. 1h8) itemp1 = 8
279           if (ipchar .eq. 1h9) itemp1 = 9
280           idgval = itemp1
```

```
281             return
282             end
283             subroutine initla
284       c   inits the lexical analysis system, as well as the system to be
285       c   simulated.
286       c
287             common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
288             common /charac/ iprptr, ipchar, ictype, ipcard(82)
289             common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
290             common /errfla/ ierflg, iersvi, ismesi
291             common /endofx/ noipdk, noipr, noilr, noicst, nologi
292       c
293             logical         noipdk, noipr, noilr, noicst, nologi
294       c
295       c
296       c   first initialize the program variables etc.
297       c
298       c   ( initializations of variables )
299             ierflg = -1
300             iersvi = -1
301             ismesi = 0
302       c
303             noipdk = .false.
304             noipr  = .false.
305             noilr  = .false.
306             noicst = .false.
307             nologi = .false.
308       c
309       c   ( (initialization of processes)  (( process variables )) )
310             call gcard
311             call gchar
312             call gnsep
313       c
314       c   initialize the photolithographic system to be analysed (i.e.
315       c    the defaultable variables ( and their flags etc.) ).
316       c
317             return
318             end
319             function ipchty( jargch )
320       c
321       c   tells the type of the input character
322       c   0 => $ (last char ),  1 => letter,  2 => digit-char,
323       c   3 => sign-char,       4 => period,  5 => separator
324       c   6 => * (start symb),  7 => / ( continuation symb )
325       c   -1 < ipchty < 8
326       c
327       c   the dummy argument is not used in this function
328             common /charac/iprptr, ipchar, ictype, ipcard(82)
329       c
330       c   a giant case-stmt
331             if ( ipchar .eq. 1h$ ) goto 1
332             if ( ( 1ha .le. ipchar) .and. (ipchar .le. 1hz) ) goto 10
333             if ( (1h0 .le. ipchar) .and. (ipchar .le. 1h9) ) goto 20
334             if ( (ipchar .eq. 1h+) .or. (ipchar .eq. 1h-) )  goto 30
335             if ( ipchar .eq. 1h. ) goto 40
336             if ( (ipchar .eq. 1h ) .or. (ipchar .eq. 1h,) .or.
```

```
337            1     (ipchar .eq. 1h() .or. (ipchar .eq. 1h)) ) goto 50
338             if ( ipchar .eq. 1h* ) goto 60
339             if ( ipchar .eq. 1h/ ) goto 70
340      c
341             ipchty = -1
342             return
343      c
344          1  ipchty = 0
345             return
346         10  ipchty = 1
347             return
348         20  ipchty = 2
349             return
350         30  ipchty = 3
351             return
352         40  ipchty = 4
353             return
354         50  ipchty = 5
355             return
356         60  ipchty = 6
357             return
358         70  ipchty = 7
359             return
360             end
```

