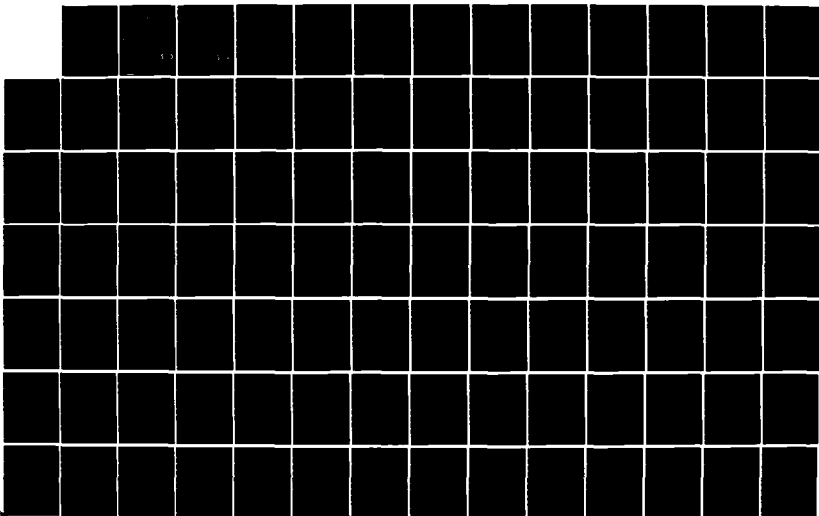
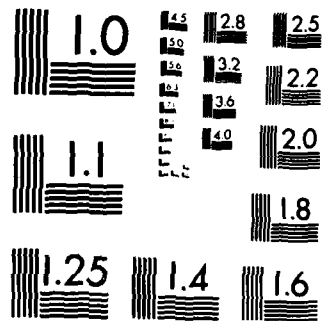


AD-A151 715 A MICROCOMPUTER-BASED PROGRAM FOR PRINTING CHECK PLOTS 1/4
OF INTEGRATED CIRC. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. K S HORTON
UNCLASSIFIED DEC 84 AFIT/GE/ENG/84D-35 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

①

AD-A151 715



A MICROCOMPUTER-BASED PROGRAM
FOR PRINTING CHECK PLOTS OF
INTEGRATED CIRCUITS SPECIFIED
IN CALTECH INTERMEDIATE FORM

THESIS

AFIT/GE/ENG/84D-35

Kirk S. Horton
Captain USAF

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

DTIC
ELECTE
S **D**
MAR 28 1985

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

85 03 13 134

DTIC FILE COPY

AFIT/GE/ENG/84D-35

A MICROCOMPUTER-BASED PROGRAM
FOR PRINTING CHECK PLOTS OF
INTEGRATED CIRCUITS SPECIFIED
IN CALTECH INTERMEDIATE FORM

THESIS

AFIT/GE/ENG/84D-35

Kirk S. Horton
Captain USAF

DTIC
ELECTE
MAR 28 1985
S B

Approved for public release; distribution unlimited

AFIT/GE/ENG/84D-35

A MICROCOMPUTER-BASED PROGRAM FOR PRINTING
CHECK PLOTS OF INTEGRATED CIRCUITS SPECIFIED
IN CALTECH INTERMEDIATE FORM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

by

Kirk S. Horton, B. S.

Captain USAF

Graduate Electrical Engineering

December 1984

Approved for public release; distribution unlimited

Preface

This thesis was primarily concerned with the development of a integrated circuit design tool, mcifplot, on a CPM based microcomputer. It provided a unique opportunity to learn more about a variety of different fields of study. In addition to the obvious application to computer aided design, the project was my first exposure to two-dimensional graphics. It was also a tremendous learning experience in the area of software design and engineering.

mcifplot is designed to implement the function of Dan Fitzpatrick's VLSI design tool cifplot. In the program itself, I relied heavily on algorithms in A Guide to LSI Implementation[8] and Principals of Interactive Computer Graphics[15]. The function of the program depends on the use of Tom Speer's CPM graphics utility PLOT.COM.

I offer special thanks to Hal Carter, my thesis advisor, for suggesting this project and for his ideas and guidance during its execution. I would also like thank my wife Julie for her patience and encouragement throughout the project.

Kirk S. Horton

Table of Contents

	<u>Page</u>
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	I-1
Background	-1
Problem	-4
Summary of Current Research	-7
Approach	-9
Sequence of Presentation	-10
II. System Requirements and Justification	II-1
Objectives	-1
Requirements Definition	-3
Program Description	-4
Operation	-5
Implementation Requirements	-6
Performance Requirements	-8
Functional Requirements	-11
III. System Design	III-1
Overview	-1
Interpret Commands	-6
Parse CIF File	-8
Build Plot File	-11
Print Error Messages	-13
IV. Detailed Design	IV-1
Design Goals	-2
Design Procedure	-3
Design Format.	-5
Detailed Design	-9
Procedure Descriptions	-12
int_command	-12
parse_cif	-16
bld_file	-19
V. Analysis	V-1
mcifplot Design	-1
mcifplot Function	-3
mcifplot Performance	-4

<u>mcifplot</u> Status	V-8
VI. Conclusions and Recommendations	VI-1
Conclusions	-1
Recommendations	-2
Bibliography	BIB-1
Appendix A: <u>mcifplot</u> Manual	A-1
Appendix B: SADT Data Dictionary	B-1
Appendix C: Structure Charts & Headers/Code	C-1
Appendix D: Data Dictionary	D-1
Appendix E: C Procedures / PC Procedures	E-1
Appendix F: C80 Procedures	F-1
Appendix G: Check Plot Examples	G-1



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
Ex	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

List of Figures

<u>Figure</u>		<u>Page</u>
III-1	<u>mcifplot</u> - Check Plotting Utility SADT-C1 .	III-2
III-2	<u>mcifplot</u> - Check Plotting Utility SADT-C2 .	III-4
III-3	Interpret Commands SADT-C3	III-7
III-4	Parse CIF SADT-C4	III-9
III-5	Build Plot File SADT-C5	III-12
III-6	Print Error Messages SADT-C6	III-14
IV-1	<u>main</u> Procedure	IV-13
IV-2	<u>int_command</u> Procedure	IV-15
IV-3	<u>parse_cif</u> Procedure	IV-17
IV-4	<u>bld_file</u> Procedure (Part 1)	IV-20
IV-5	<u>bld_file</u> Procedure (Part 2)	IV-21

List of Tables

<u>Table</u>		<u>Page</u>
II-1	<u>cifplot/mcifplot</u> Options	II-28
V-1	<u>mcifplot</u> Performance	V-5
V-2	<u>mcifplot</u> Divisions and Composition	V-8

Abstract

mcifplot, a microcomputer-based program which prints check plots of integrated circuits on a dot-matrix printer, was designed, implemented, and tested. The SADT Design, Detailed Design, Data Dictionary, algorithms, Structure Charts, and program code are presented. mcifplot interprets a geometric description file written in Caltech Intermediate Form (CIF) and produces an output file which can be printed using the public domain program PLOT.COM. mcifplot is written in C for the CPM operating system and closely matches the function of the Unix based VLSI design tool cifplot.

mcifplot interprets all standard CIF 2.0 commands with the exception of Poly Commands, Delete Definition Commands, and commands which define geometric elements in non-Manhattan directions. Program execution is primarily limited by disk access delays and depends greatly on the format and structure of the input CIF file.

*Final Report
Computer aided design, Computer graphics,
VLSI design, algorithms, theses*

A Microcomputer-based Program for Printing
Check Plots of Integrated Circuits Specified
in Caltech Intermediate Form

I. Introduction

Background

When Fairchild introduced the first Integrated Circuit (IC) in 1961, it contained only four transistors[5]. Each year since that time, the semiconductor industry has consistently improved the quality of the fabrication techniques used in manufacturing IC's. This has resulted in an almost geometric growth in complexity, with the number of devices per IC almost doubling yearly. Today it is possible to manufacture IC's which contain 400K or more devices. The physical limits of device density has still not been reached, and densities of million's of devices per circuit are predicted for the near future.

Unfortunately, the engineer's ability to design complex electronic systems has not kept pace with the improvements in fabrication technology. With this dramatic increase in density and computational power has come an increasingly difficult design problem. The first IC's were designed and laid out by hand. The layout was done on Mylar sheets which were used as patterns to cut Rubylith[11]. The layout was then checked manually by the designer. This was possible because the designs were relatively simple and they contained a small number of devices. As IC's became

more complex it became increasingly time consuming to design and layout a circuit.

This is not unexpected; it is almost impossible for a single engineer to trace and understand the operation of a circuit that contains hundreds of thousands of devices, much less design and lay it out by hand. IC design tools began to appear in the late 1960's to aid the engineer in the design process. The first design tools were used to automate the production of masks for fabrication. Next came tools for layout digitization and maintenance of hand-drawn designs[16]. These were followed by tools for discovering design rule violations and simulators used to verify the functions performed by circuits that had been laid out.

The design tools were created by IC manufactures primarily for in-house IC research and development. In the same way that IC manufactures improved their fabrication techniques, they also developed the design tools required for their unique applications and design requirements. The operation and function of these tools varied from company to company and they were usually proprietary.

As circuit density and complexity grows, it is predicted that increasingly more powerful design tools will be required to allow the engineer to cope with the magnitude of the design problem. The current design tools have a productivity of approximately 50-100 transistors per day. Ten years from now, design tools will need to produce transistors at an average rate of almost 10,000 or more per day to

completely utilize predicted circuit densities[12]. The improvement of existing tools and the development of different and more powerful tools will become an even greater priority as the gap between the capability of IC's and the engineer's ability to design these circuits increases in the future.

Universities across the country are becoming more involved in research in this area and a growing number of schools are incorporating IC design courses into their curriculum. Until recently, research and training in IC design rarely took place at universities and was almost exclusively confined to industry. IC design techniques were developed, tested, and taught only in the manufacturer's facility. Two people are largely responsible for the introduction of this field to the university environment: Carver Mead and Lynn Conway. In the early 1970's, Carver Mead began to teach IC design at the California Institute of Technology in one of the first courses taught in a university curriculum. He worked with Lynn Conway to develop a course where students not only studied IC design, but also were able to layout their circuits and have them fabricated for testing.

It was not until 1978 that Mead and Conway collaborated to produce the first book for the classroom[13]. Their Introduction to VLSI Systems is still considered the definitive work in the field and introduces the theory behind circuit operation and fabrication as well as the more practical aspects of circuit design methodology. It opened

the number of pages is larger than desired). The program then interprets "filename.cif" and if there are no errors, writes the output file to "filename.vec" (the PLOT file). "Filename.vec" can be plotted by PLOT.COM to produce the desired check plot. If errors occur, they are displayed on the standard output and then program execution is affected in one of three ways: 1) after warnings program execution continues as before, 2) after runtime errors program execution continues with all other errors being reported, but the output file is not written, 3) after fatal errors program execution is aborted and no output file is written.

Implementation Requirements.

The implementation requirements are the result of project objectives and they will directly influence the performance and functional requirements. These requirements are as follows:

- 1) The program will execute on a CPM-based microcomputer with a minimum of 64K of main memory. This computer will have a minimum of two disk drives with a capacity of at least 180K each (the lower limit of the capacity of the average 5 1/4" single-sided double-density disk).

- 2) The program and required temporary storage used during program execution will not exceed the capacity of one disk or 180K.

The above description of the program is very similar to the program description given in the cifplot manual. mcifplot performs five major actions. First, it interprets the command line and initializes the runtime environment as necessary to execute the requested options. Next, the CIF file is read and a Symbol Table is constructed of all the symbols. The Symbol Table is then traced to determine the size of the layout and the scale required for plotting it. The Symbol Table is traced again to create the PLOT file of elementary plot commands for plotting by PLOT.COM. The fifth major action which takes place throughout the operation of the other sections is error checking. The command line and the syntax of the CIF file itself is checked for errors and all errors are displayed on the standard output.

Operation

A CIF file is first created by any of a variety of methods. These methods range from hand layout and coding to automatic generation by other software tools which interpret a functional circuit description to produce a CIF file. To run the program, the user types:

```
mcifplot [options] filename.cif (return)
```

With no special options the program executes in the following way. First, the program reads "filename.cif" (the CIF file), computes the size of the plot, and asks the user if a plot is desired (this allows the user to abort the plot if

because the environment that the program operates in is a central part of the program objectives. Implementational requirements are included because they are a primary objective of the program and they will have a direct influence on the performance and functional requirements of the program. In many cases the requirements are simply a more concise statement of the system goals which were previously defined.

Program Description.

mcifplot reads a geometric description of an integrated circuit (from a file to be referred to as a CIF file), and produces a file of plot commands (to be referred to as a PLOT file), which can be read by PLOT.COM to produce a check plot. The CIF input file to the program is written in Caltech Intermediate Form (CIF) which is a low-level 2-dimensional graphics language suitable for describing integrated circuits. The commands in the CIF file describe the dimensions of geometric elements (like a Box or a Wire) and where they are located. A detailed description of the syntax of CIF is contained in chapter four of Introduction to VLSI Systems[13]. mcifplot interprets a CIF 2.0 description (with the exception of Poly Commands and geometric elements in non-Manhattan directions) including the ability to rename symbols. There are restrictions on the size and complexity of the CIF files that can be plotted because of the memory constraints of the microcomputer environment.

7) Provide warnings and error messages in the same format as cifplot. Provide additional, more detailed error messages to better specify ambiguous errors. Provide options to simplify use in the microcomputer environment. Provide error messages which relate to the microcomputer environment.

These objectives will be transformed into specific requirements in the next section. With the exception of additional error handling capabilities and features to ease its use in the microcomputer environment, the program will be designed to function identically to cifplot within the constraints imposed by memory and disk space. The goal is not to change the function of the program or to dramatically improve it, but to duplicate existing functions in a new implementation for a different environment. It is stressed that mcifplot is not a translation of cifplot's code into a slightly different implementation of C, but is a new design and development specifically for the special requirements of execution on a microcomputer.

Requirements Definition

This section defines the requirements for mcifplot. A description of what the program does and how it will be used is given first. Next the requirements for the program are listed in three categories: implementation, performance, and functional. Requirements are listed first for implementation

mcifplot will duplicate the applicable functions of the VLSI design tool cifplot. It will operate on CPM-based microcomputer with a minimum configuration of 64K memory and two disk drives with a capacity of 180K each. mcifplot will produce a file of plot commands which will be read and plotted by the public domain utility PLOT.COM to produce the desired check plot.

In addition to this primary goal, the program will be designed to meet the following objectives:

- 1) Provide all applicable options of cifplot. These options will be activated using the same commands as in cifplot.
- 2) Minimize the amount of disk space required for the program and the program's temporary storage so that it can be contained on one disk.
- 3) Minimize the amount of main memory required for program execution.
- 4) Minimize the execution speed of the program.
- 5) Divide program functions to allow the finished program to be split into several program modules if the completed program exceeds available main memory.
- 6) Avoid system or compiler unique features so that the program will be executable on all CPM-based microcomputers and be easily translated to other operating system environments.

II. System Requirements and Justification

The most important stage of any software development project is the specification of requirements. This specification is the foundation on which the system design, detailed design, and program coding and testing is based. Well written, specific requirements allow the succeeding steps of the system development to proceed quickly and efficiently. On the other hand, a small error or omission in the requirements will cause major problems and rework if it is not discovered until the coding stage. This chapter presents the project objectives and the implementation, performance, and functional requirements for mcifplot. The justification for the specified requirements is presented as each one is listed.

Objectives

One of the most important parts of establishing general design objectives is that the goals be implementation independent. However, the design of mcifplot is somewhat unique in that the program will duplicate the function of an existing program in a new operating environment. For this reason, some implementation dependent features are of necessity included in the design objectives. These goals are listed first because they are the primary objectives of the system design. The design of mcifplot has the following primary goal:

are the result of the limitations imposed by the operating environment will also be discussed.

Chapter III is a description and explanation of the system design as documented by SADT. Charts from the SADT design are used as required to clarify the description.

Chapter IV documents the detailed design of the program. It includes descriptions of the required algorithms and how they were derived, as well as structure charts to decompose module functions.

Chapter V contains an analysis of the design, the function of mcifplot, the performance of mcifplot, and a comparison to the performance of cifplot.

Chapter VI presents the conclusion to the thesis. It includes recommendations for future work on mcifplot and similar microprocessor based IC design tools.

The first stage of the project will involve analysis of the way cifplot functions and the limitations imposed by the microcomputer operating environment. A detailed list of requirements will be derived from this analysis. These requirements will then be used to formulate a general system design. The system design will be implemented using the Structured Analysis and Design Technique (SADT). SADT charts and documentation will define the program to at least the major module level.

The system design will then form a basis for the detailed design. Additional detail as required in the form of data dictionary expansions, structure charts, and specific algorithms should be contained in the detailed design. This design will be implemented on CPM-based microcomputer (the same environment in which it will operate) in the C language. In particular, the TOOLWORKS C/80 compiler will be used primarily because of its high cost/performance ratio. The final phase of the project will be the evaluation of mcifplot to determine if the established requirements were met and the quality of its performance and implementation.

Sequence of Presentation

Chapter II defines the requirements for mcifplot and a justification of how these requirements were derived. A large portion of the chapter describes which features of cifplot will and will not be implemented. Requirements which

the clarity of the resulting check plots is limited, it is certainly a low cost alternative to producing the required plots.

Tom Alamy of Tektronix has developed a set of IC design tools for the TRS 80 Model I which are written in the Forth language[3]. These programs have over 200,000 lines of code and have been used to design circuits of up to 15,000 transistors. The system can be used for low level cell design or for high level layout using module generators and cell placement.

Probably the most active area of research in low cost design tools is at the University of Utah, under the direction of Professor Kent Smith. A set of VLSI tools has been developed using a design methodology called Path-Programmable Logic (PPL). Circuits are designed by arranging symbols which represent circuit elements on a grid. The circuit densities achieved are close to that of custom VLSI circuits. Early work involved the use of a remote terminal and a minicomputer host[14]. More recently, a PPL system has been implemented on a 16-bit IBM personal computer[3]. Chips as large as 250 X 250 mils have been designed and simulated on this system.

Approach

The approach used for this thesis should closely parallel the approach required for any large software project.

computer system. Although much research is taking place in developing VLSI design tools and even design systems for high performance "personal" computers[7;18;23;24], this thesis concerns the design of VLSI tools for computers on the low end of this scale. Specifically, systems of interest will be microcomputers with limited memory, floppy disk storage, and total system costs of under \$5000.

Educational CAD. There is increasing interest in using microprocessors as design aids in education[2;5]. The main reason for the interest is the availability of these computers and their low cost. In most cases, student design projects do not require high performance systems or high resolution displays. While there are hundreds of low cost microcomputers to choose from, the software situation is a different story. There are a variety of CAD software packages available like PC-Draw, CADdraft, AutoCAD, and others, but none of these packages is specifically for IC design or produces the required geometric layout files[9].

Microcomputer Design Tools. With the exciting possibility of allowing IC design to be taught with a minimum of hardware expenditure, it is surprising how little research is taking place in this field. The author is aware of efforts in three areas. A program developed at New York University at Stony Brook allows printing of IC geometric file check plots on an inexpensive line printer[22]. This program is written in Pascal and uses capital letters to represent the various layers and overlays of an IC layout. Although

microcomputer-based check plot program, especially when working with small subsets of the entire circuit.

The purpose of this thesis is to design and implement the IC design tool cifplot on a CPM-based microcomputer. Written at Berkeley by Dan Fitzpatrick, cifplot reads a file of primitive geometric commands written in Caltech Intermediate Form (CIF) and produces a check plot. The new microcomputer based tool, mcifplot, will incorporate all of the applicable features of cifplot allowing for constraints imposed by memory and disk space. mcifplot, along with a group of other microcomputer based tools, will allow students to do IC design work on existing school or privately owned microcomputers, without the expense of a minicomputer system or the performance impact on general purpose mainframes.

Summary of Current Research

This summary of current research is divided into three sections. The first section defines the scope of interest and gives a working definition for "low cost personal computer." The second section describes the general status of CAD on microcomputers in education. The final section describes current research in VLSI design tools for low cost microcomputers.

Scope of Interest. The phrase "low cost personal computer" is applied in the literature to computers ranging in cost from a \$3000 Apple II system to a \$30,000 LSI-11 mini-

Beyond cost savings, the use of microcomputers would alleviate many of the problems associated with Computer Aided Design (CAD) on large mainframes: heavy use of system time, complex support and maintenance management, lack of special purpose programming, and delays associated with serving many users[2;5]. A group of microcomputer-based design tools would allow a small circuit or a subset of a larger circuit to be designed and tested on the student's computer. The size of the circuit that could be designed would be limited by the speed of the processor and the size of memory. However, in most educational design problems, these limitations would not seriously affect student projects. An integrated set of design tools would require programs to do logic design and simulation, chip layout, design rule and electrical checking, feature extraction, circuit simulation, and timing analysis[4].

If implemented on a microcomputer, a design tool for producing layout check plots would be especially useful. While a program like this would be a necessary part of an integrated set of microcomputer design tools, it would also have application when used alone. A check plot program would allow remote users of a mainframe to produce plots independently from their office or home. A large amount of time is usually spent during a design connecting circuit subsets together and checking the connections with check plots. This activity could also be done easily with a

hardware and software. Most existing IC design tools are written for execution on mainframe computers or expensive dedicated minicomputer systems. IC design tools produced by the schools listed above are available in the public domain, but require a large mainframe on which to execute[2]. While the quality and performance of these tools is excellent and many are available at no charge, the cost of the equipment required for their operation is excessive for many universities. The alternative, a special purpose design system using a minicomputer, can cost from \$150,000 to \$800,000[5].

This problem has two aspects: 1) the cost of the purchase and maintenance of a computer dedicated to IC design is prohibitive for many universities, 2) when a time-shared mainframe is available for use in IC design courses, the memory used and execution time of the design tools has a serious impact on the performance of the system. One solution to this problem is to utilize less expensive and readily available personal computer systems to do the IC design required for educational curriculums. Based on price alone, microcomputer-based IC design appears to be an attractive alternative to the mainframe or special purpose design system. A microcomputer system with the minimal required features could cost as little as \$2,000 and many students would already own the necessary equipment for the design work.

the door for more extensive involvement of schools in IC design research and education. Today, almost 100 universities teach IC design courses[20], many of which involve fabrication of student circuits.

University research in IC design is moving in many directions. One of the most active areas is the development of new design tools. While most of these tools are being used within the educational community, it is not uncommon for industry to borrow ideas and even particular programs from these university designed tools. The Very Large Scale Integration (VLSI) tool SPICE2, which was developed at Berkeley, has seen wide distribution to both the IC industry and other universities[17]. At present, universities are developing IC design tools at such a rate and in such quantities that they are sometimes outdated before they are used at distribution sites. In particular, the following universities are leading in the creation of new design tools: California Institute of Technology, Carnegie-Mellon University, Massachusetts Institute of Technology, Stanford University, and the University of California at Berkeley[19].

Problem

Unfortunately, the average university is working with far fewer resources than the schools listed above. One of major stumbling blocks for developing a quality program in IC design for most schools is the cost of the required

3) The program will be written in the C programming language.

4) The output PLOT file generated by the program must be compatible with the public domain plotting utility PLOT.COM. For example, a CIF Box command must be translated into the Draw and Fill commands that are recognized by PLOT.COM. Compatibility with PLOT.COM must include such factors as multiple page plots and the precision of numbers.

The justification for these requirements is that they are part of the project objectives which were independently established. However, these features are largely based on what is currently available in inexpensive microcomputer systems. CPM is one of the more popular and established operating systems for personal computers. There is an extensive group of public domain programs available for CPM, including the utility PLOT.COM, which will greatly simplify the program's scope and complexity.

The memory and disk sizes listed above were once fairly large by personal computer standards. Furthermore, as personal computer performance and capability improves, memory and disk capacity will continue to grow. By restricting the program to the above memory and disk size, compatibility is assured with thousands of existing systems, as well as more powerful future microcomputer systems.

Finally, the C language was chosen for it's portability as well as it's robust operators and commands. To accomplish the implementation of mcifplot in the limited memory availa-

ble, it will be necessary to have the extensive control over low level I/O functions which is possible with C. This is also the language that cifplot is written in. In particular, the TOOLWORKS C/80 compiler will be used because of its high cost/performance ratio.

Performance Requirements

This is the most difficult set of requirements to establish. Although an implementation of this program already exists, the operating environment that it is designed for (Unix operating system on a VAX 11-780) is vastly different than that of a microcomputer. The size, speed, and performance of cifplot provide little information about the type of results to be expected on a microcomputer. However, two performance requirements are specified as follows:

- 1) The program will read and produce an output PLOT file for a CIF file of up to 35K in length for the computer system specified in the objectives. It is expected that each executable CIF command will require an average of 5 - 15 plot commands which will occupy approximately four times as much disk space as most CIF commands. A maximum size 35K CIF file is established to allow the output PLOT file (which is estimated to be four times as large - 140K) to be placed on the same disk.

The 35K size limit will be used as a guideline for program development and evaluation. The actual possible maximum size of an input CIF file will depend on the disk

capacity of the computer on which the program is executed and the characteristics of the CIF file (see factors below). For all input CIF files, the program will write the output PLOT file until execution is complete or the capacity of the disk is reached. If the size of the PLOT file reaches disk capacity, then the file will be closed at the maximum possible size so that a check plot can be made of the CIF file up to this point. Because the output PLOT file will vary greatly in size depending on the structure and format of the input CIF file, no error checking will be done on the size of the input file before processing begins.

Three factors will affect the validity of the estimated CIF file to PLOT file ratio of 1:4. First, it is assumed that only Manhattan directions are used in the CIF file (see Functional Requirements). Non-Manhattan directions would require many additional PLOT file commands because of the greater number of fill plot commands required to "color" the irregular shape. The second factor is the number of Call commands. Each Call will cause all the executable commands specified (and called) in a symbol to be transformed into the required plot commands in the PLOT file. This could result in a single CIF command generating 100's of plot commands.

Finally, the length of the plot (x direction with no rotation) will effect the number of plot commands. PLOT.COM uses memory to create an image to be plotted on a page by page basis. For example, if a plot requires three pages of

output, then wires and boxes which run the length of the plot would have to be specified with plot commands three different times. Any of these factors could cause the output PLOT file of a CIF file that was less than 35K to exceed the estimated 140K. For this reason, the maximum size file that can be plotted will depend on the characteristics of the input file as well as the system disk capacity.

2) The execution time of the program on a maximum size CIF file (35K) will not exceed one hour for the system specified in the objectives. Of course, execution speed will not depend solely on the size of the file, but also on many other factors. These factors include the CPU (Central Processing Unit) clock rate, disk access speed, the dimensions of the CIF file, and the hierarchical nature of the CIF file. The characteristics of the CIF file will affect execution speed in proportion to the number of commands that must be processed. The disk access speed and CPU clock rate will vary from system to system. The one hour time limit is an arbitrary limit beyond which the author feels the usefulness of the program for IC development would be seriously impaired. It should be noted that this time estimate does not include the time required to execute PLOT.COM and obtain a plot of the PLOT file.

The execution time for PLOT.COM also depends on the type of input file which it processes. PLOT.COM has two stages of execution. First, the input file is read and the elementary plot commands are used to create a bit map in

memory which corresponds to the commands. The execution time for this stage depends on the type of input file. The author has observed execution times in this stage which range from a few seconds to eight minutes. Fill commands are especially CPU intensive (unfortunately the output of mcifplot will have many Fill commands).

The second stage of PLOT.COM execution is the output of the memory map to the printer. This time varies from printer to printer, but is constant for each full page output. With the Okidata 92, output takes approximately seven minutes for one full page. This time will be less if there are "blank" output lines which require no printing (PLOT.COM uses line feeds to quickly output these lines).

The combined time of both stages must be considered in order to estimate the execution time of PLOT.COM. Of course, multiple page plots increase the execution time by at least a factor of the number of pages times the printer output time.

Functional Requirements

The functional requirements of the program are concerned with what it does and does not do. Because the program is intended to duplicate the functions of cifplot, the manual for cifplot will be examined function by function to illustrate which functions and options mcifplot will and will not perform. These requirements were developed with the

aid of the CIF syntax description in A Guide to LSI Implementation[8] and will be divided into three categories.

Each function or option in cifplot will be labeled as Not Required (NR), Required (R), or Desired (D). Options listed as Desired will be implemented if possible within the constraints imposed by memory size, program size and development time. Defining a priority for each requirement at this point will simplify the choices which will be necessary if the scope of the program needs to be reduced during development. This will also suggest directions for future enhancement of the final program.

In order to delineate between the cifplot manual and the requirements description, the following conventions will be adapted. The manual text will be indented and single spaced. All requirements text will have normal margins and begin with one of the three categories (R, NR, D). A justification for the decision to implement or not implement an option or function is given directly following each requirement.

NAME

cifplot - CIF interpreter and plotter

SYNOPSIS

cifplot [options] file1.cif [file2.cif ...]

R: mcifplot will use this format for the command line with the exception of reading more than one CIF file. More than one file in the command line will produce a fatal error.

D: Allow reading and interpreting of multiple CIF files.

This would be implemented primarily for consistency with cifplot. In the operating environment for which the program is designed, this feature would be required only rarely. The maximum total size of the input CIF file would still be restricted as above.

DESCRIPTION

Cifplot takes a description in Cal-Tech Intermediate Form (CIF) and produces a plot. CIF is a low-level graphics language suitable for describing integrated circuit layouts. Although CIF can be used for other graphics applications, for ease of discussion it will be assumed that CIF is used to describe integrated circuit designs. Cifplot interprets any legal CIF 2.0 description including symbol renaming and Delete Definition commands. In addition, a number of local extensions have been added to CIF, including text on plots and include files. These are discussed later. Care has been taken to avoid any arbitrary restrictions on the CIF programs that can be plotted.

R: mcifplot will also read and interpret CIF files including symbol renaming. However, geometric features in non-Manhattan directions as well as Poly Commands will produce runtime errors. The purpose of the Poly Command is to create non-Manhattan geometric elements and is not implemented. The implications of this restriction are discussed in more detail at the end of the chapter. Implementation of the Delete Definition command would require that construction of the PLOT file would start before the entire CIF file is read. This would be difficult to implement in the limited disk space of a microcomputer. The local extensions supported will be discussed below. Some arbitrary restrictions on the size of the file are required by the operating environment and these were discussed under implementation and perfor-

mance requirements.

To get a plot call cifplot with the name of the CIF file to be plotted. If the CIF description is divided among several files call cifplot with the names of all files to be used. Cifplot reads the CIF description from the files in the order that they appear on the command line. Therefore the CIF End command should be only in the last file since cifplot ignores everything after the End command. After reading the CIF description but before plotting, cifplot will print a estimate of the size of the plot and then ask if it should continue to produce a plot. Type y to proceed and n to abort. A typical run might look as follows:

```
% cifplot lib.cif sorter.cif
Window -5700 174000 -76500 168900
Scale: 1 micron is 0.004075 inches
The plot will be 0.610833 feet
Do you want a plot? y
```

After typing y cifplot will produce a plot on the Benson Varian plotter.

R: mcifplot will function in the manner described above with the following exceptions: 1) will not accept multiple input files (multiple CIF files in the command line will produce a fatal error), 2) no plotting will take place, a PLOT file will be produced which can be plotted using PLOT.COM. The diagnostics will be in the same format as illustrated above. A complete trace of the Symbol Table will be required before the diagnostics are printed.

D: Allow multiple CIF files to be read as stated above.

Cifplot recognizes several command line options. These can be used to change the size and scale of the plot, change default plot options, and to select the output device. Several options may be selected. A dash(-) must precede each option specifier. The following is a list of options that may be included on the command line:

-w xmin xmax ymin ymax
(window) The -w options specifies the window; by default the window is set to be large enough to contain the entire plot. The windowing commands lets you plot just a small section of your chip, enabling you to see it in better detail. Xmin, xmax, ymin, and ymax should be specified in CIF coordinates.

R: Fully implemented.

-s float
(scale) The -s option sets the scale of the plot. By default the scale is set so that the window will fill the whole page. Float is a floating point number specifying the number of inches which represents 1 micron. A recommended size is 0.02.

R: A subset of this option will be implemented. The plot in the X direction will be consistent with float scaling value. If the Y dimension plot bounds exceed the size of the page, they will be truncated to the maximum page width (cifplot would produce as many strips as required to plot the CIF file at the input scale). The allowable scale value also depends on the size of the output PLOT file (a large scale factor could possibly require a PLOT file larger than available disk space).

D: Implement full strip plotting for consistency with cifplot.

-l layer_list
(layer) Normally all layers are plotted. The -l option specifies which layers NOT to plot. The layer_list consists of the layer names separated by commas, no spaces. There are some reserved names: allText, bbox, outline, text, pointName, and symbolName. Including the layer name allText in the list suppresses the plotting of text; bbox suppresses the bounding box around symbols.

outline suppresses the thin outline that borders each layer. The keywords text, pointName, and symbolName suppress the plotting of certain text created by local extension commands. text eliminates text created by user extension 2. pointName eliminates text created by user extension 94. symbolName eliminates text created by user extension 9. allText, pointName, and symbolName may be abbreviated by at, pn, and sn respectively.

R: All options will be implemented with the exception of the "text" keyword (User Extension 2 is not implemented). The options bbox and outline are concerned with the way that CIF syntax is interpreted. The options allText, pointName, and symbolName are concerned with eliminating text created by user extensions. Incorrect format will produce a fatal error.

D: Implement the "text" keyword in conjunction with User Extension 2.

-c n
(copies) The -c option specifies the number of copies of the plot you would like. This allows you to get many copies of a plot with no extra computation. Makes n copies of the plot. Works only for the Varian and Versatec. Default is 1 copy.

NR: Multiple copies can be obtained by multiple runs of PLOT.COM. However, since CPM converts all lower case letters to upper case in the command line, use of '-c' in the command line will be interpreted as the 'C' (Comments) option as described below.

D: This option could be implemented with a modification to PLOT.COM using the PLOT.COM Extend Command.

-d n

(depth) This option lets you limit the amount of detail plotted in a hierarchically designed chip. It will only instantiate the plot down n levels of calls. Sometimes too much detail can hide important features in a circuit.

R: Fully implemented. This option may also allow larger CIF files to be plotted because less detail is required. The default depth setting is 10 to prevent stack overflow during symbol tracing.

-g n

(grid) Draw a grid over the plot with spacing every n CIF units.

R: Implemented with the exception of labeling the grid lines with dimensions.

D: Implement labeling the grid lines with dimensions.

-h

(half) Plot at half normal resolution. (Not yet implemented.)

NR: Options not supported by cifplot will not be implemented. Use will produce a fatal error.

-e

(extensions) Accept only standard CIF. User extensions produce warnings.

R: Fully implemented.

-I

(non-Interactive) Do not ask for confirmation. Always plot.

R: Fully implemented.

-L

(List) Produce a listing of the CIF file on standard output as it is parsed. Not recommended unless debugging hand-coded CIF since CIF code can be rather long.

NR: This would be an option of questionable value in mcif-plot's operating environment and would increase the already long program execution time. Use will produce a fatal error.

D: Implement for consistency with cifplot.

-a n

(approximate) Approximate a roundflash with an n-sided polygon. By default n equals 8. (I.e. roundflashes are approximated by octagons.) If n equals 0 then output circles for roundflashes. (It is best not to use full circles since they significantly slow down plotting.) (Full circles not yet implemented.)

NR: Default approximation is always a square box. Implementation is too costly in execution time and size of the output PLOT file. Use will produce a fatal error.

D: Implement for consistency with cifplot.

-b "text"

(banner) Print the text at the top of the plot.

R: Fully implemented using PLOT.COM Text Command.

-C

(Comments) Treat comments as though they were spaces. Sometimes CIF files created at other universities will have several errors due to syntactically incorrect comments. (I.e. the comments may appear in the middle of a CIF command or the comment does not end with a semicolon.) Of course, CIF files should not have any errors and these comment related errors must be fixed before transmitting the file for fabrication. But many times fixing these errors seems to be more trouble than it is worth, especially if you just want

to get a plot. This option is useful in getting rid of many of these comment related syntax errors.

R: Fully implemented.

-r
(rotate) Rotate the plot 90 degrees.

R: Fully implemented.

-V
(Varian) Send output to the varian. (This is the default option.)

NR: Not applicable to mcifplot's mode of operation. Use will produce a fatal error.

-W
(Wide) Send output directly to the versatec.

NR: Not applicable to mcifplot's mode of operation. However, the '-W' option will be interpreted as the '-w' (window) option because of the CPM command line conversion to upper case characters.

-S
(Spool) Store the output in a temporary file then dump the output quickly onto the Versatec. Makes nice crisp plots; also takes up a lot of disk space.

NR: This is the standard mode of operation of mcifplot except that the temporary file (PLOT file) is not plotted. However, because of the CPM command line conversion to upper case, the '-S' option is interpreted as the '-s' (scale) option.

- T (Terminal) Send output to the terminal. (Not yet fully implemented.)
- Gh -Ga (Graphics terminal) Send output to terminal using it's graphics capabilities. -Gh indicates that the terminal is an HP2648. -Ga indicates that the terminal is an AED 512.

NR: These options are not applicable to mcifplot's operating environment. However, because of the CPM command line conversion to upper case, the '-Gh' or '-Ga' option will be interpreted as the '-g' (grid) option. '-T' will be interpreted as the '-t' (transfer) option which is described below.

- X basename (eXtractor) From the CIF file create a circuit description suitable for switch level simulation. It creates two files: basename.sim which contains the circuit description, and basename.node which contains the node numbers and their location used in the circuit description.

When this option is invoked no plot is made. Therefore it is advisable not to use any of the other options that deal only with plotting. However, the window, layer, and approximate options are still appropriate. To get a plot of the circuit with the node numbers call cifplot again, without the -X option, and include basename.nodes in the list of CIF files to be plotted. (This file must appear in the list of files before the file with the CIF End command.)

NR: Although this is a working cifplot option, it is considered out of scope of the thesis project and will not be implemented. Use will produce a fatal error.

- P pattern_file (Pattern) The -P option lets you specify your own layers and stipple patterns. Pattern_file may

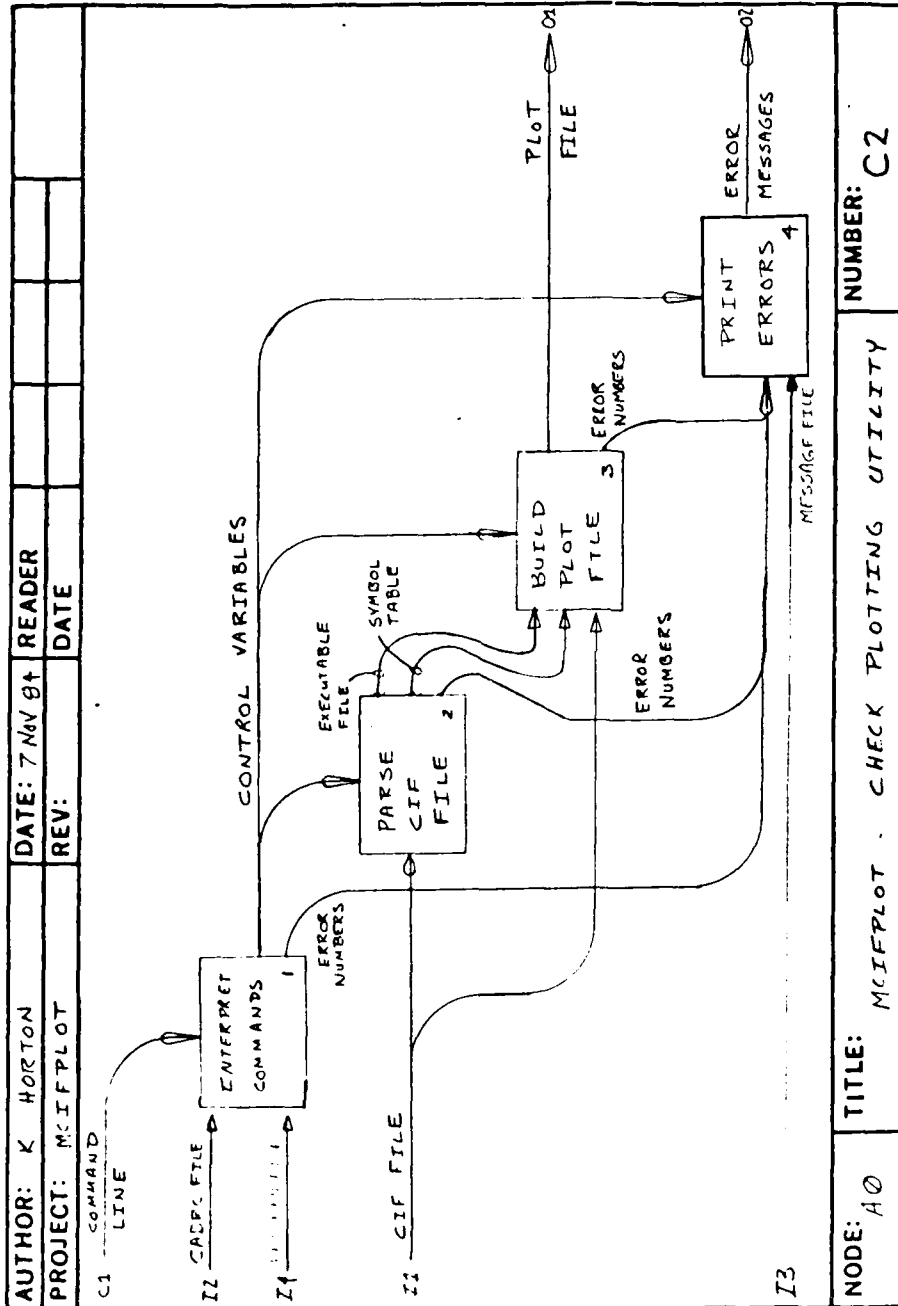


Figure III-2

The purpose of this is to emphasize the relationship between the two programs. mcifplot is very similar in function to a language compiler. In this case, the language to be compiled is CIF. The standard compiler functions of reading an input file, checking syntax and semantics, and writing an output file in a different format are all accomplished.

The program reads a CIF file (file of elementary plot commands which conforms to the CIF 2.0 syntax as defined in [13] including the restrictions listed in chapter 2) and produces an output PLOT file. The PLOT file is composed of commands which can be read by the public domain program PLOT.COM to produce the desired check plots. mcifplot interprets the input CIF file and checks for correct syntax and semantics. Incorrect syntax, semantics, or runtime overflow will produce the second output: error messages to the standard output. The type of error (fatal, runtime, or warning) will determine whether or not the PLOT file can (or will) be produced. The program terminates with no output when it encounters a fatal error.

The function of the program is determined by built-in defaults, the PATTERN file, the CADRC file, and the Command Line. These items control optional features of the program and are used to tailor the output file as desired.

The program activities can be divided into four major modules or sections. These sections are called Interpret Commands, Parse CIF File, Build Plot File, and Process

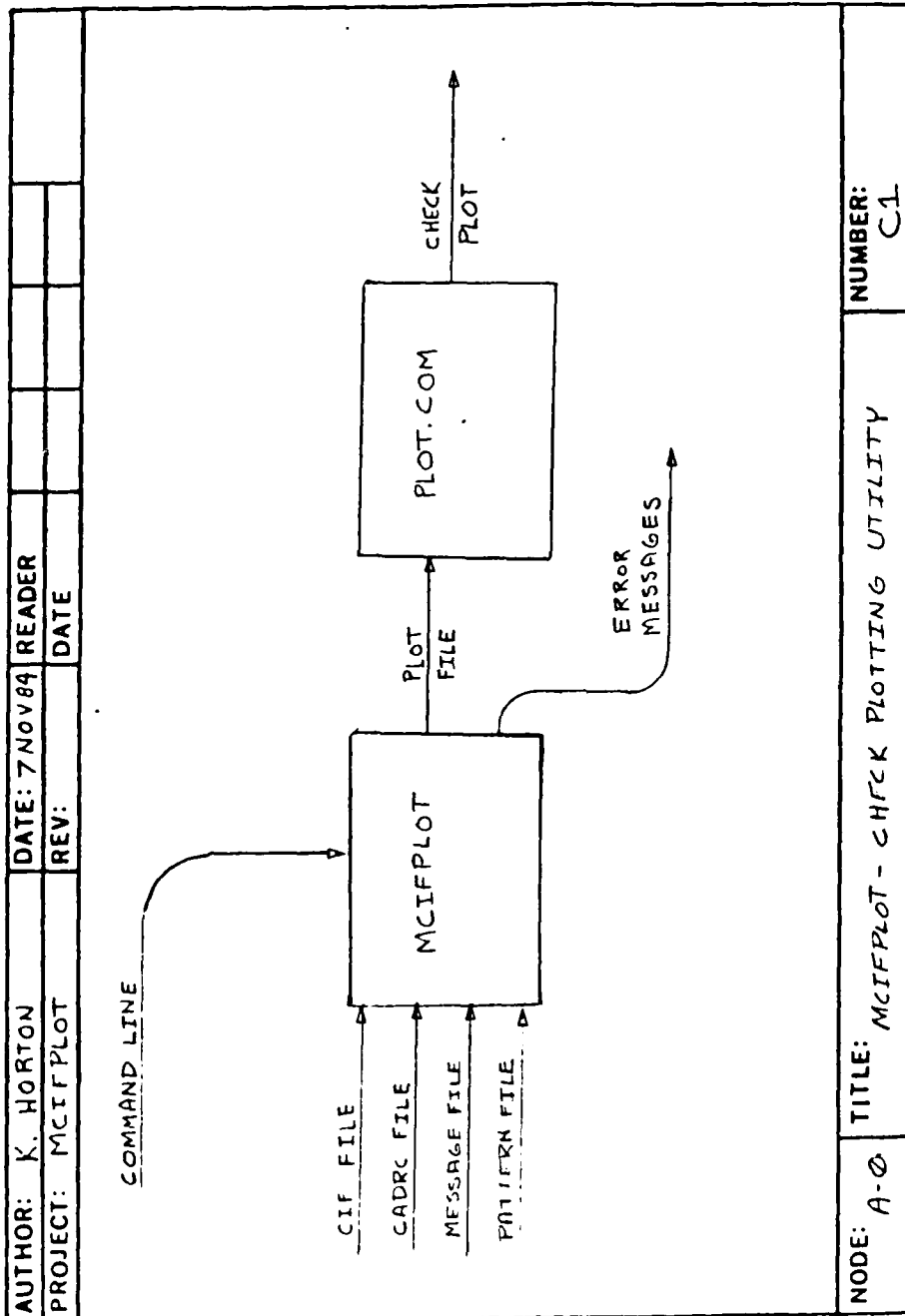


Figure III-1

III. System Design

The overall plan which describes the general approach to the implementation of a problem's solution is described as the System Design. The System Design is the beginning of the translation of detailed requirements into a finished product. Because this thesis is concerned with the development of a software program, it is appropriate to use existing software engineering tools for its design and implementation.

The Structured Analysis and Design Technique (SADT) was chosen for the System Design because it is very effective for design at the system level and because it was familiar to the author. The SADT design of mcifplot is included in this chapter (process descriptions which are normally written on the back of the SADT charts are included in the text of the chapter). The associated Data Dictionary which contains a detailed description of all the SADT data items is included in Appendix B. At this stage the design is intended to be independent of a particular language or data structure format.

Overview

The SADT diagram C1 in Figure III-1 presents the overall picture of the inputs, outputs, and controls of mcifplot. This top level SADT diagram has the nonstandard feature of including a second program process block (PLOT.COM).

the clarity of the the check plot. Of course, the outlines can be eliminated with a command line option.

Finally, because of the expected length of program execution time, an additional Desired requirement would be to allow execution to be suspended and then resumed at a later time. The status of the program and all intermediate values would be saved on disk and the program could be restarted and would resume execution by using a special command.

operating environment and are required in order to decrease execution speed and memory utilization.

First, non-Manhattan directions are not allowed and will produce runtime errors if detected. This restriction has many implications. Directions specified in Box Commands and transformations in Call Commands must be in Manhattan directions. Wire Commands (Poly Commands are not implemented) must also lie in Manhattan directions and will be approximated with boxes. The Roundflash Command will be approximated with a square box. Any CIF Commands that do not meet these restrictions will not be accepted and will produce a runtime error. An additional Desired requirement would be to allow the acceptance of non-Manhattan directions and this should be implemented if development time allows.

Secondly, the outlines of two or more geometric elements on the same layer will intersect if located together. The standard mode of operation for cifplot is to outline the outside of each layer with a black line. Thus, a rectangle is normally drawn with a rectangular region of a particular stipple pattern and then this region is outlined with a black line to clarify its dimensions. With cifplot, if two geometric features on the same layer intersect (for example, when two wires overlap) they are outlined only on the outside of the overlapping regions. With mcifplot the commands for each feature will be generated as it is encountered in the CIF file. This will cause the outlines of overlapping regions to intersect and will slightly degrade

cifplot's options, all options are interpreted as upper case letters. A new users manual for mcifplot with a listing of all supported options and directions for program operation is included in Appendix B.

Table II-1 cifplot/mcifplot Options

<u>cifplot</u> Option	Option Name	Status	<u>mcifplot</u> Option	Option Name
a	approximate	NI	-	-
b	banner	I	b	banner
c	copies	NI	-	-
C	Comments	I	C	comments
d	depth	I	d	depth
F	Font	NI	-	-
g	grid	I	g	grid
Ga/t	Graphics term	NI	-	-
h	half resolut.	NI	-	-
I	interactive	I	I	interactive
l	layer	I	l	layer
-	-	I*	m	make page
-	-	I*	n	notify
O	Output	NI	-	-
-	-	I*	o	output
P	Pattern	I	P	pattern
-	-	I*	q	quiet
r	rotate	I	r	rotate
s	scale	I	s	scale
S	Spool	NI	-	-
T	Terminal	NI	-	-
-	-	I*	t	transfer
V	Varian	NI	-	-
w	window	I	w	window
W	Wide	NI	-	-
X	eXtractor	NI	-	-

NI = Not Implemented

I = Implemented

* = options unique to mcifplot

In addition to the differences listed above, mcifplot will have two other additional functional restrictions. These restrictions are the result of the limitations of the

After execution of the program is successfully completed, transfers control from mcifplot to filename with the name of the output PLOT file as input. A default extension of ".COM" is assumed for filename if no extension is specified. The most useful choice for filename is the current version of PLOT.COM.

The execution time of mcifplot will be lengthy for large or complex input CIF files. Messages will be displayed on the standard output during execution to notify the user of the status of program execution. The following new option is required:

-q (quiet)

Inhibits displaying status messages on the standard output so that mcifplot executes in the same manner as cifplot.

mcifplot will frequently be used to prepare one page plots for inclusion in reports or presentations. In order to allow the automatic generation of these one page plots, the following new option is required:

-m (make page)

Forces the x dimension to one page or less in length, and rescales the y dimension if required. Adds a border of white space around the plot.

The cifplot options and those options required for implementation in mcifplot are summarized in Table II-1. Although the mcifplot options are listed in both lower and upper case letters in order to have a similar format as

the other options. Space before semi-colons in local extensions can cause syntax errors.

The -O option produces simple cif with no scale factors in the DS commands. Because of this you must supply a scale factor to some programs, such as the -l option to cif2ca.

R: Correct interface problems caused by -r option. Correct syntax error problems caused by spaces in local extensions.

NR: Correction of -O option is not required because it is not implemented.

Because of the long execution time expected for mcif-plot it is important to clearly define the status of all options. This will help prevent the user from executing the program and producing a check plot only to discover that a desired option was set incorrectly. The following new option is required:

-n (notify)

Displays the status of program option defaults and new options specified from the command line or CADRC file on the standard output. Also displays diagnostics concerning the number of commands and symbols in the input CIF file. This text output can be redirected to a file for a saved copy if desired.

In order to print a check plot, PLOT.COM must be executed with the output PLOT file as input. The following new option is required so that the check plots can be automatically produced:

-t filename (transfer)

D: Implement for consistency with cifplot.

```
9 name;
  (Name symbol) name is associated with the current
  symbol.

94 name x y;
94 name x y layer;
  (Name point) name is associated with the point
  (x, y). Any mask geometry crossing this point is
  also associated with name. If layer is present
  then just geometry crossing the point on that
  layer is associated with name. For plotting this
  command is similar to text on plot. When doing
  circuit extraction this command is used to give
  an explicit name to a node. Name must not have
  any spaces in it, and it should not be a number.
```

R: Fully implemented.

```
FILES
~cad/.cadrc
~/.cadrc
~cad/bin/vdump
/usr/lib/vfont/R.6
/usr/tmp/cif*
```

R: Implement checking the disk for a CADRC file at the start of program execution. In the CADRC file, options which are listed on the line starting with the word "mcifplot" will also be used in addition to command line options to modify program execution.

```
ALSO SEE
mcp(cad1),vdump(cad1), cadrc(cad5)
A Guide to LSI Implementation by Hon and Sequin,
Second Edition (Xerox PARC, 1980) for a
description of CIF.
```

```
AUTHOR
  Dan Fitzpatrick
```

```
BUGS
```

The -r is somewhat kludgy and does not work well with

NR: Not implemented for the same reasons that multiple input CIF files are not allowed.

D: Implement for consistency with cifplot.

0A s m n dx dy ;
(Array) Repeat symbol s m times with dx spacing in the x-direction and n times with dy spacing in the y-direction. s, m, and n are unsigned integers. dx and dy are signed integers in CIF units.

NR: Although this is a working cifplot option, it is considered out of scope of the thesis project and will not be implemented.

D: Implement for consistency with cifplot.

1 message;
(Print) Print out the message on standard output when it is read.

R: Fully implemented. Will prove useful in program development.

2 "text" transform ;
2C "text" transform ;
(Text on Plot) Text is placed on the plot at the position specified by the transformation. The allowed transformations are the same as the those allowed for the Call command. The transformation affects only the point at which the beginning of the text is to appear. The text is always plotted horizontally, thus the mirror and rotate transformations are not really of much use. Normally text is placed above and to the right of the reference point. The 2C command centers the text about the reference point.

NR: User Extension 9 provides the same function as User Extension 2, therefore it will not be implemented.

include and array commands (see next section) and replaces them with equivalent standard CIF statements. The resulting file is suitable for transmission to other facilities for fabrication.

NR: The include and array commands are not implemented so there is no requirement to "remove" them. Instead, a new option is implemented as follows:

-o output_file

(output) The output PLOT file is written to the file with name output_file. Normally, the output PLOT file is written to a file on the same drive with the same name as the input CIF file and a ".VEC" extension. This option allows the specification of a new name or drive. output_file must be a legal CPM file name.

In the definition of CIF provisions were made for local extensions. All extension commands begin with a number. Part of the purpose of these extensions is to test what features would be suitable to include as part of the standard language. But it is important to realize that these extensions are not standard CIF and that many programs interpreting CIF do not recognize them. If you use these extensions it is advisable to create another CIF file using the -O options described above before submitting your circuit for fabrication. The following is a list of extensions recognized by cifplot:

R: All extensions that are not implemented will produce warnings and be ignored (with the exception of User Extension 0 which will produce a fatal error). The extensions implemented are described below.

01 filename;

(include) Read from the specified file as though it appeared in place of this command. Include files can be nested up to 6 deep.

```

1
; example pattern file of format 1
NP 27
NC 13
NB 127

```

D: Implement a second format which allows respecification of the PLOT.COM default "colors" (patterns). This format is signified by the character "2" as the first character in the file on the first line. The following lines contain one CIF layer name and eight two-digit hex numbers separated by spaces. The hex numbers correspond to the PLOT.COM Upload Command as specified on page 11 of the PLOT DOCUMENTATION [21]. One layer is specified per line and lines that begin with ';' are ignored. For example:

```

2
; example pattern_file of format 2
NM 88 44 88 44 88 44 88 44
NC AA 00 AA 00 AA 00 AA 00
NP 00 01 00 02 00 03 00 04

```

An additional desired requirement would be to allow reading and interpretation of cifplot format pattern_files.

```

-F font_file
  (Font) The -F option indicates which font you
  want for your text. The file must be in the
  directory '/usr/lib/vfont'. The default font is
  Roman 6 point. Obviously, this option is only
  useful if you have text on your plot.

```

NR: Not applicable to mcifplot's operating environment or supported by PLOT.COM. Use will produce a fatal error.

```

-O filename
  (Output) After parsing the CIF files, store an
  equivalent but easy to parse CIF description in
  the specified file. This option removes the

```

contain an arbitrary number of layer descriptors. A layer descriptor is the layer name in double quotes, followed by 8 integers. Each integer specifies 32 bits where ones are black and zeroes are white. Thus the 8 integers specify a 32 by 8 bit stipple pattern. The integers may be in decimal, octal, or hex. Hex numbers start with '0x'; octal numbers start with '0'. The CIF syntax requires that layer names be made up of only uppercase letters and digits, and not longer than four characters. The following is example of a layer description for poly-silicon:

```
"NP" 0x08080808 0x04040404 0x02020202 0x01010101  
      0x80808080 0x40404040 0x20202020 0x10101010
```

R: This option is modified to be compatible with the features of PLOT.COM and the operating environment. The specified pattern_file must have a legal CPM file name. It will contain a slightly different format. When a pattern_file is specified, the default mcifplot layers and their corresponding PLOT.COM "colors" are erased and replaced with the new layers and colors. A maximum of 15 layers can be specified. CIF layer names and PLOT.COM colors must be legal for their respective environments, else fatal errors will be produced. Default PLOT.COM colors which closely correspond to the standard NMOS stipple patterns of cifplot will be built into mcifplot.

The pattern_file format is signified by the character "1" as the first character in the file on the first line. It is followed by a list of layer names and a corresponding PLOT.COM color with a decimal value from 0 to 127. One layer is specified per line and lines that begin with ';' are ignored. For example:

Errors. Figure III-2 decomposes the main program into these four processes.

The Interpret Commands section parses the command line and the CADRC file and modifies the Control Variables to include these new options. Initially all Control Variables are set to the Default Control Values which are the standard default actions for the program. The Control Variables control the action and function of the rest of the programs modules.

The Parse CIF File section reads and parses the input CIF File. The syntax and a portion of the semantics of the file is checked and appropriate errors are generated. In this section, a Symbol Table is constructed which contains information on bounds, scale, and location for each symbol in the CIF file. CIF commands not inside a symbol (Executable Commands) are saved as they are encountered in a temporary file called the Executable File. This saves the overhead of storing the location of each Executable Command for later interpretation when the output PLOT file is built.

The Build Plot File section reads this Executable File and generates the appropriate plotting commands which are written to the PLOT file. The PLOT file contains the elementary plot commands in the format required for reading and execution by PLOT.COM. The Executable File is read and when calls are encountered, the called symbol is located in the symbol table and traced as required. Symbols are read from disk (CIF file input) and the CIF commands from within the

symbol are used to generate additional plotting commands in the output file. This process continues until all commands in the Executable File are read and interpreted. The semantics of the commands are checked and Error Numbers are output as required.

The final major activity in mcifplot is the Print Errors section. It receives all Error Numbers, and prints the associated error message to the standard output. The error messages are contained in a special file called a Message File to save space in the main program.

These four sections and their function are further decomposed and described in the sections below. The SADT diagram that is associated with each section is included immediately following the description of the section and should be referred to while reading.

Interpret Commands (Figure III-3)

Set Control Variables is an initialization module which initially sets all the Control Variables to their default values.

The Get CADRC Options process opens and reads the CADRC File (there are no errors or additional actions if the file does not exist). The file is searched for a line that begins with the word "mcifplot". If found, the words following mcifplot until a carriage return are formed into CADRC Options and output. The CADRC options are identical in function and format to the options allowed in the Command

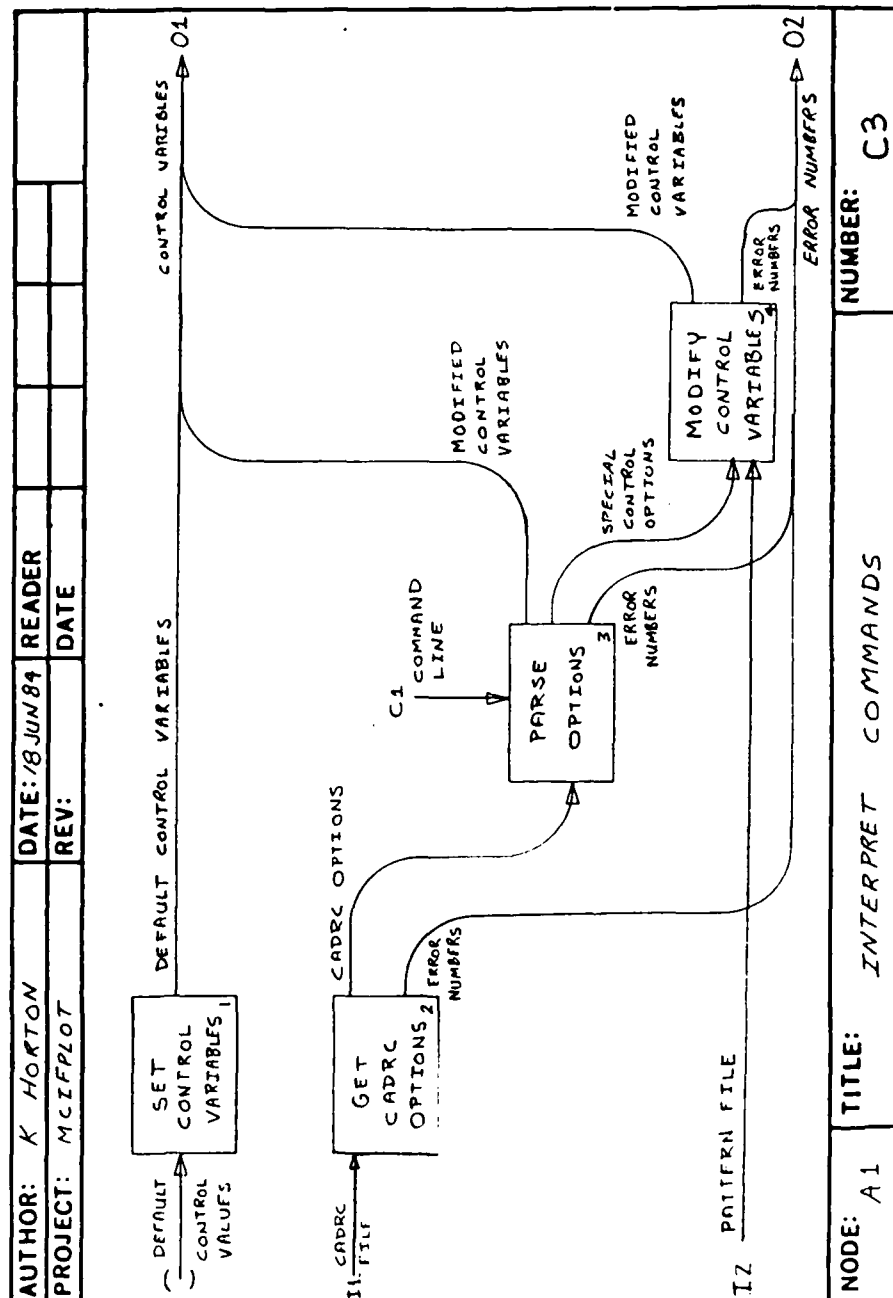


Figure III-3

Line with the exception of the specification of a CIF input file.

The Parse Options process reads and parses the CADRC Options and the Command Line and produces the Modified Control Variables and Special Control Options for output. This action takes place for each word of both inputs (or no action takes place if there are no options from either source). The Special Control Options are parameters which are passed to a group of modules called Modify Control Variables for special option processing. The syntax of the options are checked and fatal errors are generated as necessary.

The Modify Control Variables process accepts Special Control Options and produces a group of Modified Control Variables which are used to control the actions of other program sections. These special options include printing a banner and changing the default stipple patterns to those in an optional pattern file. The Command Line and CADRC Options control the function of the rest of the program through the Control Variables.

Parse CIF File (Figure III-4)

The Open File module opens the main input CIF File and reads it into an internal buffer. A table of data which contains information about the CIF File is output to the Get

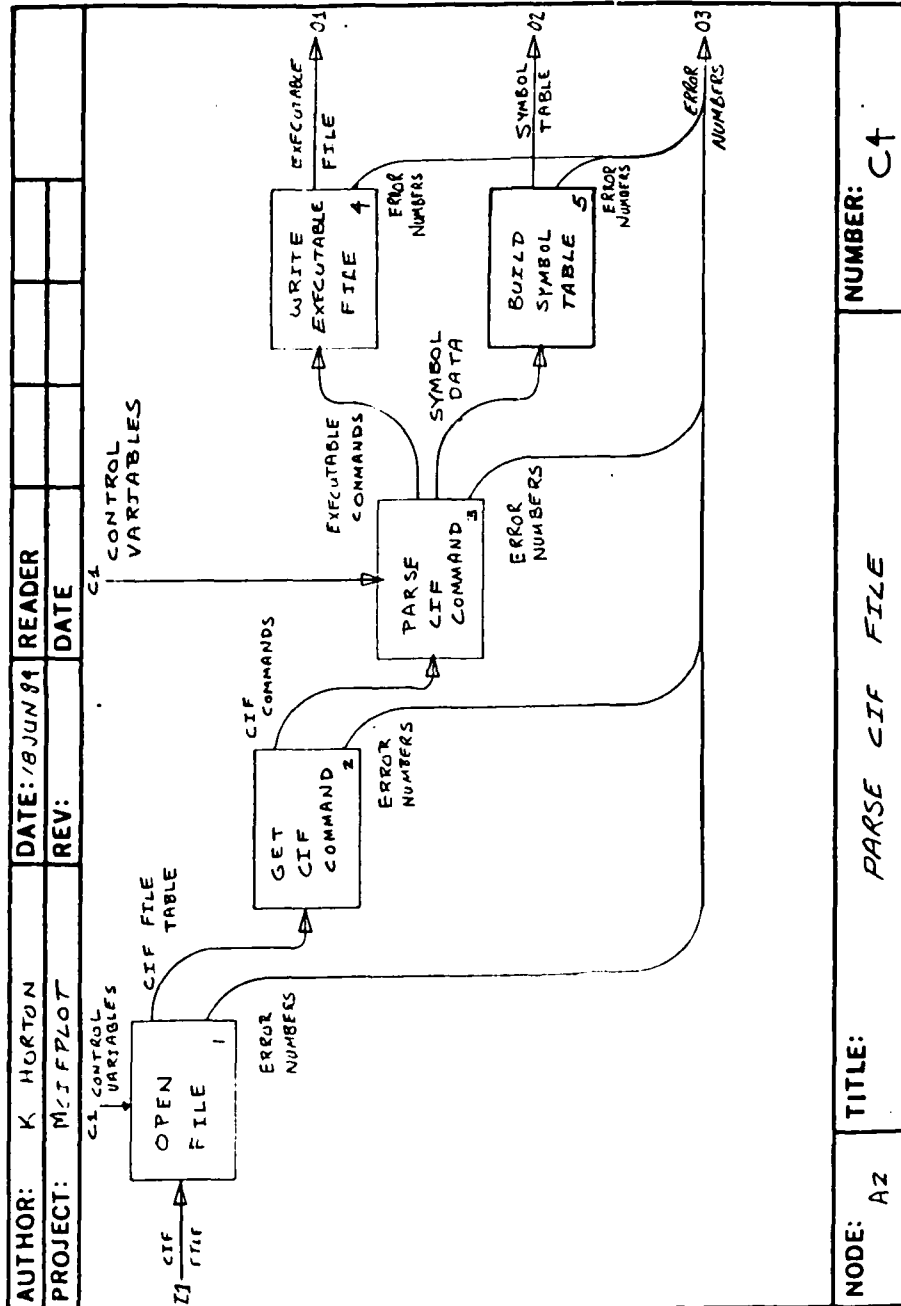


Figure III-4

Cif Command process. A fatal error is produced if the file does not exist.

The Get CIF Command process gets one CIF command from the internal buffer referenced by the CIF File Table and outputs it to the Parse Cif Command process. The commands are moved into a temporary buffer for processing one at a time. Comments are skipped and errors are produced for empty or incomplete commands.

The Parse Cif Command process takes each command and breaks it into tokens to check syntax. Each Executable Command is output to the Write Executable File process and Symbol Data is output to Build Symbol Table. All syntax and semantic errors produce error messages. The Symbol Data has several formats which are used to signify the beginning or end of a symbol, it's scale, and the bounding box of each CIF command within a symbol.

The Write Executable File process creates the EXECUTABLE File and writes all Executable Commands to this file. The CIF End Command should be the last command written to the file. A fatal error is generated if the file cannot be written or exceeds available disk space.

The Build Symbol Table process creates an internal Symbol Table which contains a list of all CIF symbols by number, their bounds, scale, type (whether the symbol contains Call commands or not), and their location on disk. This information is derived from the Symbol Data. The bounds recorded in the Symbol Table at this time are only accurate

for symbols with no internal Call commands. An additional trace of the Symbol Table is done by a later process to determine the bounds of symbols which contain calls. A fatal error is generated if the symbol table requires more memory than is available.

Build Plot File (Figure III-5)

The Scale Plot process is the controlling process of the Build Plot File section. It determines the CIF units which correspond to the maximum and minimum coordinates in the PLOT.COM coordinate system by tracing the Symbol Table completely from top to bottom. The bounding box of symbols which contain calls is modified during this trace. If the window option is selected then the Y dimension is scaled to fill the page left to right and the X dimension is scaled relative to this. The scale option takes precedence over the window option and forces a micron per inch scaling. A scale option value that will generate a plot too big for the page in the Y direction (i.e. across the page) will generate window values which truncate the plot in the Y dimension.

Once the size of the plot is determined, the user is queried whether to produce a plot. The question, window size, and length of the plot in the X direction is written to the standard output. A negative reply produces an internal Error Number which terminates the program. If more than one PLOT.COM page is required, this process initiates re-

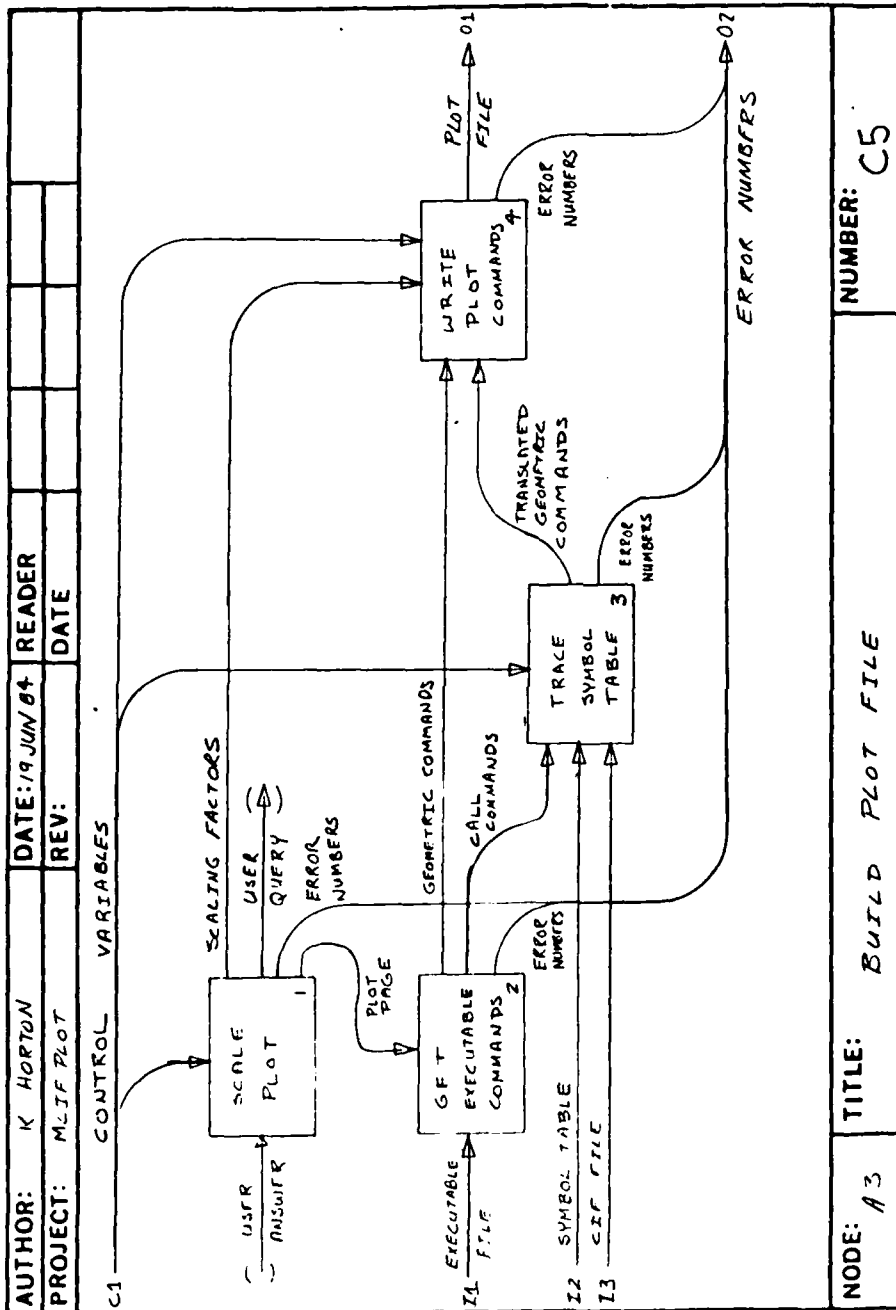


Figure III-5

peated reading and tracing of the Executable File using different sizes.

The Get Executable Commands section opens the Executable File and divides the commands into two categories. Call Commands are output to the Trace Symbol Table section. All other commands (Geometric Commands) are output to the Write Plot Commands Section.

The Trace Symbol Table section parses the Call Commands and finds the symbol in the Symbol Table referenced by the Call Command. If this symbol is already in a memory Symbol Buffer, then it is processed and the Translated Geometric Commands are output. The correct translation for each command is determined using a transformation matrix which contains all the transformations of higher level calls multiplied together. If the symbol is not in memory, the symbol is read into a Symbol Buffer (from the CIF file) and then processed. This sequence continues until all Call Commands from the Executable File are read and processed. Symbols which would fall outside the window of the plot are not processed.

The Write Plot Commands section clips the dimensions of the input commands using the scaling factors as a control. The input commands are translated into the required elementary plot commands with the correct dimensions for PLOT.COM and are written to the PLOT File.

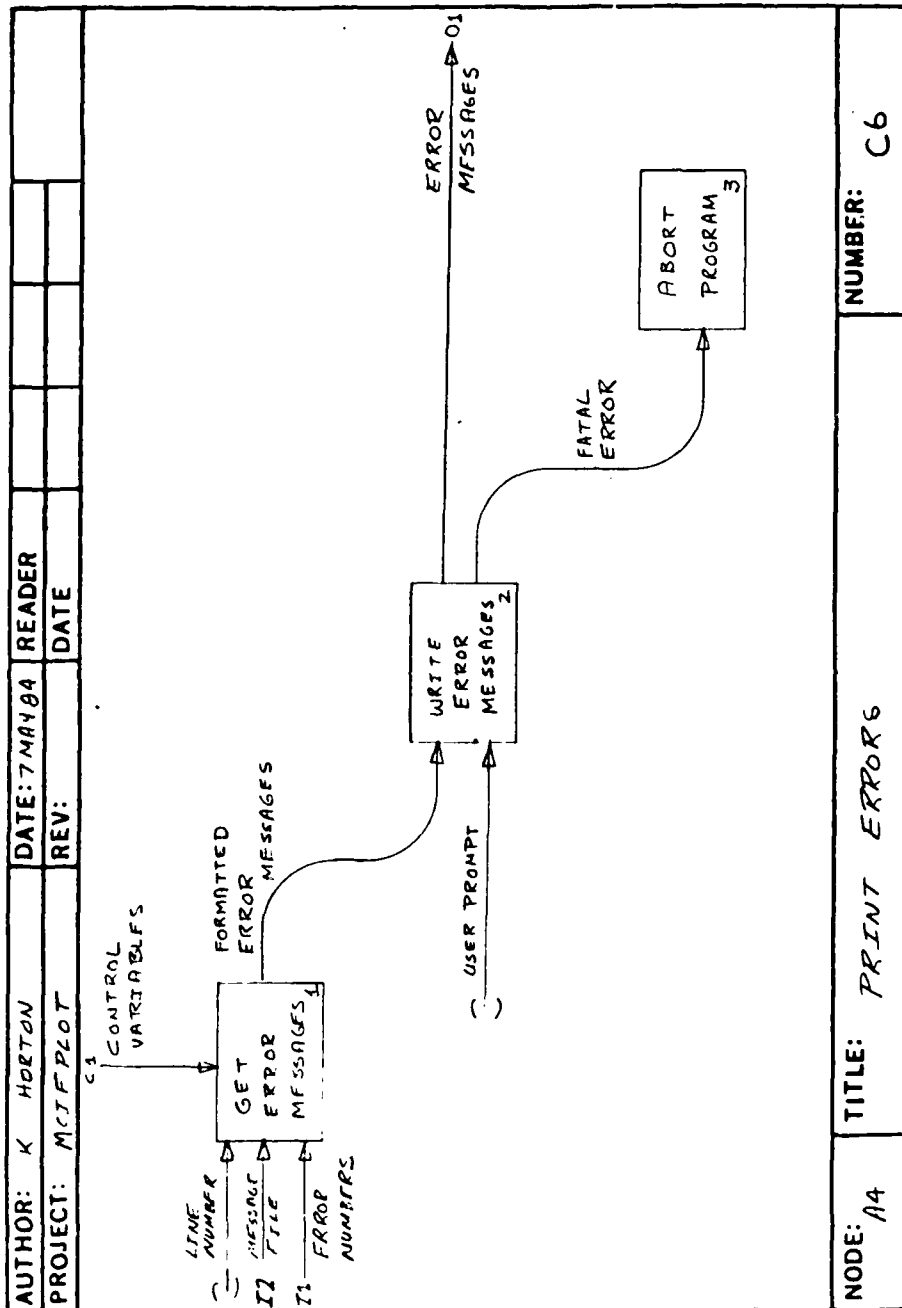


Figure III-6

Print Error Messages (Figure III-6)

The Get Error Messages process receives Error Numbers and reads the associated message from Message File. This message is then formatted and output. Fatal error messages always begin with the word "Fatal" for special processing by Write Error Messages.

The Write Error Messages section prints the input messages to the standard output until a screen of errors has been output. It then waits for a prompt from the user to continue. If the error is fatal, the Abort Program process is started. The Abort Program process erases all temporary files and stops program execution immediately.

IV. Detailed Design

The Detailed Design translates the general specification documented in the System Design into a specific, detailed plan for a particular implementation. There is a tendency during software development to move directly from the System Design into the actual coding of the program. With complex projects, this usually results in wasted time and effort when early mistakes and incorrect assumptions require correction and modification. The Detailed Design of mcifplot was written in very specific detail so that problems in the implementation phase would be kept to a minimum.

In fact, the goal of the Detailed Design is to make the implementation phase primarily a C syntax exercise, rather than a creative effort. Thus, the major time spent on implementation took place during the formulation of the Detailed Design. Chapter IV describes the design goals or philosophy, the design procedure, the format of the Detailed Design, and the design itself. This chapter is intended to introduce the major body of documentation of the Detailed Design which is contained in Appendixes C, D, E, and F. These appendixes contain a complete set of structure charts, procedure headers and code, and a detailed Data Dictionary of the data elements of the design.

Design Goals

The major design goals have, of course, been stated in detail in the requirements section of Chapter II. What follows is a summary of some of the specific goals for the implementation of the requirements into code. The general goals for the implementation are to use "good coding practices", develop loosely coupled modules with strong cohesion, and to use abstraction to enhance understandability and reduce complexity. In addition, because of the microcomputer environment, the efficiency of the implementation must be maximized. These goals all fall under the general category of good software engineering.

"Good coding practices" is a rather nebulous quality which encompasses a variety of different areas of software engineering and relates to programming style. Specifically, the code should be easy to understand and avoid "correct" but confusing structures or branches (very easy to do with C). Boundary conditions in looping constructs should be carefully analyzed and processed. Code should be developed which works correctly before time is spent making it faster or more compact. Variable names should be descriptive of the quantities they contain. In general, procedures should be used to perform any functions which are done repetitively.

The program procedures should have two important qualities. First, they should be loosely coupled with each other. This means that they are relatively independent of each other, and as much as possible, their interaction is through

the exchange of data. At the same time, the procedures should exhibit strong cohesion. This concerns the type of function that is accomplished within a procedure. The goal is to have the procedure perform one specific function as a "black box" with the method of performance unknown, or hidden, from the calling module.

Correct use of design abstraction results in procedures and functions which perform as "black boxes". The program should be broken down hierarchically into procedures which closely abstract the solution of the real world problem. Lower levels of procedures should define more specific details of the implementation of a problem's solution.

Finally, since memory space will be at a premium, the space efficiency of the program is very important. Whenever possible, procedures should implement a general purpose function so that they can be used at different times by more than one calling procedure. In addition, the space in the program used for storing messages, tables, and arrays should be minimized whenever possible. These goals, when implemented consistently, will produce code which is both easier to understand and easier to debug, maintain, and modify.

Design Procedure

As with the rest of the design process, the Detailed Design adds more detail to the previous level of design. In this case, the previous level of design is the System Design

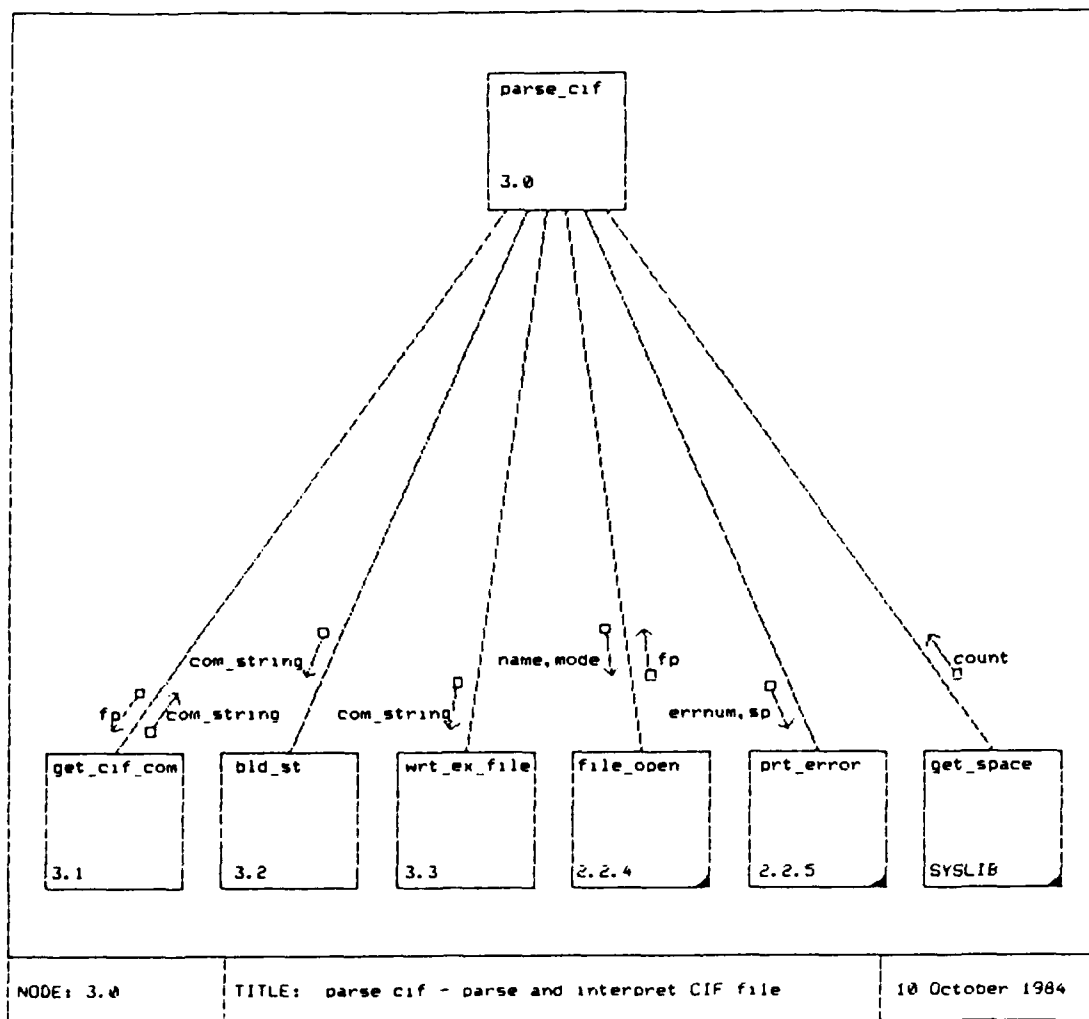


Figure IV-3

sponding error message on the standard out. The error messages are not a part of mcifplot's code and must be read in from the MESSAGE file on the default disk drive. If this file is not found, then just the error numbers are displayed. prt_error calls abort if fatal errors are found, or sets the global variable err_flag to TRUE if errors are found which should inhibit the final output of the PLOT file.

3.0 parse_cif. The parse_cif procedure is responsible for parsing and partially interpreting the input CIF file (see Figure IV-3). First, it sets the charcount global variable to contain the available space on the default drive using a call to the procedure get_space. This allows a graceful recovery from the problem of filling the disk while trying to write the EXECUTABLE file. The next step is to call file_open to open and initialize the EXECUTABLE file for output. As the CIF file is processed, all commands which are not part of a symbol definition are written to this file.

The CIF file is processed with a loop which repeatedly calls get_cif_com to get one CIF command from the input CIF file. If this command is not a DS command (used to signify the start of a symbol definition) then wrt_ex_file is called to check the syntax of the command and write it to the temporary EXECUTABLE file. Otherwise, bld_st is called to check the syntax of the entire symbol and add an entry to the Symbol Table.

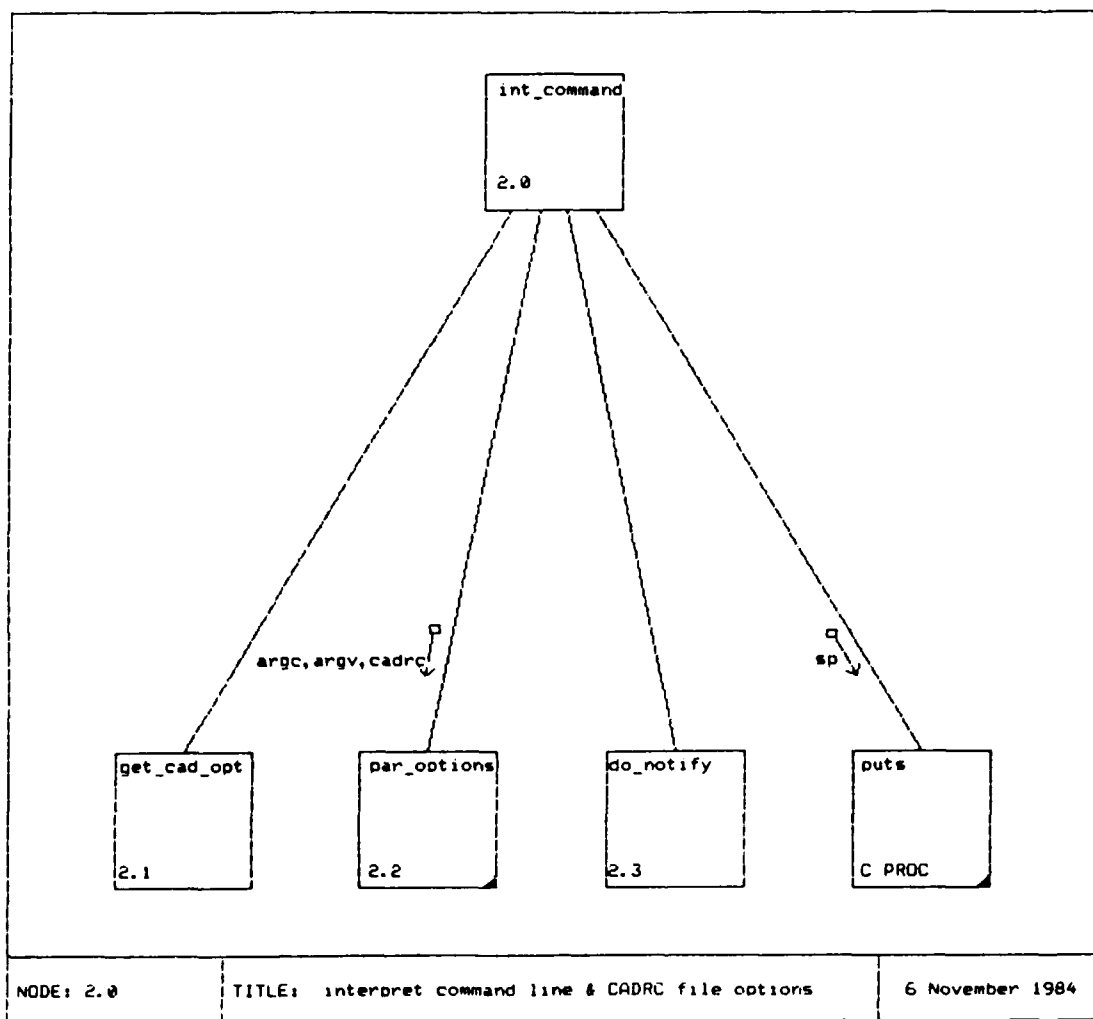


Figure IV-2

line and the CADRC file. The procedure calls `get_cad_opt` to read and parse the `mcifplot` line in the CADRC file (if it exists). `get_cad_opt` also calls `par_options` to do the actual parsing and modification of the Control Variables found in the CADRC file.

One of the most complex procedures in `mcifplot` is `par_options` which parses the options in the command line and the CADRC file. It includes a large case statement with entries for each of the accepted program options. If the options are input correctly, then the state of the appropriate Control Variable is modified. For example if the '-i' option is detected in the command line, then the control variable `interactive` is set to FALSE and the user is not queried to see if a plot is desired or not. This procedure also calls `get_pattern` if required by the '-p' option to read and process new layers from a PATTERN file.

The procedure `do_notify` is called `int_command` if the notify ('-n') option is detected in the command line or CADRC file. This procedure displays the status of the Control Variables on the standard output as well as any new input values such as a window or banner value. `puts` is called by `int_command` to display the `mcifplot` version on the standard output when the command line and CADRC file has been processed without error.

A lower level procedure in `int_command` that is used throughout the `mcifplot` is `prt_error`. This procedure accepts an error number and string as input and displays the corre-

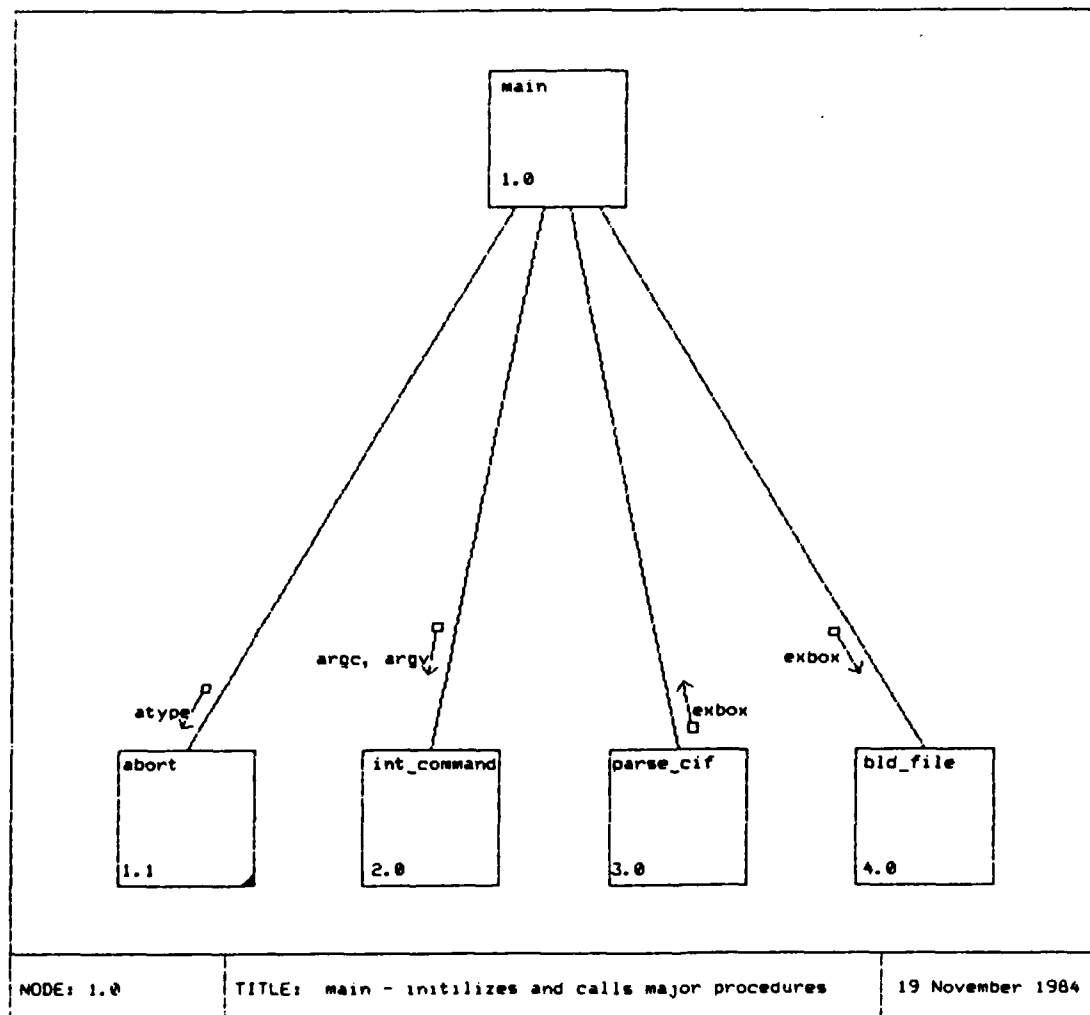


Figure IV-1

Procedure Descriptions

Figure IV-1 is the top-level structure chart of the mcifplot. The main procedure is only responsible for setting the default values of several global variables and calling the three primary sub-procedures. The abort procedure is included at this level in case there are syntax errors in the input CIF file that prevent the output PLOT file from being written correctly. If the exbox variable returned from parse_cif is equal to NILL (0), then abort is called to abort the program. abort gracefully exits, closing all open files, erasing temporary files, and displaying an exit message on the standard output. abort also displays program diagnostics on the standard output if the notify variable is set TRUE.

The three main sub-procedures are numbered in a slightly unusual manner (2.0, 3.0, 4.0 rather than 1.2, 1.3, 1.4) primarily to prevent lengthy procedure numbers near the bottom of the hierarchical structure. The standard C command line variables argv and argc are passed to the int_command procedure for processing. The parse_cif procedure returns a pointer to a box structure (exbox) which contains the CIF bounds of the executable commands which are contained in the EXECUTABLE file. If this is not NILL, it is passed to the bld_file procedure to use as a starting value to determine the true bounds of the plot.

2.0 int_command. int_command is decomposed in Figure IV-2 and is responsible for reading and parsing the command

The most complex structure is the **symbol** structure which contains necessary descriptive information about the CIF symbols found in the input CIF file. This includes the symbol number, the symbol bounding box, the symbol scale, and the offset and record used to determine the symbol's location in the CIF file on disk. The **symbol** structure has the following format:

```
struct symbol
{
    long    number;
    int     offset;
    int     record;
    float   scale;
    long    sxmin;
    long    symin;
    long    sxmax;
    long    symax;
    char    calls;
    char    inuse;
    struct symbol *next;
};
```

The scale value represents the division of two integer values and is stored as a float to maintain the accuracy of values that are less than one. The offset and record are in the format required by the C80 **seek** procedure. The calls variable is used as a boolean and is TRUE if the symbol contains any Call Commands. The inuse variable is YES when a symbol is being traced and is used to detect recursive symbol calls. The Symbol Table is constructed as a linked list and the next variable points to the next symbol in the list. More detailed justification of the format of the **symbol** structure is contained in the Data Dictionary of Appendix D.

There are also a small group of global variables associated with the entire program and with the `parse_cif` and `bld_file` procedures. Their purpose is to allow many procedures to access a single variable such as `line_number` and to allow an upper level procedure to initialize a variable that should be set correctly the first time certain lower level procedures are called.

Structures. The purposes of structures in the program are to provide a way to pass multiple variables between procedures and to simplify the access and manipulation of groups of variables. It should be noted that structures cannot be passed between procedures, and pointers to structures must be used to pass structure information.

The idea of a box (four long integers which represent the the lower left and upper right corners of a box) and a point (two long integers to represent a point in two-dimensional space) are integral to the CIF geometric description. The `box` and `point` structures represent these two data items throughout the design.

The `cliprec` structure (four integers which represent the status of four sides of a box) is used to record the results of clipping a box to the plot window. The values (TRUE or FALSE) in a `cliprec` structure determine whether or not the outline for a box side is drawn (the outline for a box side that has been clipped is not drawn). The `file` structure represents data about a file and contains elements for it's name and mode.

Detailed Design

The Detailed Design follows the basic form of the System Design with some changes made to better organize the control of the major modules. First, the program variables and structures are discussed. Then the important top level procedures are described and presented with the appropriate structure charts. This description is intended to be a general overview of important procedures. A complete set of structure charts is included in Appendix C along with the procedure headers which describe the function of each procedure. The major algorithms used in the program are also discussed.

Global Variables. It is obvious from the Structure Charts that the data interchange between procedures frequently uses global variables. In general, this is not considered good software engineering and results in tight coupling between procedures. There were several functional types of global variables used in the program.

One type has been given the name Control Variables. These are concerned with the actions and function of the program and are set and modified by the command line or CADRC file. The alternative method for access to this group of data items was to create a C structure and pass a pointer to it. In most cases this resulted in similar coupling to that of the group of global variables and it added additional complexity.

```

/*****
* DATE: date of last modification
* VERSION: number of version in form 1.0, 1.1 etc.
* NAME: name of procedure
* SCNUMBER: Structure Chart number of procedure
* FUNCTION: description of what the program does
* INPUTS: data passed to procedure
* OUTPUTS: data passed from procedure
* GLOBAL VARIABLES USED: global variables used by this
* procedure but not changed
* GLOBAL VARIABLES CHANGED: global variables changed by
* the procedure
* FILES READ: files read by this procedure
* FILES WRITTEN: files written by this procedure
* MODULES CALLED: modules called directly by this procedure
* CALLING MODULES: modules which call this procedure
* HISTORY: history of major modifications
*****/

```

In addition to providing part of the Detailed Design's documentation, the headers will be used with each procedure during implementation.

Data Dictionary. The final form of documentation for the Detailed Design is the Data Dictionary (Appendix D). It contains four types of data items and is different in format from the Data Dictionary prepared for the System Design. All constants used in the program are listed and described. The origin of the constant and the method of calculation (if the constant was derived) is included in this section. There is also a description and justification for each C structure used in the design and a listing of the form required for implementation in C. Next, each file used by the program is described. Finally, each variable which is global or passed between procedures is described (including the Control Variables). These data items are arranged in alphabetical order.

of their own, were developed independently of mcifplot and are used by calls within mcifplot to write all plot commands for PLOT.COM into the PLOT file. Headers and the code required for implementation of both the C and PC procedures is included in Appendix E.

The final type of procedure is labeled "SYSLIB". This procedure implements the most system specific type of function: computing available disk space. This procedure does not contain C code, but contains inline assembly code taken directly from Rick Conn's assembly language library SYSLIB3. It provides an easy to use, and efficient means of implementing a system directory function which is required by mcifplot. The major emphasis of the actual design description in Chapter IV is the presentation and "walk-through" of the program Structure Charts.

Procedure Headers. The procedure headers are standard program headers which usually appear before modules or procedures in a program. These headers provide detailed information about each procedure in the Structure Charts and are listed in alphabetical order in Appendix C. The headers contain the following information about each procedure:

procedure and how they communicate. In general, the charts are drawn in standard Structure Chart format. Vectors indicate control relationships between procedures. The name of each procedure is contained in a box and small arrows are used to show data flow. A triangle in the lower-left corner of a procedure's box indicates that it is called by more than one parent procedure. Each procedure is broken down until the lowest level contains procedures which call no others.

There are basically five "types" of procedures shown in the SC's. The procedures which were designed specifically for mcifplot are numbered hierarchically starting with the number 1.0 for **main**. Each C80 library procedure which is used is indicated by the the heading "C80 PROC" in the procedure's box. These procedures are not broken down further or documented with headers. Additional information on them is contained in the C80 Manual[1] and a short summary of their function and inputs/outputs is included in Appendix F.

There are also a number of procedures labeled "C PROC". These procedures are "standard" C procedures which are either taken directly from The C Programming Language[10] or are slight modifications to C80 procedures. For example **latoi** (long ascii to integer) was created by changing the type of the output variable in the C80 procedure **atoi**.

The fourth type of procedure is labeled "PC PROC" (Plot Command Procedure). These procedures, which form a library

tion of variables at the beginning of the main procedure.

In general, an entire process was designed before proceeding to the next. Thus, Interpret Commands was designed completely before proceeding to the Parse CIF process. As the design developed, the structured English included in the header became closer and closer to actual C code. If a better method of implementing a function was discovered, the previously designed procedures were changed accordingly. For example, several global variables were converted to local variables by passing pointers to them between the necessary procedures.

Design Format

The final format of the Detailed Design of mcifplot uses three types of documentation which are contained in Appendixes C-F. First, Structure Charts are used to show each procedure designed for the program, the hierarchy of the design, and the data communication that takes place between procedures. The procedure headers contain detailed information about each procedure and are included with the C code which implements the procedure's function. Finally, the Data Dictionary documents all global and passed variables, C structures, constants, and all input, output, and temporary files. These three documentation types are explained in more detail below.

Structure Charts. The Structure Charts begin with the top level procedure of the program (main) and show each sub-

as described by SADT. Because this was the author's first experience with the implementation of a large scale software project, the entire design procedure was somewhat experimental and a learning process at each step. The design procedure is described before the design format, because some of the intermediate steps of the procedure are no longer a part of the finished design.

The first step in the design process was to begin translating the processes of the SADT into descriptions of specific procedures to be implemented in C. These descriptions were documented in the code headers which are described below. In most cases, SADT processes mapped directly into corresponding C procedure descriptions. Of course, since the SADT description was at very general level, these C procedure descriptions were just the start of a hierarchical structure of C procedures which performed the required high level function. Included in the procedure header was a structured English/C pseudocode section which described in specific detail the method or algorithm of performing the desired function.

In parallel with the construction of the procedure headers, a new Data Dictionary was developed, and structure charts were drawn to illustrate the hierarchical nature of the design and the interaction between the different procedures. Some SADT processes were changed in scope during the translation process. For example, Set Control Variables did not map into a procedure but was accomplished by initializa-

Both `wrt_ex_file` and `bld_st` use a large case statement with an entry for each type of CIF command. The syntax of each command is checked with a call to a specific lower level procedures like `ck_box` or `ck_call`. If errors are found, an error number is passed to `prt_error`. The checking procedures also calculate the bounds of the the geometric commands (Box, Wire, Roundflash) and return this value to the calling procedure.

In `wrt_ex_file`, all commands with correct syntax are written to the EXECUTABLE file. The bounds of the geometric commands are used to modify the value of the variable `exbox` so that it represents the bounds of all geometric EXECUTABLE commands. `Exbox` is later passed to `bld_file`. A trace of the CIF file to determine the plot bounds is only required if there are symbols in the file.

`bld_st` does a similar process of checking the syntax of all commands in a symbol definition by repeated calls to `get_cif_com` and the checking procedures until a CIF DF command is found. A new Symbol Table entry is created for the symbol being processed. If a Symbol Table entry already exists for a symbol's number, then the old symbol number is set to zero, a warning is displayed on the standard output, and the new symbol is added to the Symbol Table (symbol redefinition).

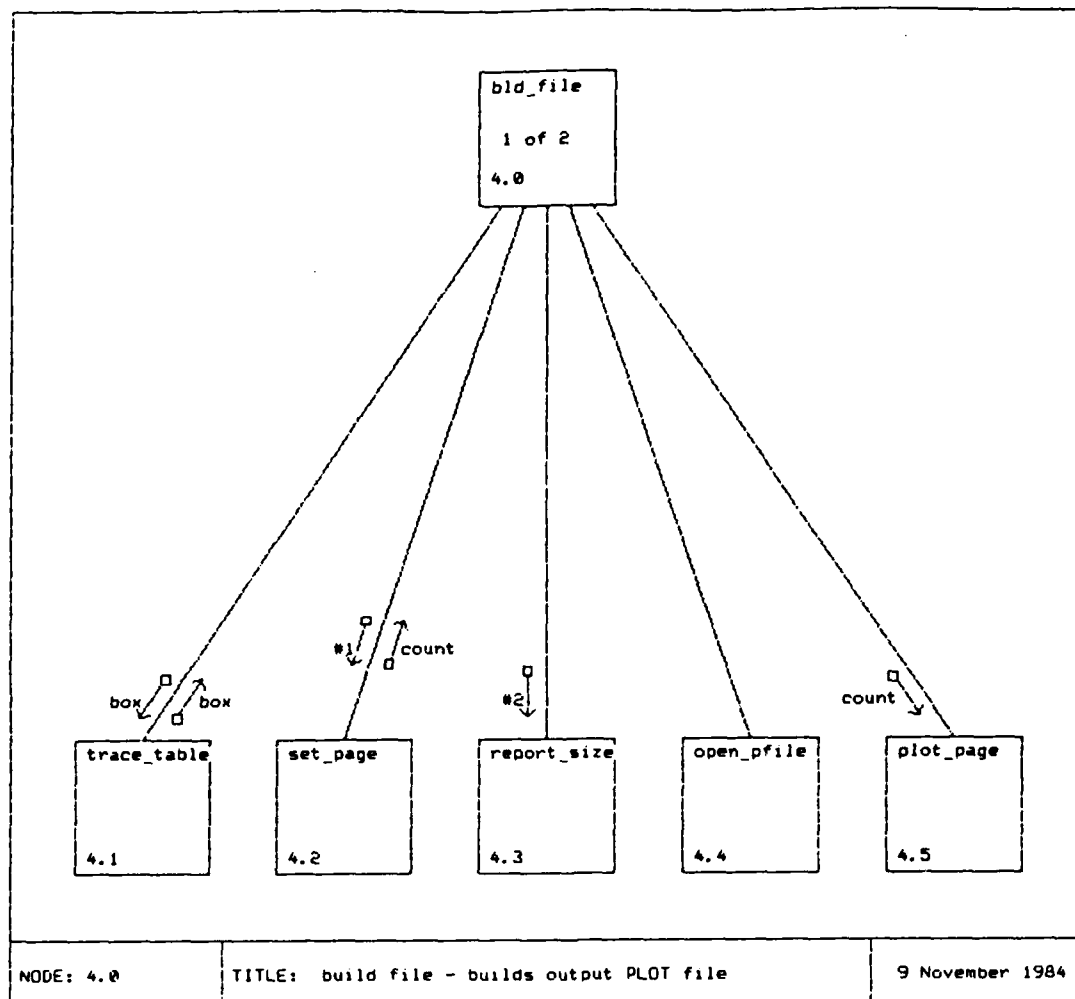
After the bounds of the all geometric commands in the symbol are found, the symbol scale value is used to convert these bound numbers to the actual bounds of the symbol which

are then stored in the Symbol Table entry for the symbol. The location on disk of the start of the symbol definition is stored in the offset and record variable of the symbol entry to allow random access to this symbol by the `bld_file` procedure (for finding true bounds and building the PLOT file). If there are Call Commands in the symbol definition, then the calls variable is set to YES. This indicates that the symbol bounds may require modification if the bounds of the called symbol exceeds the bounds of the other geometric commands in the symbol. Inuse is set to NO for each symbol.

4.0 bld_file. `bld_file` and its related sub-procedures have the largest and most complex function of the major procedures and are responsible for writing the output PLOT file for plotting by PLOT.COM (see Figures IV-4 and IV-5). The algorithms in the sub-procedures are derived from two major sources: 1) chapter seven of A Guide to LSI Implementation[8], and 2) chapter four of Principles of Interactive Computer Graphics[15].

`bld_file` first allocates a "dummy" buffer of one character to determine the top of available memory. The memory location returned from `sbrk` for the dummy buffer is offset upward by an additional "safety zone" and stored in the variable `lastalloc`. This value will be used in tracedown to insure that the program stack does not "grow" downward and overwrite the Symbol Table.

Next `trace_table` is called to determine the actual bounds of the plot. `trace_table` must read in each command



#1: box, &realxmax
 #2: box, realxmax

Figure IV-4

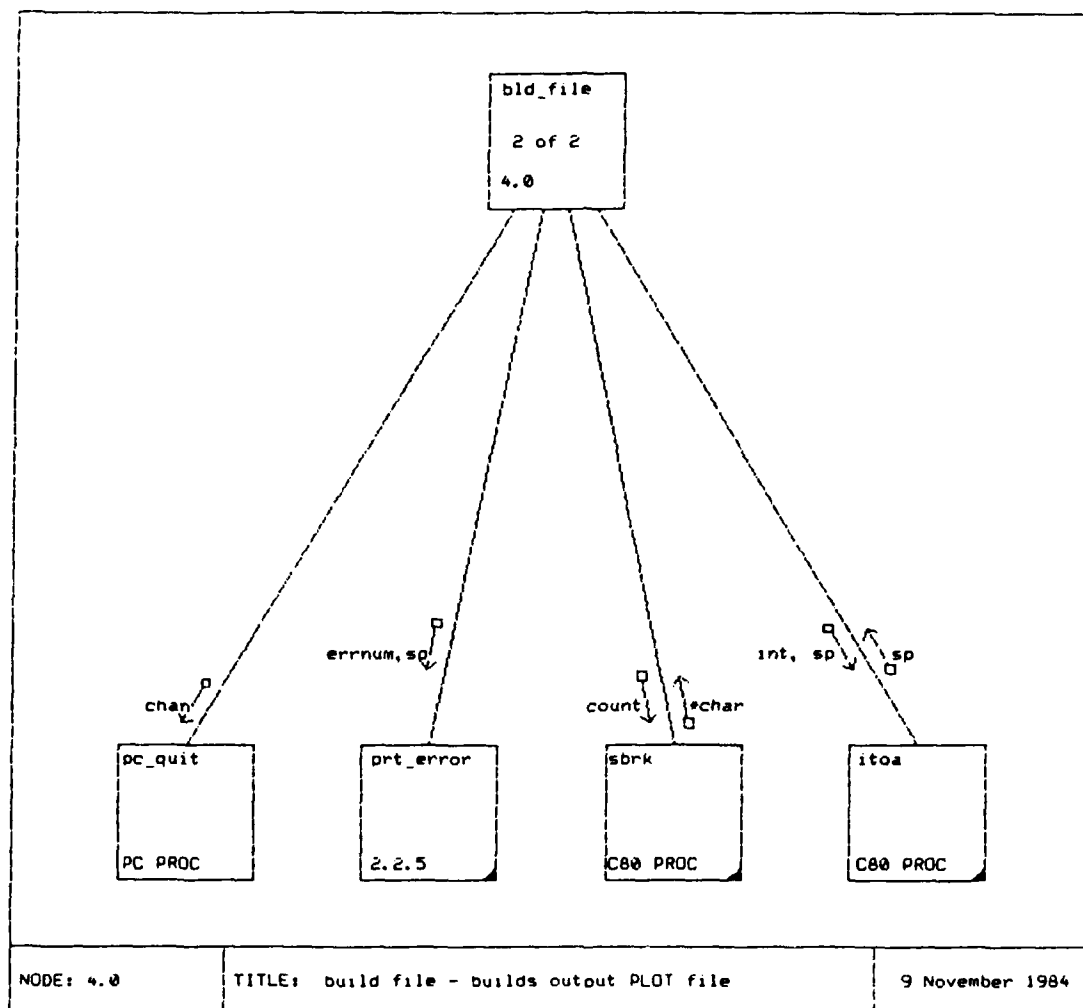


Figure IV-5

from the EXECUTABLE file and if calls are encountered, use these calls to determine the bounds of the called symbol. Many factors must be considered in this process. The symbol referred to in the Call Command is located in the Symbol Table. If this symbol's calls entry in the symbol table is NO, then the bounds found by `parse_cif` are correct and no further tracing is required of this symbol. In this case, the transformation specified in the input Call Command is applied to the bounds of the symbol and returned.

If the calls entry is YES, then two actions must take place. First, as above, the actual bounds of this symbol must be transformed and returned to the calling process. Second, to find these bounds, the called symbol is located in the Symbol Table and a random access is done to the location of the symbol in the CIF file. Commands are then read from the CIF file until a Call Command is found. Each time that a Call Command is found, `tracedown` is called recursively to trace and find the bounds of this called symbol. The value returned will modify the bounds of the current symbol if the bounds of the called symbol are outside of the current symbol bounds. When all calls have been traced, the calling symbol bounds in the Symbol Table are set to the new values, the calls variable is set to NO (one trace is all that is required), and the symbol bounds are transformed as above and returned.

The purpose of the process described above is to finish the trace with the Symbol Table containing the true bounds

of each symbol entry. This means symbols need only be traced once for bounds and during plotting, symbols do not have to be called from disk and traced if they lie outside of the current window. If the Control Variable rotate is TRUE, the final plot bounds are rotated 90 degrees before they are returned to `bld_file`.

To prevent the stack from overflowing, the value of the current depth is compared to the value of the Control Variable depth before each symbol call is traced. Calls nested too deep are ignored and a warning issued. All local variables are declared type auto so that a new set of local variables is generated with each call to `tracedown`. If the location of the last local auto variable allocated is less than `lastalloc`, then the stack is within the safety zone above the Symbol Table and all new calls below this level are ignored and a warning message is issued. If a loop is found (a symbol calling an ancestor), then a fatal error is generated.

When the true plot bounds are found, `bld_file` passes them to the procedure `set_page` to determine the size of each page in CIF units and the number of pages required for the plot. `set_page` also sets the value of the viewport global variable and modifies the input box to reflect the correct CIF bounds for one page. Three factors are considered in these calculations: the optionally specified scale of the plot, the optionally specified window of the plot, and the true bounding box of the plot. The viewport variable is set

using the formula from Newman[15], page 75. Since the PLOT.COM window is always square, one value will do for both the x and y conversion to the viewport. The basic equation for the window conversion is

$$xscreen = \frac{Vpxr - Vpxl}{Wdxr - Wdxl} (Xw - Wdxl) + Vpxl \quad (1)$$

The viewport variable is equal to the result of the division and is a constant for each page for both x and y. The viewport xr and xl are equal to the PLOT.COM values of 1.0 and 0.0 respectively. Eq (1) now reduces to

$$xscreen = viewport * (Xw - Wdxl) \quad (2)$$

A transformation is done on each point before it is output to the PLOT file to translate it from CIF units to the rotated and translated axes of PLOT.COM. This always moves Wdxl to 0. Thus, the windowing transformation reduces to a single multiplication.

bld_file next calls **report_size**, which writes the size of the plot in feet to the standard output and (if interactive is TRUE) asks the user if a plot is desired. If the answer is "y", then **open_pfile** is called to open the output PLOT file and set the global variable charcount with the available disk space. This value is tested before each write to the PLOT file to allow for a graceful exit if the available disk space is exceeded.

The output plot file is created with repeated calls to `plot_page`, once for every page to be plotted (as determined by `set_page`). All plot commands are written to the PLOT file by executing calls to the PC procedures such as `pc_color` and `pc_draw`. If the page number passed to `plot_page` is one, then the page header which consists of the name of the input CIF file, the window, and the scale of the plot is written to the PLOT file (call to `pc_text`). Plot commands to clear memory to the color white (calls to `pc_color` and `pc_erase`) are then written to the PLOT file. The major work of plotting one page is done with a call to `do_page`. `do_page` is passed a matrix which is the top level transformation for transforming points for output to the PLOT file. If `rotate` is TRUE, a matrix which represents a 90 degree rotation is passed. Otherwise, the identity matrix is passed to `do_page`. The final action of `plot_page` is to write an output command to the PLOT file (call to `pc_output`).

`do_page` first opens the EXECUTABLE file and reads in commands one at a time. `do_page` contains a large case statement with an entry for each type of CIF command. All commands which are not Call Commands are processed immediately. For example, a Layer Command results in a call to `get_layer`, which interprets the Layer Command and returns a PLOT.COM color value to be used for the output of geometric commands. Or if a Box Command is found, `do_box` is called with the Box Command string, the current color, the input transformation, and the scale (since this is the EXECUTABLE file, the scale

is set to 1.0). `do_box` does the following:

- 1) determines the bounds of the box from the Box Command string (call to `box_bounds`).
- 2) modifies these bounds with the input transformation and scale (call to `trans_box`).
- 3) clips the box to the current page window (call to `clip_box`).
- 4) transforms the clipped box to the PLOT.COM coordinate system (call to `trantoplot`).
- 5) writes the transformed box to the PLOT file with a fill and the required outlines (call to `put_box`).

If a Call Command is found, `do_page` calls `proc_call` to process the Call Command. `proc_call` also contains a large case statement like `do_page`. However, its local variables are declared type automatic so that it can be called recursively if additional Call Commands are found in the called symbol. The transformation specified in the input Call Command is used to modify the input (or current) transformation that was passed to `proc_call` from the calling procedure (either `proc_call` or `do_page`). This new transformation matrix is first applied to the symbol bounding box of the called symbol. The transformed bounds are tested to see if they are in the current page window (with a call to `clip_box`). If not, then it is not necessary to access the CIF file on disk and `proc_call` returns. Otherwise, the commands in the symbol are read in from the CIF file and processed until a DF Command is found.

If another Call Command is found in the symbol, then the current location in the CIF file is saved and a recursive call is made to `proc_call`. When `proc_call` returns, this position in the file is restored by a random access in the

CIF file and processing of commands continues as before. The current transformation, location, scale, and color are all stored automatically in the automatic local variables of `proc_call`. As in `trace_table`, the current depth is checked to prevent stack overflow.

`bld_file` continues to call `plot_page` until all pages of the plot have been written to the PLOT file. `pc_quit` is called to write the PLOT.COM Quit command to the PLOT file and control then returns to `main`.

V. Analysis

The Analysis section of the thesis is an opportunity for the author to critique his own work. This critique should not only point out the strong points of the design and product, but should also highlight weaknesses and suggestions for improvement.

This analysis will cover several areas of the program and the program's design. The following areas will be analyzed:

- 1) quality of the program design vs. design goals
- 2) program function vs. functional requirements
- 3) program performance vs. performance requirements
- 4) current status of mcifplot

mcifplot Design

The program design is specific and contains information about all aspects of mcifplot. The design hierarchy and intra-procedure communication is documented in a comprehensive set of structure charts. The passed and global variables, as well as constants, files, and structures are well documented in the Data Dictionary. The C code itself includes detailed headers which describe individual procedure I/O, access to variables, intra-procedure communication and the last modification date.

The code itself seems to be well thought out and understandable. However, because of the time pressure under which

the program was developed, it is likely that many segments could be modified for increased efficiency in space and speed of execution. For example:

- 1) long integers are used for storage of symbol bounds up until the bounding box is output to the PLOT file. Several long to float conversions take place at different stages in the program. It may be more efficient to convert to floating point numbers earlier in the program.

- 2) There are several different storage areas for command strings and arrays that are used at different times in the program. These could be combined into a global string area with an expected savings in space. It may also be possible to allocate more of the program storage areas at runtime to save space in the executable file.

In general, names for variables and procedures are descriptive and appropriate for the required function. However, there are some poorly named or organized procedures. This is especially true in procedures whose purpose was changed during program development. For example, the code which displays status information (number of lines parsed, etc.) on the standard output is scattered throughout many different, unrelated procedures. This function should be located in a single procedure.

The program is designed to minimize the size of the executable file. Procedures are frequently used more than once, rather than repeating a similar function among many slightly different procedures. For example, `put_box` is used

by `do_box`, `do_wire`, and `do_round` to write the required geometric figure to the PLOT file. The use of error numbers and an external MESSAGE file saves almost 6K of space in the executable file and centralizes the processing of errors.

mcifplot Function

mcifplot implements all of the functional requirements defined in Chapter II. This includes the following functions:

1) Reads, parses, and generates an output PLOT file for a single input CIF file. Accepts all CIF 2.0 Commands with the exception of Poly and Delete Definition Commands. The commands must define symbols and geometric elements in Manhattan directions. Additionally accepts User Extensions 1, 9, and 94.

2) Error messages are generated for syntax and semantic errors in the CIF file. Additional error messages are generated for errors in the command line and CADRC file as well as for environment specific errors (out of disk space, illegal file names, etc.).

3) Accepts and implements the following cifplot options:

- b (banner)
- c (comments)
- d (depth)
- g (grid)
- i (interactive)
- l (layer) does not include "text" keyword
- p (pattern)
- r (rotate)

- s (scale)
- w (window)

4) The output PLOT file produces a check plot with a header in the cifplot format. Multiple page check plots are also generated automatically.