```
1            nolist
2            subroutine tstlex
3       c   driver subroutine for testing the lexical-analyzer (= scanner)
4       c   of the photolithographic-processes-simulation-program.
5       c   (snn,  erl, eecs, ucb)           february 26, 1978
6       c
7            common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
8            common /charac/ iprptr, ipchar, ictype, ipcard(82)
9            common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
10           common /errfla/ ierflg, iersvi, ismesi
11           common /endofx/ noipdk, noipr, noilr, noicst, nologi
12      c
13           logical         noipdk, noipr, noilr, noicst, nologi
14      c
15      c   initialize the keywords array ( with lambda, dose, to, proj etc. )
16           data jrswdt
17           1      /1hl, 1ha, 1hm, 1hb, 1hd, 1ha,  4*1h ,
18           2        1hd, 1ho, 1hs, 1he,             6*1h ,
19           3        1ht, 1ho,                       8*1h ,
20           4        1hp, 1hr, 1ho, 1hj,             6*1h ,
21           5        1hc, 1ho, 1hn, 1ht, 1ha, 1hc, 1ht,  3*1h ,
22           6        1hl, 1hi, 1hn, 1he,             6*1h ,
23           7        1hs, 1hp, 1ha, 1hc, 1he,        5*1h ,
24           8        1hl, 1hi, 1hn, 1he, 1hs, 1hp, 1ha, 1hc, 1he,  1h ,
25           9        1he, 1ht, 1hc, 1hh, 1hr, 1ha, 1ht, 1he,  2*1h ,
26           a        1ha, 1hn, 1ha, 1hl, 1hy, 1ht, 1hi, 1hc,  2*1h ,
27           b        1hc, 1hu, 1hr, 1hv, 1he,        5*1h ,
28           c        1hd, 1he, 1hv, 1ht, 1hi, 1hm, 1he,  3*1h ,
29           d        1hr, 1he, 1hs, 1hm, 1ho, 1hd, 1he, 1hl,  2*1h ,
30           e        1hr, 1hu, 1hn,                  7*1h ,
31           f        1hl, 1ha, 1hy, 1he, 1hr, 1hs,  4*1h ,
32           g        1ht, 1hr, 1hi, 1ha, 1hl,        5*1h   /, nmrswd / 16 /
33      c
34      c   initialize the program processes, variables, etc., and also the
35      c   defaultable values of the photolithographic system to be simulated
36      c
37           call initla
38      c
39      c   now process the input cards in an interpreter like fashion
40      c   ---(for the present - just fake the processing, for test purposes )
41      c
42           call procrd
43      c
44           return
45           end
46           subroutine procrd
47      c   processes the input cards in an inetrpreter like fashion.
48      c   ---( at present just fake the processing for test purposes )
49      c
50           common /lexsca/ icsign, ricsgn, jrswdt(10,16), kwdarr(10), nmrswd
51           common /lexsem/ kwdval, rnmval, intval, fraval, rintvl, lxtkty
52      c
53      c   (only the meaning of lexemes is relevant at this level)
54      c
55      c   get the next lexeme.
56        10 call glexem
```

```
57      c
58            if (lxtkty .eq. 0) goto 100
59            if (lxtkty .eq. 1) goto 50
60            if (lxtkty .eq. 2) goto 60
61      c
62            write (6, 9000) lxtkty
63            call errlex( -1, +1, 20hwhat lexeme is this  )
64            return
65      c
66         50 write (6, 9010) lxtkty, kwdarr
67            goto 10
68         60 write (6, 9020) lxtkty, rnmval
69            goto 10
70      c
71        100 write (6, 9030)
72      c
73       9000 format(1x, 30hunknown lxtkty-out-of-range-=  , i5)
74       9010 format(1x, 8hlxtkty =, i3, 5x, 10hkeyword is , 5x, 10a1 )
75       9020 format(1x, 8hlxtkty =, i3, 5x, 10hnumber is=,  f15.5 )
76       9030 format(1x, 12hend-of-input  )
77      c
78            return
79            end
80            list
```



```
1             program main(input, output, tape5=input, tape6=output)
2             call tstlex
3             stop
4             end
```

```
1              subroutine prprst
2        c   this subroutine pretty-prints the currently parsed (formed) stmt
3        c   from its abstract syntax  (as obtained from /parsem/)
4        c
5              common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
6        c
7        c   since because of the extensive error-checking by the parser, and
8        c   stmt skipping on error, only a correct (stmt-structure wise) input
9        c   stmt will be returned by the parser.  hence no error checking is
10       c   required in this routine.
11       c
12       c   a giant case-stmt with the help of if-stmts, and a computed goto
13             if (istmty .eq. -1) goto 20
14             if (istmty .eq. 0) goto 10
15       c
16             goto (100, 200, 300, 400, 500, 600, 700, 800, 900, 1000), istmty
17       c
18       c   because of the simplicity of pretty-printing no separate procedures
19       c   are necessary for each stmt
20       c
21       c   the end-of-input-stmts stmt
22          10  write (6, 11)
23          11  format(//, 1x, 10(1h*), 5x, 18hend of lab session,
24            1               5x, 10(1h*), /, 1x, 48(1h-) )
25              return
26       c
27       c   an erroneous( = invalid)-stmt
28          20  write (6, 21)
29          21  format(//, 1x, 5h*****, 1x,
30            1            31hthat was an erroneous statement, 1x, 5h*****)
31              return
32       c
33       c   a lambda stmt (2 kinds)
34         100  write (6, 101)
35         101  format(//, 1x, 18ha lambda-statement )
36       c
37              goto (110, 120), istknd
38       c
39       c   a single wavelength in the lambda-stmt
40         110  write (6, 112) stnmls( 1 )
41         112  format(1x, 5x, 30hsingle wavelength illumination,
42            1            1x, 12hat lambda = , f10.5, 1x, 7hmicrons )
43              return
44       c
45       c   multiple wavelengths in the spectrum
46         120  write (6, 122) (stnmls(itemp1), itemp1 = 1, nminst)
47         122  format(1x, 5x, 32hmultiple wavelength illumination,
48            1            1x, 41hat the following wavelengths (in microns),
49            2            1x, 25hand relative intensities,
50            3        (/, 1x, 20x, 1h(, f10.5, 5x, f10.5, 1h) ) )
51              return
52       c
53       c   an exposure statement (2 kinds)
54         200  write (6, 201)
55         201  format(//, 1x, 21han exposure statement )
56       c
```