5) Accepts and implements additional options unique to mcifplot:

- o (output file)
- m (make page)
- n (notify)
- q (quiet)
- t (transfer)

These functions were verified by extensive testing of mcifplot. All syntax and semantic error messages were exercised. Each option was exercised alone and in conjunction with other options. Over 100 check plots were printed to test different aspects of the programs function (samples are included in Appendix G). As much as possible, the same CIF file was plotted using cifplot to compare the results of the two programs.

mcifplot Performance

mcifplot meets the performance requirements defined in Chapter II. It was discovered that the execution speed of the program depends to a great extent on the format and type of the input CIF file. Table V-1 documents the program execution speed of four different CIF files. These files were chosen to represent four different types of CIF struc-

AD-A151 715

A MICROCOMPUTER-BASED PROGRAM FOR PRINTING CHECK PLOTS
OF INTEGRATED CIRC. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. K S HORTON

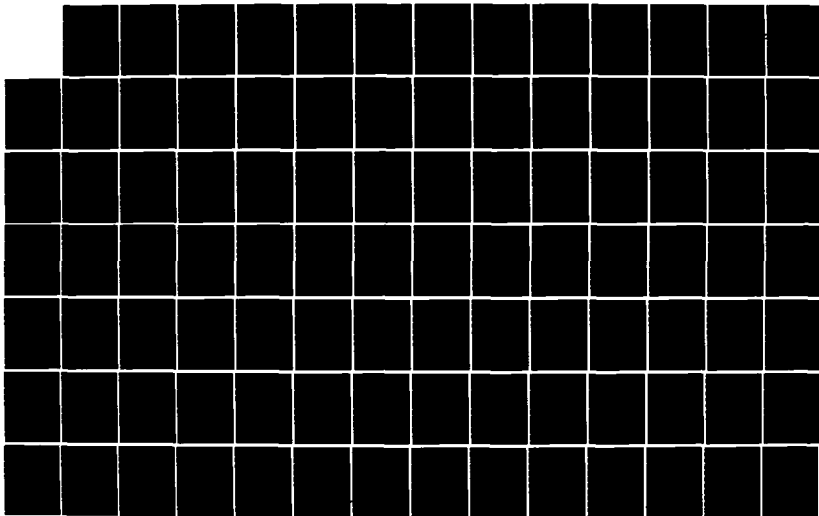
2/4

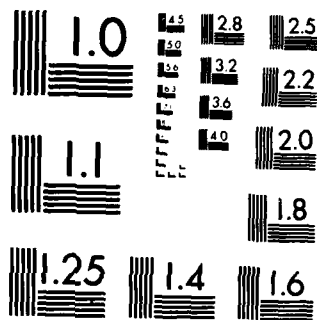
UNCLASSIFIED

DEC 84 AFIT/GE/ENG/84D-35

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ture and illustrate widely varying execution times for both mcifplot and PLOT.COM.

The program was executed on a CPM based computer with a four Mhz Z-80 microprocessor. The times are rounded to the nearest second and are all in the format minutes:seconds. The file sizes are for a disk with a 2K block size. The execution times varied slightly depending on the location of mcifplot and the input CIF file on disk.

Table V-1. mcifplot Performance

CIF File	A	B	C	D
File Size (K)	2	4	30	30
Output PLOT File Size (K)	2	128	92	186
Symbols	1	6	0	9
Box Commands	14	117	1822	453
Wire Commands	0	0	0	3
Layer Commands	7	66	5	235
Call Commands	1	13	0	840
Extension Commands	7	13	0	0
Total Commands	32	227	1830	1572
Parse CIF File (min:sec)	:17	:23	3:29	1:39
Find Plot Bounds	:01	:04	:16	1:55
Write PLOT file	:09	9:43	6:14	24:36
mcifplot total	:27	10:10	9:59	28:10
PLOT.COM computation (min:sec)	9:08	5:20	3:40	3:22
PLOT.COM printing	7:35	3:20	3:40	1:36
PLOT.COM total	16:43	8:40	7:20	4:58
Grand Total (min:sec)	17:10	18:50	17:19	33:08

It is obvious from the variety of execution times listed in Table V-1 that the contents of the CIF file are the determining factor in execution speed. It should also be noted that mcifplot itself requires several seconds to load

into memory before execution even begins. Each of the above files is discussed as it relates to the performance of mcifplot. Copies of each check plot along with a comparison check plot produced by cifplot is included in Appendix G.

File A is a short, one symbol CIF file which illustrates the NMOS stipple patterns built into mcifplot. mcifplot creates the required PLOT file in less than a minute. The total time required to obtain a check plot is over 17 minutes. PLOT.COM requires large amounts of processing time to do stipple fills in proportion to the size of the area filled. In this case, the entire PLOT.COM page is filled with stipple patterns and most areas are filled with more than one pattern. Thus, a fast execution time by mcifplot does not insure a fast check plot generation time.

File B is a small CIF file which describes a four word memory (designed by Brian Kelley). The output PLOT file requires 124K. This is a good example of how a small CIF file with a hierarchical design (a single memory cell is repeated to form a four bit word which is also repeated four times) can require a very large output PLOT file. It is clear that the size of the input CIF file gives very little indication of the final size of the PLOT file.

File C is an example of a large flat (no symbols) CIF file. It describes a double multiplier (CIF ID 105). The input CIF file of 30K produces an output PLOT file that is smaller than the one produced by the 4K CIF file B. The execution time is fairly fast. All the commands in a file

with no symbols are executable commands and are written to the temporary EXECUTABLE file.

File D pushes mcifplot to the upper limit of its capabilities. It is a large pla generated by the VLSI design tool mkpla. It is similar in size to file C, but contains over 800 Call Commands. It illustrates two limits of mcifplot. First, the output PLOT file is incomplete and the program aborted when the disk was filled (186K). The PLOT file could still be printed with PLOT.COM, but the resulting check plot is obviously incomplete.

The other limitation is the extended execution time required to process the hundreds of Call Commands. Each Call results in a least one random access to disk. This results in extreme disk activity and really increases the execution time of the program. Because of the limited amount of memory available in CPM based microcomputers, the program must rely on the disk for storage. This results in much of the execution time of the program being spent doing disk I/O. mcifplot's dependence on the disk for temporary storage, and the requirement to access symbols on the disk during creation of the PLOT file, restricts its practical use to CIF files with a limited number of Call Commands.

The example check plots in Appendix G provide a good side by side comparison of the quality of the output of mcifplot and cifplot. The format of cifplot vs. mcifplot is very similar and the CIF bounds produced by both are identical (with the exception of those files which contain Wire

Commands - cifplot incorrectly determines the bounds of CIF Wire Commands). The biggest difference is the resolution of the printers used for the plots. The Versetec V-80-OP has a 200 dots/inch resolution vs. the 72 dots/inch resolution of the Okidata 92 (typical of the printers that are interfaced to PLOT.COM). Even with the difference in resolution, the similarities are striking between the two types of plots.

mcifplot Status

In order to compile and assemble the program efficiently, it was divided into five sections. MAIN.C contains all the constant and structure definitions, as well as the main procedure. It was always included when compiling the other sections. Stubs were built into INTCOM.C, PARSECIF.C, and BLDFILE.C to allow them to be executed and tested separately. Statistics about the size and composition of the sections are given in Table V-2.

Table V-2 mcifplot Divisions and Composition

File	Size(K)	Lines Code	% Code
MAIN.C	6.3	180	66.7
INTCOM.C	26.5	709	54.9
PARSECIF.C	37.4	813	44.7
BLDFILE.C	31.7	515	38.2
MCIFPLOT.C	22.0	456	46.4
=====			
TOTAL	124.0	2682	46.6

VI. Conclusions and Recommendations

Conclusions

The program mcifplot successfully implements a micro-computer based tool for printing check plots of CIF files. In general, the program meets all functional requirements and exceeds performance expectations. Before the implementation of mcifplot began, it appeared likely that a program of this nature was feasible on a microcomputer. However, there was still some uncertainty about the size of the final program and its execution speed. The finished program not only is contained in a single executable unit, but also processed MSI and SSI circuit descriptions well within the expected execution times.

The different design tools currently used for integrated circuit design impose varying types of demands on the computer on which they execute. Even so, some general conclusions can be drawn from the successful implementation of mcifplot. It is clear though, that additional information is required to determine to what extent existing microcomputers can be used for integrated circuit design. This information could be obtained from a theoretical study of the performance requirements of CAD design tools vs. that provided by the microcomputer. Information could also be learned from the implementation of additional design tools in the micro-computer environment.

The performance of mcifplot probably provides a fairly accurate picture of how other design tools are likely to perform in the microcomputer environment. Design tools for circuit simulation and design rule checking, as well as other functions, should prove useful and can be expected to operate successfully on circuits ranging in density from SSI to MSI. Of course, the tools must be designed carefully to maximize execution speed and minimize use of memory and disk space. As the cost/performance ratio of microcomputers increases yearly, the performance limits of microcomputer based tools will also increase.

Recommendations

Recommendations are discussed in three different areas. First, recommendations are given for ways to improve mcifplot in both function and performance. Next, suggestions for implementation of a check plot program on other microcomputers are discussed. The final set of recommendations concerns the implementation of other design tools on microcomputers.

mcifplot meets the design goals and requirements defined in Chapter II. These requirements were of necessity a subset of cifplot's function and options. It may be desired to expand and improve the function and/or performance of mcifplot.

There are many desired functional requirements in Chapter two which were not implemented. Most concern compati-

lity with the function of cifplot, but would still add capabilities that would probably be useful in certain circumstances for mcifplot's environment. The following features are suggested to improve the compatibility and utility of the program:

- 1) Allow processing of CIF commands in non-Manhattan directions. The biggest difficulty with implementation of this feature is the more complex clipping and fill algorithms required. Changes would also be required in the way the program determines the bounds of geometric elements.

- 2) Implement the CIF Poly Command and the Delete Definition Command. This item, along with item number 1, will allow mcifplot to accept all standard CIF 2.0 commands.

- 3) Implement additional User Extensions defined in the cifplot manual: Extension 0 and Extension 2.

- 4) Implement additional options defined in the cifplot manual: copy, List, approximate, and a cifplot format PATTERN file. Also implement multiple strip plotting as specified by the scale option. The grid option should label the grid lines in CIF units.

- 5) Allow processing of multiple input CIF files.

In addition to functional capabilities, the performance of mcifplot can be improved in several ways:

- 1) The program should be divided into two or more executable modules which are chained together. This would

allow room for a larger Symbol Table and the implementation of the new functional capabilities listed above.

2) This additional room should also be used to implement the one item which will produce the most dramatic improvement in execution speed: input file buffers. Since the program does a random access to disk at least once for every Call Command encountered in a symbol, the implementation of multiple input buffers would allow much of the time required for this random access to be eliminated. A random access to disk would only be required when the data in an input buffer was exhausted.

3) Those procedures which require the major amount of execution should be examined for efficiency. If possible the algorithm or C code should be made more efficient. Another alternative would be to convert the procedure to assembly language.

The cost/performance ratio of microcomputers is increasing at an amazingly rapid pace. Low cost microcomputers (i.e. less than \$5000) are available which have greatly expanded memory as well as high resolution graphics terminals. The implementation of mcifplot on one of these new microcomputers would allow several advantages in performance and capability.

The availability of more memory would allow a dramatic improvement in execution speed. The input CIF file could be converted into a more compact format and stored in memory as it is read and parsed. This would eliminate the requirement

to convert a CIF command from ASCII representation to an internal machine representation (long integers) each time it is encountered in a CIF symbol. With mcifplot, for example, a Box Command in a particular symbol has to be interpreted each time the symbol is called. This involves up to six ASCII to integer conversions and the calculation of the four corners of the box. The addition of more memory would allow the Box Command to be stored internally as a one character identifier and four floating point numbers which represent the scaled coordinates of the box lower left and upper right corners.

Of course, if the CIF file was converted and stored internally, the problem with slow random disk access to the CIF file during creation of the PLOT file would be eliminated. Even the output PLOT file could be written to a large internal buffer and written to disk at a much faster rate.

Finally, the utility of the check plotting program could be enhanced if its output could be directed to a high resolution graphics display. The resolution of the terminals of some low cost microcomputers is similar to the resolution available on most printers that are driven by PLOT.COM. If a new driver was written for PLOT.COM, check plots could be quickly displayed on the graphics terminal using an input PLOT file in the same format as was used to create a printed check plot. The biggest advantage to this feature would be that check plots would be available in much less time. This

would be especially useful for checking the placement of wiring connections.

In addition to enhancements to mcifplot, it is recommended that additional design tools be implemented in the microcomputer environment. Used together, these tools would truly allow small scale (MSI or less in complexity) integrated circuits to be completely designed on a low cost microcomputer. At a minimum, three additional tools would be required:

- 1) A high level layout language such as that interpreted by the Stanford design tool cll.

- 2) A program to perform design rule checking on an input CIF file.

- 3) A program to do switch level simulations of the circuit function.

Bibliography

1. Bilofsky, Walt. "Toolworks C/80 Version 3.0", The Software Toolworks. (February, 1984).
2. Bradeberry, James E. and Kuldip S. Rattan. "Microcomputers for Engineering Design," Proceedings of the IEEE 1st Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education. 97-100. IEEE Computer Society Press, Los Alamitos, CA, (October 1982).
3. Carter, Lt Col Harold W., Assistant Professor, Air Force Institute of Technology. Personal interview. Wright-Patterson AFB, Ohio, 20 April 1984.
4. Carter, Lt Col Harold W. "Integrated Circuit Design Automation using 8- and 16-bit Microcomputers." Proposal to the Air Force Office of Scientific Research. Boiling AFB, Washington D.C., January, 1984.
5. Graef, Jean L. "Low cost Graphics for CAD/CAM: What is available?" Proceedings of the IEEE 1st Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education. 97-100. IEEE Computer Society Press, Los Alamitos, CA, (October 1982).
6. Haydamack, William J. and Daniel J. Griffin. "VLSI Design Strategies and Tools," Hewlett-Packard Journal, 32: 5-12. (June 1981).
7. Hobson, Richard F. and Warren S. Synder. "CMOS VLSI Design With a Personal Computer," Proceedings of the 1983 International Electrical and Electronics Conference. 416-419. Institute of Electrical and Electronics Engineers, (1983).
8. Hon, Robert W. and Carlo H. Sequin. A Guide to LSI Implementation, 2nd Edition. Palo Alto, California: XEROX Palo Alto Research Center, (1980).
9. Jadrnicek, Rik. "Computer-aided Design," BYTE, 9: 172-209+ (January 1984).
10. Kernigan, B. W. and D. M. Ritchie, The C Programming Language. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1978.
11. Lee, Benjamin K. and Casey Jones. "CAD tools must change to meet the needs of VLSI," Electronics, 54: 108-110 (November 17, 1981).

12. McGreivy, Dennis J. "VLSI Approaching Cost-Performance Saturation," IEEE International Conference on Circuits and Computers 620-623. IEEE Computer Society Press, Los Alamitos, CA, (October 1982).
13. Mead, Carver and Lynn Conway. Introduction to VLSI Systems. Reading, Massachusetts: Addison-Wesley Publishing Company, 1980.
14. Nelson, Brent E., and others. "Cost Effective VLSI Design System," 1983 Symposium on Circuits and Systems. 505-508. Institute of Electrical and Electronics Engineers, (1983).
15. Newman, W. M. and R. F. Sproull. Principals of Interactive Computer Graphics, 2nd Edition. New York: McGraw-Hill, 1979.
16. Newton, A. R. "A Survey of Computer Aids for VLSI Layout," 1982 Symposium on VLSI Technology. 72-75. Business Center for Academic Societies Japan, Bunkyo-ku, Japan, (September 1982).
17. Newton, A. R. and others. "Design Aids for VLSI: The Berkeley Perspective," IEEE Transactions On Circuits And Systems, CAS-28: 666-680. (July 1981).
18. Oshikiri, M. and others. "A Switch Level Simulator on a Microcomputer," Proceedings of the 1983 Custom Integrated Circuits Conference. 351-353. Institute of Electrical and Electronics Engineers, (1983).
19. Sangiovanni-Vincentelli, Alberto L. "Guest Editorial," IEEE Transactions On Circuits And Systems, CAS-28: 617. (July 1981).
20. Seitz, Charles L. "The Education of VLSI Designers at the University Level," IEEE Computer Conference Spring 82. 106-108. IEEE Computer Society Press, Los Alamitos, CA, (February 1982).
21. Speer, Tom, "PLOT Version 3.3 (PLOT33) DOCUMENTATION". Manual for the CPM public domain program PLOT33, July 31, 1984.
22. Sussman-Fort, S. E. and others. "Line Printer Plotting Programs for VLSI," Proceedings of the IEEE 1st Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education. 97-100. IEEE Computer Society Press, Los Alamitos, CA, (October 1982).

TYPE: FILE

DATE: 30 May 1984

NAME: CADRC

DESCRIPTION: The CADRC file contains additional options which are included with the command line options as input to VLSI design tools. The options for a particular program are on a single line which begins with the name of that program. These options are always executed. For example:

mcifplot -l bbox -p CMOSPAT.STI

LOCATION: Always on drive A: or the same drive as the program executed.

DATA CHARACTERISTICS: text file

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: CADRC Options

DESCRIPTION: These are options which were obtained from the CADRC file. The options are used to control the functions of the other program modules and are identical in format to the options obtained from the Command Line.

SOURCES: A12

DESTINATIONS: A13

COMPOSITION: The data is composed of two parameters:
CADRC Number - the number of words in the CADRC File line which starts with the word "mcifplot". If equal to 0 or 1 then there are no options.
CADRC Array - an array of pointers which point to the location of the option words in memory. Element [0] is "mcifplot". Element [Cadrc Number - 1] is the last element.

PART OF: N/A

DATA CHARACTERISTICS: CADRC Number - integer
Cadrc Array - array of pointers, it is dimensioned dynamically up to a maximum of CADRC Buffer pointers.

VALUES: CADRC Number : integer 0 - CADRC Buffer
CADRC Array : Array [CADRC Buffer]

Appendix B. SADT Data Dictionary

To conserve space, more than one Data Element is listed on each page. The following Data Elements are included in the Dictionary in alphabetical order.

DATA ELEMENTS:

- CADRC File
- CADRC Options
- Call Commands
- CIF Command
- CIF File
- CIF File Table
- Command Line
- Control Variables
- Default Control Values
- Default Control Variables
- Error Messages
- Error Number
- Executable Commands
- EXECUTABLE File
- Fatal Error
- File Table
- Formatted Error Messages
- Geometric Commands
- Line Number
- MESSAGE File
- Modified Control Variables
- PATTERN File
- PLOT File
- Plot Page
- Scaling Factors
- Special Control Options
- Symbol Table
- Symbol Data
- Translated Executable Commands
- User Answer
- User Prompt
- User Query

- # 522 Maximum CIF command line length exceeded
CIF commands are limited to 161 characters in length to reduce the space required in the program for character buffers.
- # 525 Wire defined with only one point

Warnings

- # 1001 User Extension 2 detected - ignored
- # 1002 'name' is number in User Extension 9
- # 1003 nonblank characters encountered after End Command
This may mean the input CIF file has an unexpected End command in the middle of it or a ';' character will produce this error. Standard CIF has no characters after the End command.
- # 1004 this symbol was previously defined
A symbol with the same number was defined previously. It will be ignored.
- # 1005 User Extension '9' outside of symbol - ignored
- # 1006 User Extension detected - ignored
The -e option is in use.
- # 1007 Poly Command detected - ignored
- # 1008 User Extension '9' repeated in symbol - ignored
- # 2000 Grid spacing smaller than minimum line resolution-grid inhibited
A grid has been specified so small that the check plot will be black.
- # 2001 Plot bounds exceed legal CIF-number range
- # 3000 Symbol tracing exceeds available memory-ignored this symbol call
The output PLOT file is still written, but the call to this symbol is ignored.
- # 3001 Nested calls to deep - ignored call to this symbol
The output PLOT file is still written, but the call to this symbol is ignored.

Memory allocation errors (not displayed)

These errors are not displayed as text because if memory is exhausted, there is not enough room to read in the error number list from the MESSAGE.BIN file.

- 5001 Symbol Table allocation
Out of memory when trying to add an entry to the symbol table.
- 5002 dynamic memory allocation
- 5003 file structure allocation
Out of memory when trying to allocate a file structure when opening a file.
- 5004 pattern string allocation
Out of memory when trying to allocate a buffer for the input of patterns from the PATTERN file.

Is this a cifplot option?

- # 102 <-- unable to open file
- # 103 <-- recursion detected in this symbol
This symbol calls itself or another symbol which eventually ends up calling this one.
- # 104 <-- pages required for plot - maximum limit exceeded
mcifplot will not process a plot that is more than 20 pages long.
- # 105 <-- non-standard or illegal CPM file name
- # 106 <-- input CIF file with .VEC extension is not allowed
The default extension for the output PLOT file is ".VEC". An input CIF file with the ".VEC" extension would be overwritten.
- # 107 <-- transfer program not found
- # 200 Delete Definition Command not supported
- # 201 DF Command encountered outside of Symbol Definition
- # 202 DS Command encountered inside Symbol Definition
- # 203 End Command encountered inside Symbol Definition

- # 300 Fatal error: call to undefined symbol
This symbol was never defined.

Runtime Errors

- # 500 mcifplot does not support User Extension '0'
- # 501 Unknown command
- # 502 No layer specified for command
All geometric commands must have a layer specified before they are listed.
- # 503 Unknown Command or User Extension
- # 504 Incomplete command
- # 505 Unknown layer
This layer is not one of the 15 layer names specified by default or the -P option.
- # 506 Can not read integer
- # 507 Unexpected signed integer
- # 508 CIF-number out of range: $-16,777,215 \leq \text{CIF-number} \leq 16,777,215$
- # 509 Symbol number out of range 1 - 16,777,215
- # 511 Scale value equals 0
- # 512 Only one scale value found
Is the DS command in the form DS 100 A:B; where the optional A:B specifies the scale?
- # 513 Non-Manhattan direction detected
- # 514 Calculation of bounds exceeds legal CIF-number range
It is possible for commands with legal CIF numbers to exceed the maximum range for legal CIF numbers when their bounds are determined.
- # 515 Unable to read point
- # 517 Unable to read transformation
- # 519 Unknown transformation
- # 520 Non-Manhattan rotation detected

ERROR MESSAGES

The error messages in mcifplot are intended to be self explanatory. However, some additional information is given with certain error messages to clarify the intended meaning. The numbers associated with the error messages are only required if the file MESSAGES.BIN is not used when executing mcifplot. In that case, the appropriate error message associated with a given error number can be found from the following list. Option specification errors are reported in the same format for both the command line and CADRC file. Remember to check the CADRC file if mysterious option errors occur that are not in the command line.

Fatal Errors

- # 1 Usage: mcifplot [options] filename.cif
- # 2 usage: mcifplot -g CIF-number filename.cif [CIF-number > 0]
- # 3 error: CIF-number out of range: -16,777,215<=CIF-number<=16,777,215
- # 4 usage: mcifplot -d CIF-number filename.cif [CIF-number > 0]
- # 5 usage: mcifplot -s floating-point-number filename.cif
- # 6 error: xmin must be less than xmax
- # 7 error: ymin must be less than ymax
- # 8 usage: mcifplot -w xmin xmax ymin ymax filename.cif
- # 9 Fatal error: unable to open EXECOM.TMP, check disk space
There is no room to write the temporary file on disk.
- # 10 Fatal error: Comment Command not closed, EOF encountered
- # 11 error: layer name "text" refers to unsupported User Extension 2
- # 12 Fatal error: EOF encountered in Symbol Definition
- # 13 Fatal error: EOF encountered before End Command
- # 14 Fatal error: could not read line in pattern file
There is a problem with reading a line in the pattern file.
- # 15 Fatal error: output disk full, cannot complete PLOT file
The PLOT file can still be plotted using PLOT.COM, however
it will not be a complete check plot as described by the
input CIF file.
- # 16 Fatal error: output disk full, cannot complete EXECUTABLE file
The temporary file could not be completed. This error might
happen if there is a large input CIF file with no symbols.
- # 17 Fatal error: incorrect format in pattern file
Make sure the format matches the one defined in the -p
option described above and colors are in the range 0 - 127.
- # 18 nothing to plot (CIF file has symbol definitions without calls)
- # 19 Previous errors prevent building PLOT file
mcifplot will continue to parse the input file as long as it
can recover from errors. This error usually occurs when a
layer specification is incorrect, thus the stipple pattern
to use for a particular layer is unknown.
- # 100 <-- no such layer
- # 101 <-- unknown or unsupported option

12) mcifplot has the following 15 layers names and PLOT.COM colors built in as defaults:

ND	49	NI	50	NP	51	NC	52
NM	53	NB	54	NG	55	CW	56
CD	57	CP	58	CS	59	CC	60
CM	61	CG	62	CE	63		

These layer names and/or colors can be modified with the '-p' option if desired.

13) There are several differences in the options allowed in mcifplot and cifplot. These options are summarized in the table below. Although the mcifplot options are listed in both lower and upper case letters in order to have a similar format as cifplot's options, all options are interpreted as upper case letters in the command line and CADRC file.

cifplot/mcifplot Options

cifplot Option	Option Name	Status	mcifplot Option	Option Name
a	approximate	NI	-	-
b	banner	I	b	banner
c	copies	NI	-	-
C	Comments	I	C	comments
d	depth	I	d	depth
F	Font	NI	-	-
g	grid	I	g	grid
Ga/t	Graphics term	NI	-	-
h	half resolut.	NI	-	-
I	interactive	I	I	interactive
l	layer	I	l	layer
-	-	I*	m	make page
-	-	I*	n	notify
O	Output	NI	-	-
-	-	I*	o	output
P	Pattern	I	P	pattern
-	-	I*	q	quiet
r	rotate	I	r	rotate
s	scale	I	s	scale
S	Spool	NI	-	-
T	Terminal	NI	-	-
-	-	I*	t	transfer
V	Varian	NI	-	-
w	window	I	w	window
W	Wide	NI	-	-
X	extractor	NI	-	-

NI = Not Implemented

I = Implemented

* = options unique to mcifplot

to clarify its dimensions. With `cifplot`, if two geometric features on the same layer intersect (for example, when two wires overlap) they are outlined only on the outside of the overlapping regions. With `mcifplot`, the commands for each geometric feature will be generated as it is encountered in the CIF file. This will cause the outlines of overlapping regions to intersect and will slightly degrade the clarity of the the check plot. Of course, the outlines can be eliminated with a '-l' command line option.

9) The maximum number of symbols in an input CIF file is limited by the size of the CPM TPA. A Symbol Table is built in memory with an entry for each symbol in the CIF file during execution of the program. `mcifplot` requires approximately 42K of the TPA for the program itself. Each entry in the symbol table requires 28 bytes. The maximum number of symbols which can be placed in the symbol table depends on the unused space in the CPM TPA. This limitation should normally affect only unusually small TPA's or CIF files with more than 300 different symbols. If `mcifplot` runs out of memory for the Symbol Table a fatal error is generated.

10) The depth of nesting of symbols (i.e. symbols that call another symbol) is also limited by the size the CPM TPA. Each nested symbol generates a recursive procedure call which causes the stack to grow down from upper memory toward the program area by approximately 100 bytes. It is possible with a small TPA, a large symbol table, and symbols nested unusually (to a depth of 100 or more) for the stack to grow too close to the dynamically allocated symbol table. If this happens, `mcifplot` will issue a warning message and ignore symbols that are nested too deep.

11) There is also a practical limitation to the size of an input CIF file that can be processed. `mcifplot` does a random access to disk at least once for every Call Command processed. If multiple pages are required for the plot, the file may have to be processed again for each page of output. With CIF files which contain hundreds of Call Commands, the execution time of the program could easily stretch to hours in length. Probably before the long execution time is a problem though, the output PLOT file will fill the available disk space and `mcifplot` will gracefully abort. The reason for this is that the output PLOT file is not hierarchically structured like most CIF files with symbols. Each time a symbol is called in a CIF file, the low level commands to plot it (lines and fills) are written to the PLOT file. For example, a single call to symbol 4 may require 2K of low level commands to be written to the PLOT file. If this symbol is called 100 times in the CIF file, then 200K of low level commands will be required in the PLOT file.

1) A check plot is not automatically produced by mcifplot. The public domain program PLOT.COM must be executed either directly on the output file or indirectly by use of the -t option described above to obtain a check plot.

2) Non-Manhattan directions in commands or symbols are not supported.

3) CIF Poly and Delete Definition commands are not supported.

4) The Wire and Roundflash Commands are approximated with boxes. It should be noted that cifplot does not correctly determine the bounds of Wire Commands. For the Wire statement

```
Wire 20000 10000,10000 10000,100000;
```

cifplot will return bounds of

```
-823 20823 -823 110823
```

rather than the correct bounds of

```
0 20000 0 110000
```

This may result in differences in the plot window displayed by the two programs when interpreting CIF files which contain Wire Commands.

5) Multiple input CIF files are not supported.

6) User Extensions 0 and 2 are not supported. The layer name in User Extension 94 is ignored.

7) The integers in CIF files are limited to the range

```
-16,777,215    <=  integer    <=  16,777,215
```

as defined in A Guide to LSI Implementation, Hon and Sequin, Second Edition (Xerox PARC, 1980). Numbers are stored internally as long integers. When odd numbers are divided, the result is truncated to the lower integer value. This sometimes presents accuracy problems when working with small CIF numbers (less than 100).

8) The outlines of two or more geometric elements on the same layer will intersect if located together. The standard mode of operation for cifplot is to outline the outside of each layer with a black line. Thus, a rectangle is normally drawn with a rectangular region of a particular stipple pattern and then this region is outlined with a black line

fers control to filename with the name of the output PLOT file as input. A default extension of ".COM" is assumed for filename if no extension is specified. The most useful choice for filename is the current version of PLOT.COM.

-q

(quiet) Status messages are normally displayed for every 100 lines read from the CIF file, every 25 Call Commands processed for determining plot bounds, and for every 2K written to the output PLOT file. The bell is also rung when plot bounds have been determined and when program execution is complete. The quiet option inhibits displaying status messages on the standard output so that mcifplot executes in the same manner as cifplot.

-m

(make page) Forces the x dimension to one page or less in length, and rescales the y dimension if required. Adds a border of white space around the plot. Useful for making one page plots for reports or slides.

In the definition of CIF provisions were made for local extensions. All extension commands begin with a number. Part of the purpose of these extensions is to test what features would be suitable to include as part of the standard language. But it is important to realize that these extensions are not standard CIF and that many programs interpreting CIF do not recognize them. The following is list of extensions recognized by mcifplot:

1 message;

(Print) Print out the message on standard output when it is read.

9 name;

(Name symbol) name is associated with the current symbol.

94 name x y;

(Name point) name is associated with the point (x, y). The name is printed at the specified point on the plot. Name must not have any spaces in it, and it should not be a number.

MCIFPLOT/CIFPLOT DIFFERENCES

The following list summarizes the functional differences between mcifplot and cifplot. the performance limitations of mcifplot are also discussed.

(I.e. the comments may appear in the middle of a CIF command or the comment does not end with a semicolon.) Of course, CIF files should not have any errors and these comment related errors must be fixed before transmitting the file for fabrication. But many times fixing these errors seems to be more trouble than it is worth, especially if you just want to get a plot. This option is useful in getting rid of many of these comment related syntax errors.

-r

(rotate) Rotate the plot 90 degrees.

-P pattern_file

(Pattern) The -P option lets you specify your own layers and PLOT.COM colors from among the 127 available (see the documentation for PLOT.COM). The pattern_file must have a legal CPM file name. When a pattern_file is specified, the default mcifplot layers and their corresponding PLOT.COM "colors" are erased and replaced with the new layers and colors. A maximum of 15 layers can be specified. CIF layer names and PLOT.COM colors must be legal for their respective environments. The format of pattern_file requires the character "1" as the first character in the file on the first line. It is followed by a list of layer names and a corresponding PLOT.COM color with a decimal value from 0 to 127. One layer is specified per line and lines that begin with ';' are ignored. For example:

```
1
; example pattern file of format 1
NP 27
NC 13
NB 127
```

-o output_file

(output) The output PLOT file is written to the file named output_file. Normally the output PLOT file is written to a file on the same drive, with same name as the input CIF file and a ".VEC" extension. This option allows the specification of a new name or drive. output_file must be a legal CPM file name.

-n

(notify) Displays status of program option defaults and new options specified from the command line or CADRC file on the standard output. Also provides diagnostics concerning the number of commands and symbols in the input CIF file. The output can be redirected to a file for a saved copy if desired.

-t filename

(transfer) After execution of mcifplot is complete, trans-

fault the scale is set so that the window will fill the whole page in the y dimension. The scale option takes precedence over the window option. If a scale is specified along with a window that will exceed the paper width in the y dimension, then the window value is truncated to fit the page. No multiple strips are generated. Float is a floating point number specifying the number of inches which represents 1 micron.

-l layer_list

(layer) Normally all layers are plotted. The -l option specifies which layers NOT to plot. The layer_list consists of the layer names separated by commas, no spaces. There are some reserved names: allText, bbox, outline, pointName, and symbolName. Including the layer name allText in the list suppresses the plotting of text; bbox suppresses the bounding box around symbols. outline suppresses the thin outline that borders each layer. The keywords pointName and symbolName suppress the plotting of certain text created by local extension commands. pointName eliminates text created by user extension 94. symbolName eliminates text created by user extension 9. allText, pointName, and symbolName may be abbreviated by at, pn, and sn respectively.

-d n

(depth) This option lets you limit the amount of detail plotted in a hierarchically designed chip. It will only instantiate the plot down n levels of calls. Sometimes too much detail can hide important features in a circuit. The default depth value of 20 (limited to prevent stack overflow) can be changed with this option.

-g n

(grid) Draw a grid over the plot with spacing every n CIF units. Grid axes are not labeled, but the grid spacing is printed at the top of the plot.

-e

(extensions) Accept only standard CIF. User extensions produce warnings.

-I

(non-Interactive) Do not ask for confirmation. Always produce a PLOT file.

-b "text"

(banner) Print the text at the top of the plot.

-C

(Comments) Treat comments as though they were spaces. Sometimes CIF files created at other universities will have several errors due to syntactically incorrect comments.

```
A> mcifplot pla.cif
Version 1.0      10 November 1984
Window -5700 174000 -765000 168900
Scale: 1 micron is 0.004075 inches
The plot will be 0.610833 feet
Do you want a plot? y
```

After typing y, mcifplot will produce an output PLOT file which can be plotted by PLOT.COM. Or if the -t option is used, mcifplot could automatically execute PLOT.COM upon completion of the PLOT file to produce the check plot. The text output of mcifplot which is normally displayed on the terminal can be redirected to a file as follows:

```
A> mcifplot -i pla.cif >filename
```

There should be no space between the ">" and the output filename. Error messages are contained in a file called MESSAGE.BIN. This file must be on the disk from which mcifplot is executed for the text of the error messages to be displayed. If it is not found, only error numbers will be displayed.

mcifplot recognizes several command line options. These can be used to change the size and scale of the plot, change default plot options, and to select the name and drive of the output file. Several options may be selected. A dash(-) must precede each option specifier. Options may also be specified in a file named CADRC on the same drive as that from which mcifplot is executed. The purpose of the CADRC file is to allow a user to specify options which should always be used with mcifplot without having to repeatedly type the option in the command line. mcifplot checks for the existence of the CADRC file and looks for options listed on the line starting with the word "mcifplot". These options are interpreted in the same way as command line options. An example line in the CADRC file might be:

```
mcifplot -t PLOT.COM -n -l bbox
```

The following is a list of options accepted by mcifplot:

-w xmin xmax ymin ymax
(window) The -w options specifies the window; by default the window is set to be large enough to contain the entire plot. The windowing commands lets you plot just a small section of your chip, enabling you to see it in better detail. Xmin, xmax, ymin, and ymax should be specified in CIF coordinates.

-s float
(scale) The -s option sets the scale of the plot. By de-

23 November 1984

NAME

mcifplot - CIF interpreter and plotter for displaying LSI circuits

SYNOPSIS

mcifplot [options] filename.cif

INTRODUCTION

mcifplot is a CPM based program which produces check plots on a dot matrix printer using the public domain program PLOT.COM. It was designed to operate in the same way as the Unix based VLSI design tool cifplot (written by Dan Fitzpatrick at Berkeley) with some restrictions imposed due to the microcomputer environment. In fact, this manual is derived from the manual for cifplot. The manual is divided into three sections. Section one is a Unix style manual on how to use the program and a description of the available options. Section two describes the major differences between mcifplot and cifplot. Section three lists the program error messages with explanations on what they mean.

DESCRIPTION

mcifplot takes a description in Cal-Tech Intermediate Form (CIF) and produces an output PLOT file which can be interpreted by the public domain program PLOT.COM to produce a check plot. CIF is a low-level graphics language suitable for describing integrated circuit layouts. Although CIF can be used for other graphics applications, for ease of discussion it will be assumed that CIF is used to describe integrated circuit designs. mcifplot interprets CIF 2.0 descriptions with some limitations. The CIF Poly and Delete Definition Commands are not implemented. In addition, mcifplot will only accept Commands which describe geometric elements and symbols which lie in Manhattan directions (i.e. parallel to x or y axis) There are also restrictions on the size and complexity of the input CIF file which are imposed by the microcomputer environment. In addition, a number of local extensions have been added to CIF, including text on plots. These are discussed later.

To get a plot, call mcifplot with the name of a CIF file (a default option of ".CIF" is assumed) to be plotted. Multiple input CIF files are not supported. After reading the CIF description but before producing the output PLOT file, mcifplot will print an estimate of the size of the plot and then ask if it should continue to execute. Type y to proceed and anything else to abort. A typical run might look as follows:

23. Weinreb, Daniel L. "High Performance Personal Computation for VLSI CAD," IEEE Computer Conference Spring 82. 263-266. IEEE Computer Society Press, Los Alamitos, CA, (February 1982).
24. Xian-long, Hong and others. "QCADS -- A LSI CAD Tool for Minicomputers" 19th Design Automation Conference Proceedings. 706-709. IEEE Computer Society Press, Los Alamitos, CA, (June 1982).

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: Call Commands
DESCRIPTION: This is a string of characters which contains a
"Call" Command from the Executable CIF File.
The last character will always be a semicolon.
SOURCES: A32
DESTINATIONS: A33
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: string of characters
VALUES: Any legal character string

TYPE: FILE
DATE: 31 May 1984
NAME: CIF
DESCRIPTION: The CIF file is the main input to mcifplot. It
contains the CIF commands that are read to
produce the output PLOT file for producing
check plots. The default extension is .CIF and
the file should not have an extension of .VEC
(this extension is used for the output PLOT
file.
LOCATION: B: or drive specified in command line
DATA CHARACTERISTICS: text file

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: Command Line
DESCRIPTION: This is a string of options from the program
command line.
SOURCES: The command line which starts program execution
(actually the CPM OS).
DESTINATIONS: A13
COMPOSITION: The data is composed of two parameters:
Option Number - the number of words in the
command line 0 or 1 then there are no options.
Option Array - an array of pointers which point
to the location of the option words in memory.
Element [0] is "mcifplot". Element [Option
Number - 1] is the last element.
PART OF: N/A
DATA CHARACTERISTICS: Option Number - integer
Option Array - array of pointers, it is dimen-
sioned dynamically.
VALUES: Option Number : integer ≥ 0
Option Array : Array [CADRC Number]

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: CIF Commands
DESCRIPTION: Strings of characters which contain a single
CIF command.
SOURCES: A22
DESTINATIONS: A23
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: A single string of characters
VALUES: Any legal character string

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: CIF File Table
DESCRIPTION: The CIF File Table is a group of parameters
which contain specific information about the
input CIF file.
SOURCES: A21
DESTINATIONS: A22
COMPOSITION: See File Table
PART OF: N/A
DATA CHARACTERISTICS: See File Table.
VALUES: Values are set by the Command line, the CADRC
Options, and storage pointers.

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Control Variables

DESCRIPTION: The Control Variables are variables which are set to default values or are modified by Command Line or CADRC Options. These global variables control the function and operation of the entire program.

SOURCES: A12 / A13 / A14

DESTINATIONS: A2 / A3 / A4

COMPOSITION: The composition, Data characteristics and Default and allowed values are listed in the following table (for implementation in the C language):

Name	Type	Default Val	Range
Window	int	FALSE	TRUE or FALSE
Xmin	long	INFINITY	long
Xmax	long	-INFINITY	long
Ymin	long	INFINITY	long
Ymax	long	-INFINITY	long
Scale	int	FALSE	TRUE or FALSE
Scale Value	float	0	float > 0
AllText	int	TRUE	TRUE or FALSE
BBox	int	TRUE	TRUE or FALSE
Outline	int	TRUE	TRUE or FALSE
Text	int	TRUE	TRUE or FALSE
PointName	int	TRUE	TRUE or FALSE
SymbolName	int	TRUE	TRUE or FALSE
Depth	long	INFINITY	long > 0
Grid	int	FALSE	TRUE or FALSE
Grid Space	long	0	long > 0
Extension	int	TRUE	TRUE or FALSE
Interactive	int	TRUE	TRUE or FALSE
Comments	int	FALSE	TRUE or FALSE
Rotate	int	FALSE	TRUE or FALSE
Layer[1]	char[4]	"ND"	char[4]
Layer[2]	char[4]	"NP"	char[4]
Layer[3]	char[4]	"NC"	char[4]
Layer[4]	char[4]	"NM"	char[4]
Layer[5]	char[4]	"NI"	char[4]
Layer[6]	char[4]	"NB"	char[4]
Layer[7]	char[4]	"NG"	char[4]
Layer[8]	char[4]	" "	char[4]
Layer[9]	char[4]	" "	char[4]
Layer[10]	char[4]	" "	char[4]
Color[1-10]	int	TBD	PLOT colors

PART OF: N/A

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Default Control Values

DESCRIPTION: These are the default values of the Control Variables which are set immediately upon program execution.

SOURCES: Built into program code

DESTINATIONS: All

COMPOSITION: The composition is listed under Control Variables.

PART OF: N/A

DATA CHARACTERISTICS: See Control Variables

VALUES: See Control Variables.

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Default Control Variables

DESCRIPTION: These are the Control Variables after they are set to the Default Control Values.

SOURCES: All

DESTINATIONS: Part of Control Variable output of Interpret Commands process.

COMPOSITION: See Control Variables

PART OF: N/A

DATA CHARACTERISTICS: See Control Variables

VALUES: Values are set by Default Control Values.

TYPE: DATA ELEMENT

DATE: 31 May 1984

NAME: Error Messages

DESCRIPTION: These are the final error messages which are written to the standard output.

SOURCES: A42

DESTINATIONS: Standard Output

COMPOSITION: N/A

PART OF: N/A

DATA CHARACTERISTICS: Array of characters

VALUES: legal Ascii characters

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Error Numbers

DESCRIPTION: Error Numbers are integer numbers which correspond to errors in the Message File. They are passed to the Print Errors module which prints the appropriate error message to the standard output.

SOURCES: A1 A2 A3

DESTINATIONS: A4

COMPOSITION: N/A

PART OF: N/A

DATA CHARACTERISTICS: integer

VALUES: Fatal Errors: 0 > 500

Runtime Errors: 500 > 1000

Warnings: -500 > -1

The numbers which correspond to particular error are listed in the Error Table.

TYPE: FILE

DATE: 18 JUN 1984

NAME: EXECUTABLE

DESCRIPTION: This is a temporary file which contains all CIF commands from the input CIF files that are not in symbols. It will always contain the "E" command and should contain at least one call if symbols are used.

LOCATION: Drive from which program executes.

DATA CHARACTERISTICS: text file

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Executable Commands

DESCRIPTION: These are character strings of one Executable CIF command. The last character will always be a semicolon except for the "E" command.

SOURCES: A23

DESTINATIONS: A24

COMPOSITION: N/A

PART OF: N/A

DATA CHARACTERISTICS: character strings

VALUES: Any legal character string

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Fatal Error

DESCRIPTION: This parameter signifies that a fatal error has been encountered and the process executes an immediate abnormal execution abort.

SOURCES: A42

DESTINATIONS: A43

COMPOSITION: N/A

PART OF: N/A

DATA CHARACTERISTICS: integer flag

VALUES: a value of 1 signifies program abort.

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: File Table

DESCRIPTION: This is the table which contains all information required for file I/O. Using this table, files can be accessed using a pointer to the table.

SOURCES: Any Process which passes file information.

DESTINATIONS: Any process which requires file information.

COMPOSITION: see data characteristics.

PART OF: All specific file tables (CIF File Table)

DATA CHARACTERISTICS:

Element	Type	What
name	char[14]	name of file
mode	char[2]	mode of file interface
buffer start	*char	first character in file buffer
next character	*char	next character in file buffer
buffer size	int	size of character buffer
current offset	int	offset used for random access

VALUES: Values are determined dynamically during program execution.

TYPE: DATA ELEMENT
DATE: 31 May 1984
NAME: Formatted Error Messages
DESCRIPTION: These are strings which have been constructed by combining messages from the message file and other required information such as line numbers and character pointers to produce a finished error message.
SOURCES: A41
DESTINATIONS: A42
COMPOSITION: See Description; the finished product is one string.
PART OF: N/A
DATA CHARACTERISTICS: Array of Characters
VALUES: legal Ascii values

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: Geometric Commands
DESCRIPTION: These are character strings of one Geometric CIF command. This is either a Polygon, Box, Wire, or Roundflash Command. The last character will always be a semicolon.
SOURCES: A23
DESTINATIONS: A24
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: character strings
VALUES: Any legal character string

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: Line Number
DESCRIPTION: This is a variable which identifies the current line number during the first reading of the CIF file by the Parse CIF File process.
SOURCES: variable modified by A22
DESTINATIONS: A41
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: integer
VALUES: any legal integer value > 0

TYPE: FILE

DATE: 30 May 1984

NAME: MESSAGE

DESCRIPTION: This file contains all messages which are required by mcifplot. These include all errors and warnings. The program code will only contain message numbers which reference a particular message in this file.

LOCATION: The same drive as for program execution.

DATA CHARACTERISTICS: This file will be most easily constructed using another program and it has the following format:

First Element in File-> Number of messages unsigned integer

Table of Message

Numbers and Offsets	->	Message Number	integer
of message from		Message Offset	unsigned integer
the start of file			

Message strings	->	First Message	char array, '\0'
		...	
		Last Message	

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Modified Control Variables

DESCRIPTION: These variables are simply the Default Control Variables which have been modified by options in the Command Line or CADRC file.

SOURCES: A13 / A14

DESTINATIONS: A2 / A3 /A4

COMPOSITION: See Control Variables

PART OF: N/A

DATA CHARACTERISTICS: See Control Variables

VALUES: See Control Variables

TYPE: FILE

DATE: 30 May 1984

NAME: PATTERN

DESCRIPTION: This file is used to specify new stipple patterns (really color values) in mcifplot. The file contains a list of up to 15 new layers and colors. It has the following form:

```
1
;comments set off by ';'
LAY1      23
LAY2      55
```

LOCATION: The drive specified in the pattern option.

DATA CHARACTERISTICS: Ascii characters.

TYPE: FILE

DATE: 30 May 1984

NAME: PLOT

DESCRIPTION: This file is the output file of mcifplot. It contains elementary plot commands in the format required for PLOT.COM. The file is named filename.VEC, where filename is the name of the input CIF file. Check plots are produced by using this file as input to PLOT.COM.

LOCATION: The same drive as the input CIF file.

DATA CHARACTERISTICS: The file is a mixture of text and binary data as required by PLOT.COM

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Plot Page

DESCRIPTION: This is an integer flag which starts the reading and tracing of the Executable File.

SOURCES: A31

DESTINATIONS: A32

COMPOSITION: N/A

PART OF: N/A

DATA CHARACTERISTICS: Integer

VALUES: 1 = start reading file

2 = continuing reading file (check to see if complete file is in memory to avoid disk access)

0 = trace complete, erase Executable File

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Scaling Factors

DESCRIPTION: These are four long integers in CIF units which correspond to the x and y minimum (0.0) and x and y maximum (1.0) of PLOT.COM. The values are determined using the scale value, window value, and plot bounding box.

SOURCES: A31

DESTINATIONS: A34

COMPOSITION: Four long integers: Scale X Minimum
Scale Y Minimum
Scale X Maximum
Scale Y Maximum

PART OF: N/A

DATA CHARACTERISTICS: integers

VALUES: Any legal integer value

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Special Control Options

DESCRIPTION: The majority of the command line and CADRC options are processed in the Parse Options process. However, two special options require more complex activity than simply modifying global variables. These two activities use the parameters grouped under the heading Special Control Options.

SOURCES: A13

DESTINATIONS: A14

COMPOSITION: Banner - pointer to string for banner
Pattern File - pointer to string which contains the name of a pattern file

PART OF: N/A

DATA CHARACTERISTICS: Both are pointers

VALUES: legal pointer values

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Symbol Data

DESCRIPTION: The Symbol Data is a group of parameters which are used to construct the elements of the Symbol Table. These parameters have three formats as defined by the first parameter and described below.

SOURCES: A23

DESTINATIONS: A25

COMPOSITION: The first parameter is an integer flag which specifies the format of the rest of the parameters. It can have the value of 0, 1, or 2 which specifies the following three types:

Type 0: Symbol start

Type 1: Bounds of Executable Commands (excluding Calls) in Symbol

Type 2: Symbol End

The three formats are specified below:

Format 0:	Element	Type	What
	(2)	long	symbol number
	(3)	long	scale of symbol
	(4)	long	offset in CIF file to Symbol start
	(5)	long	unused
Format 1:	(2)	long	x minimum of Geometric Command
	(3)	long	y minimum of Geometric Command
	(4)	long	x maximum of Geometric Command
	(5)	long	y maximum of Geometric Command
Format 2:	(2)	long	unused
	(3)	long	unused
	(4)	long	unused
	(5)	long	unused

Values: legal values for long variables

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Symbol Table

DESCRIPTION: This table contains all necessary information to process and access all symbols in the CIF file. It is built using information from the Symbol Data. Note that the type data in the Symbol Table varies from the more general data types in the Symbol Data in order to conserve space.

SOURCES: A25

DESTINATIONS: A33

COMPOSITION: The table is a linked list that is dynamically allocated. Each entry in the list contains the following elements (types listed are for implementation in the C language):

<u>Name</u>	<u>Type</u>	<u>What</u>
number	unsigned	number of symbol
offset	int	offset from start of file
record	unsigned	record number on disk of start of symbol, if > than 255 then the offset must be interpreted relative to the record
x min	int	minimum x of symbol bounding box
y min	int	minimum y of symbol bounding box
x max	int	maximum x of symbol bounding box
y max	int	maximum y of symbol bounding box
next symbol	*int	pointer to next symbol in table

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: Translated Geometric Commands

DESCRIPTION: This is a character string which contains one Geometric command that has been translated according to the the transformations and scaling of the symbol that contains it. The numbers in the command are in absolute CIF units.

SOURCES: A33

DESTINATIONS: A34

COMPOSITION: N/A

PART OF: N/A

DATA CHARACTERISTICS: character string

VALUES: Any legal character string

TYPE: DATA ELEMENT
DATE: 18 JUN 1984
NAME: User Answer
DESCRIPTION: This is a character string read from the standard input in response to the User Query.
SOURCES: Standard Input
DESTINATIONS: A31
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: String of characters.
VALUES: The first character of User Answer[0] = 'y' or 'Y' signifies continue execution, any other character will abort the program.

TYPE: DATA ELEMENT
DATE: 31 May 1984
NAME: User Prompt
DESCRIPTION: Any legal character which signifies that the user is ready for another page of errors. Error messages are written to the standard output until a full screen has been output, the output pauses until a prompt is received from the user.
SOURCES: Standard Input
DESTINATIONS: A42
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: character
VALUES: Any Ascii character

TYPE: DATA ELEMENT

DATE: 18 JUN 1984

NAME: User Query

DESCRIPTION: This query consists of four lines of text which report the window size, plot length, and scale. The user is asked if a plot is desired. The lines are written to the standard output.

SOURCES: A31

DESTINATIONS: Standard Output

COMPOSITION: The following lines with appropriate values inserted for window, length, and scale:

Window 0 20000 0 345000
Scale: 1 micron is 0.00034 inches
The plot will be 0.610833 feet
Do you want a plot?

PART OF: N/A

DATA CHARACTERISTICS: Window - long
Scale - float
Feet - float

VALUES: Legal values for long and float (float values are >0)


```

        sypt->number = synnum;
else build = FALSE;
sypt->scale = (float) sym_yscale / sym_bscale;
while (*(com_string = get_cif_com(cif_file,FALSE)) != NULL &&
        !(*com_string == 'D' && *(com_string+1) == 'F'))
    {
        if (isdigit(*com_string) && !extension)
            {
                prt_error(1006,com_string);
            }
        else
            {
                switch (*com_string)
                {
case 'B':
                    if (no_layer) prt_error(502,com_string);
                    newbox = ck_box(com_string);
                    if (newbox != NULL) change_bds(&oldbox,newbox);
                    else build = FALSE;
                    break;
case 'R':
                    if (no_layer) prt_error(502,com_string);
                    newbox = ck_round(com_string);
                    if (newbox != NULL) change_bds(&oldbox,newbox);
                    else build = FALSE;
                    break;
case 'W':
                    if (no_layer) prt_error(502,com_string);
                    newbox = ck_wire(com_string);
                    if (newbox != NULL) change_bds(&oldbox,newbox);
                    else build = FALSE;
                    break;
case 'P':
                    prt_error(1007,com_string);
/*                    if (no_layer) prt_error(502,com_string);          */
/*                    newbox = ck_poly(com_string);                  */
/*                    if (newbox != NULL) change_bds(&oldbox,newbox);  */
/*                    else build = FALSE;                             */
                    break;
case 'L':
                    ck_layer(com_string);
                    no_layer = FALSE;
                    break;
case 'C':
                    if (ck_call(com_string) == TRUE) build = FALSE;
                    else sypt->calls = YES;
                    break;
case '1':
                    do_lcom(com_string);
                    break;
case '2':
                    prt_error(1001,com_string);
/*                    if (text) ck_2(com_string);                      */
                    break;

```

```

        puts(itoa(i,sp));
        puts(" K written to ");
        puts(outfname);
        puts("\n");
    }
    pc_quit(plot_file->channel);
}

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: bld_st
* SCNUMBER: 3.2
* FUNCTION: Adds an entry to the symbol table or starts symbol
*           table if this is the first symbol. Checks syntax of all
*           commands in the symbol. Calls prt_error if errors are
*           found and sets build = FALSE.
* INPUTS: com_string
* OUTPUTS: none
* GLOBAL VARIABLES USED: com_offset, com_record
* GLOBAL VARIABLES CHANGED: last_symbol, table_top, build
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: ck_box / ck_call / ck_round / ck_layer
*                 ck_start / ck_wire / do_lcom / ck_9 / isdigit / sbrk
*                 change_bds / get_cif_com / prt_error
* CALLING MODULES: parse_cif
* HISTORY: N/A
*****/
bld_st(com_string)
char    *com_string;
{
    static struct symbol    *sypt;
    static struct box      oldbox, *newbox;
    static int             found9, no_layer;
    static long            symnum, sym_ascale, sym_bscale;
    struct box             *ck_box(),*ck_wire(),*ck_round();
    char                   *get_cif_com();

    found9 = FALSE;
    no_layer = TRUE;
    oldbox.xmin = oldbox.ymin = INFINITY;
    oldbox.xmax = oldbox.ymax = -INFINITY;
    sypt = ((struct symbol *) sbrk(sizeof(struct symbol)));
    if (sypt == -1) prt_error(5001,NILL);
    if (table_top == NILL) table_top = sypt;
    sypt->offset = com_offset;
    sypt->record = com_record;
    sypt->next = NILL;
    sypt->inuse = NO;
    sypt->calls = NO;
    sym_ascale = sym_bscale = 1L;
    sypt->number = 0L;
    if (ck_start(com_string,&symnum,&sym_ascale,&sym_bscale) == FALSE)

```

```

*      TRUE (program aborts in report size) and if there is
*      no room to write the output file then the program
*      aborts using prt_error after the size is reported, but
*      before writing begins.
* INPUTS: expox
* OUTPUTS: none
* GLOBAL VARIABLES USED: charcount, wrtspace, nextk, quiet
*      plot_file
* GLOBAL VARIABLES CHANGED: pwindow, lastalloc
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: trace_table / set_page / report_size / itoa
*      plot_page / open_pfile / prt_error / pc_quit / sbrk puts
* CALLING MODULES: main
* HISTORY: N/A
*****/
bld_file(expox)
struct box      *expox;
{
    struct box      *trace_table();
    static struct box      *allbounds;
    static int      i, pagenum;
    static long      realxmax, pagex;
    static char      sp[7];
    char      *itoa();

    lastalloc = sbrk(1);
    if (lastalloc == -1) prt_error(5002,NILL);
    lastalloc += SAVEZONE;
    allbounds = trace_table(expox);
    if (ck_bounds(allbounds)) prt_error(2001,NILL);
    pwindow.xmin = allbounds->xmin;
    pwindow.xmax = allbounds->xmax;
    pwindow.ymin = allbounds->ymin;
    pwindow.ymax = allbounds->ymax;
    pagenum = set_page(&pwindow,&realxmax);
    pagex = pwindow.xmax - pwindow.xmin;
    report_size(&pwindow,realxmax);
    open_pfile();
    if (charcount < 1024L) prt_error(15,NILL);
    for (i=1; i<=pagenum; ++i)
    {
        plot_page(i);
        pwindow.xmin = pwindow.xmax;
        pwindow.xmax += pagex;
        if (((i+1) == pagenum) && (pwindow.xmax > realxmax))
            pwindow.xmax = realxmax;
    }
    if (!quiet)
    {
        pagex = wrtspace - charcount;
        pagex = pagex / 1024;
        i = (int) pagex + 1;
        puts("\r-> ");
    }
}

```

```

/*****
* DATE: 5 NOV 1984
* VERSION: 1.0
* NAME: add_line
* SCNUMBER: 3.1.3
* FUNCTION: Increments line_number and if not quiet, displays
*           the line count to the std out.
* INPUTS: none
* OUTPUTS: none
* GLOBAL VARIABLES USED: quiet
* GLOBAL VARIABLES CHANGED: line_number
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: puts / ltoa
* CALLING MODULES: get_cif_com
* HISTORY: N/A
*****/

```

```
add_line()
```

```

{
    static char    ap[10];
    char          *ltoa();

    ++line_number;
    if (!quiet && (line_number * LNOTIFY) == 0)
    {
        puts("\r-> parsing ");
        puts(cif_file->name);
        puts(": ");
        puts(ltoa(line_number,ap));
        puts(" lines read ");
    }
}

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: bld_file
* SCNUMBER: 4.0
* FUNCTION: Memory allocation for the symbol table is complete
*           at this time. (The last memory allocation is done in
*           open_pfile when space is allocated for the PLOT file
*           record). Sbrk is called to get the top of available
*           memory and lastalloc is set to this including a safety
*           zone. Next, the symbol table is traced to determine
*           true bounds of the entire plot. Sets global variable
*           pwindow with these bounds. Calls set_page to determine
*           window for each PLOT.COM page. Calls report_size to
*           display size and scale of plot to std out. If control
*           returns, open_pfile is called to open the output file
*           and set the available disk space in charcount. The
*           program aborts through prt_error if the available space
*           is less than 1 K. Next plot_page is called to write one
*           PLOT.COM page to the PLOT file for each page in
*           sucession. No output file is built if err_flag is

```

```

        puts(co);
        puts(ltoa(totbox,comstore));
        puts("\n");
        puts("Wire");
        puts(co);
        puts(ltoa(totwire,comstore));
        puts("\n");
        puts("Roundflash");
        puts(co);
        puts(ltoa(totround,comstore));
        puts("\n");
        puts("Layer");
        puts(co);
        puts(ltoa(totlayer,comstore));
        puts("\n");
        puts("Call");
        puts(co);
        puts(ltoa(totcall,comstore));
        puts("\n");
        puts("Comment");
        puts(co);
        puts(ltoa(totccoments,comstore));
        puts("\n");
        puts("Extension");
        puts(co);
        puts(ltoa(totextension,comstore));
        puts("\n");
        puts("total");
        puts(co);
        puts(ltoa(totcommand,comstore));
        puts("\n");
    }
    switch (type)
    {
    case 0:
        puts("\nmcifplot killed - abort\n");
        break;
    case 1:
        puts("\nFatal error(s) encountered - abort\n");
        break;
    case 2:
        puts("\n");
        break;
    case 3:
        puts("\n");
        if (transfer) exec(trname,outfname);
        break;
    case 4:
        puts("\n");
        break;
    }
    if(!quiet) puts("\007");
    exit();
}

```

```

/*                                GLOBALS                                */

float      viewport;
float      identity[9] =      ( 1.0,    0.0,    0.0,
                                0.0,    1.0,    0.0,
                                0.0,    0.0,    1.0 );
float      rotmatrix[9] =      ( 0.0,    1.0,    0.0,
                                -1.0,   0.0,    0.0,
                                0.0,    0.0,    1.0 );

int         deplimit = 200;
int         curdepth;
struct box  pwindow;
char        pban[162];
long        callcount, wrtspace;
long        nextk = WNOTIFY;
unsigned    lastalloc;

/*****
* DATE: 25 OCT 84
* VERSION: 1.0
* NAME: abort
* SCNUMBER: 1.1
* FUNCTION: Closes all open files (it does this rather crudely
*           by executing fclose on channel #'s 1-6). Erases the
*           Temporary EXECUTABLE file (if type < 3) and depending
*           on the type of abort - prints different messages to the
*           standard out. This module is responsible for all program
*           controlled termination of mcifplot.
* INPUTS: type
* OUTPUTS: none
* GLOBAL VARIABLES USED: quiet, transfer, trname, totcommand
*           totbox, totwire, totroundflash, totextension, totcomment
*           totcall, totsymbol
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: all files closed
* MODULES CALLED: fclose / puts / unlink / exec
* CALLING MODULES: main / report_size
* HISTORY: N/A
*****/
abort(type)
int      type;
{
    static int      i;
    static char      *co = " Commands: ";
    char             *ltoa();
    for (i=1; i<=6; ++i) fclose(i);
    if (type < 4) unlink(exec_file->name);
    if (notify && (type == 0 || type == 3))
    {
        puts("\n\nSyabola: ");
        puts(ltoa(totsymbol,comstore));
        puts("\n");
        puts("Box");
    }
}

```

```

#ifdef TEST
float      viewport;
float      identity[9] =      ( 1.0,    0.0,    0.0,
                                0.0,    1.0,    0.0,
                                0.0,    0.0,    1.0 );

float      rotmatrix[9] =      ( 0.0,    1.0,    0.0,
                                -1.0,   0.0,    0.0,
                                0.0,    0.0,    1.0 );

long      nextk = WNOTIFY;
long      wrtaspce, callcount;
int       deplimit = 200;
unsigned  lastalloc;
int       curdepth;
struct box pwindow;
char      pban[162];
#else
extern float      viewport;
extern float      identity[9], rotmatrix[9];
extern int       deplimit, curdepth;
extern struct box pwindow;
extern char      pban[162];
extern long      callcount, nextk, wrtaspce;
extern unsigned  lastalloc;
#endif

/*****
* DATE: 21 NOV 84
* VERSION: 1.0
*
* TITLE: MCIFPLOT.C
* OWNER: Kirk S. Horton
* OPERATING SYSTEM: CPM 2.2
* LANGUAGE: Software Toolworks C/80
* USE: Comlile and link with INTCOM.REL, PARSECIF.REL,
*      BLDFILE.REL, MAIN.REL, and CLIB.REL
* CONTENTS:
*      do_9xcom      4.5.2.6
*      do_box        4.5.2.1
*      do_grid       4.5.1
*      do_page       4.5.2
*      do_round      4.5.2.3
*      do_wire       4.5.2.2
*      plot_page     4.5
*      proc_call     4.5.2.5
*      put_box       4.5.2.1.3
*      test_space    PC PROC
*****/
#include "a:main.c"

```

```

*      find_sym      3.2.7.1      *
*      fskp_blank    3.1.2        *
*      fskp_comment  3.1.1        *
*      get_cif_com    3.1          *
*      get_integer   3.2.1.1      *
*      get_point     3.2.1.2      *
*      put_com       3.3.1        *
*      skip_blank    3.2.4.1      *
*      skip_sep      3.2.1.4      *
*      wrt_ex_file   3.3          *
*  FUNCTION: Parses and checks syntax of the input CIF file. The *
*            executable commands are written to the EXECUTABLE file *
*            and symbol table entries are created for each symbol. *
*
*...../
#include "a:main.c"

/*                                GLOBALS                                */

struct symbol *last_symbol;
int          build, end, exlayer;
struct box   exbox;

/...../
*  DATE: 12 NOV 84
*  VERSION: 1.0
*
*  TITLE: BLDFILE.C
*  OWNER: Kirk S. Horton
*  OPERATING SYSTEM: CPM 2.2
*  LANGUAGE: Software Toolworks C/80
*  USE: Compile and link with MAIN.REL, PARSECIF.REL, INTCOM.REL,
*      MCIFPLOT.REL, and CLIB.REL
*  CONTENTS:
*      bld_file      4.0
*      clip_box      4.5.2.1.1
*      get_ex_com    4.1.1
*      get_layer     4.5.2.4
*      get_tran      4.1.2.1
*      mult33        4.1.2.1.1
*      open_pfile    4.4
*      report_size   4.3
*      set_page      4.2
*      trace_table   4.1
*      tracedown     4.1.2
*      trans_box     4.1.3
*      trantoplot    4.5.2.1.2
*...../
#include "a:main.c"

```



```

/*****
* DATE: 21 NOV 84
* VERSION: 1.0
*
* TITLE: INTCOM.C
* OWNER: Kirk S. Horton
* OPERATING SYSTEM: CPM 2.2
* LANGUAGE: Software Toolworks C/80
* USE: Compile and link with MAIN.REL, PARSECIF.REL, BLDFILE.REL,
*      MCIFPLOT.REL, and CLIB.REL to form MCIFPLOT.COM.
* CONTENTS:
*      int_command      2.0
*      ck_range         2.2.1
*      cpm_file         2.2.6
*      do_notify        2.3
*      file_open        2.2.4
*      file_seek        2.2.5.1
*      get_cad_opt      2.1
*      get_pattern      2.2.3
*      par_layer        2.2.2
*      par_options      2.2
*      prt_error        2.2.5
*
* FUNCTION: Interprets the command line and CADRC file to modify
*           the status of the Control Variables.
*****/

```

```
#include "a:main.c"
```

```

/*****
* DATE: 12 NOV 84
* VERSION: 1.0
*
* TITLE: PARSECIF.C
* OWNER: Kirk S. Horton
* OPERATING SYSTEM: CPM 2.2
* LANGUAGE: Software Toolworks C/80
* USE: Compile and link with MAIN.REL, INTCOM.REL, BLDFILE.REL
*      and MCIFPLOT.REL AND CLIB.rel to form MCIFPLOT.COM
* CONTENTS:
*      parse_cif        3.0
*      add_line         3.1.3
*      bld_at           3.2
*      box_bounds       3.2.1.3
*      change_bds       3.2.9
*      ck_9             3.2.5
*      ck_bounds        3.2.1.3.1
*      ck_box           3.2.1
*      ck_call          3.2.5
*      ck_layer         3.2.6
*      ck_round         3.2.3
*      ck_start         3.2.7
*      ck_wire          3.2.2
*      do_lcom          3.2.8

```

```

#else
/*          CONTROL VARIABLES          */

extern int      banner;
extern char     bantext[BANLENGTH];
extern int      bbox;
extern int      color[LAYERMAX];
extern int      comments;
extern long     depth;
extern int      extension;
extern int      grid;
extern long     gridspace;
extern int      interactive;
extern char     layer[LAYERMAX][5];
extern int      makepage;
extern int      notify;
extern int      outfile;
extern char     outfname[15];
extern int      outline;
extern int      pattern;
extern int      pointname;
extern int      quiet;
extern int      rotate;
extern int      scale;
extern float    scalenum;
extern int      symbolname;
extern int      text;
extern int      transfer;
extern char     trname[15];
extern int      window;
extern long     wxmin, wxmax, wymin, wymax;
extern char     *patstring;

/*          GLOBALS          */

extern struct file      *er_file, *cif_file, *exec_file, *plot_file;
extern struct symbol    *table_top;
extern long             charcount, line_number, totextension, totcomment, totlayer;
extern long             totbox, totwire, totround, totcall, totsymbol, totcommand;
extern int              inrestab, mesfiletest, com_offset, com_record, err_flag;
extern char             comstore[MAXCOMLEN];

#endif

```

```

long    symin;
long    sxmax;
long    symax;
char    calls;
char    inuse;
struct symbol *next;
);

```

```
#ifdef TEST
```

```
/*
```

```
CONTROL VARIABLES
```

```
*/
```

```

int      banner                = FALSE;
char     bantext[BANLENGTH]    = NULL;
int      bbox                  = TRUE;
int      color[LAYERMAX]       =
( 49 , 50 , 51 , 52 , 53 , 54 , 55 , 56 , 57 , 58 , 59 , 60 , 61 , 62 , 63 );
int      comments              = FALSE;
long     depth                 = NESTLIMIT;
int      extension              = TRUE;
int      grid                  = FALSE;
long     gridspace              = 0;
int      interactive            = TRUE;
char     layer[LAYERMAX][5]    =
( "ND","NI","NP","NC","NM","NB","NG","CW","CD","CP","CS","CC","CM","CG","CE" );
int      makepage               = FALSE;
int      notify                = FALSE;
int      outfile                = FALSE;
char     outfname[15]          = NULL;
int      outline                = TRUE;
int      pattern                = FALSE;
int      pointname              = TRUE;
int      quiet                  = FALSE;
int      rotate                 = FALSE;
int      scale                  = FALSE;
float    scalenum               = 0;
int      symbolname             = TRUE;
int      text                   = TRUE;
int      transfer               = FALSE;
char     trname[15]            = NULL;
int      window                 = FALSE;
long     wxmin                  = INFINITY;
long     wxmax                  = -INFINITY;
long     wymin                  = INFINITY;
long     wymax                  = -INFINITY;
char     *patstring             = NULL;

```

```
/*
```

```
GLOBALS
```

```
*/
```

```

struct file    *er_file, *cif_file, *exec_file, *plot_file;
struct symbol  *table_top;
long           charcount, line_number, totlayer, totextension, totccomment;
long           totbox, totwire, totround, totcall, totsymbal, totcommand;
int            inrestab, nesfiletest, com_offset, com_record, err_flag;
char           comstore[MAXCOMLEN];

```

```

#define NULL          0
#define NO            '0'
#define NULL         '\0'
#define PAGEMAX       20
#define PAGERIM       10
#define PAGEWIDTH     8.0
#define PLOTCHARS     96L
#define PLOTMAX       32767
#define RETCURDISK    25
#define SAVEZONE      200
#define SELECTDISK    14
#define TRUE          1
#define VERSION       "Version 1.0\t21 November 1984\t(c) 1984 Kirk S. Horton"
#define WHITE         0
#define WNOTIFY       2048
#define YES           '1'

```

```

/*                      STRUCTURES                      */

```

```

struct box
(
    long    xmin;
    long    xmax;
    long    ymin;
    long    ymax;
);

struct cliprec
(
    int     top;
    int     bottom;
    int     left;
    int     right;
);

struct file
(
    char     name[15];
    char     mode[3];
    char     *next_char;
    char     *buff_start;
    int      channel;
    int      buff_size;
);

struct point
(
    long     x;
    long     y;
);

struct symbol
(
    long     number;
    int      offset;
    int      record;
    float    scale;
    long     axmin;

```

```

/*****
* DATE: 12 NOV 84
* VERSION: 1.0
*
* TITLE: MAIN.C
* OWNER: Kirk S. Horton
*
* MCIFPLOT AND IT'S DOCUMENTATION ARE COPYRIGHT (C) 1984 BY
* KIRK S. HORTON IN ASSOCIATION WITH THE UNITED STATES AIR FORCE
* USERS MAY COPY AND EXCHANGE THE PROGRAM WITH THE RESTRICTION
* THAT SAID DISTRIBUTION IS NON-COMMERICAL IN NATURE. RESALE
* FOR PROFIT REQUIRES THE EXPRESS WRITTEN CONSENT OF THE
* COPYRIGHTER.
*
* OPERATING SYSTEM: CPM 2.2
* LANGUAGE: Software Toolworks C/80
* USE: Comlile and link with INTCOM.REL, PARSECIF.REL,
*      BLDFILE.REL, MCIFPLOT.REL, AND CLIB.REL to form
*      MCIFPLOT.COM
* CONTENTS:
*      global definitions
*      structure definitions
*      Control Variables
*      global variables
*      main          1.0
*
* FUNCTION: Main calls the 3 major functions which implement
*      MCIFPLOT.COM - int_command, parse_cif, and bld_file
*****/

```

```

/*          CONSTANTS          */

#define BANLENGTH      83
#define BLACK           127
#define CADNAME         "CADRC"
#define CADOPTMAX       30
#define CLOSEROOM       5L
#define CNOTIFY         10L
#define CONVERT         1024L
#define EOF             -1
#define ERRSTGBUF       81
#define EXNAME          "EXCOM.TMP"
#define FALSE           0
#define FEETMICRON      2.1167e-4
#define INFINITY        16777215
#define LAYERMAX        15
#define LNOTIFY         100
#define MAXCOMLEN       161
#define MAXSYMNUM       65535L
#define MESFILE         "MESSAGE.BIN"
#define MINLINE         .001
#define MINFILL         .002
#define NESTLIMIT       10
#define NEWLINE         '\n'

```

APPENDIX C: Structure Charts and Headers/Code

APPENDIX C includes three items: the file headers for the four mcifplot divisions, the headers and code for mcifplot with the procedures in alphabetical order, and the program Structure Charts.

```

case '9':
    if (*(com_string+1) != '4' && found9)
    {
        prt_error(1008,com_string);
        break;
    }
    found9 = TRUE;
    if (pointname || symbolname) ck_9(com_string);
    break;
case '0':
    prt_error(500,com_string);
    break;
case 'D':
    if (*(com_string+1) == 'D') prt_error(200,com_string);
    else if (*(com_string+1) == 'S') prt_error(202,com_string);
    else prt_error(501,com_string);
    break;
case 'E':
    prt_error(203,com_string);
    break;
default:
    prt_error(503,com_string);
    }
    }
    }

if (*com_string == NULL) prt_error(12,NILL);

sypt->sxmin = oldbox.xmin * sym_ascale / sym_bscale;
sypt->sxmax = oldbox.xmax * sym_ascale / sym_bscale;
sypt->symin = oldbox.ymin * sym_ascale / sym_bscale;
sypt->symax = oldbox.ymax * sym_ascale / sym_bscale;

if (last_symbol != NILL) last_symbol->next = sypt;
last_symbol = sypt;

```

```

/*****
* DATE: 10 OCT 1984
* VERSION: 1.0
* NAME: box_bounds
* SCNUMBER: 3.2.1.3
* FUNCTION: Uses box_data to determine bounds of a box and
*           modifies the input box (via pointer) to these bounds.
*           Returns error = TRUE if bounds exceed legal CIF-numbers
*           when calculated.
*           Assumes Manhattan directions. Since dx or dy must be
*           zero the procedure calculates the bounds of the box if
*           it lies parallel to the x or y axis. Should be
*           rewritten to allow for no restrictions on directions.
*           Round down occurs if numbers are not even.
* INPUTS: box_data
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: ck_bounds
* CALLING MODULES: ck_box
* HISTORY: N/A
*****/
box_bounds(length,width,cx,cy,dx,dy,bpt)
long          length, width, cx, cy, dx, dy;
struct box    *bpt;
(
    length /= 2;
    width /= 2;
    if (dx)
    {
        bpt->xmin = cx - length;
        bpt->ymin = cy - width;
        bpt->xmax = cx + length;
        bpt->ymax = cy + width;
    }
    else
    {
        bpt->xmin = cx - width;
        bpt->ymin = cy - length;
        bpt->xmax = cx + width;
        bpt->ymax = cy + length;
    }
    if (ck_bounds(bpt)) return(TRUE);
    return(FALSE);
)

```



```

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: change_bds
* SCNUMBER: 3.2.9
* FUNCTION: Compares the bounds in the new input box to the old
*           input box. Modifies the old input box to the maximum
*           values of the two input boxes (the new symbol bounds).
* INPUTS: oldbox, newbox
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: bld_st / wrt_ex_file
* HISTORY: N/A
*****/
change_bds(oldbox,newbox)
struct box      *oldbox, *newbox;
{
    if (newbox->xmin < oldbox->xmin) oldbox->xmin = newbox->xmin;
    if (newbox->xmax > oldbox->xmax) oldbox->xmax = newbox->xmax;
    if (newbox->ymin < oldbox->ymin) oldbox->ymin = newbox->ymin;
    if (newbox->ymax > oldbox->ymax) oldbox->ymax = newbox->ymax;

    return;
}

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: ck_9
* SCNUMBER: 3.2.5
* FUNCTION: Checks syntax of Extension 9 and 94 (layer is not
*           read or checked for 94). Returns error = TRUE if syntax
*           errors are found.
* INPUTS: com_string
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: prt_error / isdigit / get_point / isspace
*                 isalpha
* CALLING MODULES: build_st
* HISTORY: N/A
*****/
ck_9(sp)
char      *sp;
{
    static char      *tp;
    static int      four;

```

```

static struct point    pt;

tp = sp + 1;
if (*tp == '4')
(
    four = TRUE;
    ++tp;
)
else four = FALSE;

while (isspace(*tp)) ++tp;
if (isdigit(*tp)) prt_error(1002,sp);
if (four)
(
    while (isalpha(*tp) || isdigit(*tp)) ++tp;
    if (get_point(sp,&tp,&pt))
    (
        prt_error(515,sp);
        return(TRUE);
    )
)

if (notify) ++totextension;
return(FALSE);
)

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: ck_bounds
* SCNUMBER: 3.2.1.3.1
* FUNCTION: Uses input box and checks to insure that each
*           coordinate is a legal CIF number. Returns error = FALSE
*           if ok, else error = TRUE.
* INPUTS: box
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: ck_range
* CALLING MODULES: box_bounds
* HISTORY: N/A
*****/
ck_bounds(bpt)
struct box    *bpt;
(
    if (ck_range(bpt->xmin) ||
        ck_range(bpt->xmax) ||
        ck_range(bpt->ymin) ||
        ck_range(bpt->ymax))    return(TRUE);
    return(FALSE);
)

```

```

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: ck_box
* SCNUMBER: 3.2.1
* FUNCTION: Checks CIF Box Command for correct syntax and
*           reports errors. Returns pointer to box with correct
*           bounds or NULL if errors are found.
*           Checks for Non-Manhattan directions in the box direction
*           if found. NM directions produce syntax errors. With
*           Manhattan directions dx or dy is always equal to 0, so
*           it is not necessary to scale the direction.
* INPUTS: sp
* OUTPUTS: *box
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: totbox
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: box_bounds / skip_sep / get_integer / isdigit
*                 prt_error / get_point
* CALLING MODULES:
* HISTORY: N/A
*****/
struct box      *ck_box(sp)
char      *sp;
{
    static char      *tp;
    static long      length, width, cx, cy, dx, dy;
    static struct box newbox;
    static struct point pt;

    dx = 1L;
    dy = 0L;
    tp = sp + 1;
    if (get_integer(sp,&tp,&length)) return(NULL);
    if (get_integer(sp,&tp,&width)) return(NULL);
    if (get_point(sp,&tp,&pt))
    {
        prt_error(515,sp);
        return(NULL);
    }
    cx = pt.x;
    cy = pt.y;
    skip_sep(&tp);
    if (isdigit(*tp) || *tp == '-')
    {
        if (get_point(sp,&tp,&pt))
        {
            prt_error(515,sp);
            return(NULL);
        }
        dx = pt.x;
        dy = pt.y;
        if ((dx != 0L && dy != 0L) || (dx == 0L && dy == 0L))

```

```

        {
            prt_error(513,sp);
            return(NILL);
        }
    )

    if (box_bounds(length,width,cx,cy,dx,dy,&newbox))
    {
        prt_error(514,sp);
        return(NILL);
    }

    if (notify) ++totbox;
    return(&newbox);
)

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: ck_call
* SCNUMBER: 3.2.4
* FUNCTION: Checks syntax of call command, if errors are found
*           returns error = TRUE, else FALSE.
* INPUTS: sp
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: totcall
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: get_integer / get_point / skip_blank /
*                 prt_error
* CALLING MODULES: bld_st
* HISTORY: N/A
*****/
ck_call(sp)
char    *sp;
{
    static char    teapc, *tp;
    static struct point    pt;
    static long    sym;

    tp = sp+1;
    if (get_integer(sp,&tp,&sym)) return(TRUE);
    if (ck_range(sym) || sym < 1L)
    {
        prt_error(509,sp);
        return(TRUE);
    }
    skip_blank(&tp);
    while (*tp != ';')
    {
        switch (*tp)
        {
            case 'T':

```

```

        ++tp;
        if (get_point(sp,&tp,&pt))
        {
            prt_error(517,sp);
            return(TRUE);
        }
        break;
    case 'R':
        ++tp;
        if (get_point(sp,&tp,&pt))
        {
            prt_error(517,sp);
            return(TRUE);
        }
        if (pt.x != 0 && pt.y != 0)
        {
            prt_error(520,sp);
            return(TRUE);
        }
        break;
    case 'M':
        ++tp;
        skip_blank(&tp);
        if (*tp != 'X' && *tp != 'Y')
        {
            prt_error(517,sp);
            return(TRUE);
        }
        ++tp;
        break;
    default:
        prt_error(519,sp);
        return(TRUE);
    }
    skip_blank(&tp);
}
if (notify) ++totcall;
return(FALSE);

```

}

```

/*****
* DATE: 26 OCT 1984
* VERSION: 1.0
* NAME: ck_layer
* SCNUMBER: 3.2.6
* FUNCTION: Checks syntax of layer and if layer name is in layer
* table. Returns error = TRUE if errors, else error=FALSE.
* INPUTS: sp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: totlayer
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: skip_blank / isupper / isdigit / prt_error
* CALLING MODULES: build_st
* HISTORY: N/A
*****/
ck_layer(sp)
char *sp;
{
    static int found, limit, i, j;
    static char *tp, shortname[4];

    tp = sp + 1;
    skip_blank(&tp);
    if (!isupper(*tp) && !isdigit(*tp))
    {
        prt_error(504,sp);
        return(TRUE);
    }

    i = 0;
    while ((isupper(*tp) || isdigit(*tp)) && (i < 4))
    {
        limit = i;
        shortname[i++] = *tp;
        ++tp;
    }

    for (i=0; i<=LAYERMAX; ++i)
    {
        for (j=0; j<=limit; ++j)
        {
            if (layer[i][j] == shortname[j]) found = TRUE;
            else found = FALSE;
        }
        if (found) break;
    }

    if (i > LAYERMAX)
    {
        prt_error(505,sp);
        return(TRUE);
    }

    if (notify) ++totlayer;
    return(FALSE);
}

```

```

/*****
* DATE: 9 OCT 1984
* VERSION: 1.0
* NAME: ck_range
* SCNUMBER: 2.2.1
* FUNCTION: checks the range of the input long integer to
*           verify that it is a valid CIF-number, that is:
*           INFINITY < CIF-number < INFINITY
*           Returns error = TRUE if out of range, else returns FALSE
* INPUTS: long
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: par_options
* HISTORY: N/A
*****/
ck_range(test)
long test;
{
    test = (test < 0) ? -test : test;
    if (test > INFINITY) return(TRUE);
    else return(FALSE);
}

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: ck_round
* SCNUMBER: 3.2.3
* FUNCTION: Checks Flash Command for correct syntax and reports
*           errors. Returns pointer to box with correct bounds or
*           NULL if errors are found.
* INPUTS: ap
* OUTPUTS: *box
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: totroundflash
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: ck_bounds / get_integer / prt_error
*               get_point
* CALLING MODULES:
* HISTORY: N/A
*****/
struct box *ck_round(ap)
char *ap;
{
    static char *tp;
    static long diam;
    static struct box newbox;
    static struct point pt;

```

```

    tp = sp + 1;
    if (get_integer(sp,&tp,&diam)) return(NILL);
    if (get_point(sp,&tp,&pt))
    {
        prt_error(515,sp);
        return(NILL);
    }
    diam /= 2;
    newbox.xmin = pt.x - diam;
    newbox.ymin = pt.y - diam;
    newbox.xmax = pt.x + diam;
    newbox.ymax = pt.y + diam;
    if (ck_bounds(&newbox))
    {
        prt_error(514,sp);
        return(NILL);
    }
    if (notify) ++totroundflash;
    return(&newbox);
}

```

```

/*****
* DATE: 29 OCT 84
* VERSION: 1.0
* NAME: ck_start
* SCNUMBER: 3.2.7
* FUNCTION: Checks the syntax of the DS command. Modifies the
*           input a and b scale values and symnum via pointer if
*           they are found correctly. Uses find_sym to verify that
*           the symbol number found is not already in the symbol
*           table. If it is (a redefined symbol), then a warning is
*           issued and the old symbol's number is set to zero.
*           Returns nonzero if errors.
* INPUTS: sp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: get_integer / find_sym / skip_sep / isdigit
*                 prt_error / ck_range
* CALLING MODULES: build_st
* HISTORY: N/A
*****/

```

```

ck_start(sp,symnum,psascale,psbscale)
char      *sp;
long      *symnum, *psascale, *psbscale;
{
    static char      *tp;
    struct symbol     *tsym, *find_sym();

    tp = sp;
    while (*tp != 'S') ++tp;

```



```

++tp;
if (get_integer(sp,&tp,sy anum)) return(TRUE);
if (ck_range(*sy anum) || *sy anum < 1L)
{
    prt_error(509,sp);
    return(TRUE);
}
if ((tsym = find_sym(*sy anum)) != NULL)
{
    prt_error(1004,sp);
    tsym->number = 0;
}
skip_sep(&tp);
if (isdigit(*tp))
{
    if (get_integer(sp,&tp,psascale)) return(TRUE);
    if (*psascale == 0)
    {
        prt_error(511,sp);
        return(TRUE);
    }
    skip_sep(&tp);
    if (!isdigit(*tp))
    {
        prt_error(512,sp);
        return(TRUE);
    }
    if (get_integer(sp,&tp,psb scale)) return(TRUE);
    if (*psb scale == 0)
    {
        prt_error(511,sp);
        return(TRUE);
    }
}
if (notify) ++tot symbol;
return(FALSE);

```

)

```

/*****
* DATE: 5 NOV 1984
* VERSION: 1.0
* NAME: ck_wire
* SCNUMBER: 3.2.2
* FUNCTION: Checks Wire Command for correct syntax and reports
*           errors. Errors are also produced for NonManhattan
*           directions. Returns error = TRUE if found, else FALSE.
*           Uses change_bds to modify bounds of each pair of points
*           on the wire and then adds wire width to all four bounds
*           allow for the width of the wire. This is all that's re-
*           quired for finding bounds with any directions.
* INPUTS: sp
* OUTPUTS: *box
* GLOBAL VARIABLES USED: notify
* GLOBAL VARIABLES CHANGED: totwire
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: prt_error / get_point / change_bds / ck_bounds
*                 / get_integer / skip_sep / isdigit
* CALLING MODULES: build_at
* HISTORY: N/A
*****/

```

```

struct box      *ck_wire(sp)
char    *sp;
{
    static char    *tp;
    static long    width;
    static struct box    oldbox, newbox;
    static struct point    pt, oldpt;
    static int    done, count;

    tp = sp + 1;
    if (get_integer(sp,&tp,&width)) return(NILL);
    if (get_point(sp,&tp,&pt))
    {
        prt_error(515,sp);
        return(NILL);
    }
    oldbox.xmin = oldbox.xmax = oldpt.x = pt.x;
    oldbox.ymin = oldbox.ymax = oldpt.y = pt.y;
    done = FALSE;
    count = 1;
    while (!done)
    {
        skip_sep(&tp);
        if (isdigit(*tp) || *tp == '-')
        {
            if (get_point(sp,&tp,&pt))
            {
                prt_error(515,sp);
                return(NILL);
            }
            if (oldpt.x != pt.x && oldpt.y != pt.y)

```

```

        {
            prt_error(513,sp);
            return(NILL);
        }

        newbox.xmin = newbox.xmax = oldpt.x = pt.x;
        newbox.ymin = newbox.ymax = oldpt.y = pt.y;
        change_bds(&oldbox,&newbox);
        ++count;
    }
    else done = TRUE;
}

if (count < 2) prt_error(525,sp);
width /= 2;
oldbox.xmin -= width;
oldbox.ymin -= width;
oldbox.xmax += width;
oldbox.ymax += width;
if (ck_bounds(&oldbox))
{
    prt_error(514,sp);
    return(NILL);
}

if (notify) ++totwire;
return(&oldbox);
}

```

```

/*****
* DATE: 18 OCT 1984
* VERSION: 1.0
* NAME: clip_box
* SCNUMBER: 4.5.2.1.1
* FUNCTION: Clips first input box to dimensions of second input
*           input box. If any portion is inside returns TRUE, else
*           FALSE. Assumes boxes with Manhattan directions. Also
*           sets the four sides of the the input cliprec to TRUE
*           for each side that has been NOT been clipped by the
*           procedure.
* INPUTS: box1, box2, cliprec
* OUTPUTS: inside - TRUE if inside, else FALSE
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: proc_call
* HISTORY: N/A
*****/
clip_box(box1,box2,clipped)
struct box    *box1,*box2;
struct cliprec *clipped;
{

```

```

    if ((box1->xmax <= box2->xmin) ||

```

```

        (box1->ymax <= box2->ymin))    return(FALSE);
if ((box1->xmin >= box2->xmax) ||
    (box1->ymin >= box2->ymax))    return(FALSE);

if (box1->xmin < box2->xmin)
    (
        box1->xmin = box2->xmin;
        clipped->left = TRUE;
    )
else clipped->left = FALSE;
if (box1->ymin < box2->ymin)
    (
        box1->ymin = box2->ymin;
        clipped->bottom = TRUE;
    )
else clipped->bottom = FALSE;
if (box1->xmax > box2->xmax)
    (
        box1->xmax = box2->xmax;
        clipped->right = TRUE;
    )
else clipped->right = FALSE;
if (box1->ymax > box2->ymax)
    (
        box1->ymax = box2->ymax;
        clipped->top = TRUE;
    )
else clipped->top = FALSE;

return(TRUE);
)

```

```

/*****
* DATE: 10 OCT 1984
* VERSION: 1.0
* NAME: cpm_file
* SCNUMBER: 2.2.2
* FUNCTION: Verifies that the input string is a valid CPM file
*           name. Returns flag = TRUE if so, else FALSE for error.
*           Checks for:
*               Correct position & number of ':' and '.'
*               Characters "/=;,"
*               Length > 14
*           User areas are NOT expected in the filename
* INPUTS: name
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: par_options
* HISTORY: N/A
*****/

```

```

    if (record < 127) seek(fp->channel,offset,0);
    else
    {
        seek(fp->channel,record,3);
        if (record < 255) offset = offset * 256;
        seek(fp->channel,offset,1);
    }
}

```

```

/*****
* DATE: 26 OCT 1984
* VERSION: 1.0
* NAME: find_sym
* SCNUMBER: 3.2.7.1
* FUNCTION: Find pointer to symbol in the symbol table, NILL
*           returned for not found.
* INPUTS: sym_number
* OUTPUTS: *symbol
* GLOBAL VARIABLES USED: table_top
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: ck_start
* HISTORY: N/A
*****/
struct symbol *find_sym(symnum)
long symnum;
{
    static struct symbol *pt;

    if (table_top == NILL) return(NILL);
    pt = table_top;
    while (pt != NILL)
    {
        if (pt->number == symnum) return(pt);
        pt = pt->next;
    }
    return(NILL);
}

```

```

static int      temp;
static struct file *fp;

temp = fopen(name,mode);
if (temp == 0) return(NILL);
fp = ((struct file *) sbrk(sizeof(struct file)));
if (fp == -1) prt_error(5003,NILL);
strcpy(fp->name,name);
strcpy(fp->mode,mode);
fp->channel = temp;
return(fp);
}

/*****
* DATE: 21 NOV 1984
* VERSION: 1.0
* NAME: file_seek
* SCNUMBER: 2.2.5.1
* FUNCTION: Does a random access in the input file to the offset
*           specified by the input record and offset. If the record
*           is greater than 255, then the input offset is interpreted
*           as a byte position in the current record.
*           If the record > 255 then seek is done to the specified
*           record and then another seek is done to move the pointer
*           to the correct location in the record
* NOTE: THE SEEK (VERSION DATED 1/28/84) FUNCTION CALLED
* BY THIS PROCEDURE HAS BEEN MODIFIED AS FOLLOWS:
*
* extern char IOch[];
*
* ADDED AFTER THE LINE: extern int IOfcb[];
*
* IOch[fdes] = 0;
*
* ADDED AFTER THE LINE: #ifdef CPM
* IN THE pos PROCEDURE
*
* INPUTS: fp,offset, record
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: fp - the input file is closed, opened, accessed
* FILES WRITTEN: none
* MODULES CALLED: seek
* CALLING MODULES: tracedown / proc_call
* HISTORY: N/A
*****/
file_seek(fp,offset,record)
struct file *fp;
int offset, record;
{
    char c,getc();
    int top, newoffset, newsec, sec, i;

```

```

        {
            oldbox.xmin = oldpt.x;
            oldbox.ymin = oldpt.y;
            oldbox.xmax = pt.x;
            oldbox.ymax = pt.y;
        }
    else
    {
        oldbox.xmin = pt.x;
        oldbox.ymin = pt.y;
        oldbox.xmax = oldpt.x;
        oldbox.ymax = oldpt.y;
    }

    oldbox.xmin -= width;
    oldbox.ymin -= width;
    oldbox.xmax += width;
    oldbox.ymax += width;
    trans_box(&oldbox,trans);
    if (clip_box(&oldbox,&pwindow,&clipped))
    {
        trantoplot(&oldbox);
        put_box(&oldbox,curlayer,&clipped);
    }

    oldpt.x = pt.x;
    oldpt.y = pt.y;
}
else done = TRUE;
}

```

```

/*****
* DATE: 3 OCT 1984
* VERSION: 1.0
* NAME: file_open
* SCNUMBER: 2.2.4
* FUNCTION: Opens the named file in the correct mode and returns
*           a pointer to a file structure if no errors occur, else
*           returns NULL. Allocates space for a file structure and
*           places the filename, channel, and mode in the structure.
*           Allocation routine from K&R.
*           No error checking is done on the input name and mode.
* INPUTS: name, mode
* OUTPUTS: fp (pointer to a file structure)
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none (file is opened though)
* FILES WRITTEN: none
* MODULES CALLED: fopen / sbrk / strcpy / prt_error
* CALLING MODULES: par_options / open_pfile
* HISTORY: N/A
*****/
struct file *file_open(name,mode)
char *name, *mode;
{

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: do_wire
* SCNUMBER: 4.5.2.2
* FUNCTION: Reads input Wire command and determines the bounds
*           of the wire. For each two points of the wire, outputs a
*           a wire of the correct bounds until the last two points
*           are found. Modifies the boxes according to the input
*           scale and transformation. Clips each box to the current
*           page dimensions and calls put_box to output the box to
*           the PLOT file.
* INPUTS: com_string, trans, curlayer, symscale
* OUTPUTS: none
* GLOBAL VARIABLES USED: pwindow
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: get_integer / get_point / skip_sep / isdigit
*                   clip_box / trans_box / trantoplot / put_box
* CALLING PROCEDURES: proc_call / do_page
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

```

do_wire(sp,trans,curlayer,symscale)
char    *sp;
long    trans[];
int     curlayer;
float   symscale;
{
    static char    *tp;
    static long    width;
    static struct box oldbox, newbox;
    static struct point pt, oldpt;
    static int     done;
    static struct cliprec clipped;

    tp = sp + 1;
    get_integer(sp,&tp,&width);
    width = (long) width * symscale / 2;
    get_point(sp,&tp,&oldpt);
    oldpt.x = (long) oldpt.x * symscale;
    oldpt.y = (long) oldpt.y * symscale;
    done = FALSE;
    while (!done)
    {
        skip_sep(&tp);
        if (isdigit(*tp) || *tp == '-')
        {
            get_point(sp,&tp,&pt);
            pt.x = (long) pt.x * symscale;
            pt.y = (long) pt.y * symscale;
            if (oldpt.y < pt.y || oldpt.x < pt.x)

```



```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: do_round
* SCNUMBER: 4.5.2.3
* FUNCTION: Reads input Roundflash command and determines its
*           bounds (roundflash commands are output as square boxes.
*           Modifies bounds according to the input scale
*           and transformation. Clips the box to the current page
*           and calls put_box to output the box to the PLOT file.
* INPUTS: com_string, trans, curlayer, symscale
* OUTPUTS: none
* GLOBAL VARIABLES USED: pwindow
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: get_integer / get_point / clip_box /
*                   trans_box / trantoplot / put_box
* CALLING PROCEDURES: proc_call / do_page
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

```

do_round(sp,trans,curlayer,symscale)
char    *sp;
long    trans[];
int     curlayer;
float   symscale;
{
    static char    *tp;
    static long    diam;
    static struct box    newbox;
    static struct point  pt;
    static struct cliprec clipped;

    tp = sp + 1;
    get_integer(sp,&tp,&diam);
    diam = (long) diam * symscale / 2L;
    get_point(sp,&tp,&pt);
    pt.x = (long) pt.x * symscale;
    pt.y = (long) pt.y * symscale;
    newbox.xmin = pt.x - diam;
    newbox.ymin = pt.y - diam;
    newbox.xmax = pt.x + diam;
    newbox.ymax = pt.y + diam;
    trans_box(&newbox,trans);
    if (clip_box(&newbox,&pwindow,&clipped))
    {
        trantoplot(&newbox);
        put_box(&newbox,curlayer,&clipped);
    }
}

```

```

(
switch (*sp)
{
case 'B':
do_box(sp,trans,curlayer,1.0);
break;
case 'W':
do_wire(sp,trans,curlayer,1.0);
break;
case 'R':
do_round(sp,trans,curlayer,1.0);
break;
/* case 'P': */
/* do_poly(sp,trans,curlayer,1.0); */
/* break; */
case 'L':
curlayer = get_layer(sp);
break;
case 'C':
curdepth = 1;
proc_call(sp,trans,1.0);
break;
case '1':
do_1com(sp);
break;
/* case '2': */
/* if(text) do_2com(sp,trans,1.0); */
/* break; */
case '9':
if (pointname) do_9xcom(sp,trans,NILL,1.0);
break;
}
}
}
fclose(exec_file->channel);
)

```

```

        puts(" ");
        puts(litoa(wxmax,constore));
        puts(" ");
        puts(litoa(wymin,constore));
        puts(" ");
        puts(litoa(wymax,constore));
        puts("\n");
    }
    puts("input CIF file-> ");
    puts(cif_file->name);
    puts("\n");
    if(outfile)
    {
        puts("specified output file-> ");
        puts(outfname);
        puts("\n");
    }
}

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: do_page
* SCNUMBER: 4.5.2
* FUNCTION: Reads in commands from the EXECUTABLE file. All
*           executable commands are output directly to the PLOT
*           file if they fall within the current page boundaries. All
*           Call commands cause a call to proc_call with the
*           current(input) transformation
*           The EXECUTABLE file is closed on exit.
* INPUTS: trans
* OUTPUTS: none
* GLOBAL VARIABLES USED: extension, pointname
* GLOBAL VARIABLES CHANGED: exec_file, curdepth
* FILES READ: EXECUTABLE
* FILES WRITTEN: none
* MODULES CALLED: fopen / get_ex_com / get_layer / proc_call
*                 do_9xcom / do_box / do_wire / do_lcom / do_round
*                 fclose / isdigit
* CALLING MODULES: plot_page
* HISTORY: N/A
*****/
do_page(trans)
float  trans[];
{
    static int      curlayer;
    static char     *sp;
    char            *get_ex_com();

    exec_file->channel = fopen(exec_file->name,"r");
    while (*(sp = get_ex_com(exec_file)) != 'E')
    {
        if (isdigit(*sp) && !extension);
        else

```

```

if (makepage) puts(n1);
else puts(n2);
puts("accepting user extensions");
if (extension) puts(n1);
else puts(n2);
puts("treating comments as spaces");
if (comments) puts(n1);
else puts(n2);
puts("writing pointnames");
if (pointname) puts(n1);
else puts(n2);
puts("writing symbolnames");
if (symbolname) puts(n1);
else puts(n2);
/* puts("writing extension 2 text");
if (text) puts(n1);
else puts(n2);
puts("drawing layer outline");
if (outline) puts(n1);
else puts(n2);
puts("drawing symbol bounding box");
if (bbox) puts(n1);
else puts(n2);
if (banner)
{
    puts("plot banner specified-> ");
    puts(bantext);
}
if (transfer)
{
    puts("transfer control to ");
    puts(trname);
    puts(" when done\n");
}
puts ("symbol depth limit-> ");
puts(ltoa(depth,comstore));
puts("\n");
if (grid)
{
    puts("draw grid with spacing-> ");
    puts(ltoa(gridapace,comstore));
    puts("\n");
}
if (scale)
{
    puts("new scale specified-> ");
    ftoa('F',7,scaenum,comstore);
    puts(comstore);
    puts("\n");
}
if (window)
{
    puts("window specified-> ");
    puts(ltoa(wxmin,comstore));

```

```

        return;
    )
    if (pnum == 1)
    (
        dummy = ltoa(gridspace,&out[6]);
        strcat(out,"\015\012");
        pc_text(out,plot_file->channel);
    )
    times = (int) test;
    bottom = (float) viewport * (ydif - xdif);
    pc_color(BLACK,plot_file->channel);
    for(i=1; i<=times; ++i)
    (
        test = gridspace * i;
        increment = (float) viewport * test;
        if (test <= pwindow.xmax)
        pc_draw(0.0,(1.0-increment),1.0,(1.0-increment),plot_file->channel);
        pc_draw(increment,bottom,increment,1.0,plot_file->channel);
    )
}

/*****
* DATE: 25 OCT 84
* VERSION: 1.0
* NAME: do_notify
* SCNUMBER: 2.3
* FUNCTION: Displays status of Control Variables to Standard out.*
* INPUTS: none
* OUTPUTS: none
* GLOBAL VARIABLES USED: Control variables
* GLOBAL VARIABLES CHANGED: comstore
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: puts / ltoa / ftoa
* CALLING MODULES: int_command
* HISTORY: N/A
*****/
do_notify()
{
    static int      n1 = " is ENABLED\n";
    static int      n2 = " is SUPPRESSED\n";
    char            *ltoa();

    puts("\ninteractive plotting");
    if (interactive) puts(n1);
    else puts(n2);
    puts("plot rotation");
    if (rotate) puts(n1);
    else puts(n2);
    puts("quiet mode");
    if (quiet) puts(n1);
    else puts(n2);
    puts("make one page plot");
}

```

```

        dy = pt.y;
    }
    length = length * symscale;
    width = width * symscale;
    cx = cx * symscale;
    cy = cy * symscale;
    box_bounded(length,width,cx,cy,dx,dy,&newbox);
    trans_box(&newbox,trans);
    if (clip_box(&newbox,&pwindow,&clipped))
    {
        trantoplot(&newbox);
        put_box(&newbox,curlayer,&clipped);
    }
}

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: do_grid
* SCNUMBER: 4.5.1
* FUNCTION: Writes a grid separated by gridspace to the PLOT
*           file. Calculations are simplified because a square grid
*           is written to a square page. The number of grid lines
*           is calculated (long division truncates the fractional
*           part) and used to count the number of lines drawn on
*           the plot. The transformation to PLOT.COM's coordinates
*           is made by multiplying by viewport and subtracting the
*           y value in the x direction grid line from 1.0.
* INPUTS: pnum - page number being plotted
* OUTPUTS: none
* GLOBAL VARIABLES USED: cif_file, gridspace, viewport, pwindow
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: pc_color / pc_draw / pc_text / ltoa / strcat
* prt_error
* CALLING MODULES: plot_page
* HISTORY: N/A
*****/
do_grid(pnum)
int    pnum;
{
    static int    times, i;
    static float  bottom, increment;
    static long   ydif, xdif, test;
    static char   dummy, out[20] = "Grid: ";
    char          *ltoa();

    ydif = pwindow.ymax - pwindow.ymin;
    xdif = pwindow.xmax - pwindow.xmin;
    test = ydif / gridspace;
    if (test > 2200)
    {
        if (pnum == 1) prt_error(2000,NILL);
    }
}

```

```

        xc = (float) viewport * tbox.xmin;
        yc = (float) viewport * tbox.ymin;
        pc_string(xc,yc,sp,plot_file->channel);
    }
}

```

```

/*****
* DATE:  9 NOV 84
* VERSION: 1.0
* NAME: do_box
* SCNUMBER: 4.5.2.1
* FUNCTION: Reads input BOX command and determines the bounds of
*           of the box. Modifies bounds according to the input scale
*           and transformation. Clips the box to the current page
*           and calls put_box to output the box to the PLOT file.
* INPUTS: com_string, trans, curlayer, symscale
* OUTPUTS: none
* GLOBAL VARIABLES USED: pwindow
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: get_integer / get_point / skip_sep / isdigit
*                   box_bounds / clip_box / trans_box / trantoplot / put_box
* CALLING PROCEDURES: proc_call / do_page
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

```
do_box(sp,trans,curlayer,symscale)
```

```

char    *sp;
long    trans[];
int     curlayer;
float    symscale;
(
    static int             inside;
    static char            *tp;
    static long            length, width, cx, cy, dx, dy;
    static struct box      newbox;
    static struct point    pt;
    static struct cliprec  clipped;

    dx = 1L;
    dy = 0L;
    tp = sp + 1;
    get_integer(sp,&tp,&length);
    get_integer(sp,&tp,&width);
    get_point(sp,&tp,&pt);
    cx = pt.x;
    cy = pt.y;
    skip_sep(&tp);
    if (isdigit(*tp) || *tp == '-')
    {
        get_point(sp,&tp,&pt);
        dx = pt.x;
    }

```

```

static long          half, temp, width;
static struct box    tbox;
static int           four, newcount, count;
static struct cliprec clipped;
static char          *tp;
static float         xc, yc;
static struct point   pt;

width = (pwindow.ymax - pwindow.ymin) / PLOTCHARS;
++sp;
four = FALSE;
if (*sp == '4')
{
    ++sp;
    four = TRUE;
}
while (isspace(*sp)) ++sp;
tp = sp;
while (*tp != ';' && !(isspace(*tp))) ++tp;
*tp = NULL;
count = strlen(sp);
if (four)
{
    ++tp;
    get_point(sp, &tp, &pt);
    pt.x = (long) pt.x * scale;
    pt.y = (long) pt.y * scale;
    tbox.xmin = tbox.xmax = pt.x;
    tbox.ymin = tbox.ymax = pt.y;
}
else
{
    tbox.xmin = sypt->sxmin;
    tbox.xmax = sypt->sxmax;
    tbox.ymin = sypt->symin;
    tbox.ymax = sypt->syymax;
}
trans_box(&tbox, trans);
if (clip_box(&tbox, &pwindow, &clipped))
{
    trantoplot(&tbox);
    if (four) tbox.xmax = pwindow.ymax;
    temp = (tbox.xmax - tbox.xmin) / width;
    newcount = (int) temp;
    if (count > newcount) *(sp + newcount) = NULL;
    if (temp > 0)
    {
        if (!four)
        {
            tbox.xmin += width;
            tbox.ymin -= ((3 * width) / 2);
            if (tbox.xmin < pwindow.xmin) tbox.xmin = pwindow.xmin;
            if (tbox.ymin < pwindow.ymin) tbox.ymin = pwindow.ymin;
        }
    }
}

```



```

/*****
* DATE: 26 SEP 1984
* VERSION: 1.0
* NAME: do_lcom
* SCNUMBER: 3.2.8
* FUNCTION: Displays message from User Extension 1 to std out
* INPUTS: sp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: puts
* CALLING MODULES: bld_st / wrt_ex_file
* HISTORY: N/A
*****/

```

```

do_lcom(sp)
char *sp;
{
    if (notify) ++totextension;
    puts(sp);
    puts("\n");
}

```

```

/*****
* DATE: 9 NOV 84
* VERSION: 1.0
* NAME: do_9xcom
* SCNUMBER: 4.5.2.6
* FUNCTION: Writes the appropriate PLOT astring command to the
*           PLOT file for text specified by the CIF 9 or 94 extension. The text for a 9 command is clipped to the symbol
*           bounding box in the ydirection (after transformations are applied) and is not clipped in the x direction. 94
*           commands are clipped to the page boundaries only. All
*           text prints in the y direction, regardless of transformations.
* INPUTS: sp, trans, sypt, scale
* OUTPUTS: none
* GLOBAL VARIABLES USED: pwindow, viewport
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: isspace / pc_string / trantoplot / clip_box
*                   trans_box / strlen / get_point
* CALLING PROCEDURES: proc_call / do_page
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

```

do_9xcom(sp,trans,sypt,scale)
char *sp;
float scale, trans[];
struct symbol *sypt;
{

```

```

cpm_file(name)
char *name;
{
    static int place, pplace, cplace, period, colon;
    static char *sp;

    sp = name;
    period = pplace = colon = cplace = 0;
    place = 0;
    while (*sp)
    {
        if (++place > 14) return(FALSE);
        if (*sp == '/' ||
            *sp == '=' ||
            *sp == ';' ||
            *sp == ',' ) return(FALSE);
        if (*sp == '.')
        {
            ++period;
            pplace = place;
        }
        else if (*sp == ':')
        {
            ++colon;
            cplace = place;
        }
        ++sp;
    }
    if (period > 1 || colon > 1) return(FALSE);
    if (colon)
    {
        if (cplace != 2) return(FALSE);
        if (pplace > 11 || pplace == 3) return(FALSE);
        if (!period && place > 10) return(FALSE);
    }
    if (period)
    {
        if (pplace == 1) return(FALSE);
        if (!colon && pplace > 9) return(FALSE);
        if ((place - pplace) > 4) return(FALSE);
    }
    if (!colon && !period && place > 8) return(FALSE);
    return(TRUE);
}

```

```

/*****
* DATE: 26 OCT 1984
* VERSION: 1.0
* NAME: fskp_blank
* SCNUMBER: 3.1.2
* FUNCTION: Skips CIF blank characters in file specified by fp
*           and returns first non-blank character.
* INPUTS: fp
* OUTPUTS: nonblank
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: fp (CIF)
* FILES WRITTEN: none
* MODULES CALLED: isdigit / isupper / getc / add_line
* CALLING MODULES: get_cif_com
* HISTORY: N/A
*****/
char fskp_blank(fp)
struct file *fp;
{
    static char c;
    char getc();

    while ((c=(getc(fp->channel))) != EOF)
    {
        if (c == NEWLINE && !end) add_line();
        if (c == '-' ||
            c == '(' ||
            c == ')' ||
            c == ';' ||
            isdigit(c) ||
            isupper(c)) break;
    }
    return(c);
}

/*****
* DATE: 26 SEP 1984
* VERSION: 1.0
* NAME: fskp_comment
* SCNUMBER: 3.1.1
* FUNCTION: Skips all characters until ')', if EOF is found
*           then fatal error produced. Recursively calls itself if
*           '(' is encountered before end.
* INPUTS: fp
* OUTPUTS: none
* GLOBAL VARIABLES USED: end
* GLOBAL VARIABLES CHANGED: none
* FILES READ: fp (usually CIF file)
* FILES WRITTEN: none
* MODULES CALLED: getc / prt_error / fskp_comment / add_line
* CALLING MODULES: get_cif_com / fskp_comment
* HISTORY: N/A
*****/

```

```

fskp_comment(fp)
struct file      *fp;
{
    static char    tempc;
    char          getc();

    while ((tempc = getc(fp->channel)) != EOF && tempc != '\n')
    {
        if (tempc == '(') fskp_comment(fp);
        if (tempc == NEWLINE && !end) add_line();
    }
    if (tempc == EOF) prt_error(10,NILL);
}

/*****
* DATE: 25 OCT 1984
* VERSION: 1.0
* NAME: get_cad_opt
* SCNUMBER: 2.2
* FUNCTION: Opens CADRC file (if it exists) and creates two
*           parameters with the same form as of argv and argc which
*           are passed to par_options to modify the Control
*           Variables as required. Options are parsed which follow
*           the word mcifplot in the CADRC file.
* INPUTS: none
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: CADRC
* FILES WRITTEN: none
* MODULES CALLED: fopen / getc / index / fclose / par_options
*                isalpha / isspace
* CALLING MODULES: int_options
* HISTORY: N/A
*****/
get_cad_opt()
{
    static int      found, i, chan, cargc, done;
    static char     *pt, c, *cargv[CADOPTMAX];
    char            getc();

    chan = fopen(CADNAME,"r");
    if (chan == 0) return;
    found = done = FALSE;
    while (!done)
    {
        i = 0;
        while((c = getc(chan)) != EOF)
        {
            if (i > 127)
            {
                while((c = getc(chan)) != EOF && c != '\n');
            }
            constore[i] = c;

```

```

        if (c == '\n' || c == EOF)
        {
            constore[i] = NULL;
            break;
        }

        ++i;
    }

    i = index(constore, "acifplot");
    if (i == 0) found = done = TRUE;
    if (c == EOF) done = TRUE;
}

if (!found) return;
cargc = i = 0;
pt = &constore[0];
while (*pt)
{
    cargv[i] = pt;
    if (*pt == '"')
    {
        ++pt;
        cargv[i] = pt;
        while (*pt != '"' && pt) ++pt;
        if (pt == '"')
        {
            *pt = NULL;
            ++pt;
        }
    }

    while (isalpha(*pt) ||
           isdigit(*pt) ||
           *pt == '.' ||
           *pt == '-' ||
           *pt == ':' ||
           *pt == ',' ) ++pt;

    if (*pt != NULL)
    {
        *pt = NULL;
        ++pt;
    }

    while (isspace(*pt)) ++pt;
    ++cargc;
    if (++i >= CADOPTMAX) break;
}

per_options(cargc, cargv, TRUE);
fclose(chan);

```

```

/*****
* DATE: 5 NOV 1984
* VERSION: 1.0
* NAME: get_cif_com
* SCNUMBER: 3.1
* FUNCTION: Gets one CIF command from the input file structure.
*           Comment commands are filtered out and the file is read
*           sequentially on each subsequent call until EOF is
*           found (NULL returned then). If the CIF command is
*           outside of a symbol (outsym = TRUE), then com_offset and
*           com_record are set to the current read pointer.
* INPUTS: fp, outsym
* OUTPUTS: com_string
* GLOBAL VARIABLES USED: end
* GLOBAL VARIABLES CHANGED: com_offset, com_record, totcommand
*           totccomment
* FILES READ: fp (in most cases the CIF file)
* FILES WRITTEN: none
* MODULES CALLED: getc / fskp_comment / fskp_comment / prt_error
*           ftell / ftellr / add_line
* CALLING MODULES: parse_cif
* HISTORY: N/A
*****/
char *get_cif_com(fp,outsym)
struct file *fp;
int outsym;
{
    static char tempc;
    static int i;
    char getc(), fskp_blank();

    comstore[0] = NULL;
    while ((tempc = (fskp_blank(fp))) == '(')
    {
        fskp_comment(fp);
        while ((tempc = getc(fp->channel)) != EOF && tempc != ';'')
        {
            if (notify && !end)
            {
                ++totcommand;
                ++totccomment;
            }
            if (tempc == EOF) return(NULL);
        }
    }
    if (outsym)
    {
        com_offset = ftell(fp->channel);
        com_record = ftellr(fp->channel);
    }

    i = 0;
    while (tempc != EOF && tempc != ';'')
    {
        if (tempc == NEWLINE && !end) add_line();
        if (i < (MAXCOMLEN - 2))
        {

```

```

        if (comments == TRUE && tempc == '(') fskip_comment(fp);
        else comstore[i++] = tempc;
        tempc = getc(fp->channel);
    }
    else
    {
        while ((tempc = getc(fp->channel)) != EOF && tempc != ';')
        {
            if (tempc == NEWLINE && !end) add_line();
        }
        comstore[i++] = tempc;
        prt_error(522,comstore);
    }
}
if (tempc != EOF)
{
    comstore[i] = tempc;
    comstore[++i] = NULL;
    if (notify && !end) ++totcommand;
}
else comstore[i] = NULL;
return(comstore);
}

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: get_ex_com
* SCNUMBER: 4.1.1
* FUNCTION: Gets the next command from the EXECUTABLE file into
*           a string array and returns it. The last command is
*           always the CIF 'E' command.
* INPUTS: fp - pointer to EXECUTABLE file's file structure
* OUTPUTS: com_string - one CIF command from EXECUTABLE file
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: comstore
* FILES READ: EXECUTABLE file
* FILES WRITTEN: none
* MODULES CALLED: getc / fskip_blank
* CALLING MODULES: trace_table
* HISTORY: N/A
*****/

```

```

char *get_ex_com(fp)
struct file *fp;
{
    static char c;
    static int i;
    char fskip_blank(), getc();

    comstore[0] = fskip_blank(fp);
    if (comstore[0] == 'E')
    {
        comstore[1] = NULL;
        return(comstore);
    }
}

```

```

    )
    i = 1;
    while ((c = getc(fp->channel)) != ';' && c != EOF)
    {
        comstore[i] = c;
        ++i;
    }
    if (c == ';')
    {
        comstore[i] = c;
        ++i;
    }
    comstore[i] = NULL;
    return(comstore);
}

```

```

/*****
* DATE: 30 SEP 1984
* VERSION: 1.0
* NAME: get_integer
* SCNUMBER: 3.2.1.1
* FUNCTION: Gets one integer from the input string and returns
*           error = TRUE if errors are found. Errors are reported
*           for bad integer reads, signed integers and integers
*           out of range. Modifies the input pointer (the pointer
*           to its storage location is passed) to point to the first
*           non-integer character after the integer that is read.
* INPUTS: com_string, sploc, tint
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: isdigit / atoi / ck_range / prt_error
*               skip_sep
* CALLING MODULES: ck_delete / ck_box
* HISTORY: N/A
*****/

```

```

get_integer(com_string,sploc,tint)
char    *com_string, **sploc;
long    *tint;
{
    long    atoi();
    static char    ascii[11];
    static int      i;

    skip_sep(sploc);
    if (**sploc == '-')
    {
        prt_error(507,com_string);
        return(TRUE);
    }

    i = 0;

```



```

while (isdigit(**sploc) && i < 10)
{
    ascii[i++] = **sploc;
    ++(*sploc);
}
if (i == 0)
{
    prt_error(506,com_string);
    return(TRUE);
}
else ascii[i] = NULL;
*tint = atoi(ascii);
if (ck_range(*tint))
{
    prt_error(508,com_string);
    return(TRUE);
}
return(FALSE);
}

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: get_layer
* SCNUMBER: 4.5.2.4
* FUNCTION: Gets string from layer command and compares it with
*           global layer[] list. When found returns the matching
*           global color[].
* INPUTS: sp
* OUTPUTS: color
* GLOBAL VARIABLES USED: color[], layer[]
* GLOBAL VARIABLES CHANGED:
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: isupper / isdigit / skip_blank
* CALLING MODULES: do_page
* HISTORY: N/A
*****/
get_layer(sp)
char *sp;
{
    static int found, limit, i, j;
    static char shortname[4];

    ++sp;
    skip_blank(&sp);
    i = 0;
    while ((isupper(*sp) || isdigit(*sp)) && (i < 4))
    {
        limit = i;
        shortname[i++] = *sp;
        ++sp;
    }
    for (i=0; i<=LAYERMAX; ++i)

```

```

    {
        for (j=0; j<=limit; ++j)
        {
            if (layer[i][j] == shortname[j]) found = TRUE;
            else found = FALSE;
        }
        if (found) break;
    }
    return(color[i]);
}

```

```

/*****
* DATE: 7 NOV 1984
* VERSION: 1.0
* NAME: get_pattern
* SCNUMBER: 2.2.3
* FUNCTION: Opens the PATTERN file and reads in new layer names
*           and PLOT.COM colors. Up to 15 new layers can be speci-
*           fied. Fatal errors are generated if the PATTERN file
*           does not exist or if it contains incorrect syntax.
* INPUTS: name
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: color, layer, patstring
* FILES READ: PATTERN
* FILES WRITTEN: none
* MODULES CALLED: fopen / getc / fclose / prt_error / isspace
*                 isdigit / sbrk
* CALLING MODULES: par_options
* HISTORY: N/A
*****/

```

```

get_pattern(name)
char    *name;
{
    static int    done, tcolor, skip, count, j, i, chan;
    char          stg[7], c, getc(), type;

    chan = fopen(name,"r");
    if (chan == 0) prt_error(102,name);
    type = getc(chan);
    getc(chan);
    patstring = sbrk(125);
    if (patstring == -1) prt_error(5004,NILL);
    if (type == '1')
    {
        count = 0;
        done = FALSE;
        while (count < LAYERMAX && !done)
        {
            skip = FALSE;
            for (i=0; i<7; ++i) stg[i] = NULL;
            i = 0;
            while((c = getc(chan)) != EOF)
            {