```
57           goto (210, 220), istknd
58     c
59     c   only one exposure
60      210  write (6, 212) stnmls(1)
61      212  format(1x, 5x, 36hsingle exposure at the intensity of , f10.5,
62        1             5x, 30hmillijoules per sq. centimeter )
63           return
64     c   a series of exposures
65      220  itemp1 = stnmls(3)
66           write (6, 222) stnmls(1), stnmls(2), itemp1
67      222  format(1x, 5x, 13hin the range , f10.5, 2x, 3hto , f10.5, 2x,
68        1             11hmj/(sq.cm.), 2x, 3hat , i5, 2x, 16huniformly spaced,
69        2             10h intervals )
70           return
71     c
72     c   a system-stmt (5 kinds)
73      300  write (6, 301)
74      301  format(//, 1x, 21hthe imaging system is )
75     c
76           goto (310, 320, 330, 340, 340), istknd
77     c
78     c   a projection type system   (istknd = 1)
79      310  write (6, 311) stnmls(1)
80      311  format(1x, 5x, 35ha projection type system with na = , f10.5 )
81           return
82     c
83     c   a projection type system   (istknd = 2)
84      320  write (6, 321) stnmls(1), stnmls(2)
85      321  format(1x, 5x, 35ha projection type system with na = , f10.5, 2x,
86        1             12hat lambda = , f10.5, 2x, 7hmicrons )
87           return
88     c
89     c   a projection type system   (istknd = 3)
90      330  itemp1 = stnmls(3)
91           itemp2 = itemp1 + 4
92           write (6, 331) stnmls(1), stnmls(2), itemp1, (stnmls(itemp3),
93        1                                             itemp3 = 4, itemp2)
94      331  format(1x, 5x, 35ha projection type system with na = , f10.5, 2x,
95        1             12hat lambda = , f10.5, 1x, 7hmicrons, /,
96        2      1x, 5x, 22hand with the following, i5, 2x, 11hweights for,
97        3                                              1x, 7hthe mtf, /,
98        4      1x,10x, 21hin the range of 0 to , f10.5, 11h(1/microns),/,
99        5      1x,10x, 5x, (10f10.5))
100          return
101    c
102    c   a contact type system   (istknd = 4 (and 5))
103     340  write (6, 341) stnmls(1)
104     341  format(1x, 5x, 40ha contact type system with mask to chip
105       1                  13hseparation = , f10.5, 2x, 7hmicrons )
106          if (istknd .eq. 4) return
107    c
108    c   .. a contact type system   (istknd = 5 (only))
109     350  write (6, 351) stnmls(2), stnmls(3)
110     351  format(1x, 5x, 29hand the coefficients ci being, 2x, 5hc1 = ,
111       1         f10.5, 2x,                 9hand c2 = , f10.5        )
112          return
```