```

```

        if (i == 0 && c == ';' )
        {
            while((c = getc(chan)) != EOF && c != NEWLINE);
            skip = TRUE;
            break;
        }
        if (c == NEWLINE)
        {
            patstring[i] = NULL;
            break;
        }
        patstring[i] = c;
        if (++i > 100) prt_error(14,NILL);
    }
    if (c == EOF) done = skip = TRUE;
    i = 0;
    while(!skip && !isspace(patstring[i]))
    {
        if (i > 3) prt_error(17,NILL);
        layer[count][i] = patstring[i];
        i++;
    }
    if (i < 3) for (j=i; j<=3; ++j) layer[count][j] = NULL;
    while(!skip && isspace(patstring[i])) ++i;
    j = 0;
    while(!skip && isdigit(patstring[i]))
    {
        if (j > 2) prt_error(17,NILL);
        stg[j] = patstring[i];
        ++i;
        ++j;
    }
    if (!skip)
    {
        tcolor = atoi(stg);
        if (tcolor < 0 || tcolor > 127) prt_error(17,NILL);
        else color[count] = tcolor;
        ++count;
    }
    }
    else prt_error(17,NILL);
}

```

```

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: get_point
* SCNUMBER: 3.2.1.2
* FUNCTION: Gets one point from the input string and increments
*           (via pointer) the input string pointer to point to the
*           first non-digit after the point. The point is stored
*           indirectly in the input point structure. Reports errors
*           by returning error = TRUE;
* INPUTS: com_string, sploc, point
* OUTPUTS: error
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: isdigit / skip_sep / get_integer
* CALLING MODULES: ck_box
* HISTORY: N/A
*****/
get_point(com_string,sploc,pt)
char      *com_string, **sploc;
struct point *pt;
{
    static long    tint;
    static char    ascii[11];
    static int     sign;

    sign = 1;
    skip_sep(sploc);
    if (**sploc == '-')
    {
        sign = -1;
        ++(*sploc);
    }
    else if (!isdigit(**sploc)) return(TRUE);
    if (get_integer(com_string,sploc,&tint)) return(TRUE);
    pt->x = tint * sign;
    sign = 1;
    skip_sep(sploc);
    if (**sploc == '-')
    {
        sign = -1;
        ++(*sploc);
    }
    else if (!isdigit(**sploc)) return(TRUE);
    if (get_integer(com_string,sploc,&tint)) return(TRUE);
    pt->y = tint * sign;
    return(FALSE);
}

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: get_tran
* SCNUMBER: 4.1.2.1
* FUNCTION: Gets a new transformation which is the multiplica-
*           tion of an input transformation and a transformation
*           specified by the input CALL command string. A pointer to
*           the location for the new transformation is also passed
*           in and is modified by the procedure. If the input
*           transformation pointer is NILL, then it is not necessary
*           to multiply the new transformation times the old. This
*           allows the process to be used by both trace_table and
*           proc_call. In either case a symbol number is
*           returned. The input scale is used to modify the values
*           found in the CALL command. See Hon & Sequin page 109 for
*           a derivation of the algorithm used.
*           Manhattan directions are assumed to simplify call
*           calculations - the following code would implement the
*           rotation transformation without that restriction.
*           square = sqrt((fx*fx) + (fy*fy));
*           tempa[4] = tempa[0] = fx / square;
*           tempa[1] = fy / square;
*           tempa[3] = -tempa[1];
* INPUTS: sp, trans(input current transformation or NILL)
*         newtrans(location for new output transformation
*         scale(scale from the symbol above)
* OUTPUTS: snum (number of called symbol)
* GLOBAL VARIABLES USED: identity
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: skip_blank / get_point / mult33 / get_integer
* CALLING MODULES: tracedown / proc_call
* HISTORY: N/A
*****/
long  get_tran(sp,scale,tran,newtran)
char  *sp;
float  scale, tran[], newtran[];
{
    static struct point  pt;
    static long          sym;
    static float         tmatrix[9], tempa[9];
    static int           i;

    ++sp;
    get_integer(sp,&sp,&sym);
    for (i=0; i<=8; ++i) tempa[i] = tmatrix[i] = identity[i];
    skip_blank(&sp);
    while (*sp != ';')
    {
        switch (*sp)
        {
            case 'T':

```

```

        ++sp;
        get_point(sp,&sp,&pt);
        tempa[6] = (float) scale * pt.x;
        tempa[7] = (float) scale * pt.y;
        break;
    case 'R':
        ++sp;
        get_point(sp,&sp,&pt);
        if (pt.x == 0L)
        {
            tempa[0] = tempa[4] = 0.0;
            tempa[1] = (pt.y > 0L) ? 1.0 : -1.0;
            tempa[3] = (pt.y > 0L) ? -1.0 : 1.0;
        }
        else
        {
            tempa[1] = tempa[3] = 0.0;
            tempa[0] = (pt.x > 0L) ? 1.0 : -1.0;
            tempa[4] = (pt.x > 0L) ? 1.0 : -1.0;
        }
        break;
    case 'M':
        ++sp;
        skip_blank(&sp);
        if (*sp == 'X') tempa[0] = -1.0;
        else tempa[4] = -1.0;
        ++sp;
        break;
    }
    skip_blank(&sp);
    mult33(tmatrix,tempa,tmatrix);
    for (i=0; i<=8; ++i) tempa[i] = identity[i];
    }
    if (tran != NULL) mult33(tmatrix,tran,tmatrix);
    for (i=0; i<=8; ++i) newtran[i] = tmatrix[i];
    return(sym);
}

```

```

/*****
* DATE: 3 OCT 84
* VERSION: 1.0
* NAME: int_command
* SCNUMBER: 2.0
* FUNCTION: This is a primary module which does the following:
*   Calls get_cad_opt to open the CADRC file (if it
*   exists) and parses the "mcifplot" line (if found) and
*   then modifies the Control Variables as required.
*   Passes the program command line arguments argv and argc
*   to par_options for the additional modification of the
*   Control Variables as required.
*   Notify is called to inform the user of the status of the
*   control variables if notify is true.
*   All errors encountered in the procedures are fatal and
*   do not return control to main.
*****/

```

```

* INPUTS: argc,argv
* OUTPUTS: none
* GLOBAL VARIABLES USED: Control Variables (notify)
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: get_cad_opt \ par_options \ do_notify \ puts
* CALLING MODULES: main
* HISTORY: N/A
...../
int_command(argc,argv)
int      argc;
char     *argv[];
{

    get_cad_opt();
    par_options(argc,argv,FALSE);
    puts(VERSION);
    if (notify) do_notify();

}

/.....
* DATE: 25 OCT 84
* VERSION: 1.0
* NAME: main
* SCNUMBER: 1.0
* FUNCTION: primary module which initializes global variables
*           and calls all other modules:
*           int_command is called to initialize the Control Variables
*           and modify them as required by the CADRC file and com-
*           mand line.
*           parse_cif is called to read the input CIF file and build
*           the symbol table and the EXECUTABLE File
*           bld_file is called to trace the symbol table and make
*           the output PLOT file
* STRUCTURED ENGLISH:
* INPUTS: argc, argv
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: err_flag, cif_file, exec_file
*           del_def, line_number, table_top, plot_file, er_file
*           totbox, totwire, totround, totcall, totsymbol,
*           totextension, totccomment, totlayer, totcommand
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: int_command / parse_cif / bld_file / abort
* CALLING MODULES: none
* HISTORY: N/A
...../
main(argc,argv)
int      argc;
char     *argv[];
{

```

```

struct box          *parse_cif();
static struct box    *ebox;

inmestab = mesfiletest = err_flag = FALSE;
er_file = cif_file = exec_file = table_top = plot_file = NULL;
totbox = totwire = totround = totcall = totsymbol = 0;
totextension = totccomment = totlayer = totcommand = 0;
line_number = 1;

int_command(argc,argv);
if ((ebox = parse_cif()) != NULL) bld_file(ebox);
else abort(1);
abort(3);
)

```

```

/*****
* DATE: 23 OCT 1984
* VERSION: 1.0
* NAME: mult33
* SCNUMBER: 4.1.2.1.1
* FUNCTION: Multiplies two 3X3 matrices together and modifies
*           the specified matrix with the result. The matrix (read
*           array) is numbered as follows:
*
*           | 0  1  2 |
*           | 3  4  5 |
*           | 6  7  8 |
*
* INPUTS: m1 & m2 (matrices to multiplied)
*         mr (pointer to matrix in which to place result)
* OUTPUTS: none ( matrix is modified )
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: get_trans
* HISTORY: N/A
*****/

```

```

mult33(m1,m2,mr)
float  m1[],m2[],mr[];
{
    static float  mt[9];
    static int    i;

    mt[0] = m1[0]*m2[0] + m1[1]*m2[3] + m1[2]*m2[6];
    mt[1] = m1[0]*m2[1] + m1[1]*m2[4] + m1[2]*m2[7];
    mt[2] = m1[0]*m2[2] + m1[1]*m2[5] + m1[2]*m2[8];
    mt[3] = m1[3]*m2[0] + m1[4]*m2[3] + m1[5]*m2[6];
    mt[4] = m1[3]*m2[1] + m1[4]*m2[4] + m1[5]*m2[7];
    mt[5] = m1[3]*m2[2] + m1[4]*m2[5] + m1[5]*m2[8];
    mt[6] = m1[6]*m2[0] + m1[7]*m2[3] + m1[8]*m2[6];
    mt[7] = m1[6]*m2[1] + m1[7]*m2[4] + m1[8]*m2[7];
    mt[8] = m1[6]*m2[2] + m1[7]*m2[5] + m1[8]*m2[8];

```


AD-A151 715

A MICROCOMPUTER-BASED PROGRAM FOR PRINTING CHECK PLOTS
OF INTEGRATED CIRC. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. K S HORTON

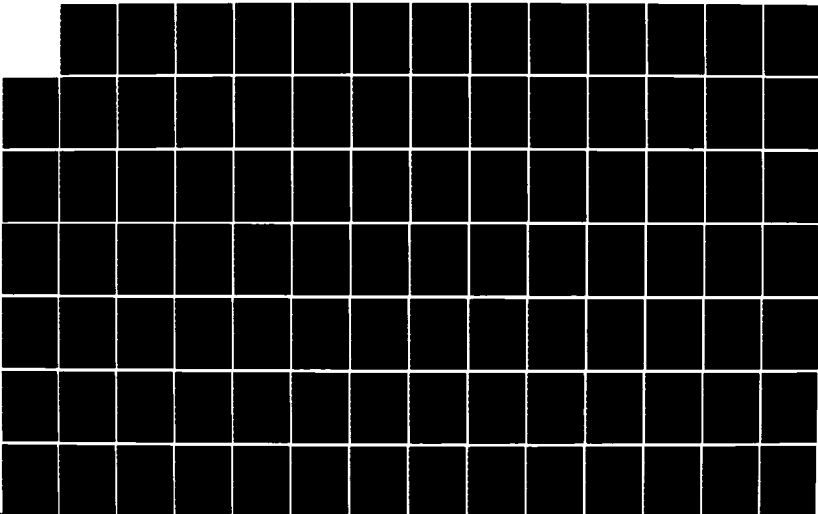
3/4

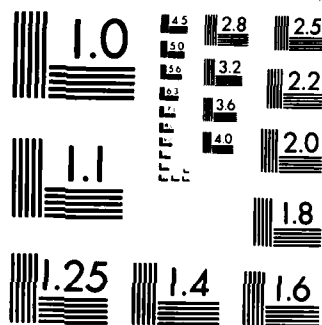
UNCLASSIFIED

DEC 84 AFIT/GE/ENG/84D-35

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

```

        for (i=0; i<=8; ++i) mr[i] = mt[i];
    )

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: open_pfile
* SCNUMBER: 4.4
* FUNCTION: Opens PLOT file for writing PLOT commands. If the
*           output file is on a different disk, gets space left
*           on that drive and puts in charcount (else gets space
*           on default drive). This procedure uses CPM specific
*           BDOS calls.
* INPUTS: none
* OUTPUTS: none
* GLOBAL VARIABLES USED: outfname
* GLOBAL VARIABLES CHANGED: plot_file, charcount, wrtspace
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: get_space / file_open / bdos
* CALLING MODULES: bld_file
* HISTORY: N/A
*****/
open_pfile()
{
    struct file      *file_open();
    static int       newdrive, cdrive, where;

    cdrive = bdos(RETCURDISK,0);
    if (outfname[i] == ':') newdrive = outfname[0] - 'A';
    else newdrive = cdrive;
    if (cdrive != newdrive)
    {
        bdos(SELECTDISK,newdrive);
        charcount = CONVERT * (get_space()) - CLOSEROOM;
        bdos(SELECTDISK,cdrive);
    }
    else charcount = CONVERT * (get_space()) - CLOSEROOM;
    wrtspace = charcount;
    plot_file = file_open(outfname,"wb");
}

```

```

/*****
* DATE: 31 OCT 1984
* VERSION: 1.0
* NAME: par_layer
* SCNUMBER: 2.2.2
* FUNCTION: Parses the layer_list from the command line or the
*           CADRC file and modifies the Control Variables as
*           required. The layer names that are accepted come from
*           the manual for cifplot.
* INPUTS: sp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: Control Variables
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: strcpy / strcmp / prt_error / toupper
* CALLING MODULES: par_options
* HISTORY: N/A
*****/
par_layer(sp)
char *sp;
{
    static char *temp;

    while (*sp != NULL)
    {
        temp = sp;
        while(*sp != ',' && *sp != NULL)
        {
            *sp = toupper(*sp);
            ++sp;
        }
        if (*sp == ',')
        {
            *sp = NULL;
            ++sp;
        }
        if (strcmp(temp,"ALLTEXT") == 0 || strcmp(temp,"AT") == 0)
            text = pointname = symbolname = FALSE;
        else if (strcmp(temp,"POINTNAME") == 0 || strcmp(temp,"PN") == 0)
            pointname = FALSE;
        else if (strcmp(temp,"SYMBOLNAME") == 0 || strcmp(temp,"SN") == 0)
            symbolname = FALSE;
        else if (strcmp(temp,"BBOX") == 0)
            bbox = FALSE;
        else if (strcmp(temp,"OUTLINE") == 0)
            outline = FALSE;
        else if (strcmp(temp,"TEXT") == 0)
            prt_error(11,NILL);
        else prt_error(100,temp);
    }
}

```

```

/*****
* DATE: 25 OCT 1984
* VERSION: 1.0
* NAME: par_options
* SCNUMBER: 2.2
* FUNCTION: Parses the options from the CADRC file and the com-
*           mand line and modifies the Control Variables as required
*           based on Kernighan and Ritchie page 113. The cadrc flag
*           allows the procedure to process the command line and the
*           CADRC file slightly differently.
* INPUTS: argc, argv, cadrc
* OUTPUTS: none
* GLOBAL VARIABLES USED: Control Variables
* GLOBAL VARIABLES CHANGED: Control Variables, cif_file
* FILES READ: CIF (is opened)
* FILES WRITTEN: none
* MODULES CALLED: toupper / atoi / strcpy / strlen / prt_error
*                 ck_range / par_layer / atof / get_pattern / file_open
*                 strcat / cpm_file / index
* CALLING MODULES: int_commands / get_cad_opt
* HISTORY: N/A
*****/
par_options(argc,argv,cadrc)
int    argc, cadrc;
char   *argv[];
{
    static char    tfile[15], *swchar;
    static int     count;
    struct file    *file_open();
    long           atoi();
    float          atof();

    if (!cadrc && argc == 1) prt_error(1,NILL);

    while (--argc > 0 && (**+argv)[0] == '-')
    {
        swchar = argv[0] + 1;
        switch(toupper(*swchar))
        {
            case 'I':
                interactive = FALSE;
                break;
            case 'R':
                rotate = TRUE;
                break;
            case 'G':
                grid = TRUE;
                --argc;
                ++argv;
                gridspace = atoi(*argv);
                if (gridspace <= 0) prt_error(2,NILL);
                if (ck_range(gridspace)) prt_error(3,NILL);
                break;
            case 'E':

```

```

        extension = FALSE;
        break;
case 'C':
    comments = TRUE;
    break;
case 'B':
    banner = TRUE;
    --argc;
    ++argv;
    count = strlen(*argv);
    if (count > (BANLENGTH - 3))
        (*argv)[(BANLENGTH-3)] = NULL;
    strcpy(bantext,*argv);
    strcat(bantext,"\015\012");
    break;
case 'D':
    --argc;
    ++argv;
    depth = atoi(*argv);
    if (depth <= 0) prt_error(4,NILL);
    if (ck_range(depth)) prt_error(3,NILL);
    break;
case 'L':
    --argc;
    ++argv;
    par_layer(*argv);
    break;
case 'M':
    makepage = TRUE;
    break;
case 'N':
    notify = TRUE;
    break;
case 'O':
    --argc;
    ++argv;
    if (cpa_file(*argv)) strcpy(outfname,*argv);
    else prt_error(105,*argv);
    outfile = TRUE;
    break;
case 'Q':
    quiet = TRUE;
    break;
case 'S':
    scale = TRUE;
    --argc;
    ++argv;
    scalenum = atof(*argv);
    if (scalenum <= 0) prt_error(5,NILL);
    break;
case 'T':
    --argc;
    ++argv;
    if (cpa_file(*argv)) strcpy(trname,*argv);

```

```

        else prt_error(105,*argv);
        count = index(*argv,".");
        if (count == -1) strcat(trname, ".COM");
        count = fopen(trname,"r");
        if (count == 0) prt_error(107,*argv);
        strcpy(trname,*argv);
        fclose(count);
        transfer = TRUE;
        break;
case 'W':
    if (!cadrc && argc <= 5) prt_error(8,NILL);
    window = TRUE;
    --argc;
    ++argv;
    wxmin = atoi(*argv);
    --argc;
    ++argv;
    wxmax = atoi(*argv);
    --argc;
    ++argv;
    wymin = atoi(*argv);
    --argc;
    ++argv;
    wymax = atoi(*argv);
    if (ck_range(wxmin) ||
        ck_range(wxmax) ||
        ck_range(wymin) ||
        ck_range(wymax)) prt_error(3,NILL);
    if (wxmax <= wxmin) prt_error(6,NILL);
    if (wymax <= wymin) prt_error(7,NILL);
    break;
case 'P':
    --argc;
    ++argv;
    if (cpa_file(*argv)) get_pattern(*argv);
    else prt_error(105,*argv);
    pattern = TRUE;
    break;
default:
    prt_error(101,*argv);
}
}
if (!cadrc)
{
    if (argc != 1) prt_error(1,NILL);
    count = index(*argv, ".VEC");
    if (count != -1) prt_error(106,*argv);
    if (cpa_file(*argv)) cif_file = file_open(*argv,"r");
    else prt_error(105,*argv);
    if (cif_file == NILL)
    {
        count = index(*argv, ".");
        if (count == -1)
        {

```

```

        strcpy(tfile,*argv);
        strcat(tfile,".CIF");
        cif_file = file_open(tfile,"r");
        if (cif_file == NULL) prt_error(102,*argv);
    }
    else prt_error(102,*argv);
}
if (!outfile)
{
    strcpy(outfname,cif_file->name);
    count = index(outfname,".");
    if (count == -1) strcat(outfname,".VEC");
    else strcpy((outfname + count + 1),"VEC");
}
unlink(outfname);
}
return;
}

```

```

/*****
* DATE: 10 OCT 84
* VERSION: 1.0
* NAME: parse_cif
* SCNUMBER: 3.0
* FUNCTION: Reads all commands from the CIF file and uses them
*           to build the EXECUTABLE file or creates new entries in
*           the symbol table for each symbol. Syntax is checked and
*           build is set = FALSE if an unrecoverable syntax error is
*           found. Returns exbox = NULL if build = FALSE.
* INPUTS: none
* OUTPUTS: exbox
* GLOBAL VARIABLES USED: cif_file
* GLOBAL VARIABLES CHANGED: exec_file, last_symbol, exlayer,
*           end, exbox, charcount
* FILES READ: none
* FILES WRITTEN: EXECUTABLE is opened
* MODULES CALLED: file_open / get_cif_com / bld_at / prt_error
*           wrt_ex_file / get_space / unlink
* CALLING MODULES: main
* HISTORY: N/A
*****/

```

```

struct box      *parse_cif()
{
    char          *get_cif_com();
    static char   *com_string;
    struct file   *file_open();

    last_symbol = NULL;
    exlayer = build = TRUE;
    end = FALSE;
    exbox.xmin = exbox.ymin = INFINITY;
    exbox.xmax = exbox.ymax = -INFINITY;

    unlink(EXNAME);
}

```



```

charcount = CONVERT * (get_space());
exec_file = file_open(EXNAME,"w");
if (exec_file == NULL) prt_error(9,NULL);
while (!end)
{
    com_string = get_cif_com(cif_file,TRUE);
    if (com_string[0] == 'D' && com_string[1] == 'S')
        bld_st(com_string);
    else wrt_ex_file(com_string);
}
if (build) return(&exbox);
else return(NULL);
}

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: plot_page
* SCNUMBER: 4.5
* FUNCTION: Writes PLOT.COM commands to plotfile for plotting
*           one page of the cifplot. If this is the first page, then
*           the plot header (file name, window, and scale) and
*           banner (if one is specified is written to the PLOT file.
*           Next the PLOT commands to erase memory to white are
*           output to the PLOT file. Then do_page is called with
*           either the identity transformation or the rotmatrix
*           to write PLOT commands for one page. Very little
*           error checking is done in this process and the processes
*           it calls, it is assumed that the syntax and semantic
*           errors have been detected prior to attempting output.
*           No warnings are issued if the symbols nested too deep
*           are ignored.
* INPUTS: trans - top level transformation
* OUTPUTS: none
* GLOBAL VARIABLES USED: banner, grid, pban, identity, rotmatrix
*           plot_file, rotate
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: plot_banner / pc_erase / do_page / do_grid
*                 pc_output /
* CALLING MODULES: bld_file
* HISTORY: N/A
*****/

```

```

plot_page(pnum)
int pnum;
{
    static char *sp;

    if (pnum == 1)
    {
        pc_text(pban,plot_file->channel);
        if (banner) pc_text(bantext,plot_file->channel);
    }
}

```

```

        pc_color(WHITE,plot_file->channel);
        pc_erase(plot_file->channel);
        if (rotate) do_page(rotmatrix);
        else do_page(identity);
        if (grid) do_grid(pnum);
        pc_output(plot_file->channel);
    )

```

```

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: proc_call
* SCNUMBER: 4.5.2.5
* FUNCTION: Using the input Call command and transformation
*           determines if the called symbol is within the current
*           page. If so, this symbol is read and each command is
*           output. If call commands are encountered in the symbol,
*           then a recursive call is made to proc_call. The nesting
*           of calls is limited by the value of depth.
* INPUTS: compt, trans, scale
* OUTPUTS: none
* GLOBAL VARIABLES USED: curdepth, depth, symbolname, pointname
*           text, extension, pwindow, bbox, com_offset, com_record
*           deplimit
* GLOBAL VARIABLES CHANGED: curdepth
* FILES READ: PLOT
* FILES WRITTEN: none
* MODULES CALLED: get_layer / proc_call do_9xcom / do_box /
*           do_wire / do_lcom / do_round / get_tran / find_sym
*           trans_box / trantoplot / clip_box / put_box
*           get_cif_com / file_seek / isdigit
* CALLING MODULES: plot_page
* HISTORY: N/A
*****/

```

```

proc_call(compt,trans,scale)

```

```

char    *compt;

```

```

float    scale, trans[];

```

```

(

```

```

    auto float          newtran[9];
    auto long           anum;
    auto struct symbol   *sypt;
    auto struct box      sbounds;
    auto int            curlayer, inside, curoffset, currecord;
    auto struct cliprec  clipped;
    long               get_tran();
    char               *get_cif_com();
    struct symbol        *find_sym();

```

```

    anum = get_tran(compt,scale,trans,newtran);
    sypt = find_sym(anum);
    sbounds.xmin = sypt->sxmin;
    sbounds.ymin = sypt->symin;
    sbounds.xmax = sypt->sxmax;
    sbounds.ymax = sypt->syymax;

```

```

trans_box(&sbounds,newtran);
inside = clip_box(&sbounds,&pwindow,&clipped);
if (!inside)
{
    --curdepth;
    return;
}
if (bbox)
{
    trantoplot(&sbounds);
    put_box(&sbounds,0,&clipped);
}
file_seek(cif_file,sypt->offset,sypt->record);
compt = get_cif_com(cif_file,TRUE);
while(!(*compt == 'D' && *(compt+1) == 'F'))
{
    if (isdigit(*compt) && !extension);
    else
    {
        switch (*compt)
        {
            case 'B':
                do_box(compt,newtran,curlayer,sypt->scale);
                break;
            case 'W':
                do_wire(compt,newtran,curlayer,sypt->scale);
                break;
            case 'R':
                do_round(compt,newtran,curlayer,sypt->scale);
                break;
            /* case 'P': */
            /* do_poly(compt,newtran,curlayer,sypt->scale); */
            /* break; */
            case 'L':
                curlayer = get_layer(compt);
                break;
            case 'C':
                if (++curdepth <= depth && curdepth < deplimit)
                {
                    curoffset = com_offset;
                    currecord = com_record;
                    proc_call(compt,newtran,sypt->scale);
                    file_seek(cif_file,curoffset,currecord);
                    compt = get_cif_com(cif_file,TRUE);
                }
                else --curdepth;
                break;
            case '1':
                do_1com(compt);
                break;
            /* case '2': */
            /* if(text) do_2com(compt,newtran,sypt->scale); */
            /* break; */
            case '9':

```

```

        if ((*compt + 1) == '4' && pointname)
        {
            do_9xcom(compt,newtran,NILL,sypt->scale);
        }
        else if (symbolname) do_9xcom(compt,newtran,sypt,1.0);
        break;
    }
}
compt = get_cif_com(cif_file,TRUE);
}
--curdepth;
return;
}

```

```

/*****
* DATE: 25 OCT 1984
* VERSION: 1.0
* NAME: prt_error
* SCNUMBER: 2.2.5
* FUNCTION: Displays the error message corresponding to the
*           input error number to the standard out. If the MESSAGE
*           file has not been opened, the file is opened and an
*           array allocated to contain the message offsets in the
*           file. Then the message is read in and printed. If the
*           file is not found, only error numbers are displayed.
*           Depending on the type of error, also displays the input
*           the input string. Certain error types set err_flag =
*           TRUE. Fatal errors call abort to exit.
* INPUTS: err_num, sp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: mesfiletest, innestab, err_flag
*           er_file
* FILES READ: MESSAGE
* FILES WRITTEN: none
* MODULES CALLED: puts / abort / itoa / ltoa / file_open /
*           putchar / sbrk / getc / file_seek
* CALLING MODULES: par_options / bld_file / bld_st / ck_9
*           do_grid / ck_wire / ck_start / ck_call / ck_box
*           ck_layer / ck_round
* HISTORY: N/A
*****/

```

```

prt_error(error,sp)
int    error;
char   *sp;
{
    static char    c, *pt, s[12];
    static char    *e1 = "<- Line ";
    static char    *e2 = " -> ";
    static int     temp, messcount, found, i, *newplace, offset;
    char           *itoa(), *ltoa(), getc();
    struct file     *file_open();

    if (error >= 5000)

```

```

    {
        error -= 5000;
        puts("memory allocation error: ");
        puts(itoa(error,s));
        abort(1);
    }
if (!mesfiletest)
    {
        er_file = file_open(MESFILE,"rb");
        if (er_file != NULL)
            {
                pt = &messcount;
                *pt++ = getc(er_file->channel);
                *pt = getc(er_file->channel);
                temp = messcount * 2;
                pt = newplace = sbrk(temp+2);
                if (pt == -1)
                    {
                        puts("error table allocation fails - error: ");
                        puts(itoa(error,s));
                        abort(1);
                    }

                pt += 2;
                for (i=1; i<=temp; ++i) *pt++ = getc(er_file->channel);
                innestab = TRUE;
            }
        else
            {
                puts("\n");
                puts(MESFILE);
                puts(" not found - errors displayed as numbers\n");
            }
        mesfiletest = TRUE;
    }
if (!quiet) puts ("\n");
if (sp != NULL) puts(sp);
if (innestab)
    {
        found = -1;
        for (i=1; i<=messcount; ++i)
            {
                if (newplace[i] == error)
                    {
                        found = i;
                        break;
                    }
            }
        if (found != -1)
            {
                offset = (messcount*2) + 2 + ((found - 1) * 80);
                file_seek(er_file,offset,0);
            }
    }
if (!innestab || found == -1)

```

```

        {
            puts("\nerror: ");
            puts(itoa(error,s));
            puts("\n");
            if (error >= 1000) return;
            if (error >= 500)
            {
                err_flag = TRUE;
                return;
            }
            abort(2);
        }
    if (error < 100)
    {
        while ((c = getc(er_file->channel))) putchar(c);
        if (error == 16 || error == 17) abort(4);
        abort(2);
    }
    else if (error < 200)
    {
        while ((c = getc(er_file->channel))) putchar(c);
        abort(2);
    }
    else if (error < 299)
    {
        puts("\n");
        puts(e1);
        puts(ltoa(line_number,s));
        puts(e2);
        puts("Fatal error: ");
        while ((c = getc(er_file->channel))) putchar(c);
        abort(2);
    }
    else if (error < 399)
    {
        puts("\n");
        while ((c = getc(er_file->channel))) putchar(c);
        abort(2);
    }
    else if (error < 1000)
    {
        puts("\n");
        puts(e1);
        puts(ltoa(line_number,s));
        puts(e2);
        err_flag = TRUE;
    }
    else if (error < 2000)
    {
        puts("\n");
        puts(e1);
        puts(ltoa(line_number,s));
        puts(" -> Warning: ");
    }

```

```

/*****
* DATE: 23 OCT 84
* VERSION: 1.0
* NAME: trantoplot
* SCNUMBER: 4.5.2.1.2
* FUNCTION: Because the PLOT.COM coordinate system is rotated 90
*           degrees from the the cifplot system and all it's coordi-
*           nates are in the first quadrant, several transformations
*           must be made before correct output. The transformations
*           are accomplished using a matrix transformation on the
*           two points in the input box to transform it into the
*           correct coordinate system for output.
*           Three transformations (and 3 matrices) are required:
*
*           | 1      0      0 |
* M1: | 0      1      0 | - translates pxmin, pymin
*           | -pxmin -pymin 0 | to 0,0
*
*           | 0      -1      0 |
* M2: | 1      0      0 | - rotates axis 90 degrees
*           | 0      0      1 |
*
*           | 1      0      0 |
* M3: | 0      1      0 | - translates to first quadrant
*           | 0      Dy      0 | using Dy (delta y)
*
*           M1 * M2 * M3 = | 0      -1      0 |
*                           | 1      0      0 |
*                           | -pymin  pxmin+Dy  1 |
*
*           Because the plot window is always square, and xmin and
*           and ymin have been translated to 0:
*
*           Dy = pymax - pymin
*           pxmin+Dy = pxmin+pymax-pymin
*
*           Because of the number of zeros and ones in the matrix
*           the transformation of a point can be reduced to two
*           equations:
*
*           xout = yin - pymin
*           yout = pxmin + pymax - pymin - xin
*
* INPUTS: bp
* OUTPUTS: modifies input box indirectly via pointer
* GLOBAL VARIABLES USED: pwindow
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: do_box
* HISTORY: N/A
*****/

```

```

*
*   the nine multiplications and six additions can be re-
*   duced to:
*
*       x' = ax + by + c
*       y' = dx + ey + f
*
*   See Newman & Sproull page 60.
*   The last test is required in the instance that a trans-
*   formation causes the box maximum point to be less than
*   the box minimum point.
* INPUTS: trans, box
* OUTPUTS: none(input box is modified via pointer)
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: none
* CALLING MODULES: tracedown
* HISTORY: N/A
*****/
trans_box(bp,trans)
struct box      *bp;
float          trans[];
{
    static long    x, y, save;

    x = (long) trans[0]*bp->xmin + trans[3]*bp->ymin + trans[6];
    y = (long) trans[1]*bp->xmin + trans[4]*bp->ymin + trans[7];
    bp->xmin = x;
    bp->ymin = y;
    x = (long) trans[0]*bp->xmax + trans[3]*bp->ymin + trans[6];
    y = (long) trans[1]*bp->xmax + trans[4]*bp->ymin + trans[7];
    bp->xmax = x;
    bp->ymin = y;
    if (bp->xmax < bp->xmin)
    {
        save = bp->xmax;
        bp->xmax = bp->xmin;
        bp->xmin = save;
    }
    if (bp->ymin < bp->ymin)
    {
        save = bp->ymin;
        bp->ymin = bp->ymin;
        bp->ymin = save;
    }
}

```



```

        if (++curdepth > depth)
        {
            prt_error(3001,(ltoa(snum,stg)));
            downok = FALSE;
        }
        if (&snum < lastalloc)
        {
            prt_error(3000,(ltoa(snum,stg)));
            deplimit = curdepth;
            downok = FALSE;
        }
        if (downok)
        {
            curoffset = com_offset;
            currecord = com_record;
            tracedown(compt,sypt->scale,&dbsounds);
            change_bds(&sounds,&dbsounds);
            file_seek(cif_file,curoffset,currecord);
            compt = get_cif_com(cif_file,TRUE);
        }
        --curdepth;
    }
    compt = get_cif_com(cif_file,TRUE);
}

bp->xmin = sypt->sxmin = sbounds.xmin;
bp->ymin = sypt->symin = sbounds.ymin;
bp->xmax = sypt->sxmax = sbounds.xmax;
bp->ymax = sypt->ymax = sbounds.ymax;
sypt->inuse = NO;
sypt->calls = NO;
trans_box(bp,newtran);
}

```

```

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: trans_box
* SCNUMBER: 4.2.3
* FUNCTION: Using an input transformation and box, this process
*           applies the transformation to each of the points of the
*           box and then modifies the input box accordingly. The
*           input transformation is a 3X3 matrix and the normal
*           procedure would be to multiply a 1X3 matrix which repre-
*           sents each point times the 3X3 transformation. Because
*           the transformation matrix is always in the form:
*
*
*           T =
*
*           | a d 0 |
*           | b e 0 |
*           | c f 1 |
*
*****/

```

```

* OUTPUTS: none
* GLOBAL VARIABLES USED: depth, lastalloc, cif_file, deplimit
*   notify, com_offset, com_record
* GLOBAL VARIABLES CHANGED: curdepth, callcount
* FILES READ: CIF
* FILES WRITTEN: none
* MODULES CALLED: get_tran / find_sym / ltoa / file_seek
*   change_bds / trans_box / prt_error / get_cif_com
*   tracedown / puts
* CALLING MODULES: trace_table
* HISTORY: N/A
*****/
tracedown(sp, scale, bp)
char      *sp;
float     scale;
struct box *bp;
{
    auto float      newtran[9];
    auto struct symbol *sypt;
    auto struct box  dbounds, sbounds;
    auto char        stg[15], *compt;
    auto int          downok, curoffset, currecord;
    auto long         snum;
    long             get_tran();
    char             *get_cif_com(), *ltoa();
    struct symbol     *find_sym();

    if (!quiet && (++callcount % CNOTIFY) == 0)
    {
        puts("\r-> finding bounds: ");
        puts(ltoa(callcount, stg));
        puts(" calls processed ");
    }

    snum = get_tran(sp, scale, NULL, newtran);
    sypt = find_sym(snum);
    if (sypt == NULL) prt_error(204, sp);
    if (sypt->inuse == YES) prt_error(103, (ltoa(snum, stg)));
    bp->xmin = sbounds.xmin = sypt->axmin;
    bp->ymin = sbounds.ymin = sypt->aymin;
    bp->xmax = sbounds.xmax = sypt->axmax;
    bp->ymax = sbounds.ymax = sypt->aymax;
    if (sypt->calls == NO)
    {
        trans_box(bp, newtran);
        return;
    }

    sypt->inuse = YES;
    file_seek(cif_file, sypt->offset, sypt->record);
    compt = get_cif_com(cif_file, TRUE);
    while ((*compt == 'D' && *(compt+1) == 'F'))
    {
        if (*compt == 'C')
        {
            downok = TRUE;

```

```

        (
            puts("\r-> finding bounds: ");
            puts(ltoa(callcount,comstore));
            puts(" calls processed\n");
        )
    return(bp);
)

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: tracedown
* SCNUMBER: 4.1.2
* FUNCTION: Traces symbol table using the input CIF CALL
*           command input (no current transformation is passed).
*           The input scale is only used to scale the numbers
*           contained in the CALL command (except the symbol number).
*           The symbol referred to in the call is located in the
*           symbol table. If this symbol is not found a fatal error
*           is produced. If this symbol's "calls" entry in the
*           symbol table is NO, then the bounds found by parse_cif
*           are correct and no further tracing is required. The
*           transformation specified in the input CALL is applied
*           to the bounds of the symbol and returned.
*           If the calls entry = YES, then two actions must take
*           place: (1) just as above the TRUE bounds of this symbol
*           must be transformed and returned to the calling process.
*           (2) To find these bounds, this symbol is found in the
*           CIF file and read until a CALL command is found. When
*           CALL commands are found, tracedown is called recursively
*           with the CALL command as found. The value returned is
*           used to modify the bounds of the current symbol if the
*           bounds of the called symbol cause it to change. When
*           all calls have been traced, the current symbol bounds
*           in the symbol table are set to the new values (the
*           "calls" variable is set to NO ) and the symbol bounds
*           are transformed as above and returned.
*           The purpose of all this is to finish the trace with the
*           the symbol table containing TRUE symbol bounds. This
*           means symbols need only be traced once for bounds and
*           during plotting symbols outside the plotting window do
*           not have to be called from disk and traced.
*           To prevent the stack from growing down into memory in
*           use by the program, the value of the stack is checked
*           and compared to lastalloc before each recursive call.
*           In addition, curdepth is checked to insure that it will
*           not exceed the value of depth before each call to trace-
*           down. Calls nested too deep are ignored and a warning is
*           generated. If a loop is found (a symbol calling an
*           ancestor) then a fatal error is generated.
* INPUTS: com_string (CALL command)
*         scale (scale of symbol called from)
*         bp (box to store new bounds in)
*****/

```

```

        puts(" K written to ");
        puts(outfname);
        puts(" ");
        nextk += WNOTIFY;
    }
}

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: trace_table
* SCNUMBER: 4.1
* FUNCTION: Traces the symbol table to determine the total size
*           of the entire plot. During the trace, the bounds of
*           symbols which call other symbols are set to the correct
*           values. Returns a box which contains the bounds of the
*           entire symbol. Exits through print error if there is
*           nothing to plot.
* INPUTS: bp - pointer to bounding box of executable commands
* OUTPUTS: bp - pointer to bounding box of entire plot
* GLOBAL VARIABLES USED: exec_file, rotate
* GLOBAL VARIABLES CHANGED: curdepth, callcount, comstore
* FILES READ: EXECUTABLE FILE
* FILES WRITTEN: none
* MODULES CALLED: fopen / get_ex_com / tracedown / fclose
*                 change_bds / prt_error / puts / ltoa / trans_box
* CALLING MODULES: bld_file
* HISTORY: N/A
*****/
struct box      *trace_table(bp)
struct box      *bp;
{
    static struct box      tbox;
    static char            *sp;
    static int             count;
    char                   *ltoa(), *get_ex_com();

    curdepth = 1;
    exec_file->channel = fopen(exec_file->name,"r");
    callcount = count = 0;
    while ((*sp = get_ex_com(exec_file)) != 'E')
    {
        if (*sp == 'C')
        {
            tracedown(sp,1.0,&tbox);
            change_bds(bp,&tbox);
        }
        ++count;
    }
    if (count < 1) prt_error(18,NILL);
    fclose(exec_file->channel);
    if (rotate) trans_box(bp,rotmatrix);
    if (!quiet)

```

```

        (**sploc != ';' ) &&
        (!(isdigit(**sploc)))) ++(**sploc);
    )

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: test_space
* SCNUMBER: (PC PROC)
* FUNCTION: Subtracts the input number from charcount, if its
*           equal to zero, then writes an output and quit command
*           to the PLOT file and calls prt_error with a fatal error.*
*           Allows error reporting on PLOT file overflow to reside
*           in one place.
*           [charcount contains the number of free character
*           locations on disk (to the nearest 1K )].
* INPUTS: count - number of chars to decrement from charcount
* OUTPUTS: none
* GLOBAL VARIABLES USED: plot_file, quiet, wrtspace, outfname
* GLOBAL VARIABLES CHANGED: charcount, nextk
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: pc_output / pc_quit / prt_error / puts / itoa
* CALLING MODULES: pc_output / pc_quit / pc_erase / pc_line /
*                 pc_fill / pc_draw / pc_color / pc_text / pc_string
* HISTORY: N/A
*****/
test_space(count,chan)
int    count, chan;
{

    static long    diff;
    static int     temp;
    static char    sp[7];
    char           *itoa();

    charcount -= count;
    if (charcount <= 0)
    {
        charcount += 30000L;           /* prevents looping */
        pc_output(chan);
        pc_quit(chan);
        prt_error(15,NILL);
    }
    if (!quiet)
    {
        diff = wrtspace - charcount;
        if (diff > nextk)
        {
            diff = nextk / 1024L;
            temp = (int) diff;
            puts("\r-> ");
            puts(itoa(temp,sp));
        }
    }
}

```

```

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: skip_blank
* SCNUMBER: 3.2.4.1
* FUNCTION: Skips CIF blank characters in input string and mod-
*          ifies string pointer to point to next non-blank char.
* INPUTS: fp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: fp (CIF)
* FILES WRITTEN: none
* MODULES CALLED: isdigit / isupper
* CALLING MODULES: ck_call
* HISTORY: N/A
*****/

```

```
skip_blank(sploc)
```

```
char **sploc;
```

```

{
    while ((*sploc != '-') &&
           (*sploc != '(') &&
           (*sploc != ')') &&
           (*sploc != ';') &&
           (!(isdigit(**sploc))) &&
           (!(isupper(**sploc)))) ) ++(*sploc);
}

```

```

/*****
* DATE: 26 OCT 1984
* VERSION: 1.0
* NAME: skip_sep
* SCNUMBER: 3.2.1.4
* FUNCTION: Using pointer to pointer to string, skips CIF sep
*          characters in string, and modifies input pointer to
*          point to first non-sep character
* INPUTS: sploc - pointer to pointer to string
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: isdigit
* CALLING MODULES: get_integer
* HISTORY: N/A
*****/

```

```
skip_sep(sploc)
```

```
char **sploc;
```

```

{
    while ((*sploc != '-') &&
           (*sploc != '(') &&
           (*sploc != ')') &&

```

```

if (scale)
(
    yt = 1.0 / scalenum * PAGEWIDTH * 100.0;
    cifmax = (long) yt;
    offset = 2L * INFINITY;
    if (cifmax > offset) cifmax = offset;
)
ylength = bp->ymin - bp->ymin;
xlength = bp->xmax - bp->xmin;
if (window)
(
    bp->ymin = ymin;
    bp->xmin = xmin;
    ylength = ymax - ymin;
    xlength = xmax - xmin;
)
if (scale) ylength = cifmax;
xt = (float) xlength;
yt = (float) ylength;
divide = xt / yt;
if (divide <= 1.000000) pnum = 1;
else
(
    if (makepage)
    {
        bp->ymin = bp->ymin + (ylength/2L) - (xlength/2L);
        ylength = xlength;
        pnum = 1;
    }
    else pnum = (int) divide + 1;
)
if (makepage)
(
    offset = ylength / PAGERIM;
    ylength = ylength + (2 * offset);
    xlength = xlength + (2 * offset);
    bp->ymin -= offset;
    bp->xmin -= offset;
    yt = (float) ylength;
)
if (pnum > PAGEMAX) prt_error(104, (itoa(pnum, comstore)));
bp->ymin = bp->ymin + ylength;
if (xlength < ylength) bp->xmax = bp->xmin + xlength;
else bp->xmax = bp->xmin + ylength;
if (!scale) scalenum = PAGEWIDTH / yt * 100.0;
viewport = 1.0 / yt;
*realxmax = bp->xmin + xlength;
return(pnum);

```



```

puts(r2);
strcat(pban,r2);
ltoa(bp->ymin,comstore);
puts(comstore);
strcat(pban,comstore);
puts(r2);
strcat(pban,r2);
ltoa(bp->yax,comstore);
puts(comstore);
strcat(pban,comstore);
puts("\n");
strcat(pban,"\015\012");
puts(r3);
strcat(pban,r3);
place = pban + strlen(pban);
ftoa('F',7,scalenum,place);
puts(place);
puts(r4);
strcat(pban,r4);
strcat(pban,"\015\012");
puts("\nThe plot will be ");
diff = (float) realx - bp->xmin;
im = diff / 1200.0 * scalenum;
ftoa('F',7,im,comstore);
puts(comstore);
puts(" feet\n");
if (err_flag) prt_error(19,NILL);
if (interactive)
{
    puts("Do you want a plot? ");
    if (!quiet) puts("\007");
    comstore[1] = getchar();
    if ((comstore[1] == 'y') || (comstore[1] == 'Y')) return;
    else abort(0);
}
}

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: set_page
* SCNUMBER: 4.2
* FUNCTION: Determines the size of a PLOT.COM page. Computes
*           and returns the number of pages required to make a
*           complete plot and modifies the input box to the correct
*           size for the first page. Three factors are considered
*           in these calculations - the scale of the plot, the
*           window of the plot, and the bounding box of the plot.
*           Also modifies the input xmax value to the real xmax
*           after windowing and scaling. Modifies the global
*           variable viewport using the formula from Newman page 75.
*           Since the window is always square one value will do for
*           both x and y conversion to the viewport.
*           The basic equation is:
*****/

```

```

/*****
* DATE: 9 NOV 1984
* VERSION: 1.0
* NAME: report_size
* SCNUMBER: 4.3
* FUNCTION: Outputs the size and scale of plot to the terminal
*           in the cifplot format. If interactive is TRUE asks if
*           plot should be made. Sets up output banner for plot in
*           the global variable pban. If user does not want plot,
*           or if err_flag is true calls abort to quit.
* INPUTS: bp - pointer to page bounds, xmax maximum x plot
*         bounds
* OUTPUTS: none
* GLOBAL VARIABLES USED: err_flag, scalenum, interactive, quiet
* GLOBAL VARIABLES CHANGED: pban, comstore
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: puts / ltoa / ftoa / getchar / abort
*                 prt_error
* CALLING MODULES: bld_file
* HISTORY: N/A
*****/

```

```

report_size(bp,xmax)
struct box      *bp;
long            xmax;
{
    static char      *r1 = "Window: ";
    static char      *r2 = " ";
    static char      *r3 = "Scale: 1 micron is ";
    static char      *r4 = " inches";
    static unsigned   place;
    static char      sp[12];
    static float      diff, im;
    static long       realx;
    char              *ltoa();

    pban[0] = NULL;
    strcat(pban,"acifplot* of ");
    strcat(pban,cif_file->name);
    strcat(pban,"\015\012");
    puts("\n");
    puts(r1);
    strcat(pban,r1);
    ltoa(bp->xmin,comstore);
    puts(comstore);
    strcat(pban,comstore);
    puts(r2);
    strcat(pban,r2);
    if (xmax > bp->xmax) realx = xmax;
    else realx = bp->xmax;
    ltoa(realx,comstore);
    puts(comstore);
    strcat(pban,comstore);

```

```

        pc_color(BLACK,plot_file->channel);
        pc_point(xb,yb,plot_file->channel);
        return;
    }

    if (outline || !color)
    {
        pc_color(BLACK,plot_file->channel);
        if (xline && !clipped->left) pc_draw(xb,yb,xt,yb,plot_file->channel);
        if (xline && !clipped->right) pc_draw(xb,yt,xt,yt,plot_file->channel);
        if (yline && !clipped->bottom) pc_draw(xb,yb,xb,yt,plot_file->channel);
        if (yline && !clipped->top) pc_draw(xt,yb,xt,yt,plot_file->channel);
    }

    if (color && xfill && yfill)
    {
        pc_color(color,plot_file->channel);
        pc_fill(xb,yt,xt,yt,yb,plot_file->channel);
    }
}

/*****
* DATE: 27 OCT 1984
* VERSION: 1.0
* NAME: put_com
* SCNUMBER: 3.3.1
* FUNCTION: Writes one string to the file specified by the input
*           file pointer. Adds NEWLINE after string.
* INPUTS: fp, sp
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: charcount
* FILES READ: none
* FILES WRITTEN: fp->channel
* MODULES CALLED: putc / prt_error / strlen / fclose
* CALLING MODULES: wrt_ex_file
* HISTORY: N/A
*****/
put_com(fp,sp)
struct file *fp;
char *sp;
{
    static int count;

    count = strlen(sp) + 2;
    charcount -= count;
    if (charcount <= 1) prt_error(16,NILL);
    while (*sp)
    {
        putc(*sp,fp->channel);
        sp++;
    }
    putc(NEWLINE,fp->channel);
}

```

```

else if (error > 1999)
{
    puts("\n");
    puts("--> Warning: ");
}

while ((c = getc(er_file->channel))) putchar(c);

}

/*****
* DATE:  9 NOV 1984
* VERSION: 1.0
* NAME: put_box
* SCNUMBER: 4.5.2.1.3
* FUNCTION: Uses plot commands to write one box to PLOT file
*           If outline is TRUE, writes lines to outline box, else
*           just fills box region with input color (an input color
*           = 0 forces output of box outline with no fill). The
*           viewport transformation is used to scale box to
*           PLOT.COM units. Assumes transformations and clipping is
*           complete and box has Manhattan directions.
* INPUTS: bp, color, cliprec
* OUTPUTS: none
* GLOBAL VARIABLES USED: plot_file, outline, viewport
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: PLOT
* MODULES CALLED: pc_color / pc_fill / pc_point
* CALLING MODULES: proc_call
* HISTORY: N/A
*****/
put_box(bp,color,clipped)
struct box      *bp;
int             color;
struct cliprec  *clipped;
{
    static float  diff, xb, xt, yb, yt;
    static int    xfill, yfill, xline, yline;

    xb = (float) viewport * bp->xmin;
    xt = (float) viewport * bp->xmax;
    yb = (float) viewport * bp->ymin;
    yt = (float) viewport * bp->ymax;

    xfill = yfill = xline = yline = FALSE;
    diff = xt - xb;
    if (diff > MINLINE) xline = TRUE;
    if (diff > MINFILL) xfill = TRUE;
    diff = yb - yt;
    if (diff > MINLINE) yline = TRUE;
    if (diff > MINFILL) yfill = TRUE;

    if (!xline && !yline)
    {

```

```

trantoplot(bp)
struct box      *bp;
{
    static long    offset, temp;

    offset = pwindow.xmin + (pwindow.ymax - pwindow.ymin);
    temp = bp->ymin - pwindow.ymin;
    bp->ymin = offset - bp->xmin;
    bp->xmin = temp;

    temp = bp->ymax - pwindow.ymin;
    bp->ymax = offset - bp->xmax;
    bp->xmax = temp;
}

/*****
* DATE: 29 OCT 1984
* VERSION: 1.0
* NAME: wrt_ex_file
* SCNUMBER: 3.3
* FUNCTION: Checks all executable commands for syntax and writes
*           them to the EXECUTABLE file if errors are not found.
*           Closes file when E command is found and prints error if
*           additional commands are found after the E command.
* INPUTS: com_string
* OUTPUTS: none
* GLOBAL VARIABLES USED: end
* GLOBAL VARIABLES CHANGED: exbounds, exlayer, build
* FILES READ: none
* FILES WRITTEN: none
* MODULES CALLED: ck_box / ck_call / ck_round / ck_layer
*                 ltoa / ck_wire / do_lcom / put_com / ltoa / puts
*                 ck_9 / prt_error / fclose / change_bds / isdigit
* CALLING MODULES: parse_cif
* HISTORY: N/A
*****/
wrt_ex_file(com_string)
char      *com_string;
{
    struct box      *ck_box(), *ck_wire(), *ck_round();
    struct box      *newbox;
    char          *dummy, ap[15], *ltoa();

    if (isdigit(*com_string) && !extension) prt_error(1006, com_string);
    else
    {
        switch (*com_string)
        {
            case 'B':
                if (exlayer) prt_error(502, com_string);
                newbox = ck_box(com_string);
                if (newbox != NULL)
                {

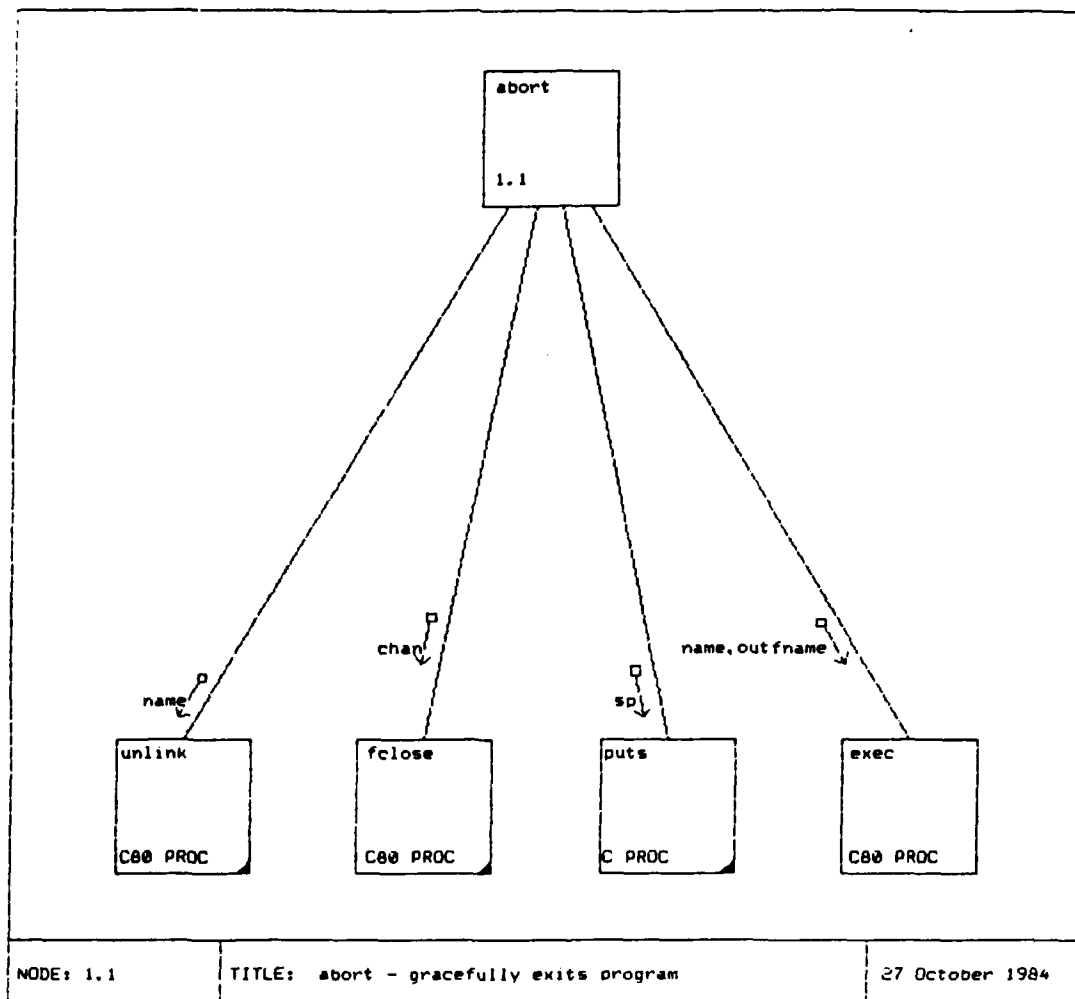
```

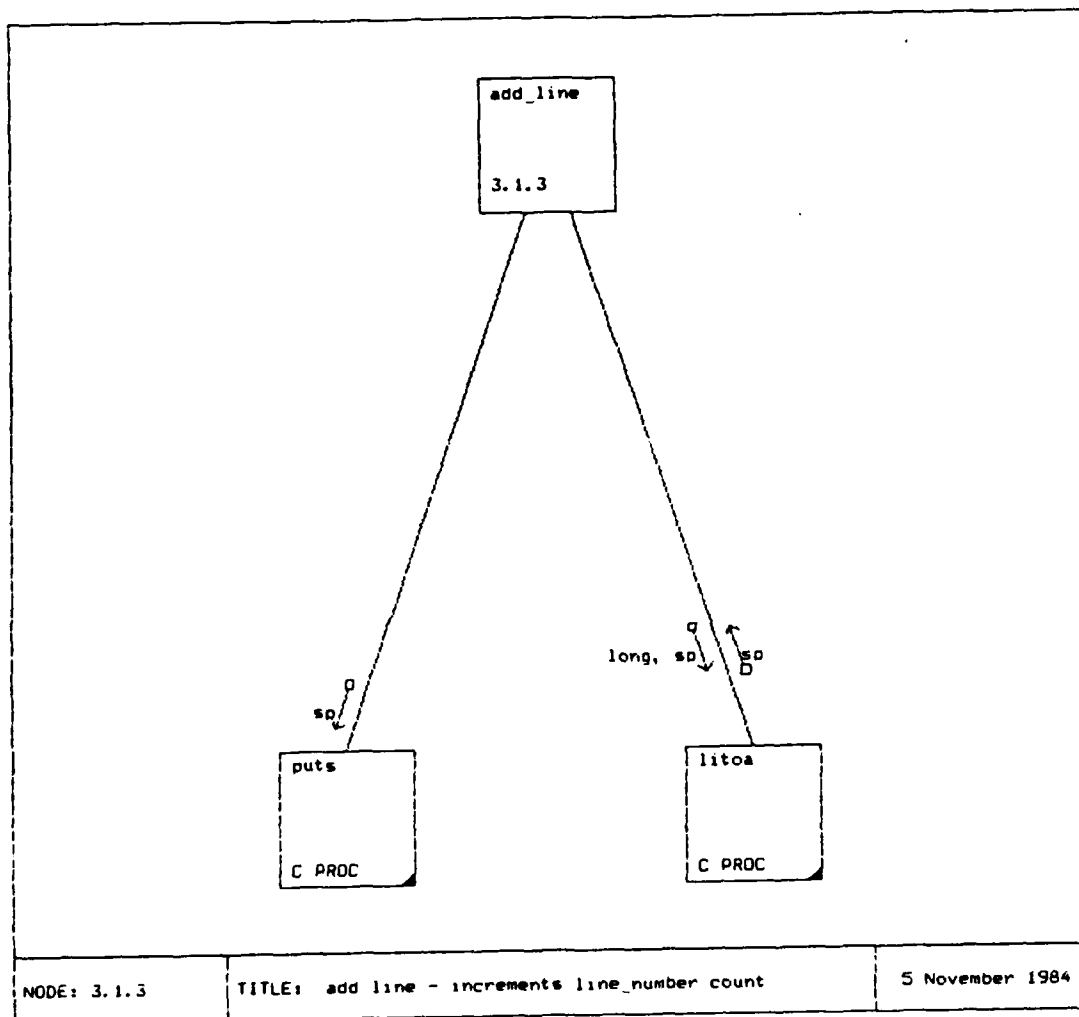


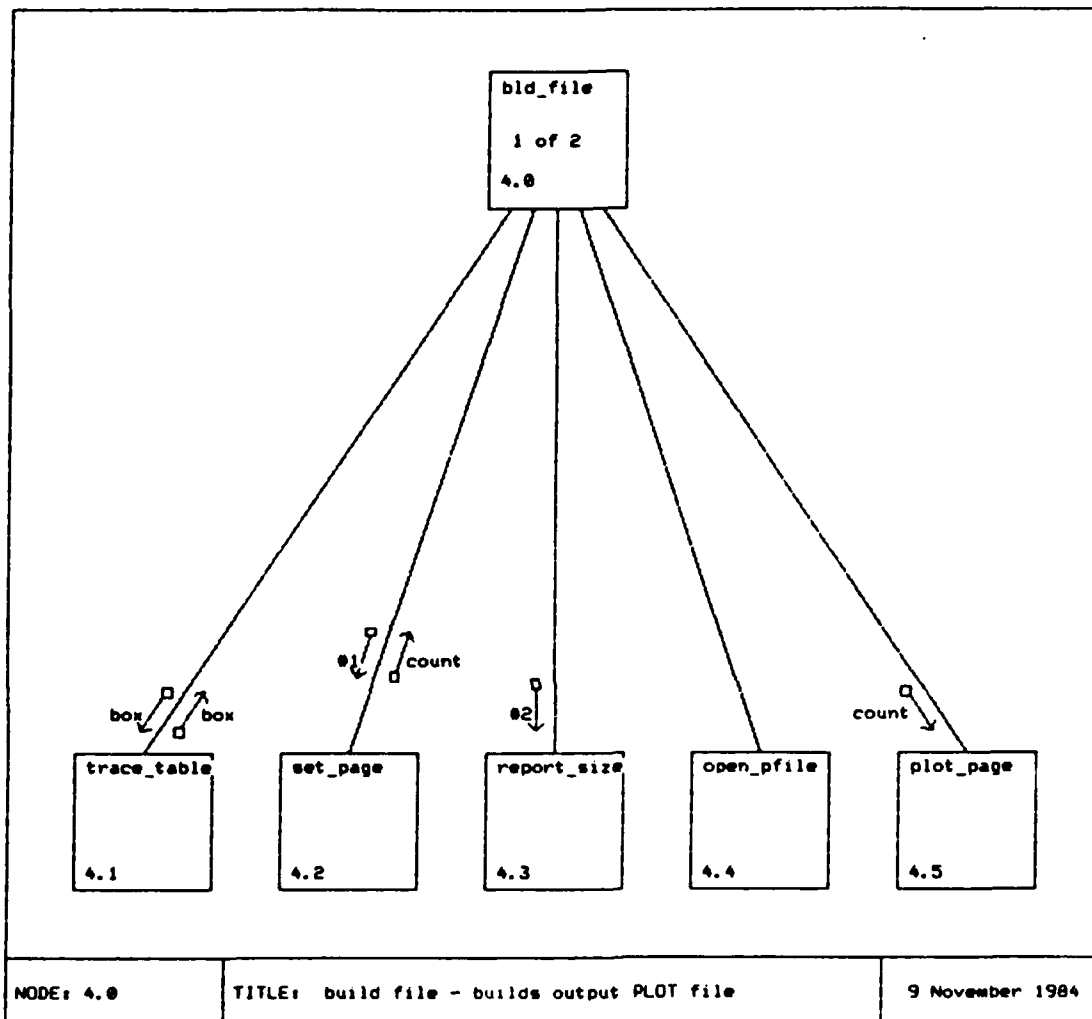
```

case '9':
    if (*(com_string+1) != '4' && symbolname)
    {
        prt_error(1005,com_string);
        break;
    }
    if (pointname)
    {
        ck_9(com_string);
        put_com(exec_file,com_string);
    }
    break;
case '0':
    prt_error(500,com_string);
    break;
case 'D':
    if (*(com_string+1) == 'D') prt_error(200,com_string);
    else if (*(com_string+1) == 'F') prt_error(201,NILL);
    else prt_error(501,com_string);
    break;
case 'E':
    if (strlen(com_string) > 1 && *(com_string+1) != NEWLINE)
    {
        prt_error(1003,com_string);
    }
    put_com(exec_file,com_string);
    end = TRUE;
    fclose(exec_file->channel);
    ++totcommand;
    if (!quiet && build)
    {
        puts("\r-> ");
        puts("parsing ");
        puts(cif_file->name);
        puts(": ");
        --line_number;
        dummy = ltoa(line_number,ap);
        puts(ap);
        puts(" lines read ");
        puts("\n");
    }
    break;
case NULL:
    if (!end) prt_error(13,NILL);
    break;
default:
    prt_error(503,com_string);
}
)

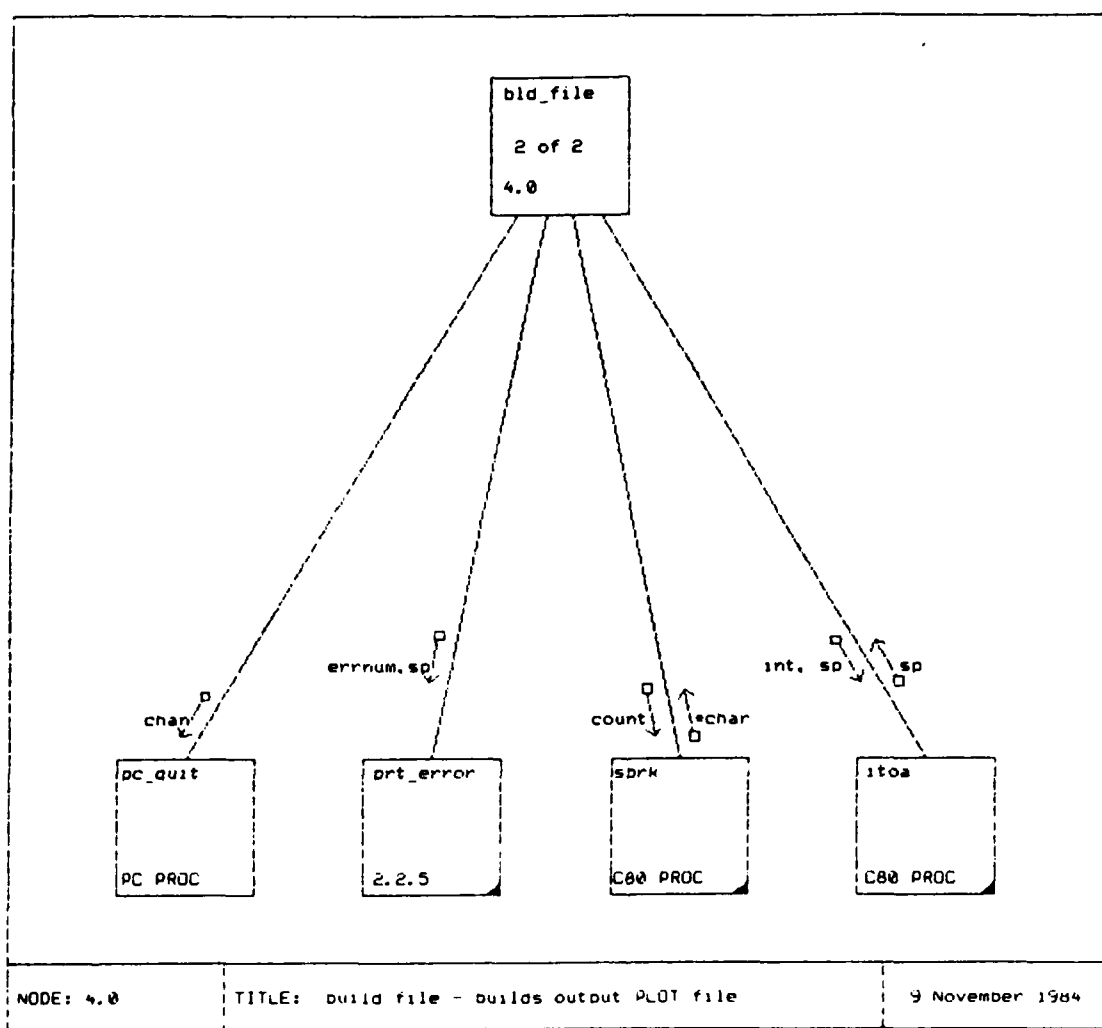
```

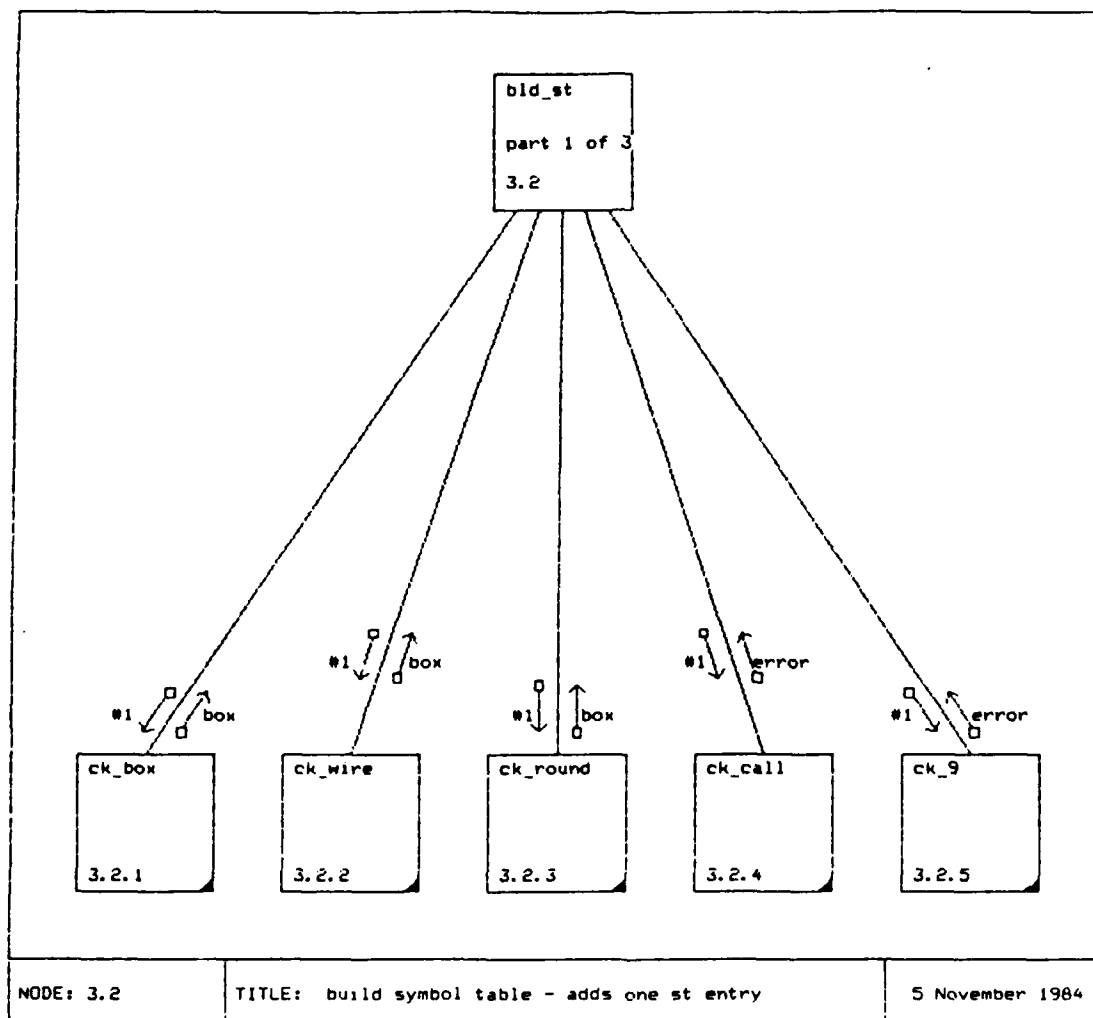




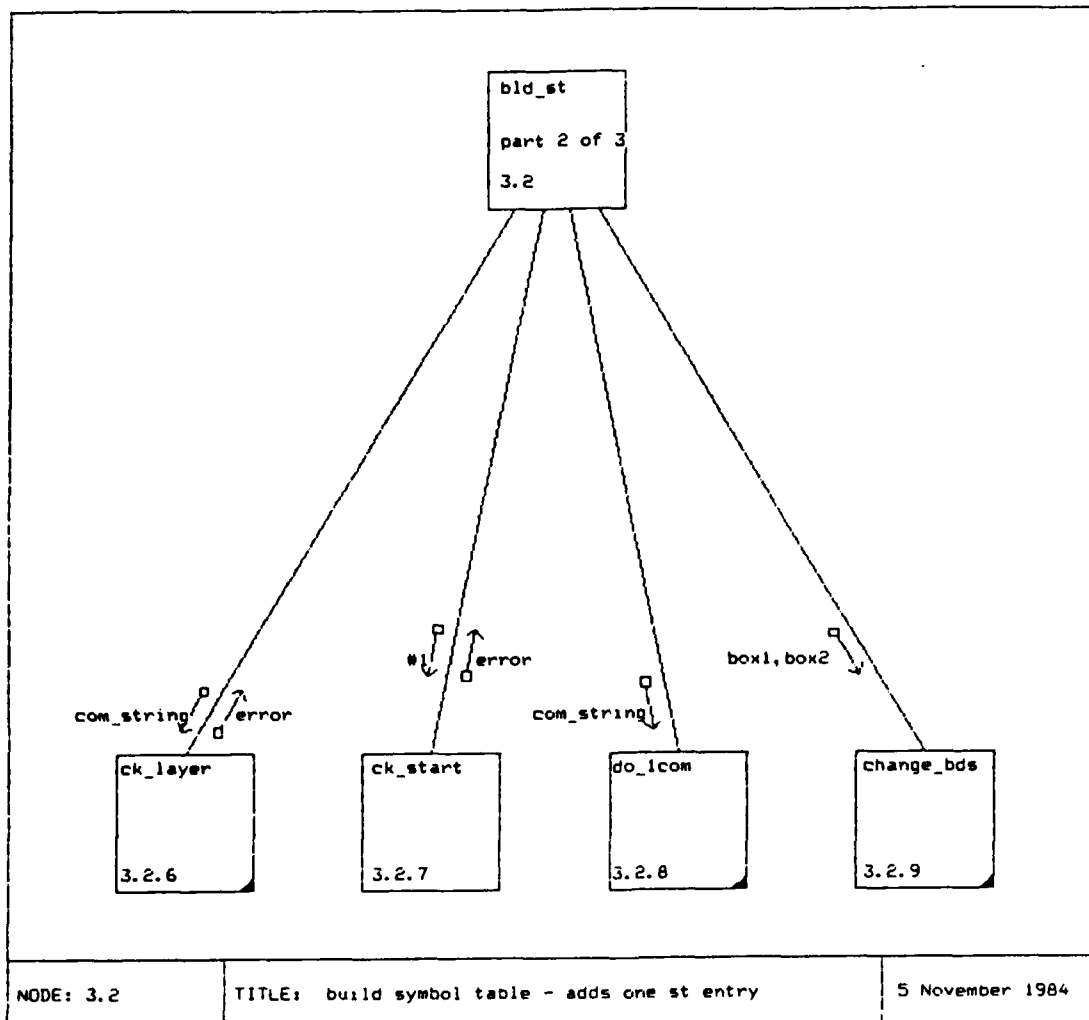


#1: box, &realmax
 #2: box, &realmax

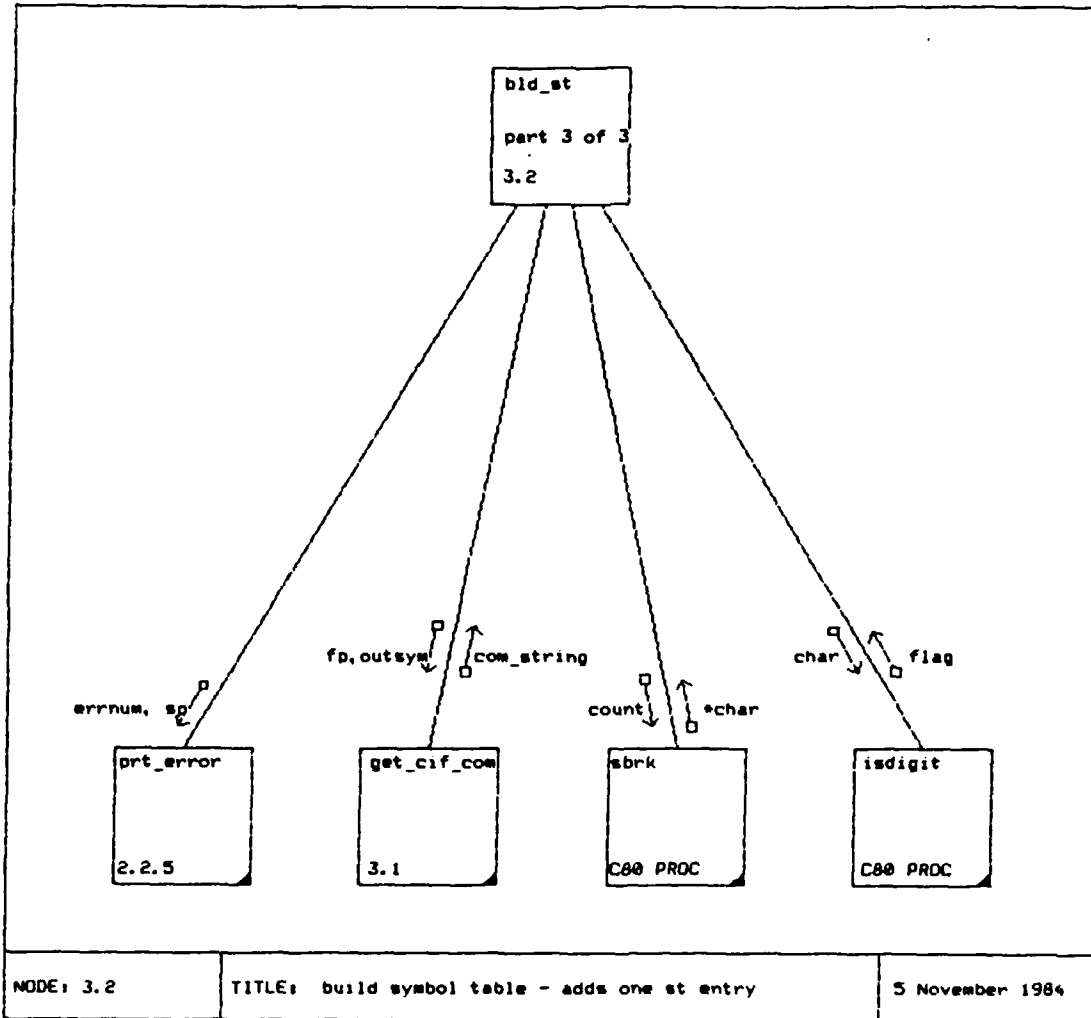


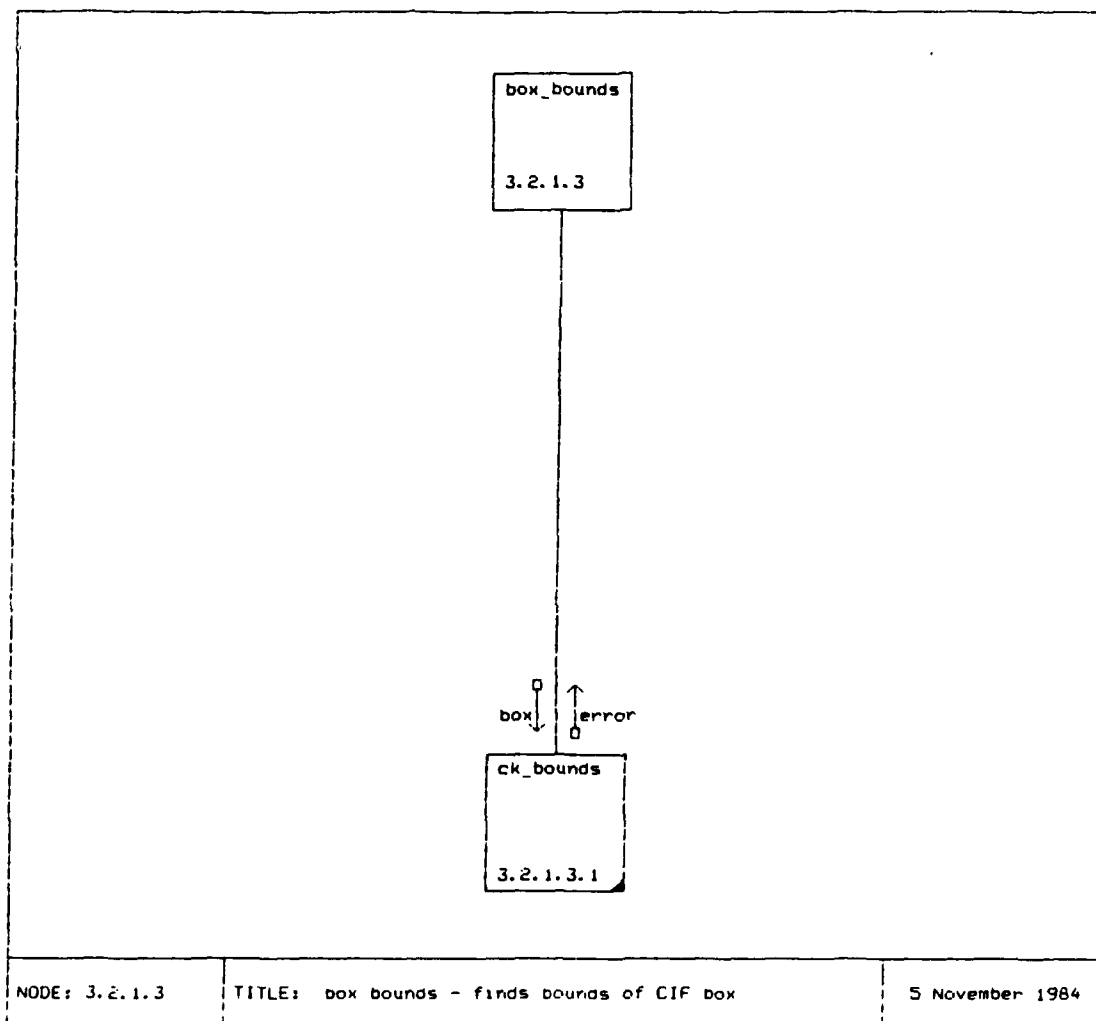


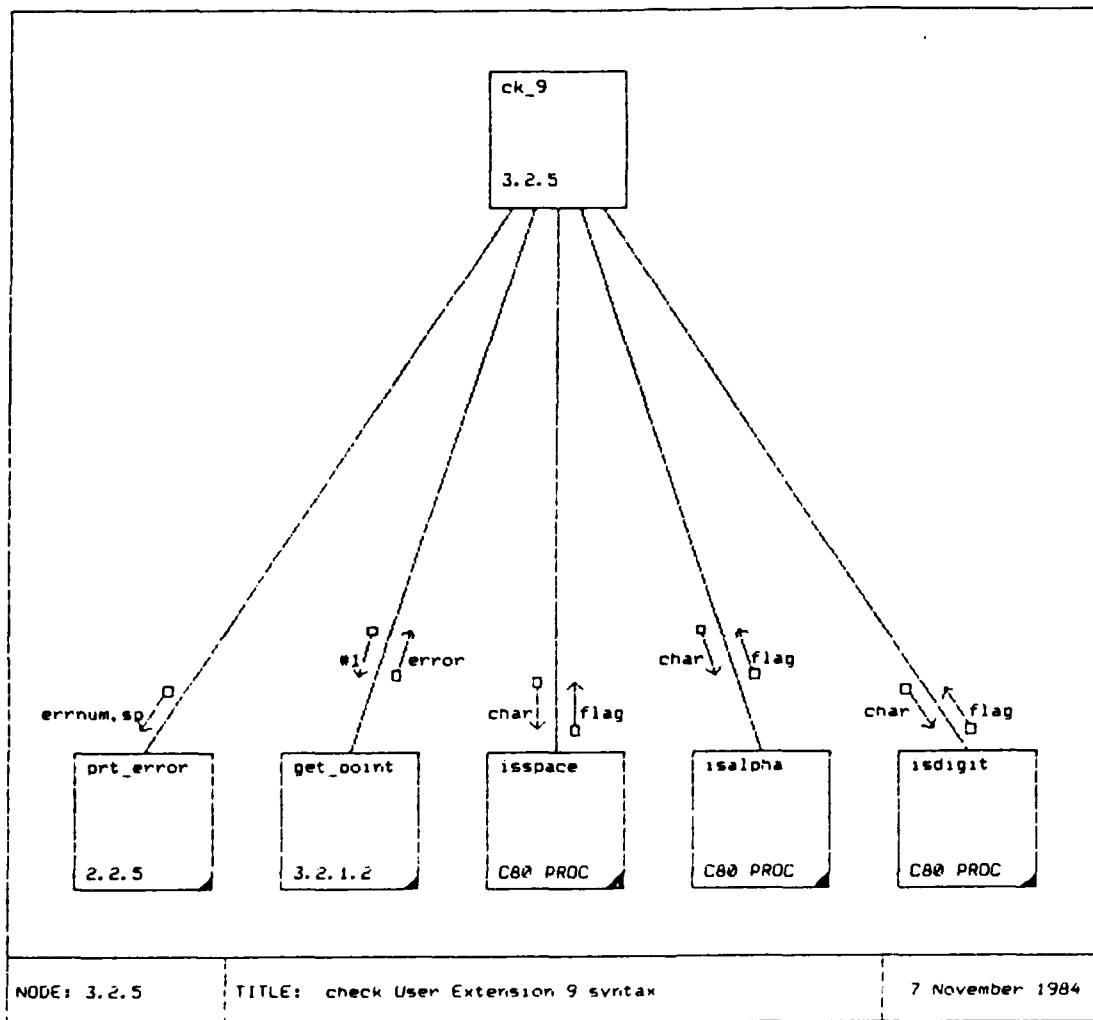
#1: com_string



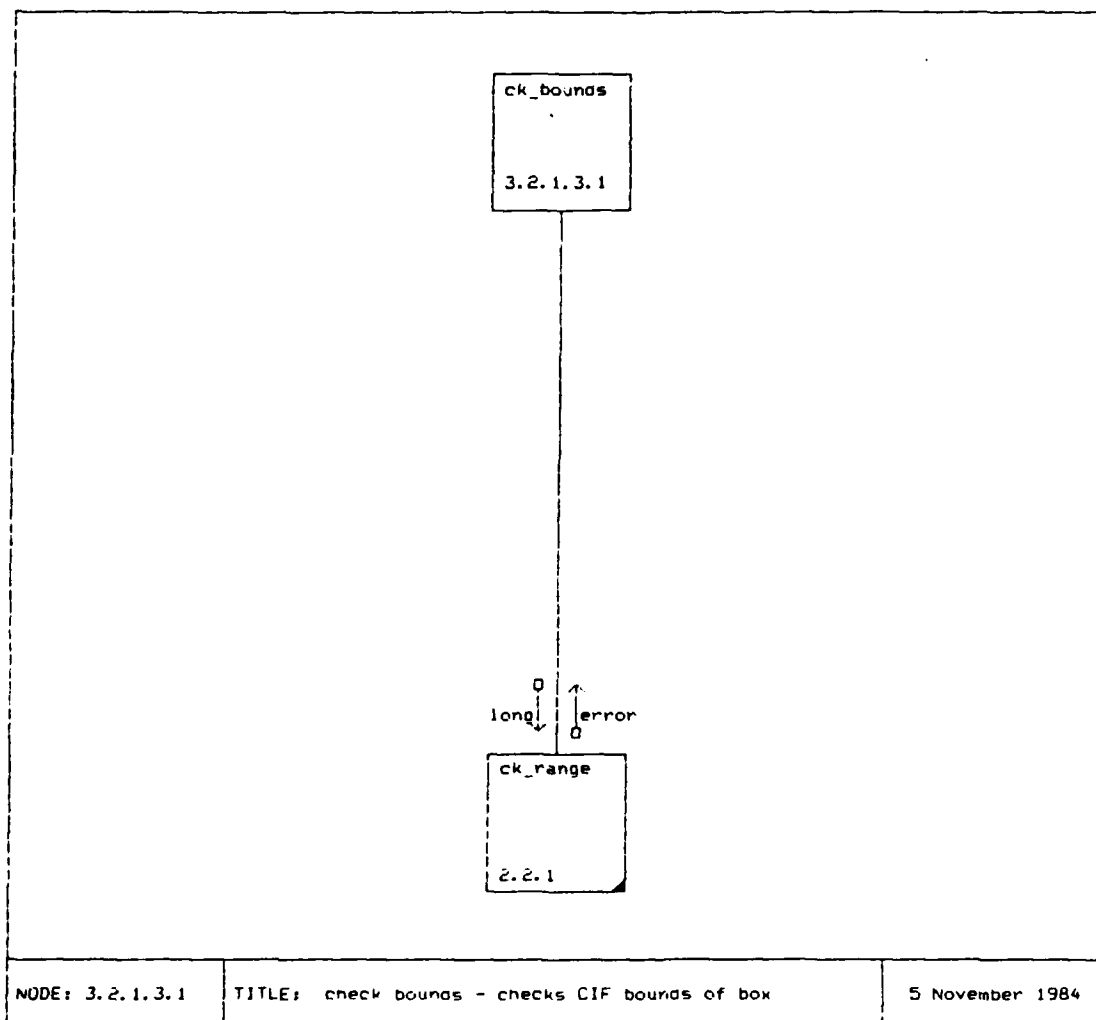
#1: com_string, &symnum, &sym_ascale, &sym_bscale

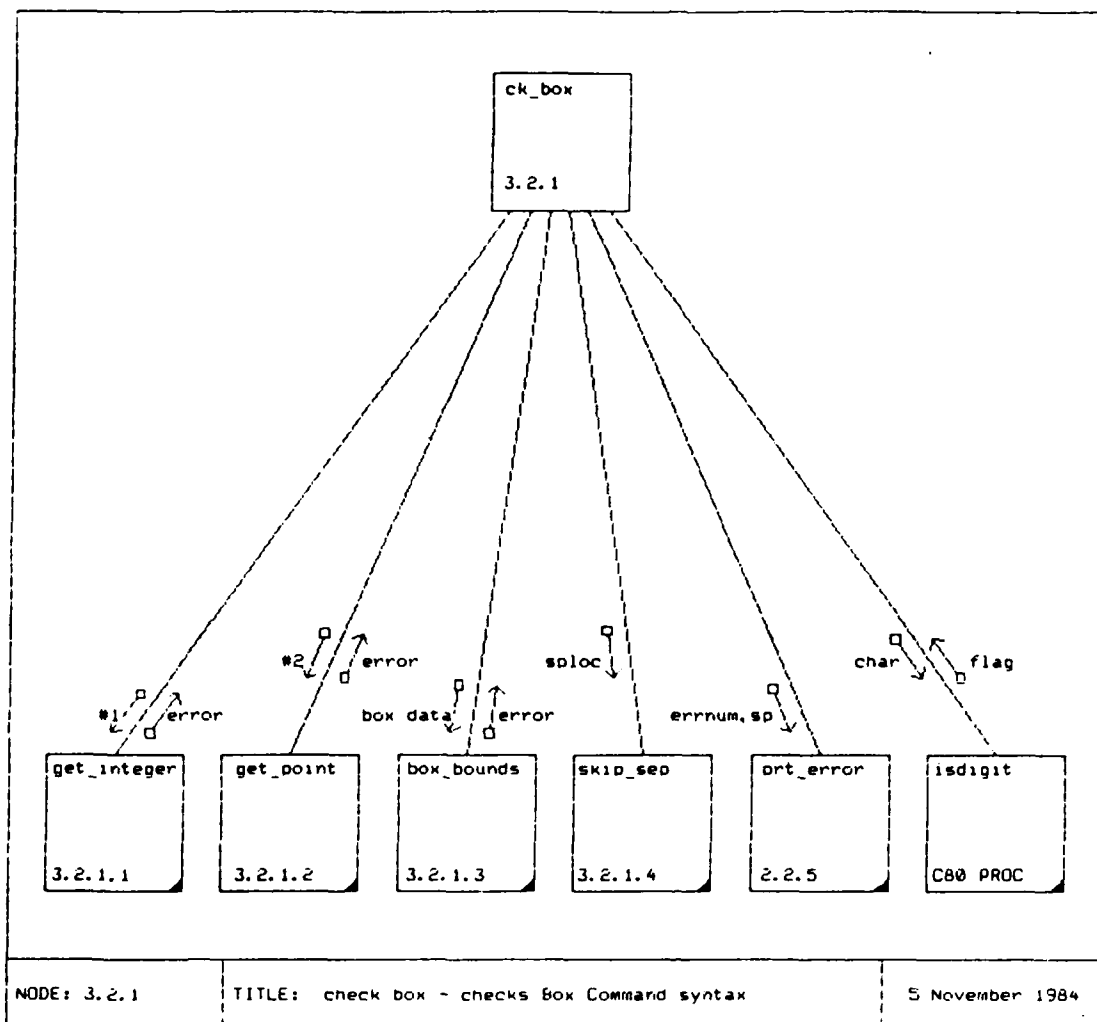




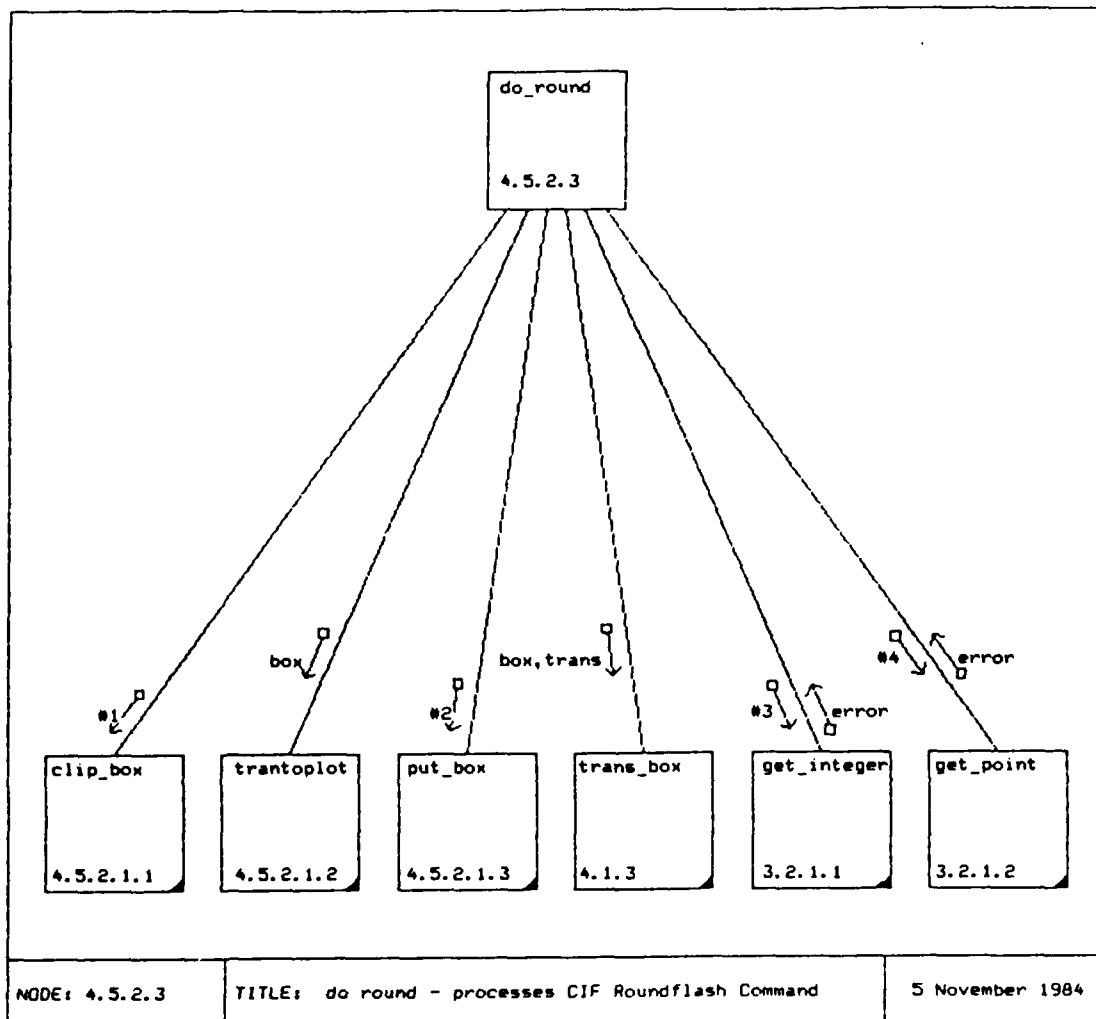


#1: com_string, sploc, point

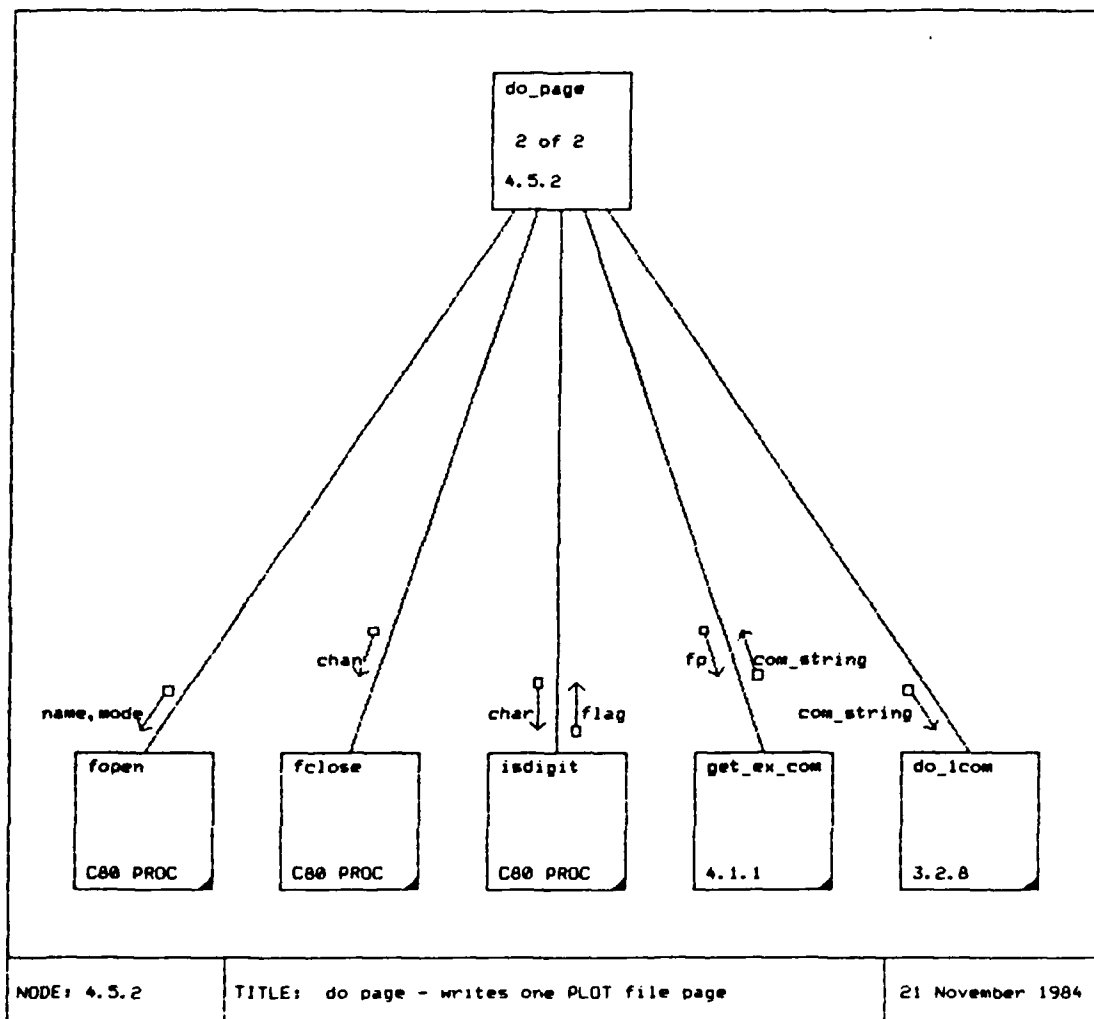


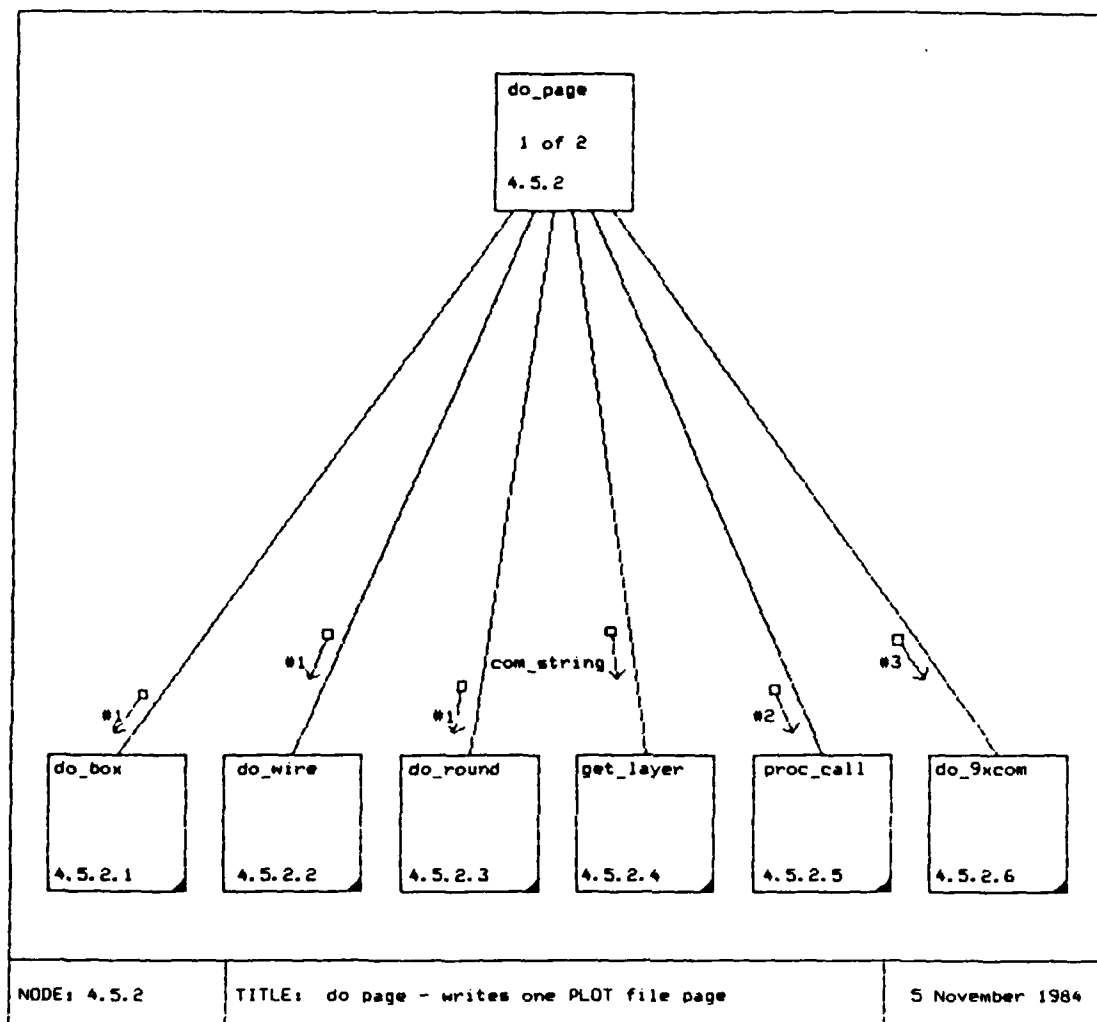


#1: com_string, sploc, &long
#2: com_string, sploc, point

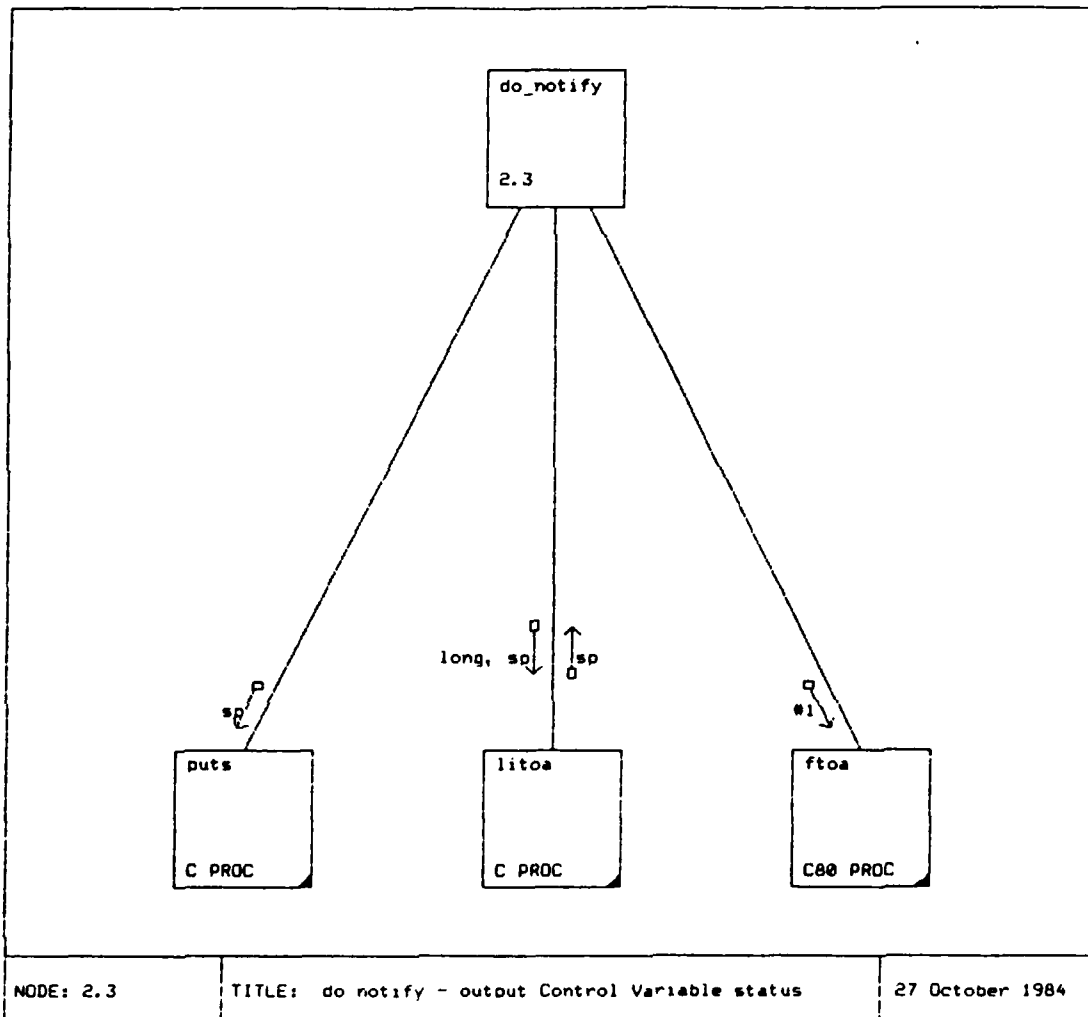


#1: box1, box2, cliprec
 #2: box, color, cliprec
 #3: com_string, sploc, &long
 #4: com_string, sploc, point

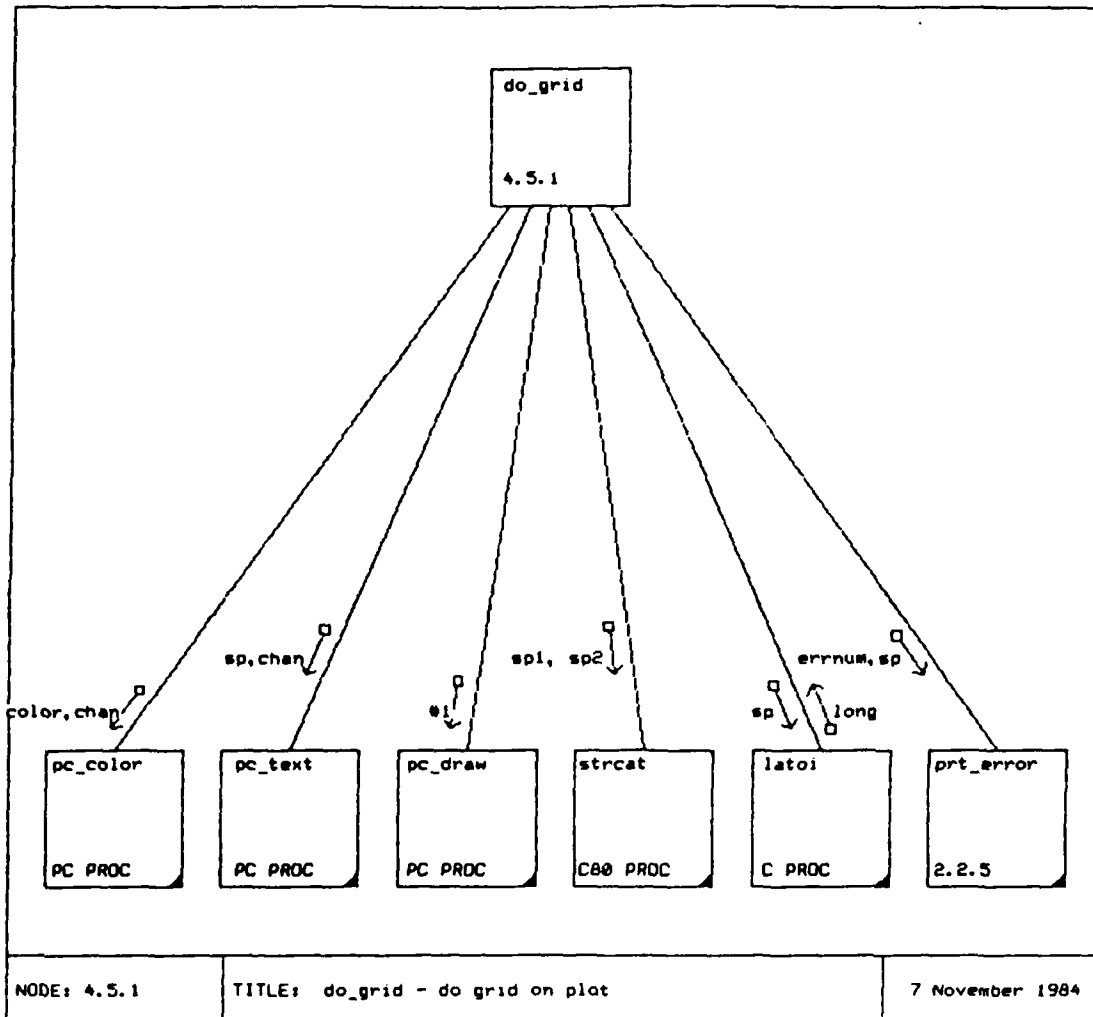




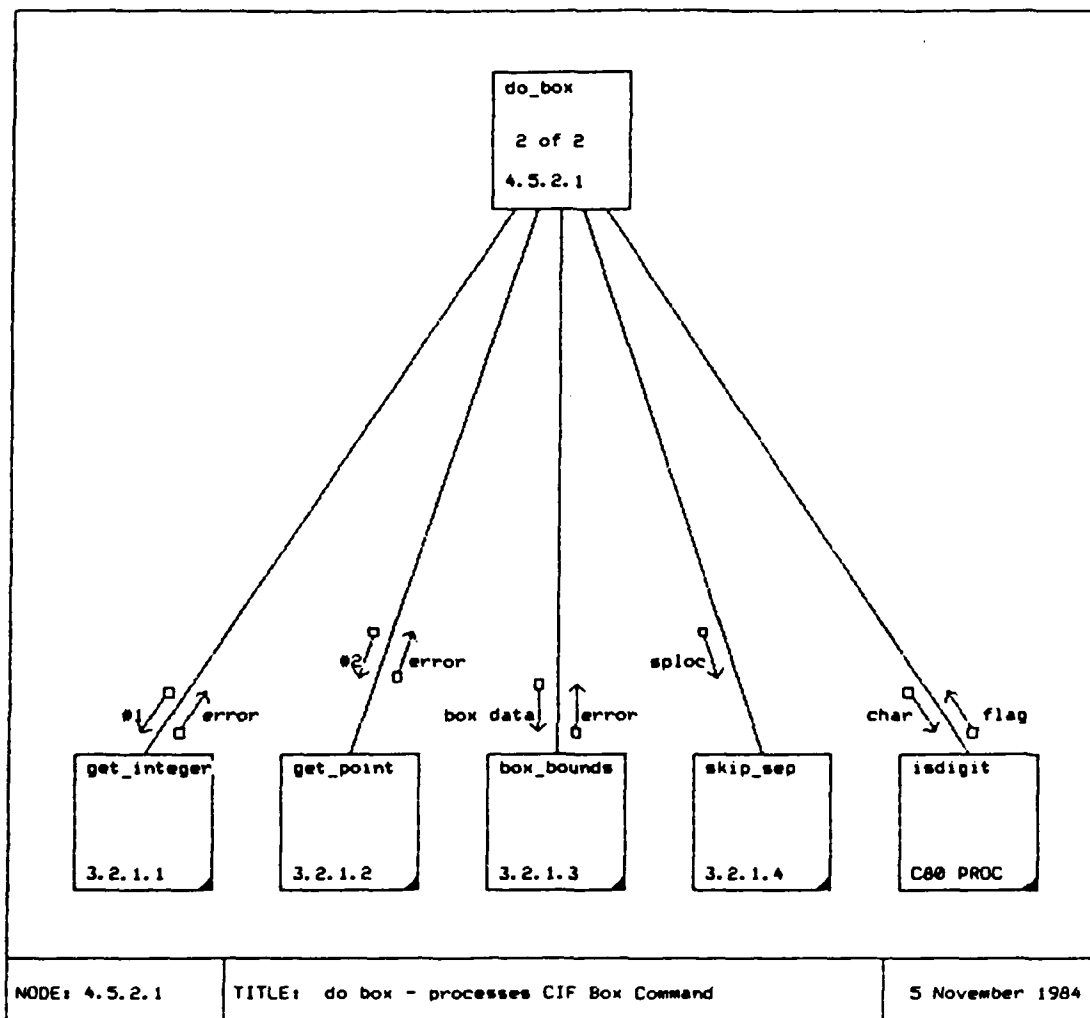
#1: com_string, trans, color, scale
 #2: com_string, trans, scale
 #3: com_string, trans, symbol, scale



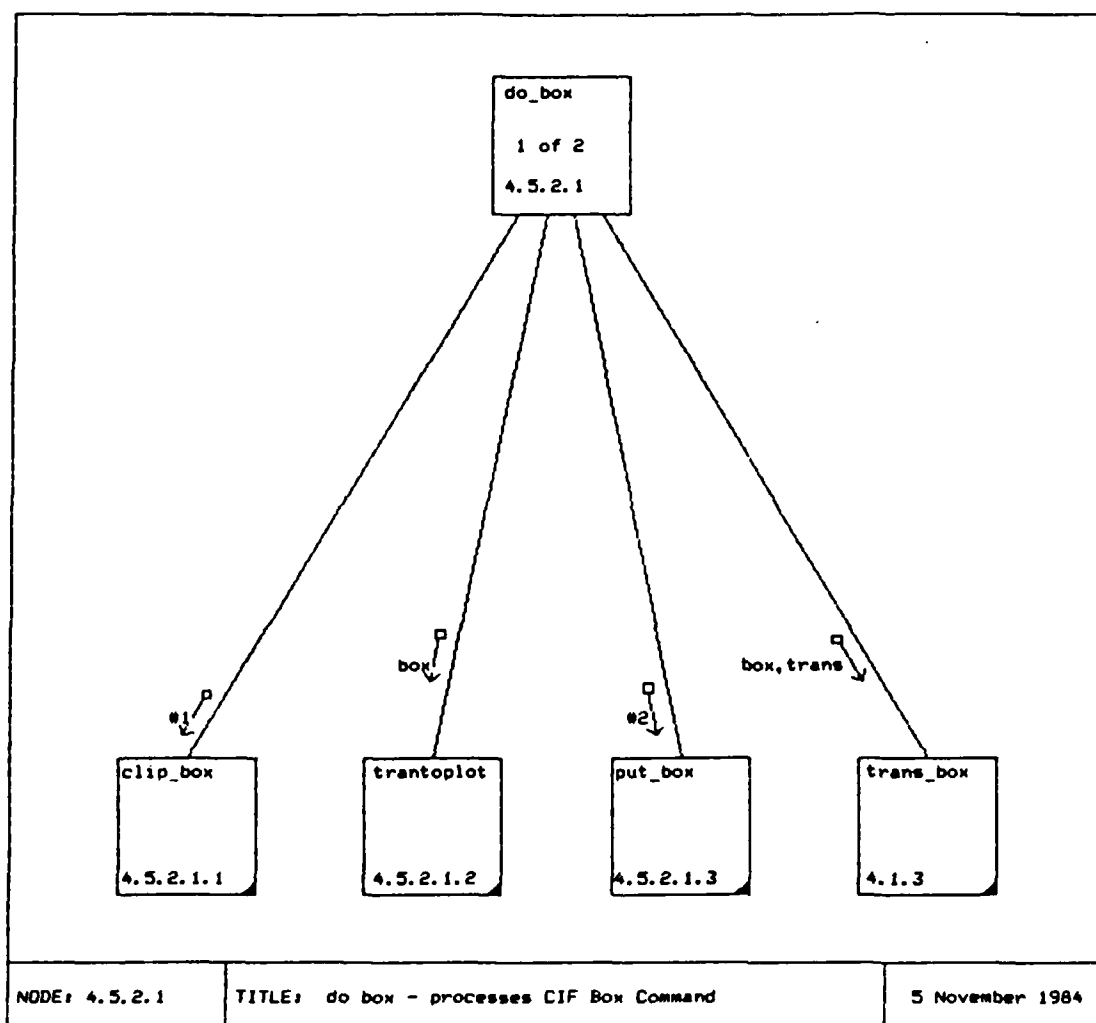
#1: format, digits, float, so



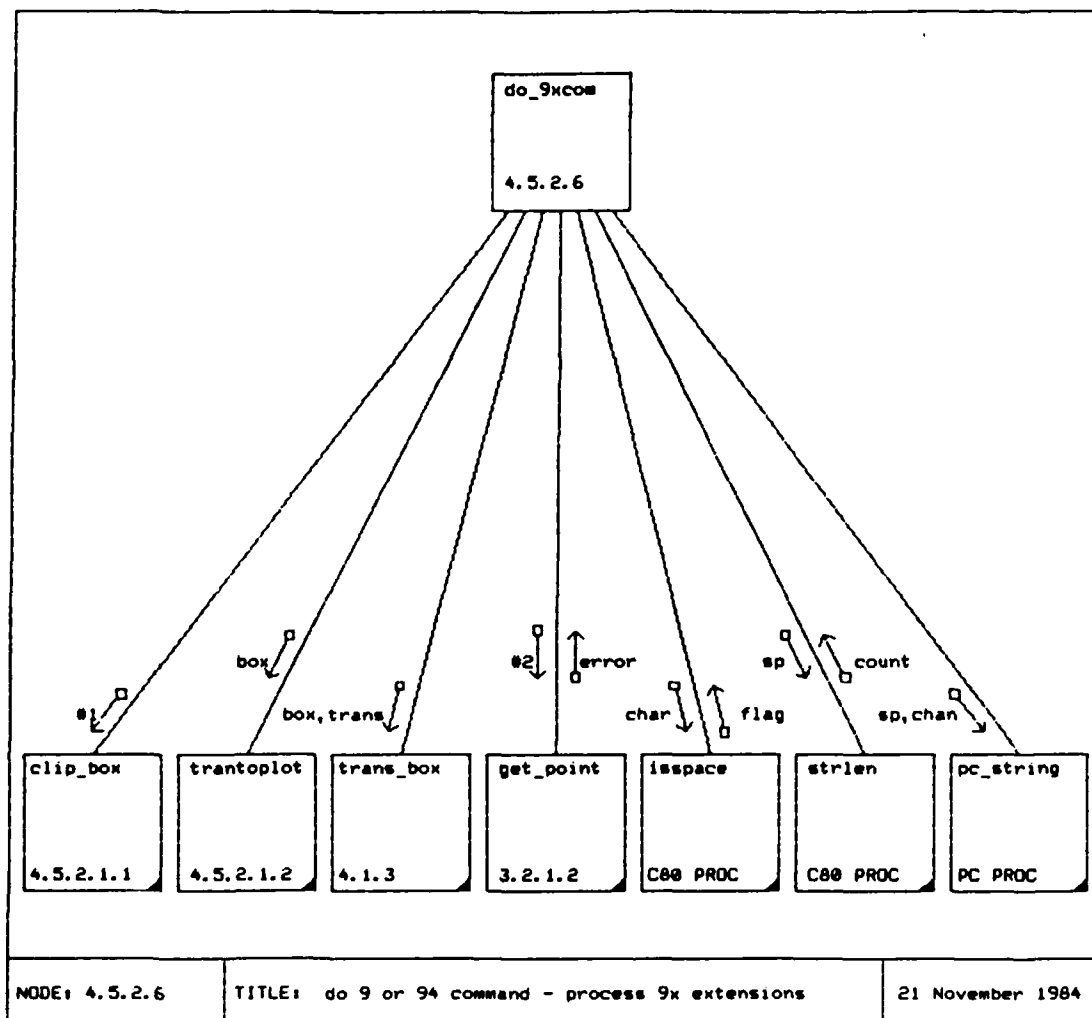
*1: x1, y1, x2, y2, chan



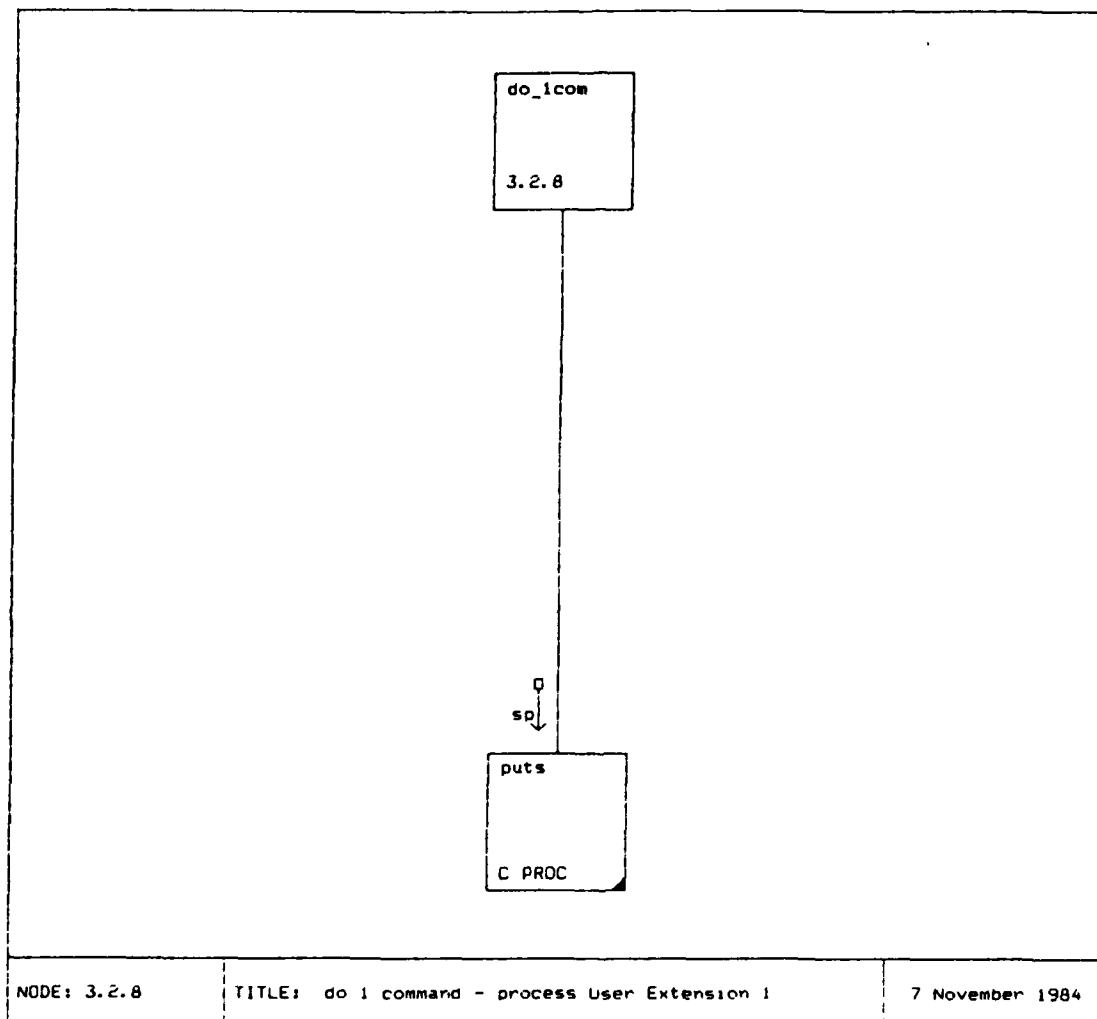
#1: com_string, sploc, &long
 #2: com_string, sploc, point