```
113     c
114     c
115     c   an object( or mask)-stmt   (3 kinds)
116      400  write (6, 401)
117      401  format(//, 1x, 19han object statement )
118     c
119           goto (410, 420, 430), istknd
120     c
121     c   a line   (istknd = 1)
122      410  write (6, 411) stnmls(1)
123      411  format(1x, 5x, 26hthe mask has only a line   , f10.5,
124          1                        14h  microns wide )
125           return
126     c
127      420  write (6, 421) stnmls(1)
128      421  format(1x, 5x, 27hthe mask has only a space   , f10.5,
129          1                        14h  microns wide )
130           return
131     c
132      430  write (6, 431) stnmls(1), stnmls(2)
133      431  format(1x, 5x, 45hthe mask is a grating with a periodic pattern,
134          1                1x, 15hof line/space   , 2f10.5, 14h  microns wide )
135           return
136     c
137     c
138     c   a devmodel stmt   (2 kinds)
139      500  write (6, 501)
140      501  format(//, 1x, 20ha devmodel statement )
141     c
142           goto (510, 520), istknd
143     c
144     c   etchrate analytic ... stmt   (istknd = 1)
145      510  write (6, 511) stnmls(1), stnmls(2), stnmls(3)
146      511  format(1x,5x,48hthe etchrate is given by the analytic expression,
147          1         /, 11x, 17hthe rate r = exp(, f10.5, 2h +, f10.5, 4h*m +,
148          2                        f10.5, 5h*m*m) )
149           return
150     c
151     c   etchrate curve ... stmt   (istknd = 2)
152      520  write (6, 521) (stnmls(itemp1), itemp1 = 2, nminst )
153      521  format(1x,5x,47hthe etchrate is given by the curve of r vs m as,
154          1          /, (11x, f10.5) )
155           return
156     c
157     c
158     c   a devtime-stmt   (2 kinds)
159      600  write (6, 601)
160      601  format(//, 1x, 19ha devtime statement )
161     c
162           goto (610, 620), istknd
163     c
164     c   a single development time   (istknd = 1)
165      610  write (6, 611) stnmls(1)
166      611  format(1x, 5x, 32hthe chip is to be developed for , f10.5, 2x,
167          1             7hseconds )
168           return
```

```
169     c
170     c   a series of development times   (istknd = 2)
171      620  itemp1 = stnmls(3)
172           write (6, 621) stnmls(1), stnmls(2), itemp1
173      621  format(1x, 5x, 33hthe chip is to be developed from , f10.5, 2x,
174         1            11hseconds to , f10.5, 2x, 11hseconds in , i5, 2x,
175         2            5hsteps )
176           return
177     c
178     c
179     c   a resmodel statement   (only one kind)
180      700  write (6, 701)
181      701  format(//, 1x, 20ha resmodel statement )
182     c   (710)   only one kind of stmt
183           write (6, 711) (stnmls(itemp1), itemp1 = 1, 7)
184      711  format(1x, 5x, 12hat lambda = , f10.5, 1x, 7hmicrons, 2x,
185         1            25hthe resist parameters are, /, 1x, 10x,
186         2            4ha = , f10.5, 18h (1/microns),  b = , f10.5,
187         3            23h (1/microns),  and c = , f10.5,
188         4            20h (sq.cm)/millijoules,
189         5            /, 1x, 10x, 23ha refractive index of (,
190         6            f10.5, 1h,, f10.5, 1h), 5x, 3hand, /, 11x,
191         7            17hthe thickness is , f10.5, 1x, 7hmicrons )
192           return
193     c
194     c
195     c   a run stmt   (only one internal form)
196     c
197     c   for the user this stmt has two forms.  but the parser converts both
198     c   of them to the same internal form.
199     c   also this is the only stmt where the value of a parameter in the
200     c   stmt decides the control flow in the prettyprinter
201      800  itemp1 = stnmls(1)
202           if ((0 .le. itemp1) .and. (itemp1 .le. 4)) goto 805
203     c
204           write (6, 802)
205      802  format(//, 1x, 41ha run-statement with an invalid parameter )
206           return
207     c
208      805  if (itemp1 .ne. 0) goto 809
209     c   hence the first kind of run-stmt (run (=run 0)).  so run all
210           write (6, 807)
211      807  format(//, 1x, 27hrun the whole system to get, /, 1x, 5x,
212         1        48h1) the normalized horizontal energy distribution,/,9x,
213         2         39hin the image of the mask resulting from, /, 9x,
214         3         41ha uniform illumination on the mask with a, /, 9x,
215         4         19htotal of 1.0 mj/cm2, /, 6x,
216         5        37h2) the standard bleaching of the chip, /, 6x,
217         6        35h3) the actual bleaching of the chip,   /, 6x,
218         7        46h4) the etched out contours of the photoresist )
219           return
220     c
221     c   now print out the proper output for each of the parts of the system
222     c   that is to be run
223      809  goto (810, 820, 830, 840), itemp1
224     c
```