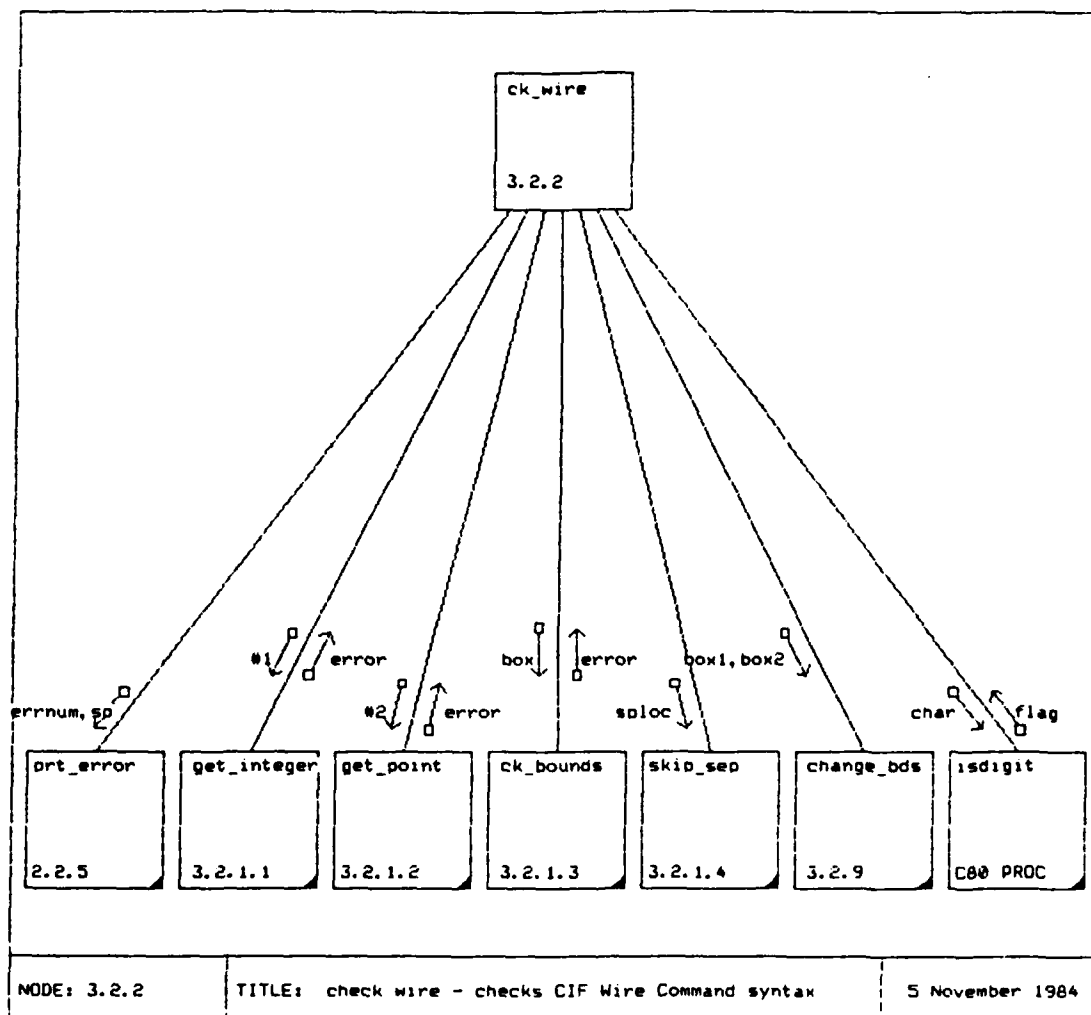


#1: box1, box2, cliprec
 #2: box, color, cliprec

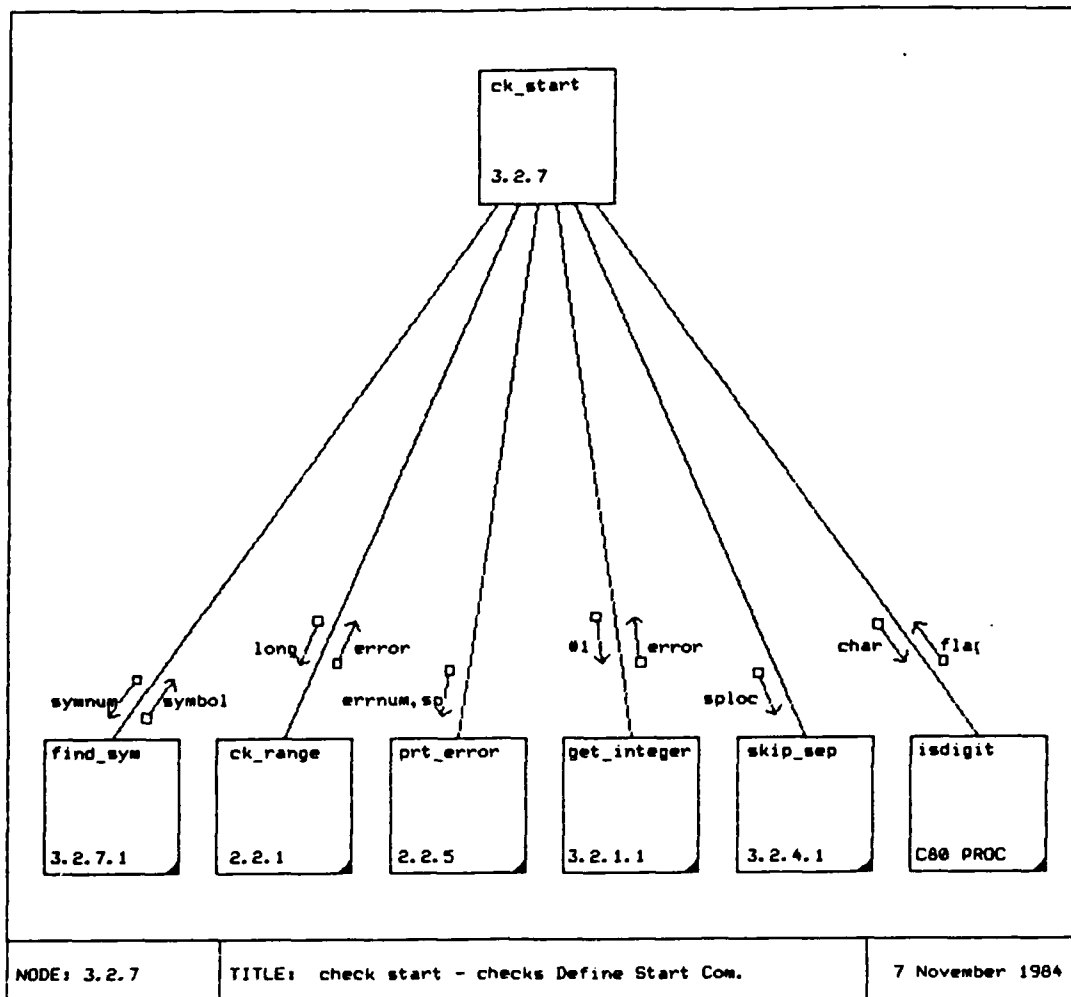


#1: box1, box2, cliprec
 #2: com_string, sploc, point

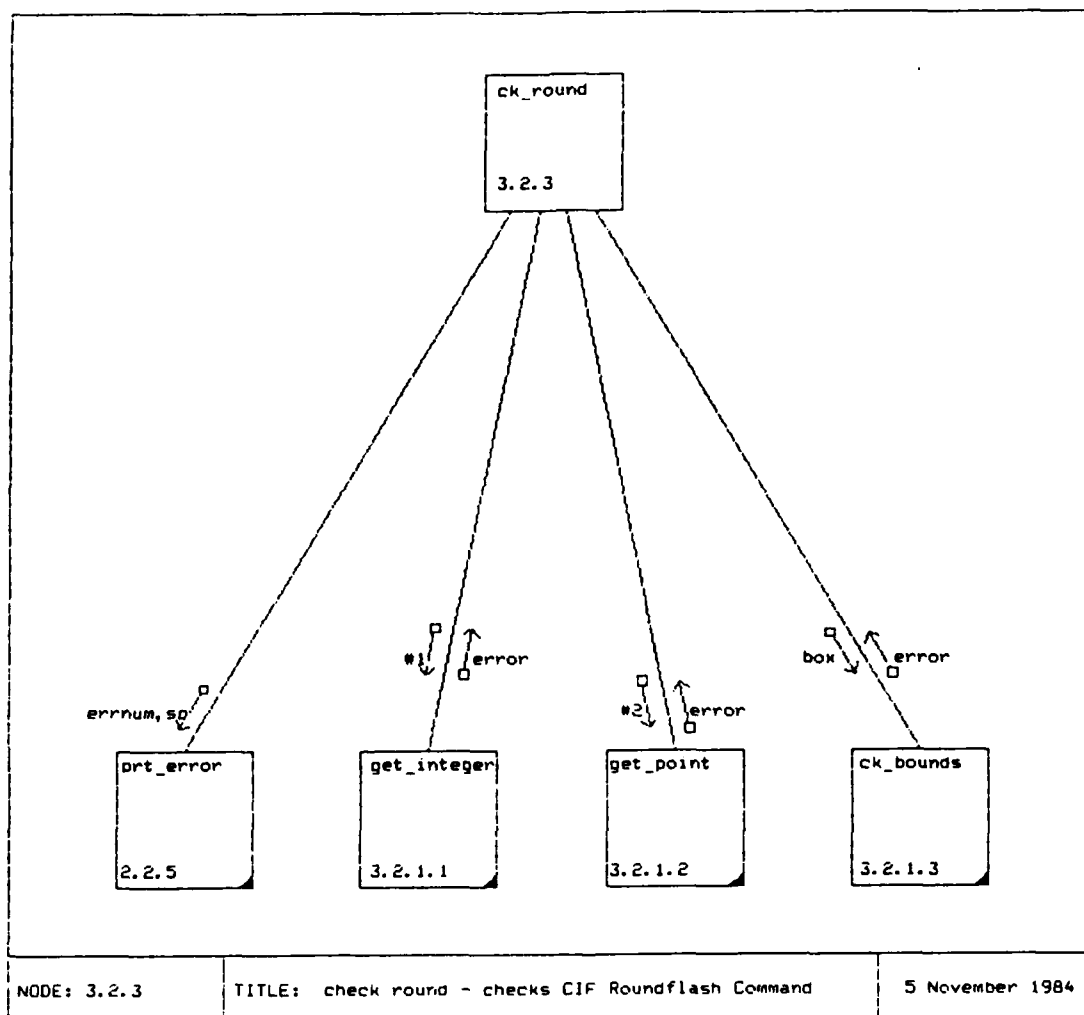




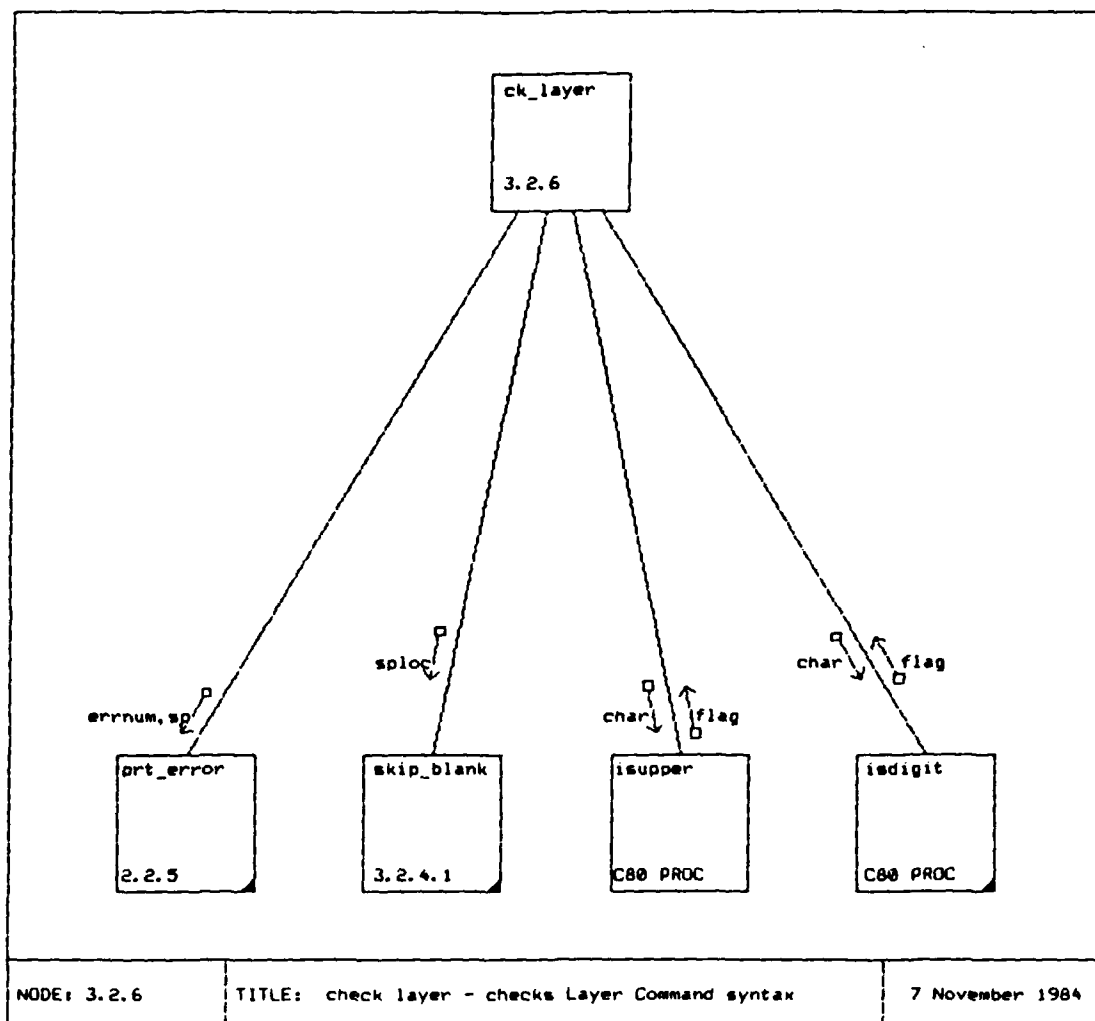
#1: com_string, sploc, &long
 #2: com_string, sploc, point

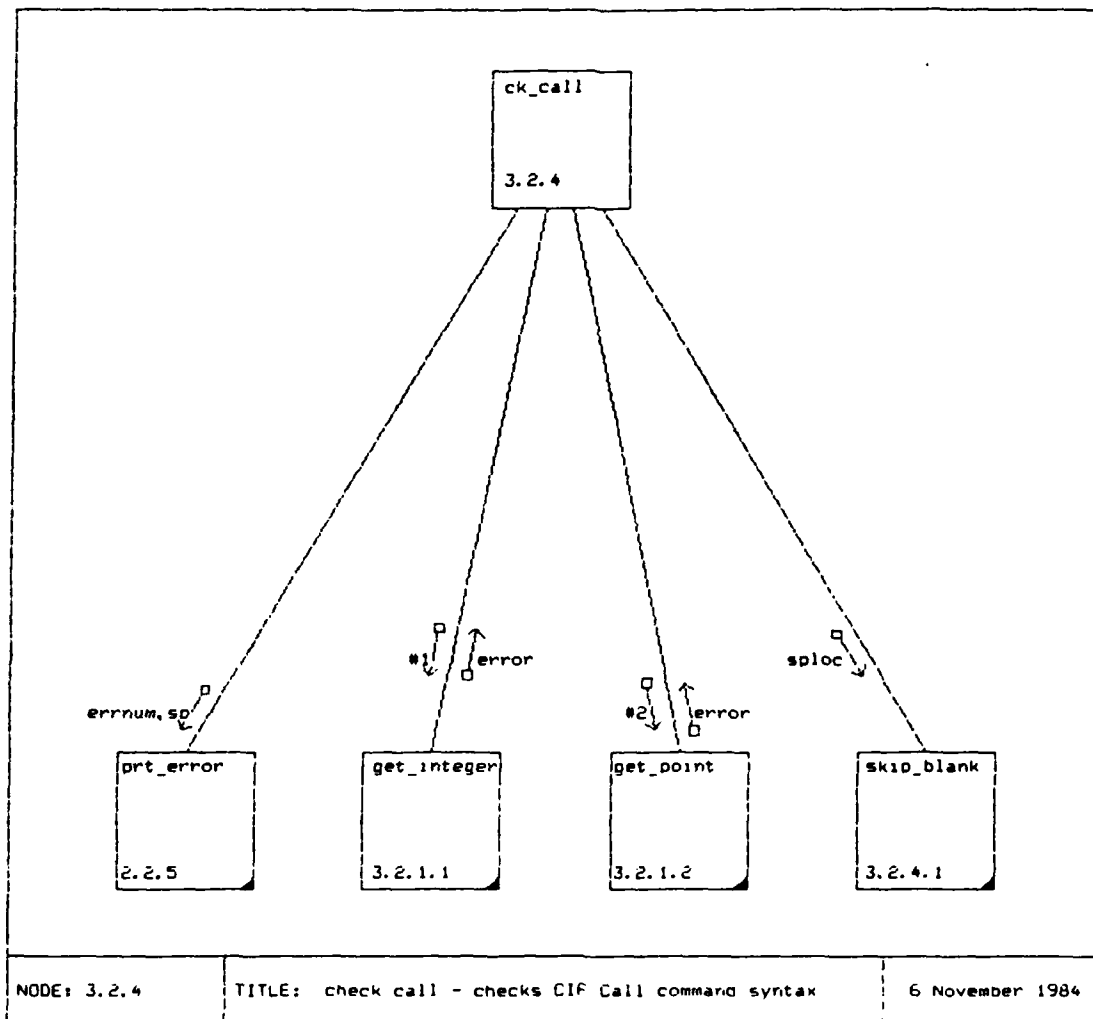


#1: com_string, sploc, &long

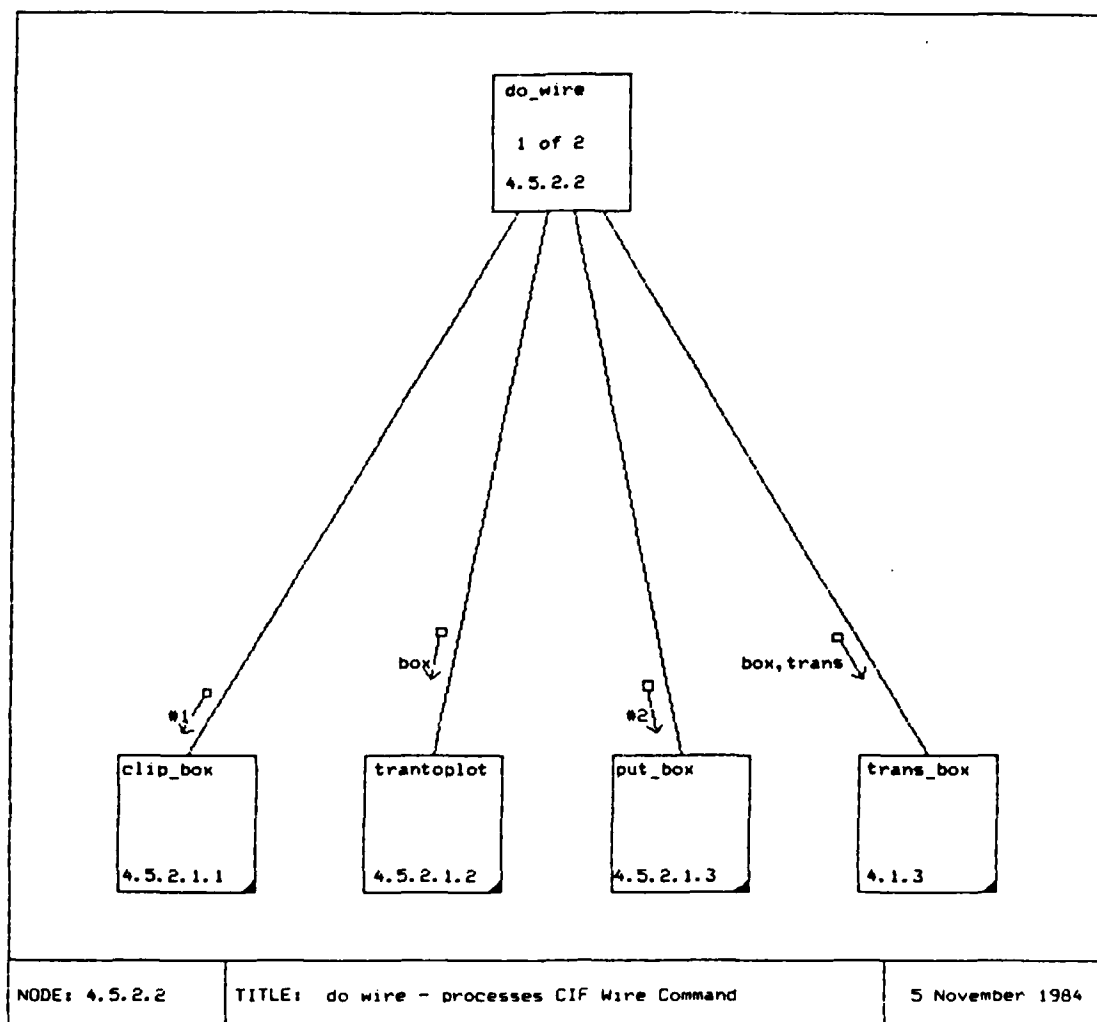


#1: com_string, sploc, &long
 #2: com_string, sploc, point

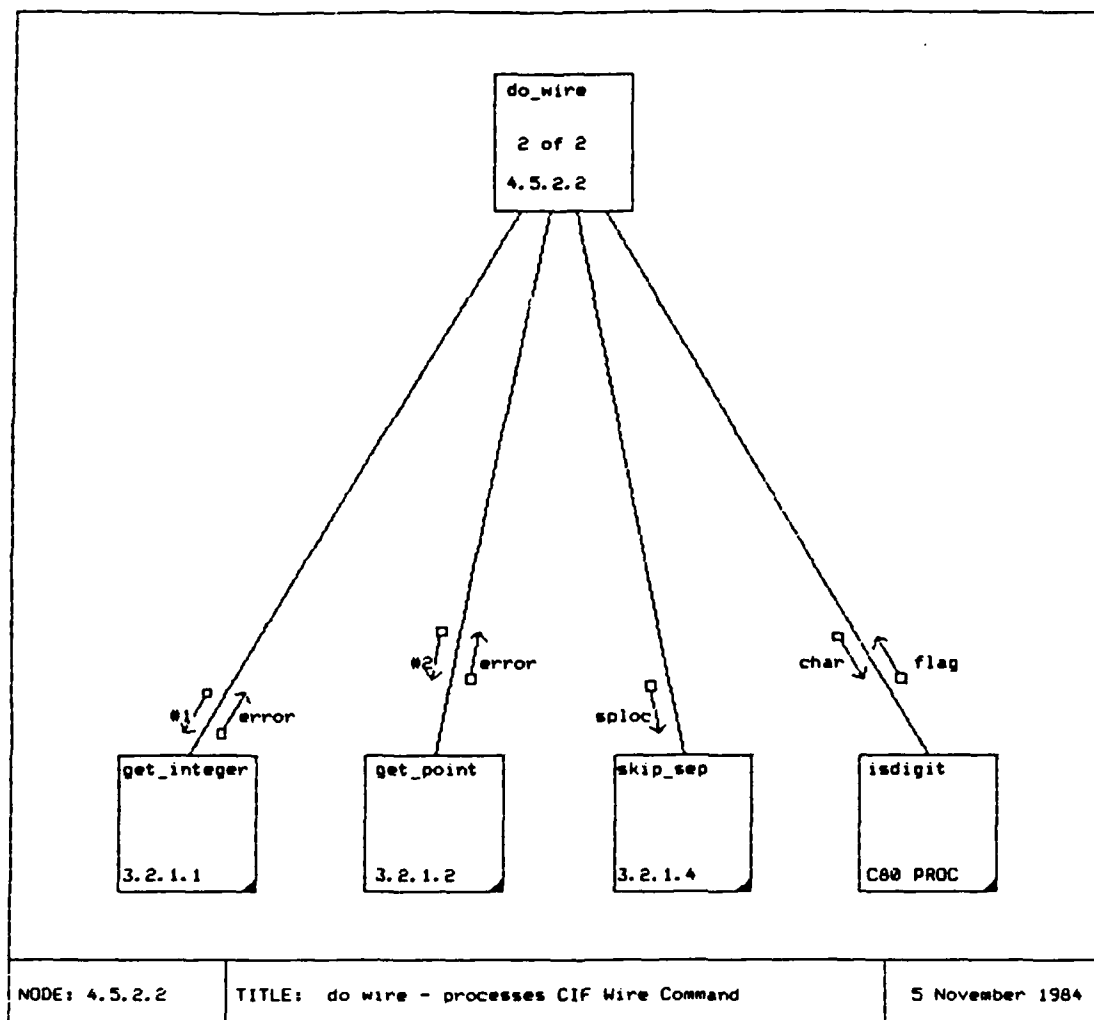




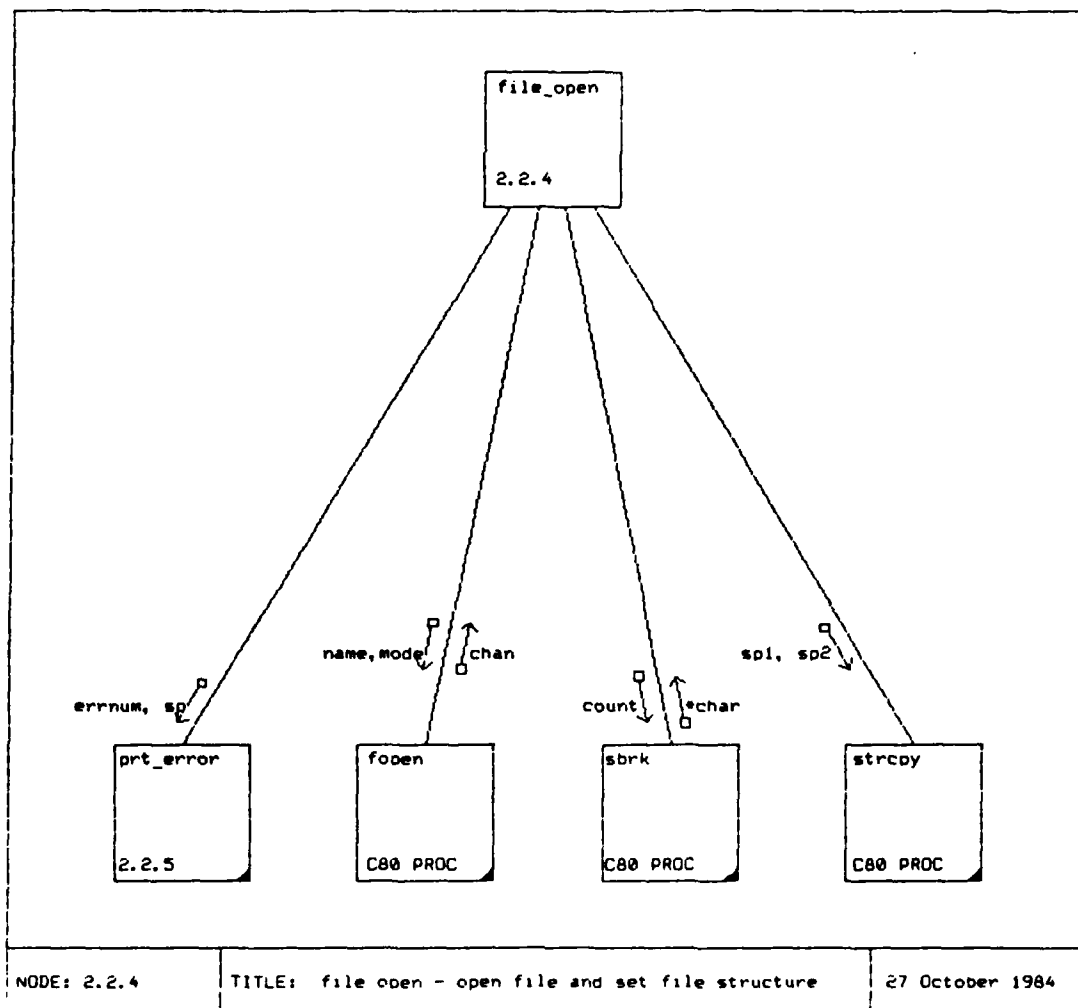
#1: com_string, sploc, &long
 #2: com_string, sploc, point

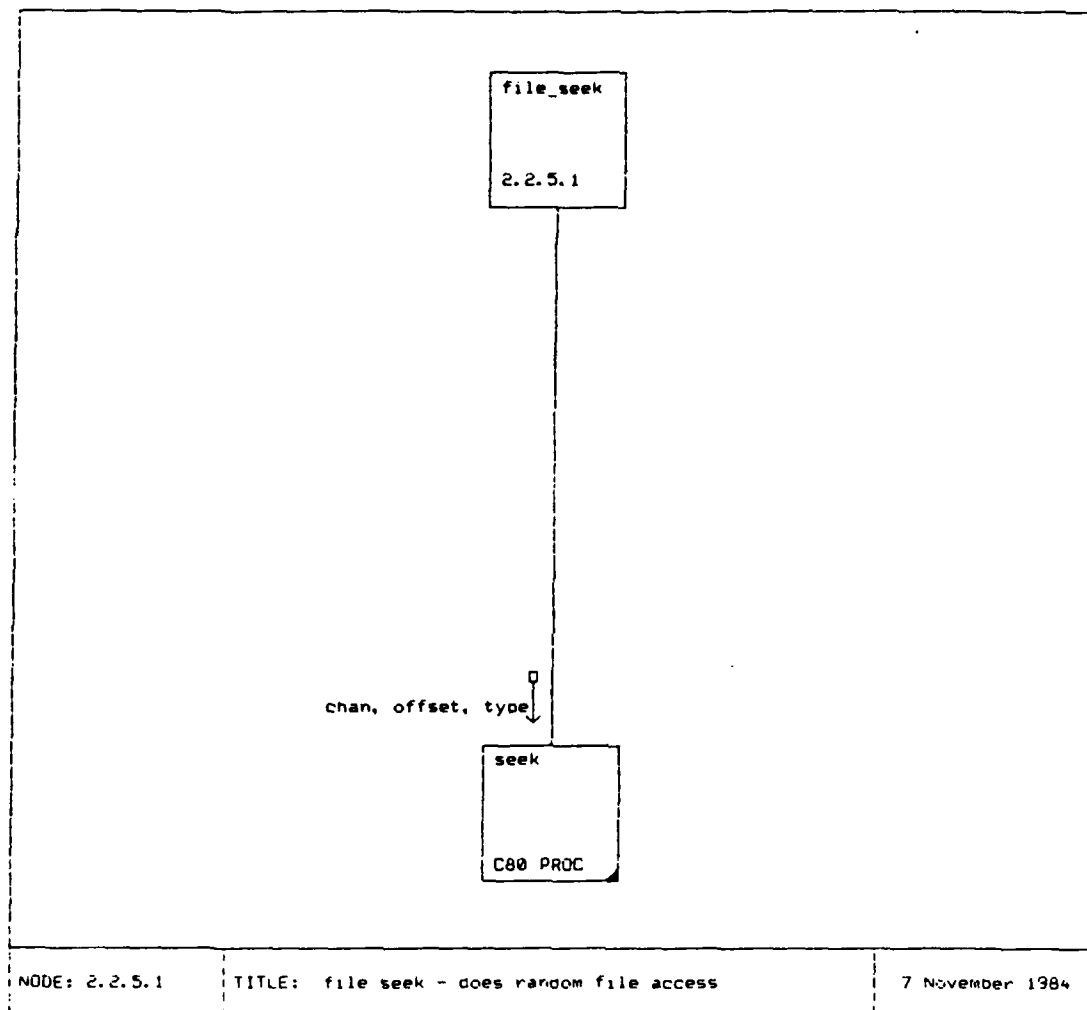


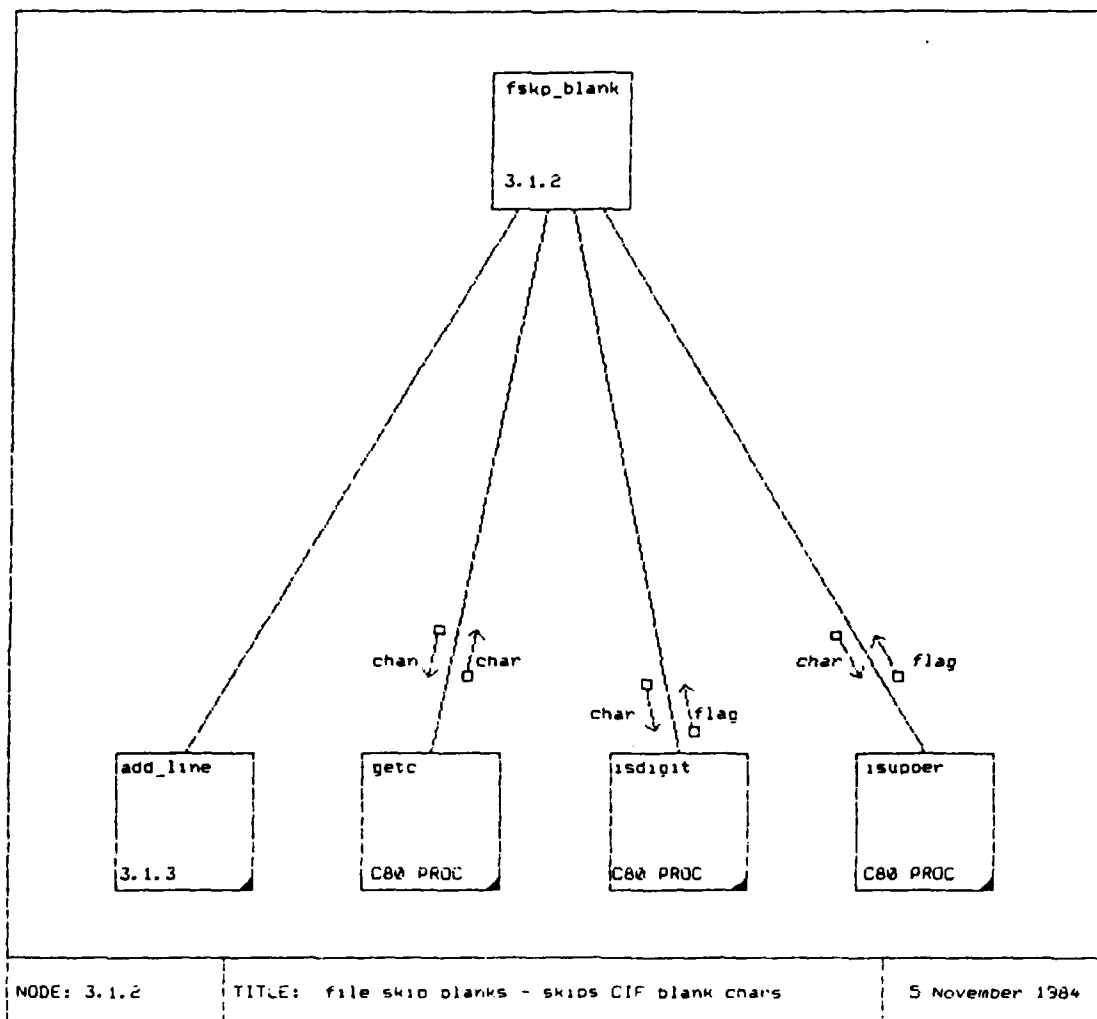
#1: box1, box2, cliprec
 #2: box, color, cliprec

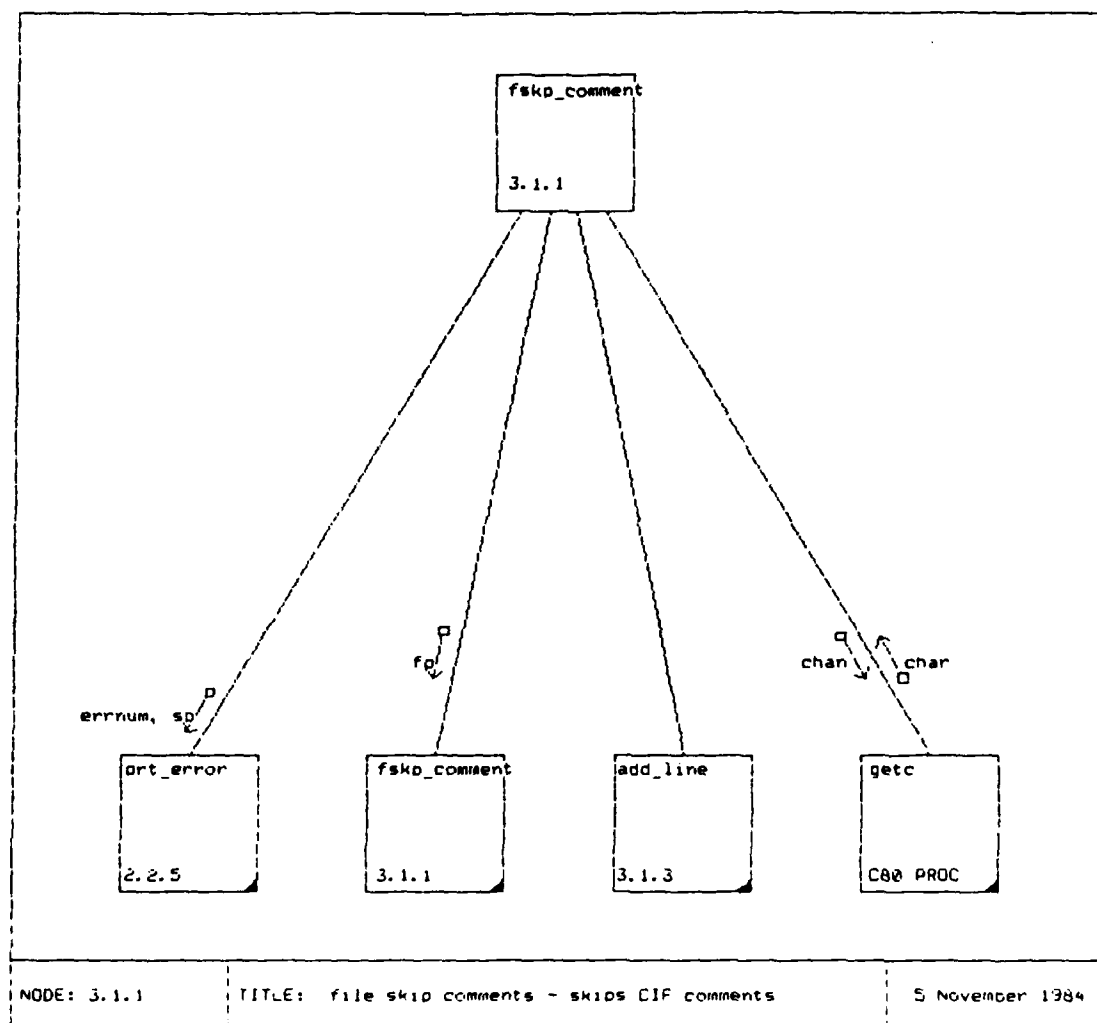


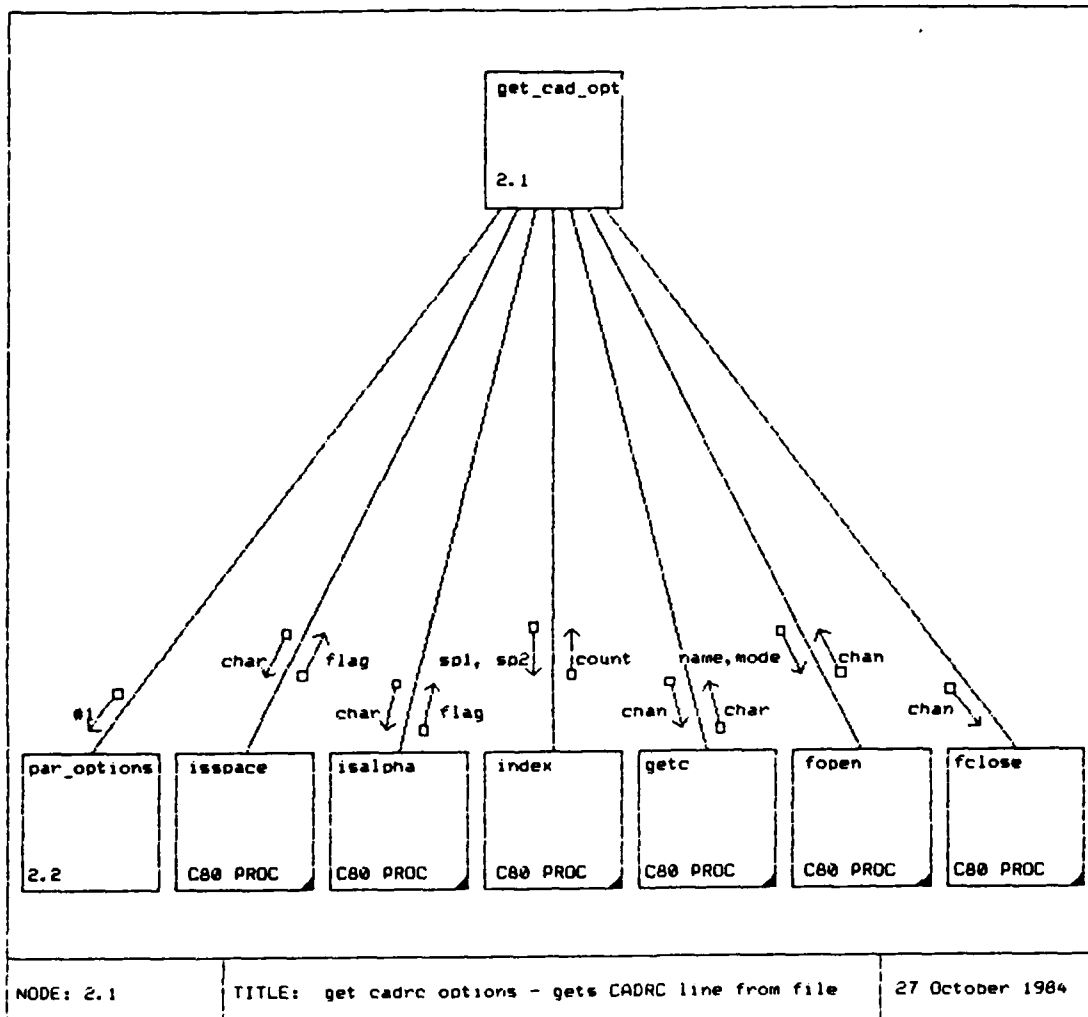
#1: com_string, sploc, &long
 #2: com_string, sploc, point