```
225       810  write (6, 811)
226       811  format(//, 1x, 32hrun the imaging subsystem to get, /, 6x,
227           1              45hthe normalized horizontal energy distribution,/,6x,
228           2              39hin the image of the mask resulting from, /, 6x,
229           3              41ha uniform illumination on the mask with a, /, 6x,
230           4              19htotal of 1.0 mj/cm2 )
231            return
232      c
233       820  write (6, 821)
234       821  format(//,1x, 40hrun the standard bleaching subsystem for,
235           1                   21h further computations )
236            return
237      c
238       830  write (6, 831)
239       831  format(//, 1x, 41hfind out the actual bleaching in the chip )
240            return
241      c
242       840  write (6, 841)
243       841  format (//, 1x, 45hfind out the etch-contours of the photoresist)
244      c
245            return
246      c
247      c
248      c   a layers-stmt   (only one kind)
249       900  write (6, 901)
250       901  format(//, 1x, 35hthe chip has the following layers- )
251      c   (910) only one kind
252            write (6, 911) (stnmls(itemp1), itemp1 = 1, nminst )
253       911  format(1x, 5x, 38ha substrate with refractive index of (,
254           1              f10.5, 2h, , f10.5, 1h), /, 1x,18x,
255           2              21hand other layers with, /, 1x, 22x,
256           3              16hrefractive index, 8x, 20hthickness in microns, /,
257           4              (1x, 18x, 1h(, f10.5, 1h,, f10.5, 1h), 8x, f10.5) )
258            return
259      c
260      c
261      c   a trial-stmt
262       1000 write (6, 1001) (stnmls(itemp1), itemp1 = 1, nminst)
263       1001 format(//, 1x, 37ha trial-statement with the parameters,
264           1         /, (6x, 10f10.5) )
265            return
266            end
```

```
1             nolist
2             subroutine testpp
3        c   a driver to test the pretty-printer subroutine by simulating
4        c   (i.e. faking) the input to it (rather, by simulating the calling
5        c   environment for it).          <-- {except for the  trial-stmt, and
6        c                                                the erroneous-stmt.}
7             common /parsem/ istmty, istknd, stnmls(25), nminst, nmpntr
8        c
9             dimension istfl(10)
10       c   this istfl(10) array holds the number of different flavours that a
11       c   statement of type istmty can have as istfl(istmty)
12       c   in short istfl(istmty) is the number of different kinds of stmts
13       c   of type istmty.
14       c
15            data istfl /2, 2, 5, 3, 2, 2, 1, 1, 1, 1/
16       c
17       c   nminst is used only in stmts (istmty.istknd =) 1.2, and 5.2
18       c   hence for this test it could be set to, say, 14.
19       c
20            nminst = 14
21       c
22            do 10 itemp1 = 1, 25
23            ritmp1 = itemp1
24            stnmls(itemp1) = ritmp1
25         10 continue
26       c
27            do 20 istmty = 1, 10
28            itemp1 = istfl( istmty )
29       c
30            do 15 istknd = 1, itemp1
31       c
32       c   only in the devmodel stmt ( etchrate curve ... ) will the
33       c   value of stnmls(1) will have an immediate effect.
34       c   hence for that stmt ,(istmty.istknd =) 5.2, we need -
35            if ((istmty .eq. 5) .and. (istknd .eq. 2)) stnmls(1) = 14.
36            call prprst
37         15 continue
38         20 continue
39       c
40       c
41       c   for the run-stmt a special treatment is to be given because it has
42       c   a prettyprinted form depending on the value of the parameter given
43       c   in it.  hence these special statements
44            istmty = 8
45            stnmls(1) = -1.0
46            call prprst
47            stnmls(1) = 0.0
48            call prprst
49            stnmls(1) = 1.0
50            call prprst
51            stnmls(1) = 2.0
52            call prprst
53            stnmls(1) = 3.0
54            call prprst
55            stnmls(1) = 4.0
56            call prprst
```

```
57      c
58      c
59      c   and finally the end-of-input statement.
60            istmty = 0
61            istknd = 1
62            call prprst
63      c
64            return
65            end
66            list
```

```
1              subroutine errm01( n )
2        c   for handling errors in machine 1
3        c   writes an error-message and returns
4        c
5              common /errlab/ lnoerr
6        c
7              logical          lnoerr
8        c
9              lnoerr = .false.
10       c
11             write (6, 9000) n
12        9000 format(//, 1x, 9h***-+-***, 10x, 20herror in machine(1)=, i5 )
13             return
14             end
```