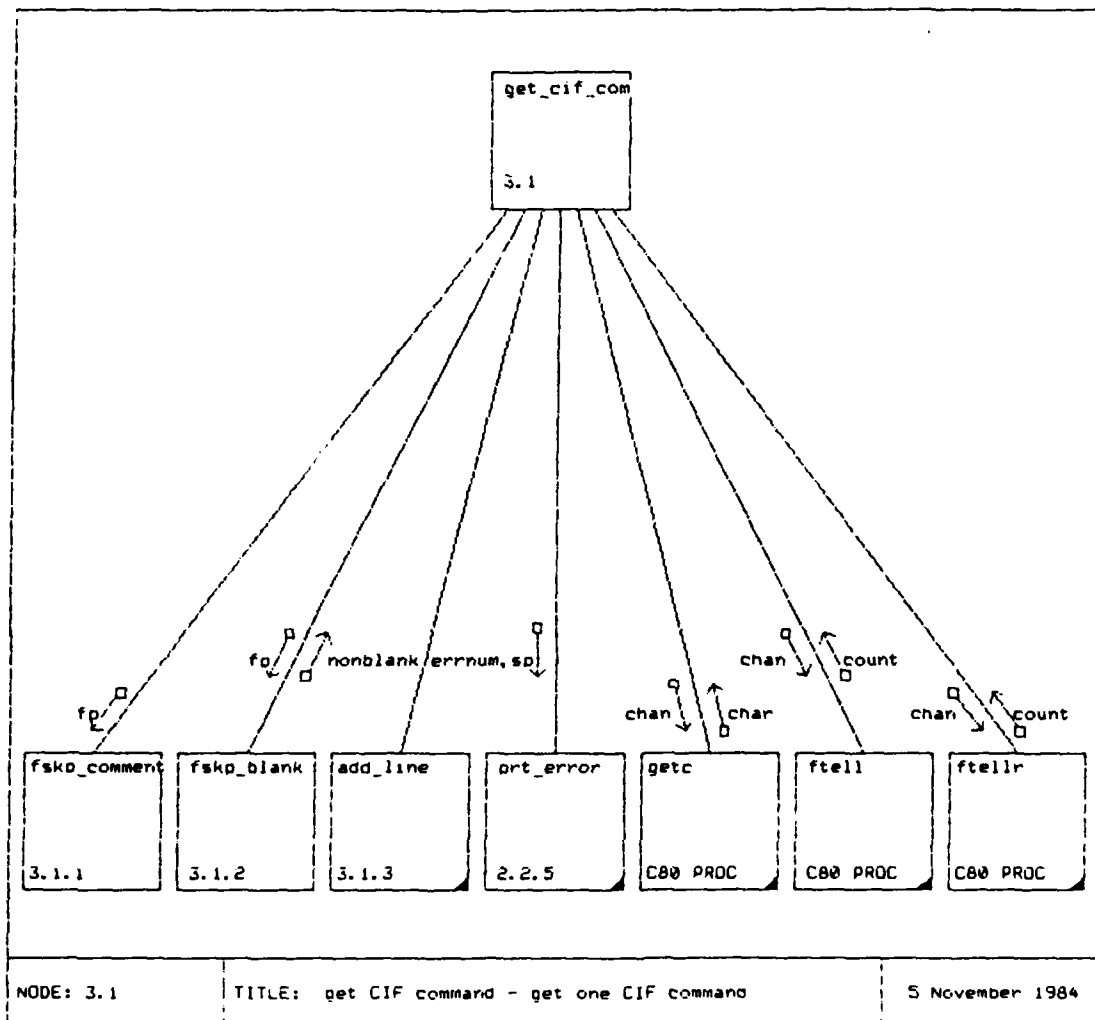


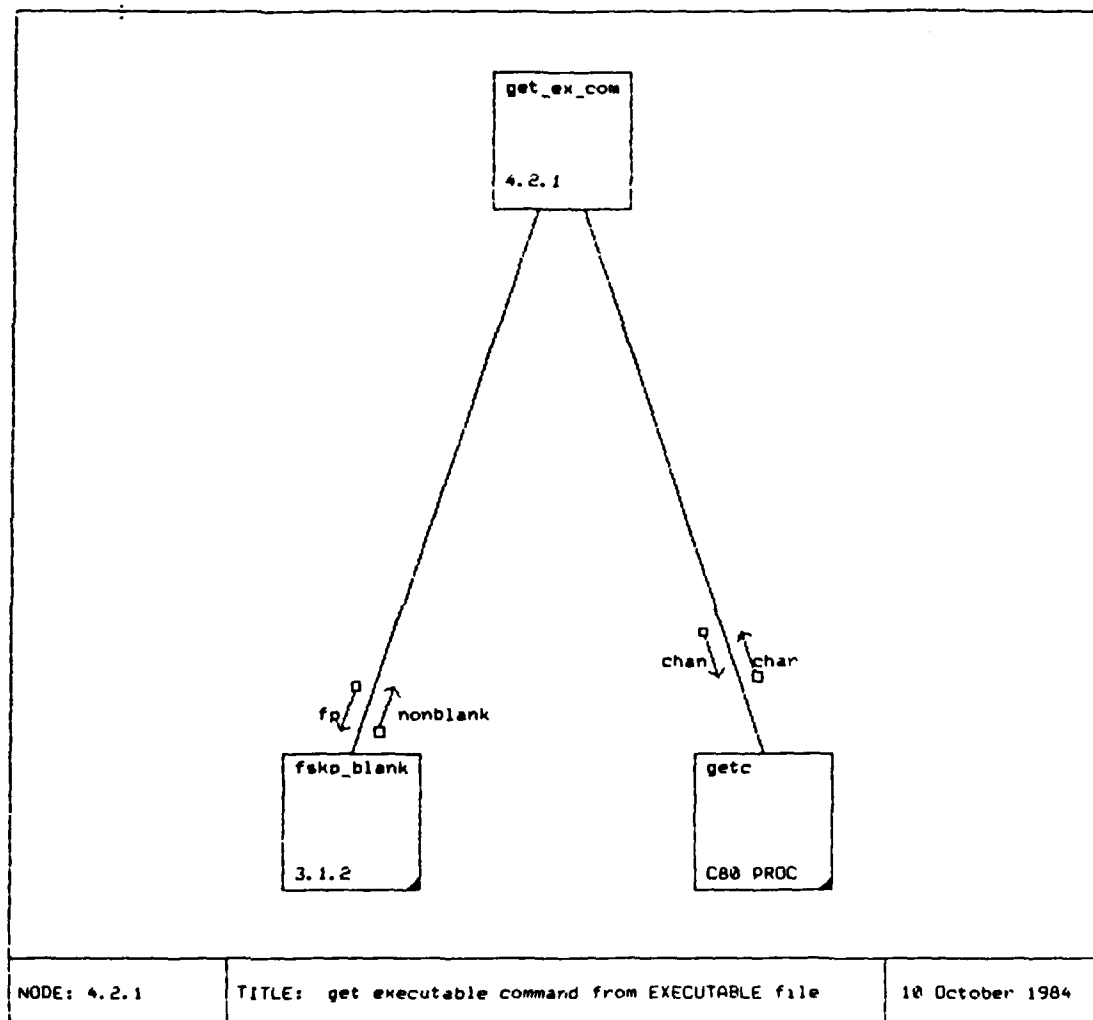


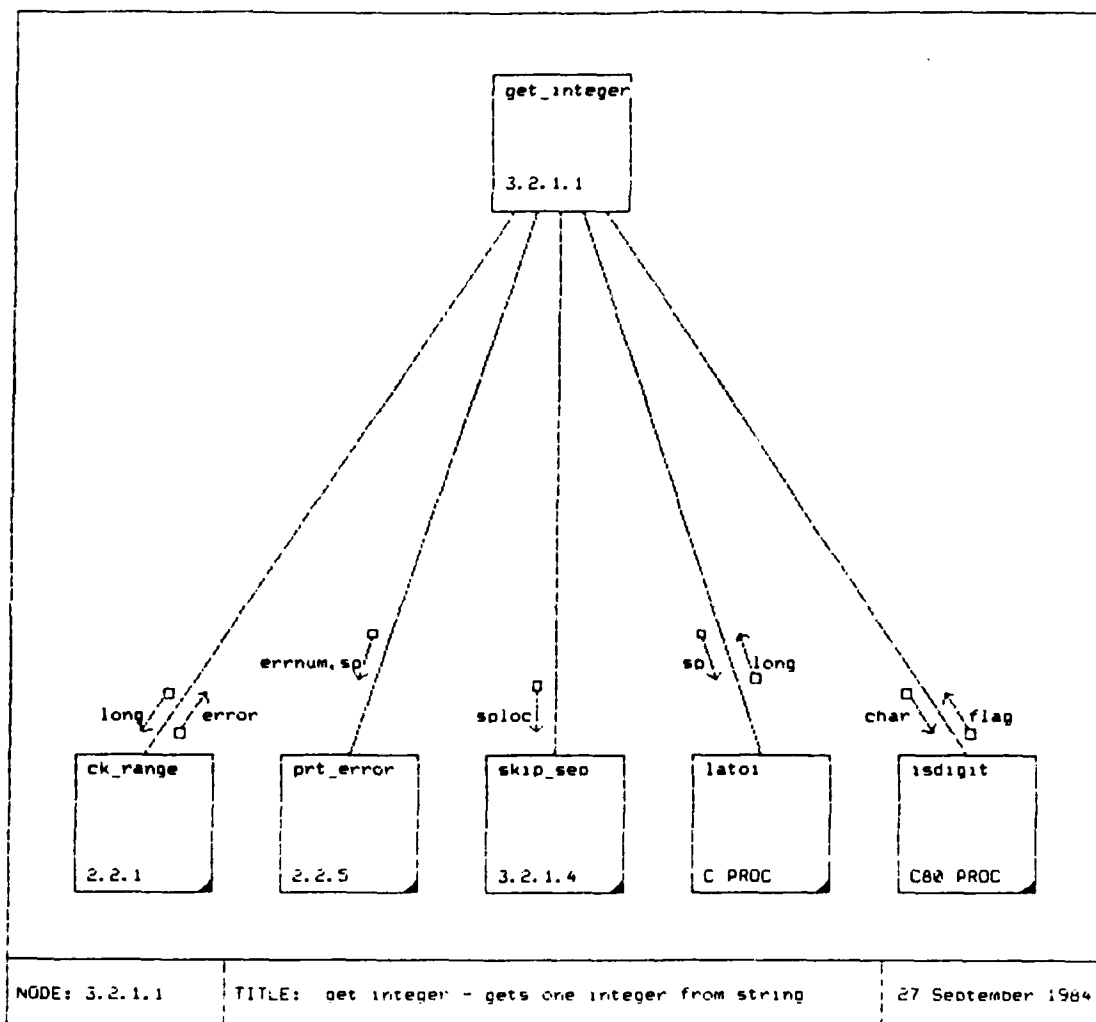


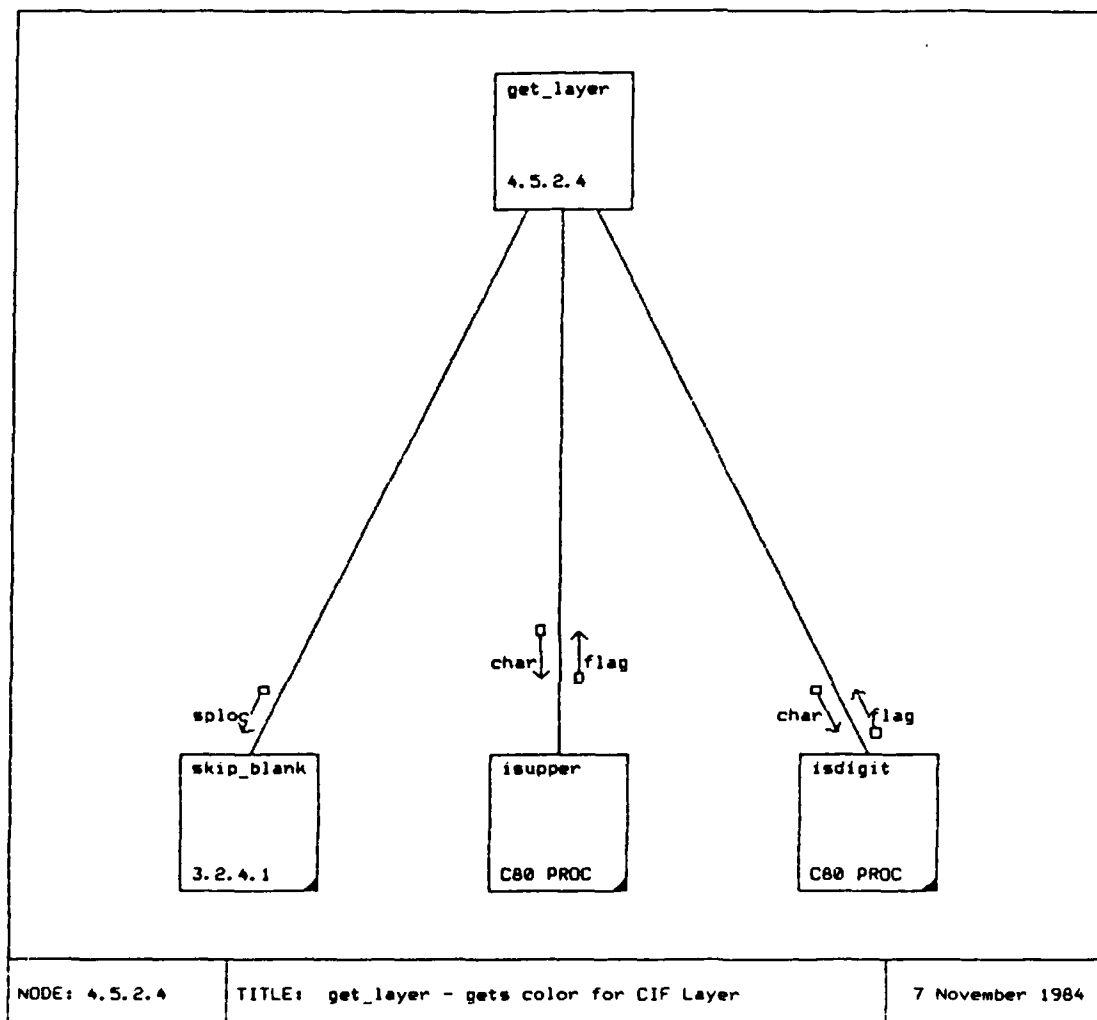


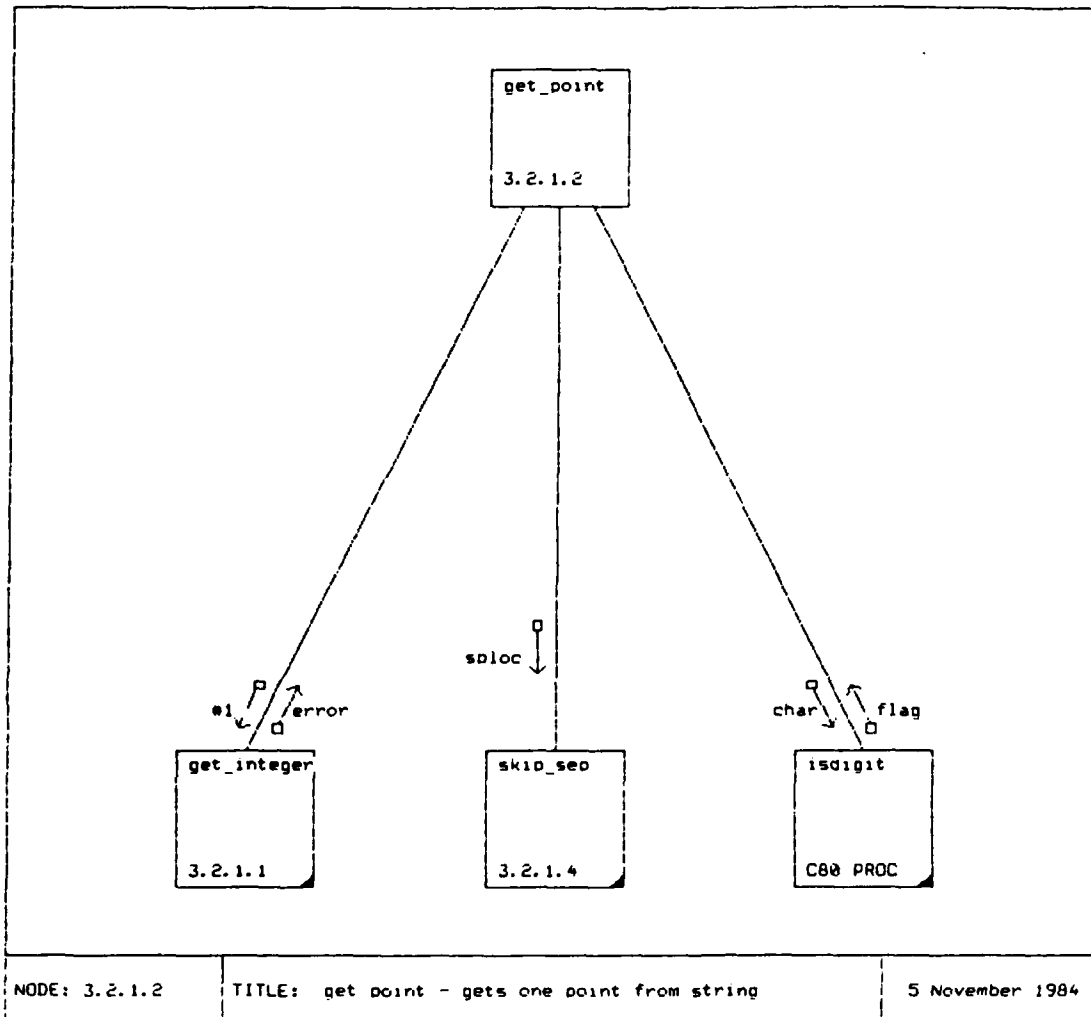
#1: argc, argv, cadrc



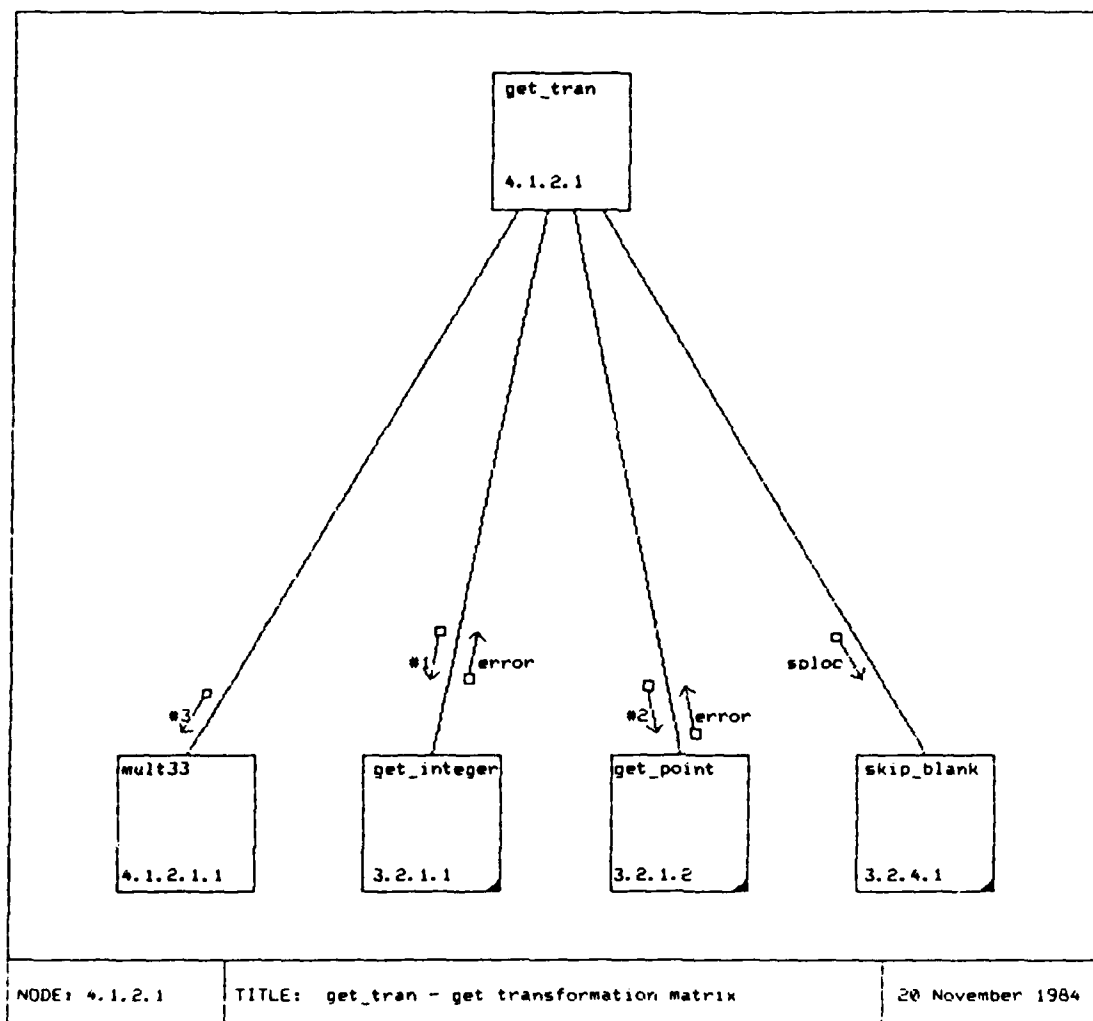




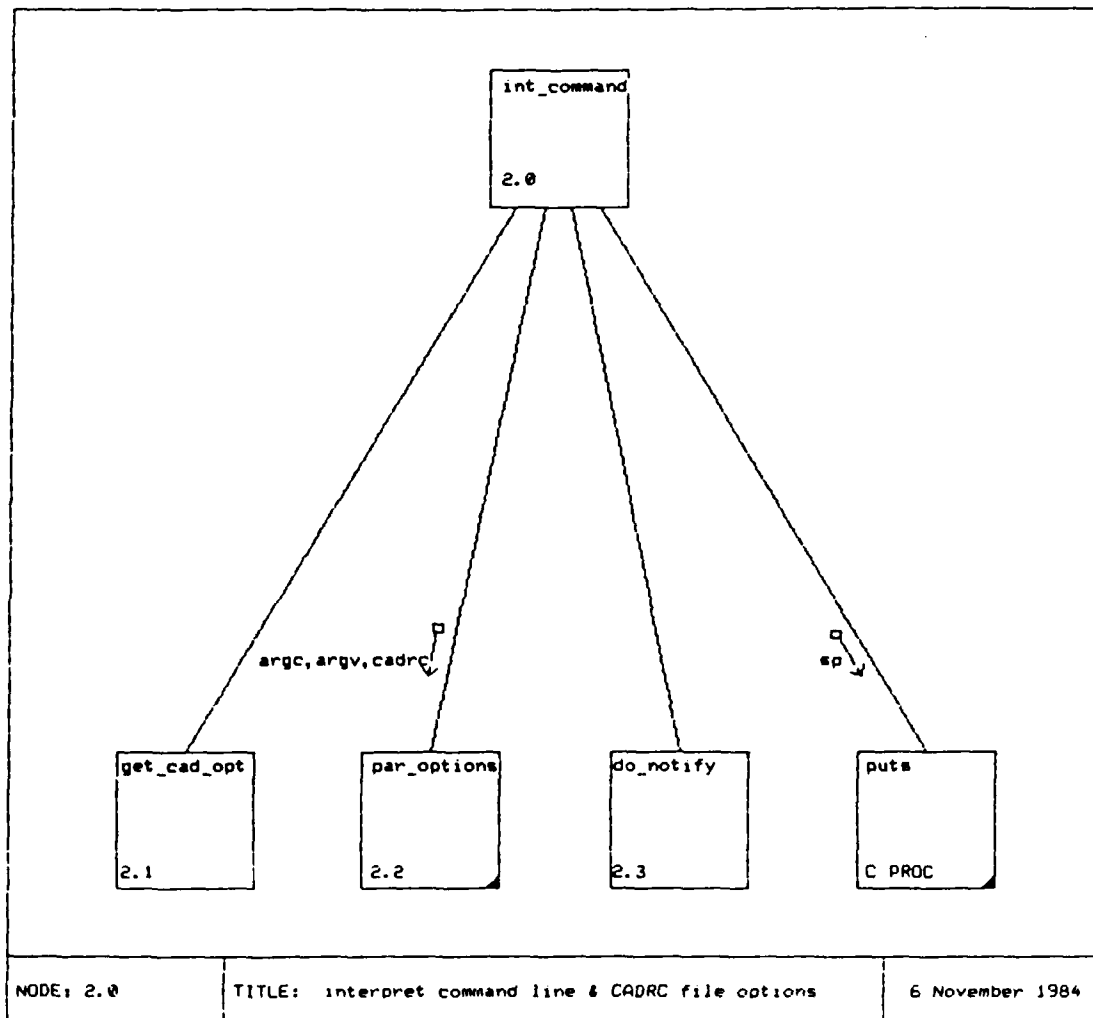


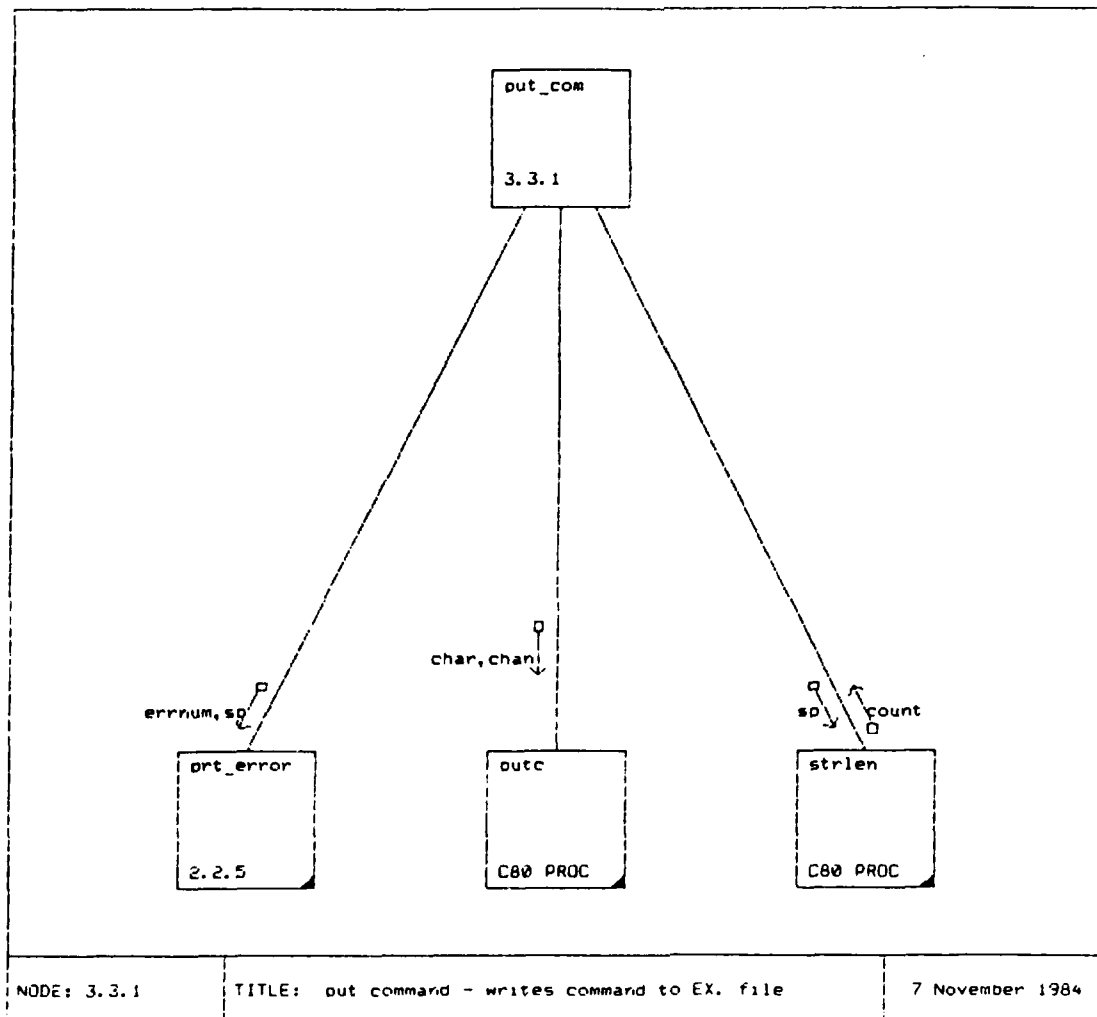


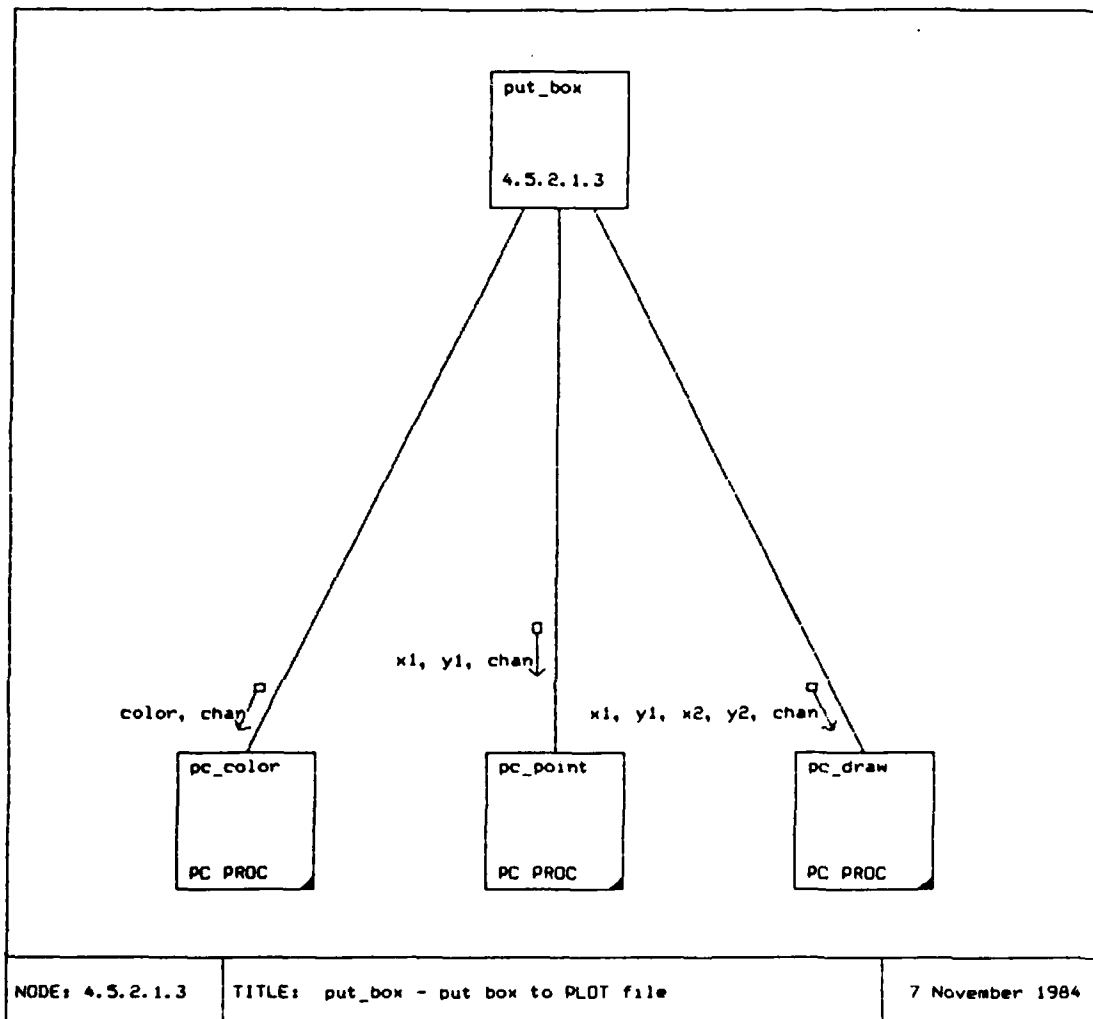
w1: ccm_string, sploc, &long

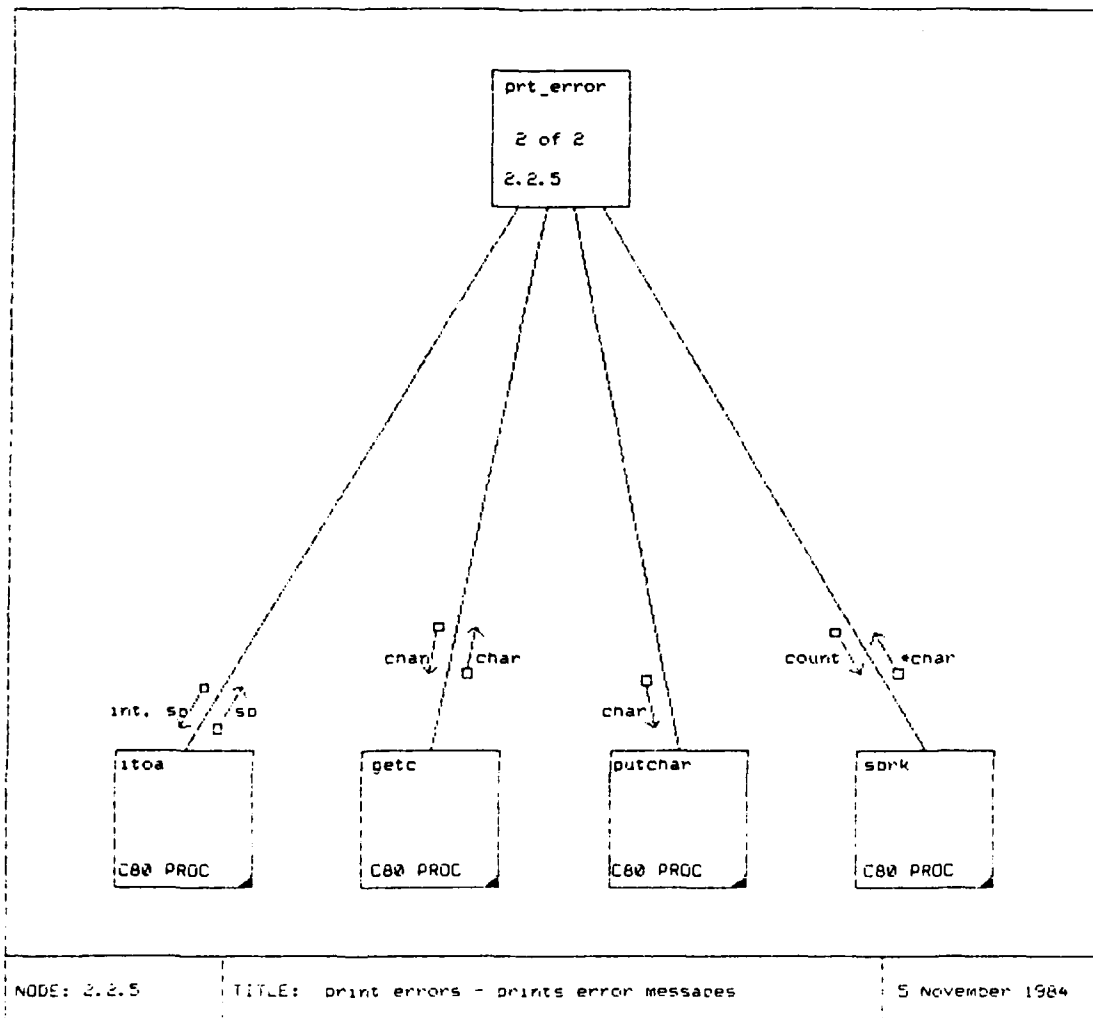


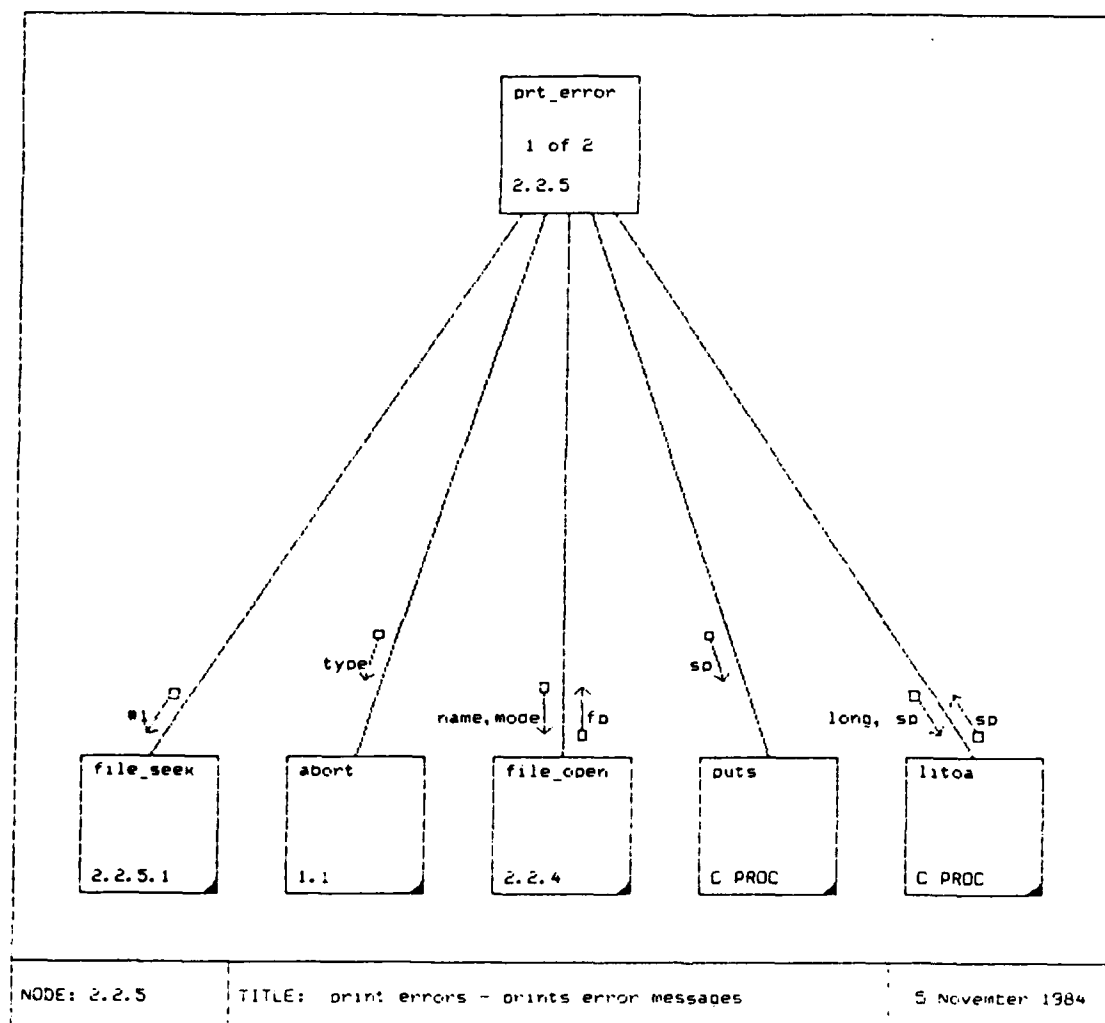
#1: com_string, sploc, &long
 #2: com_string, sploc, point
 #3: trans1, trans2, trans3



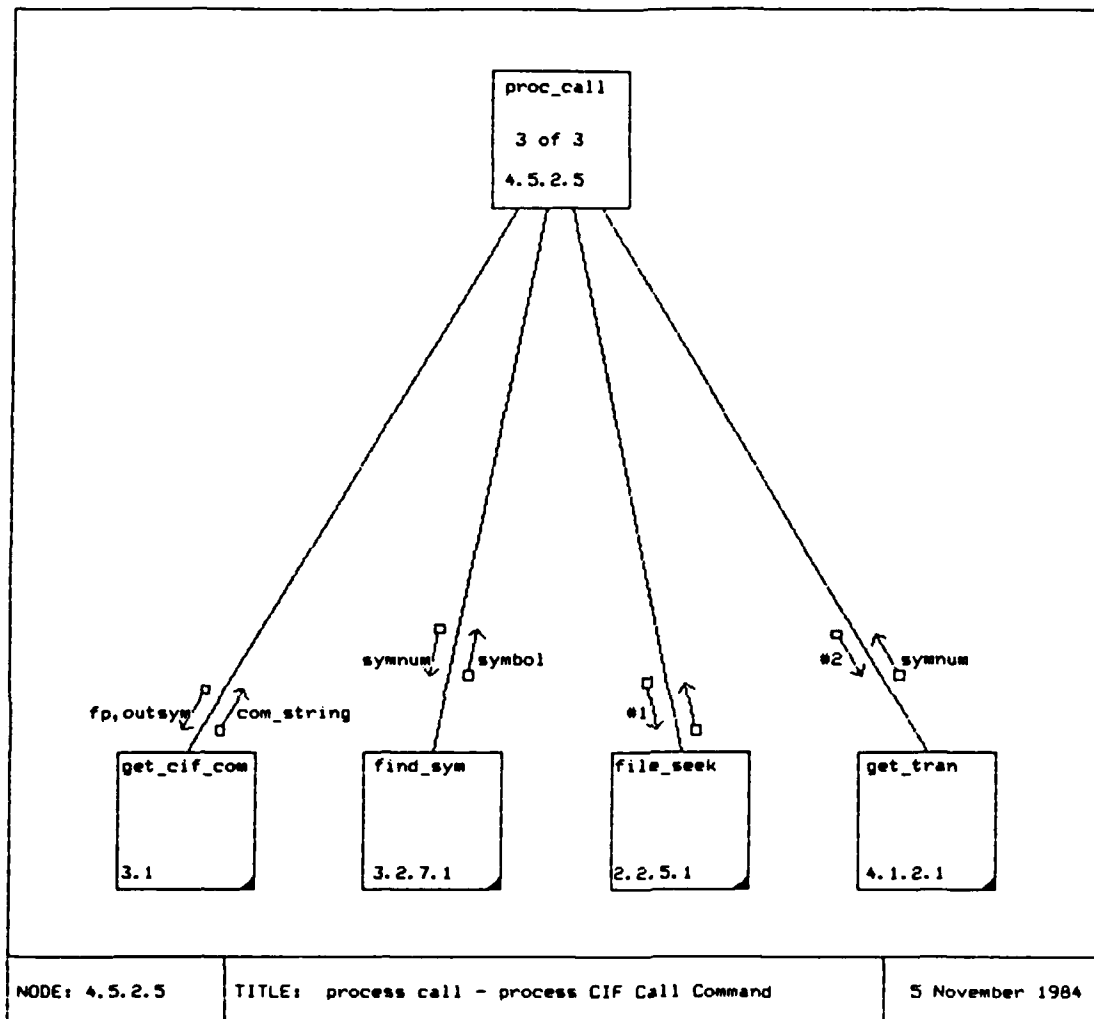




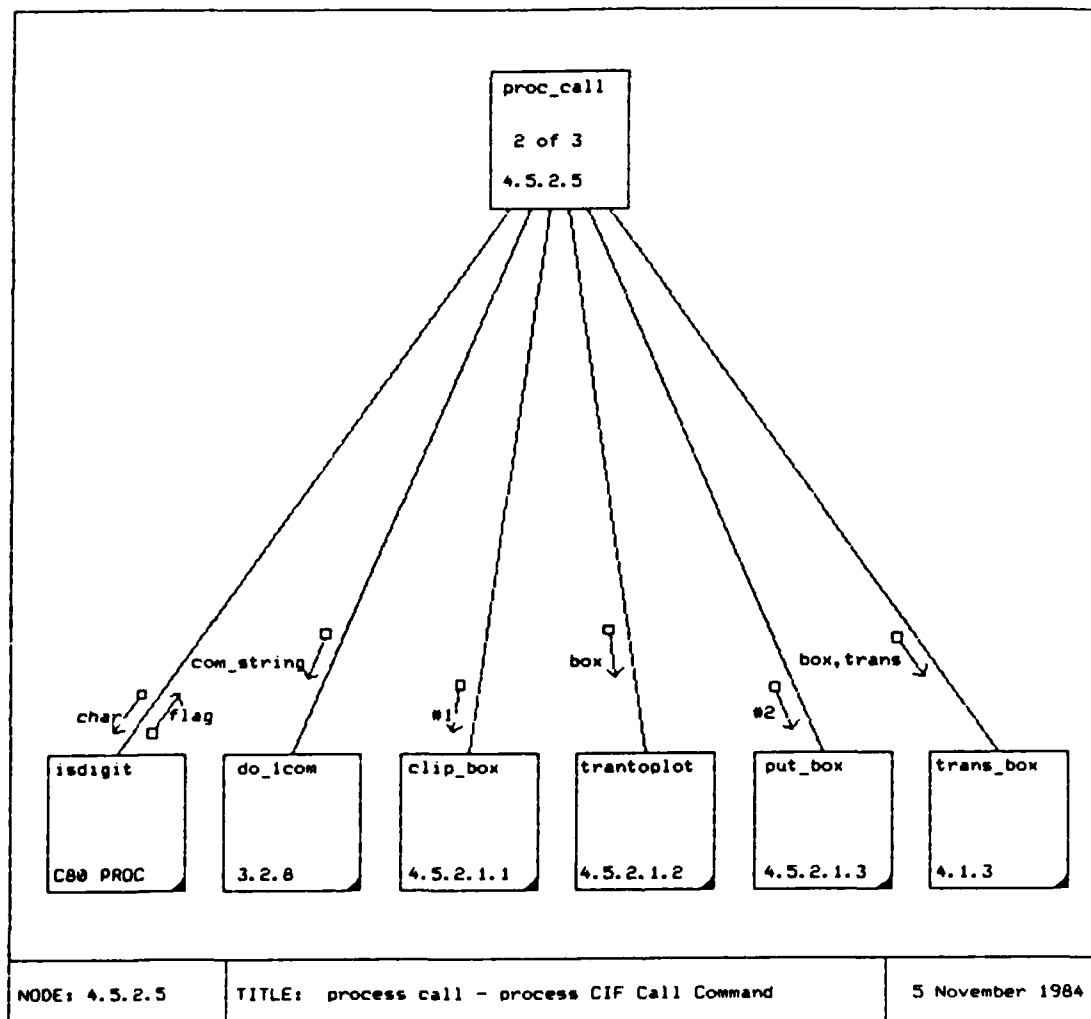




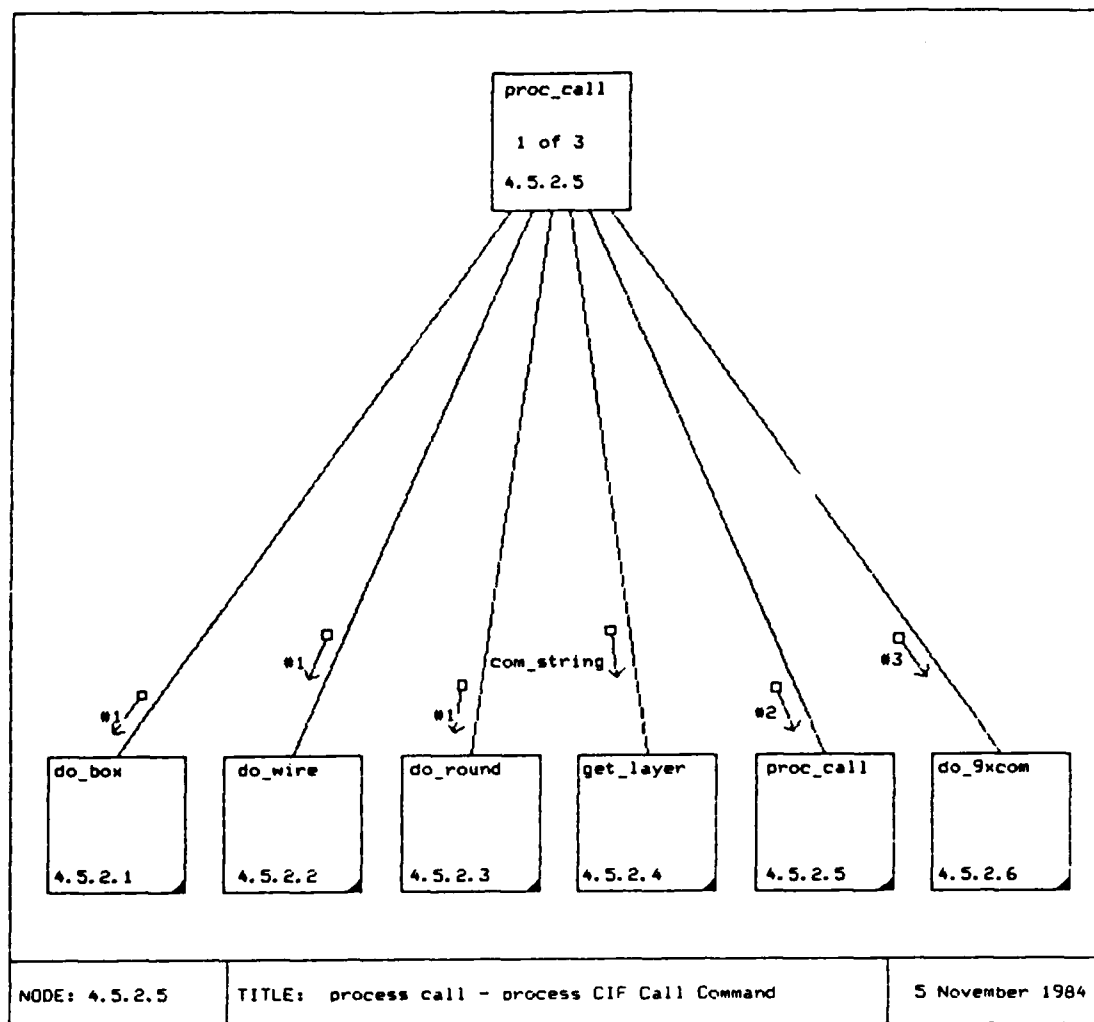
*1: fd, offset, record



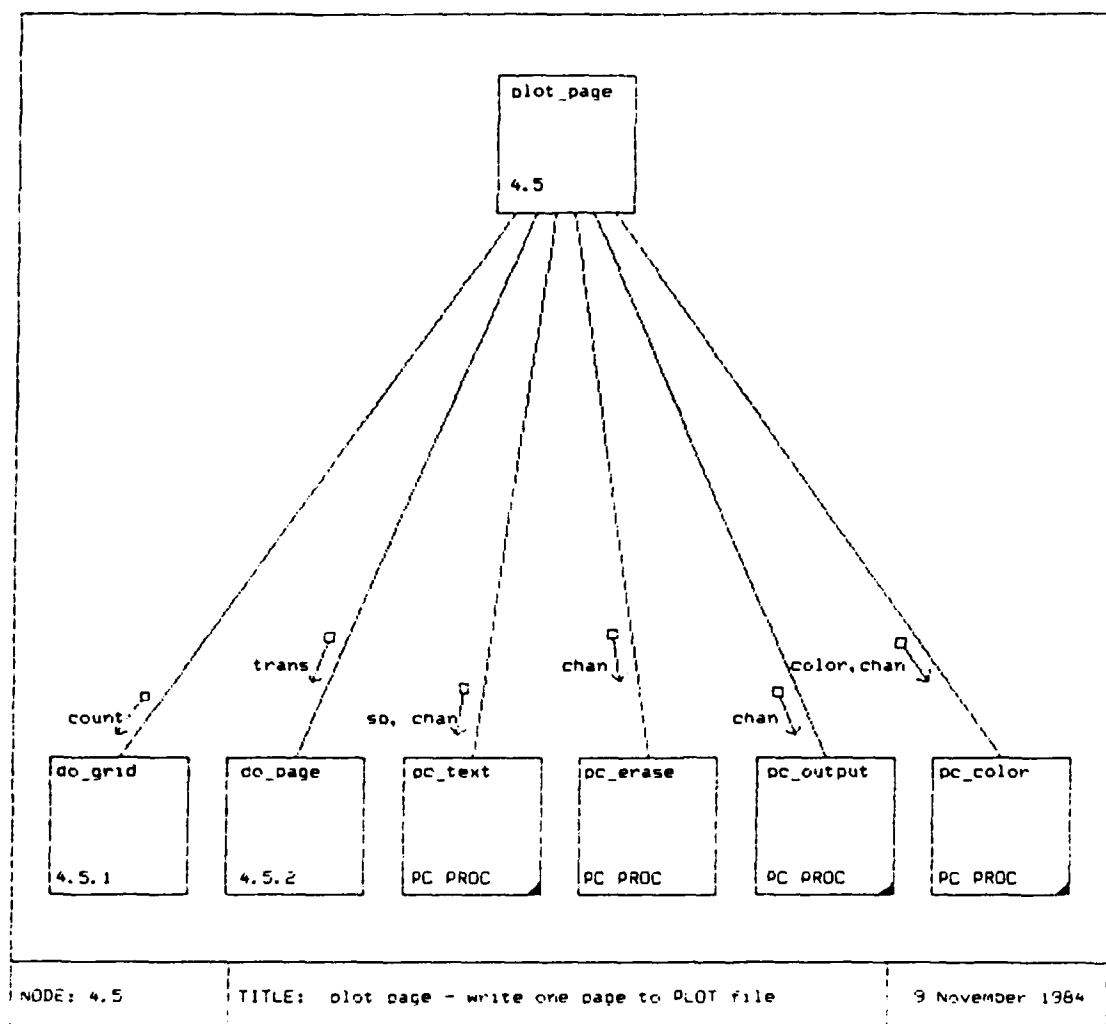
#1: fp, offset, record
#2: com_string, scale, trans1, trans2

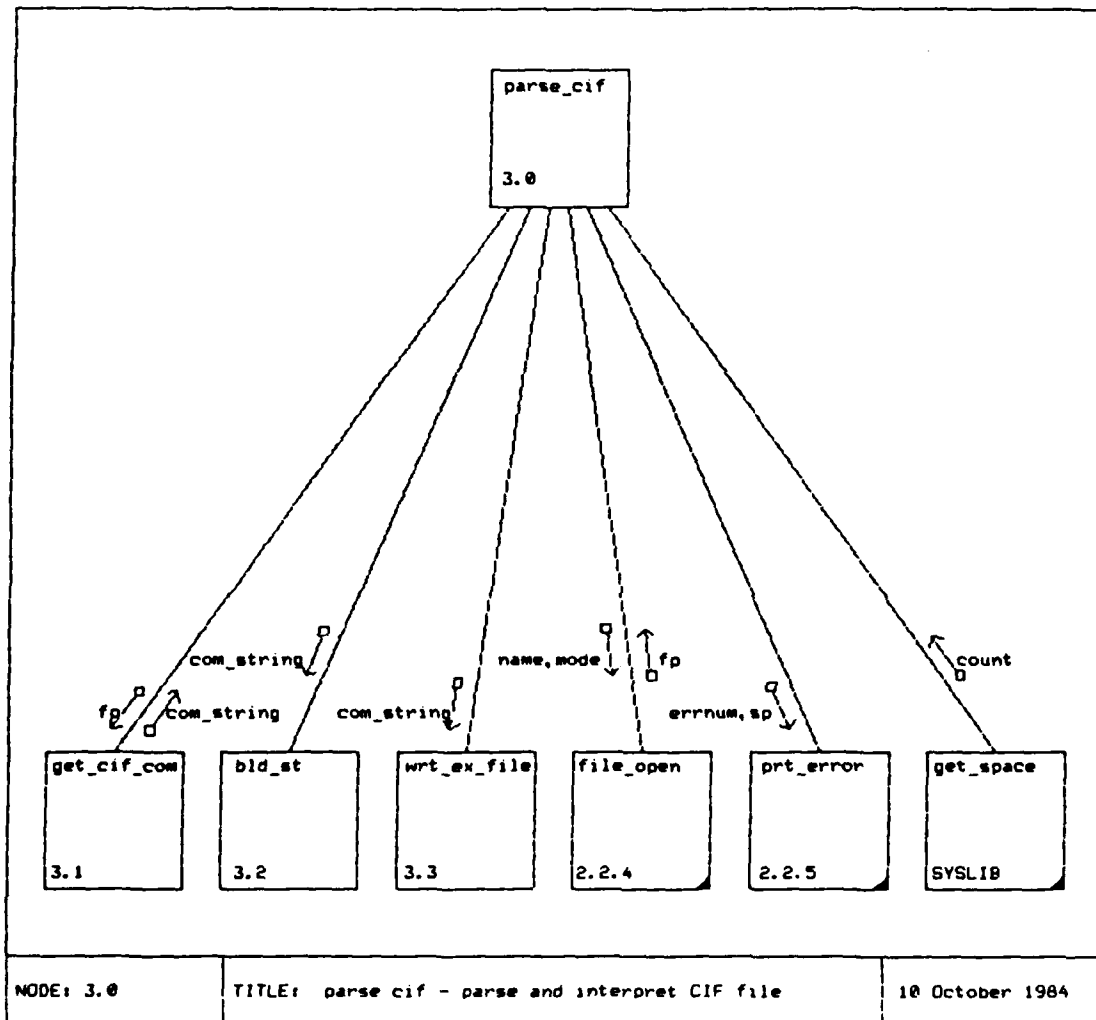


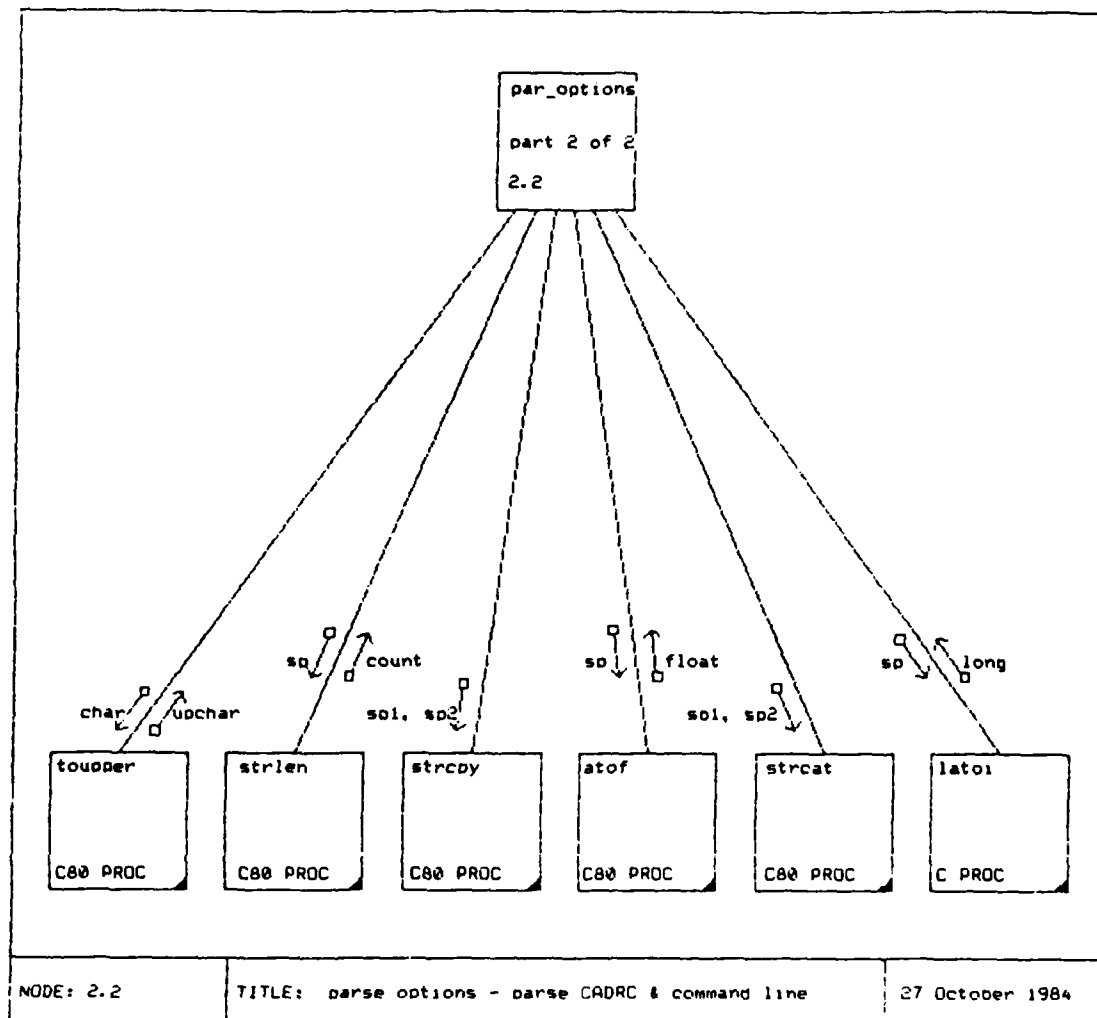
#1: box1, box2, cliprec
 #2: box, color, cliprec

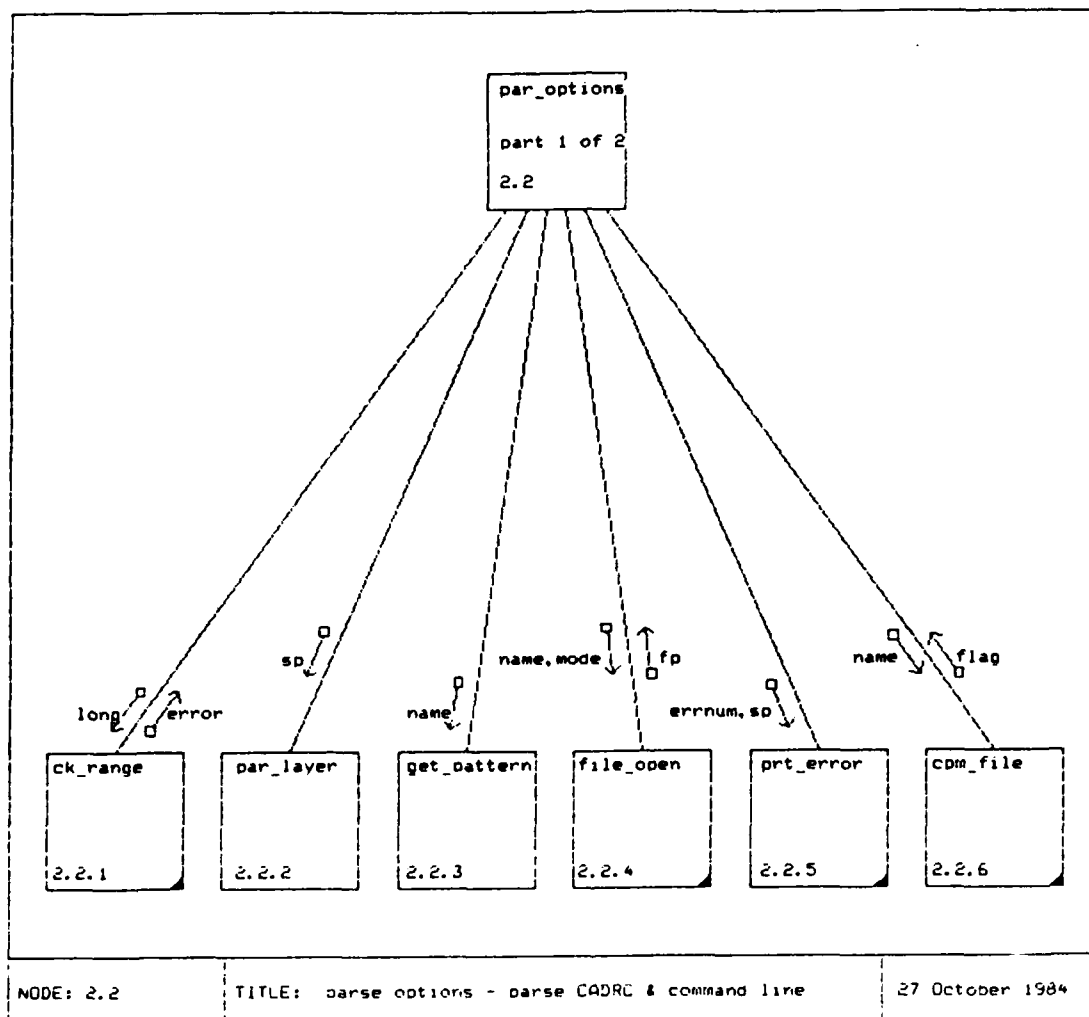


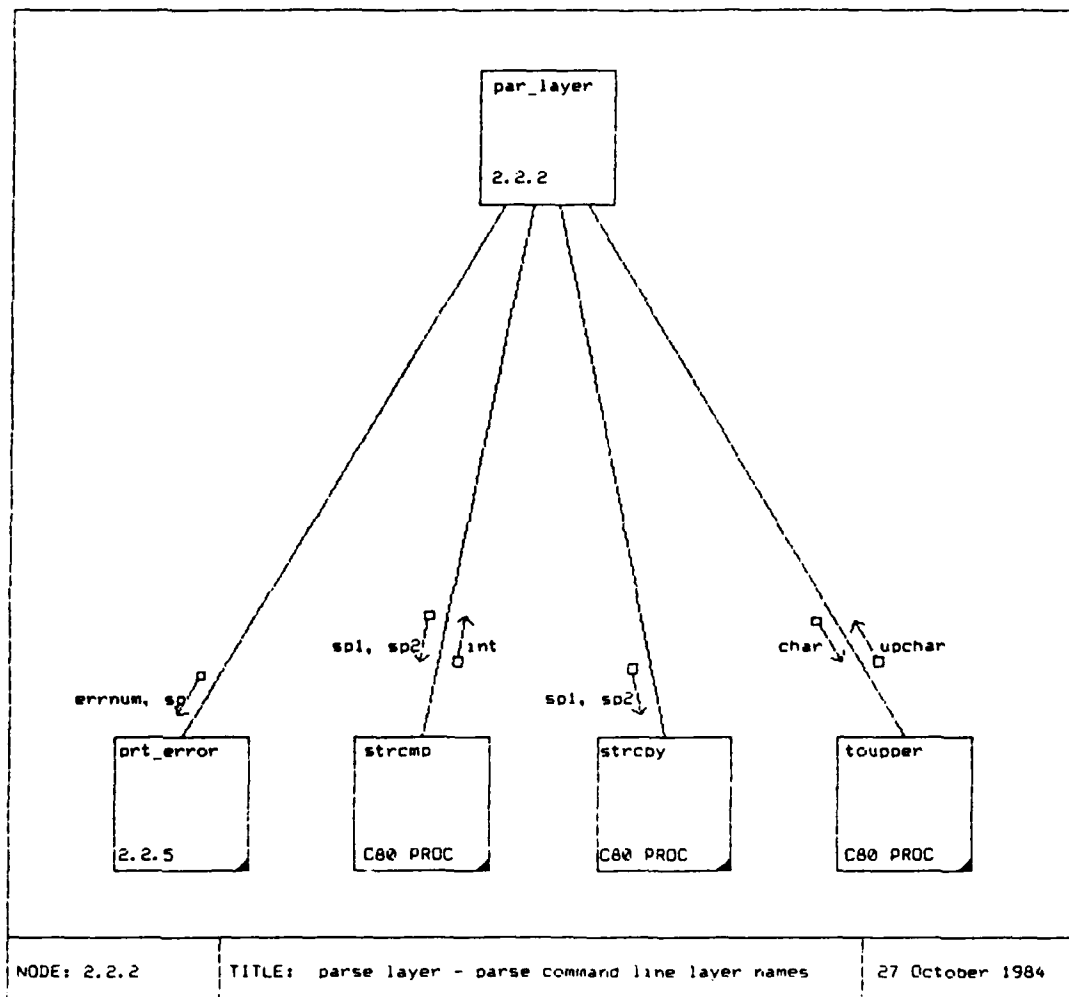
#1: com_string, trans, color, scale
 #2: com_string, trans, scale
 #3: com_string, trans, symbol, scale

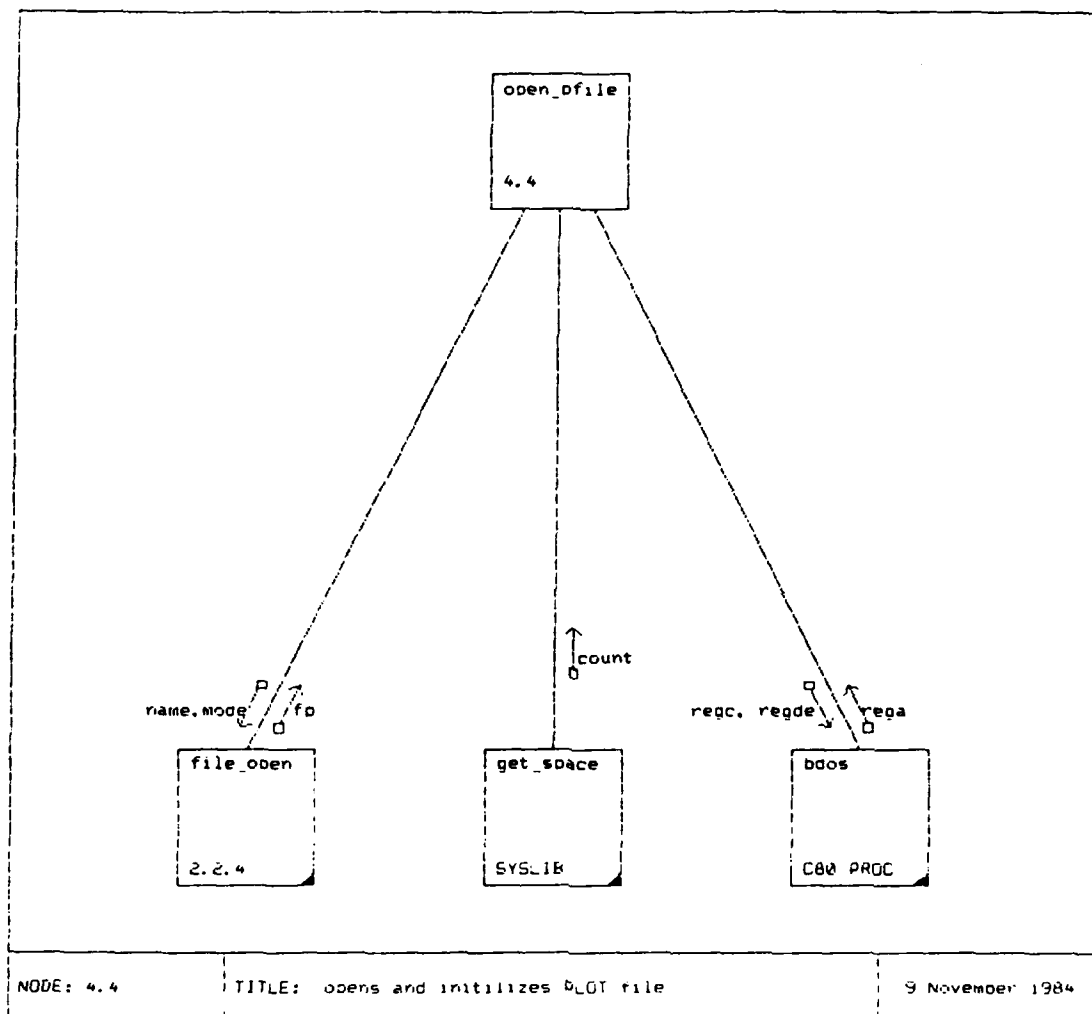


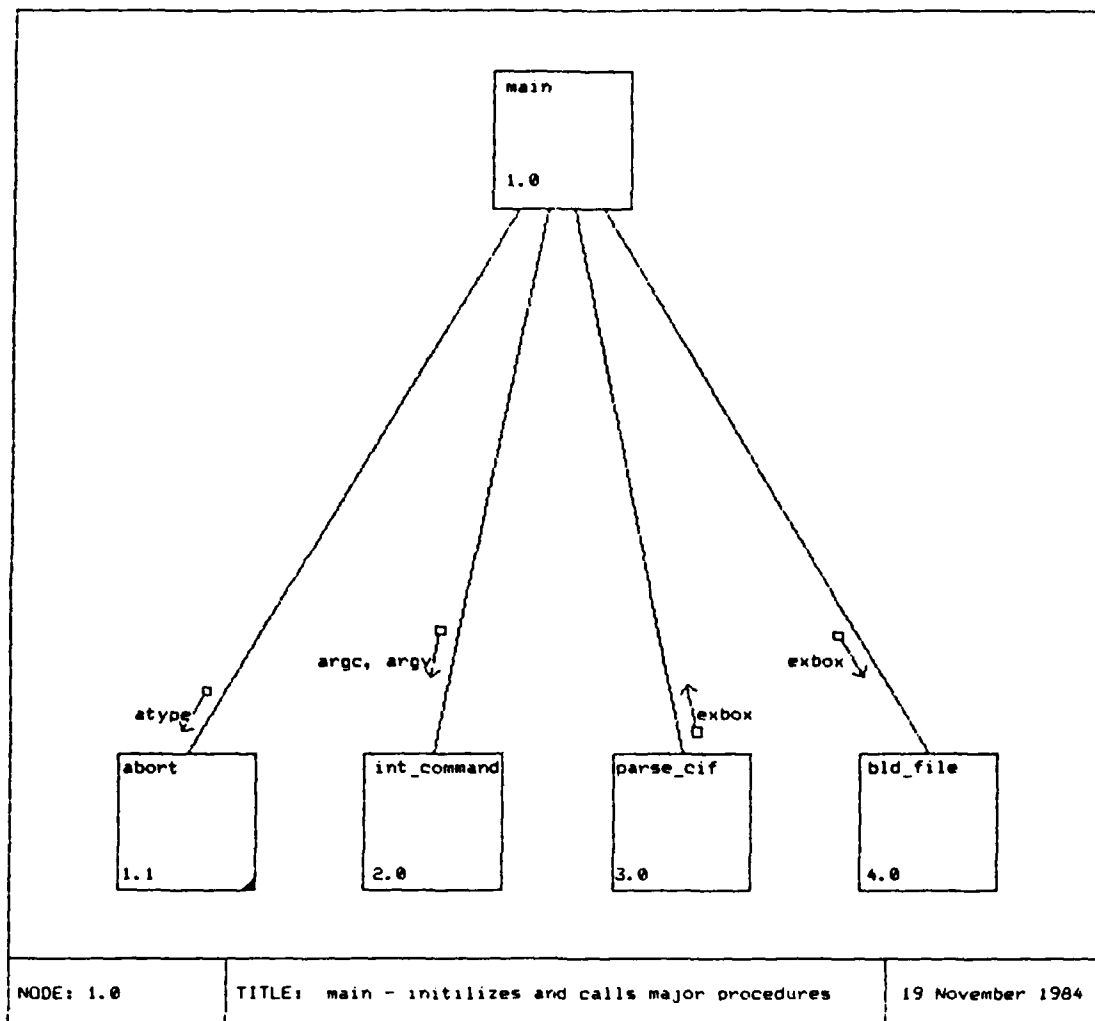


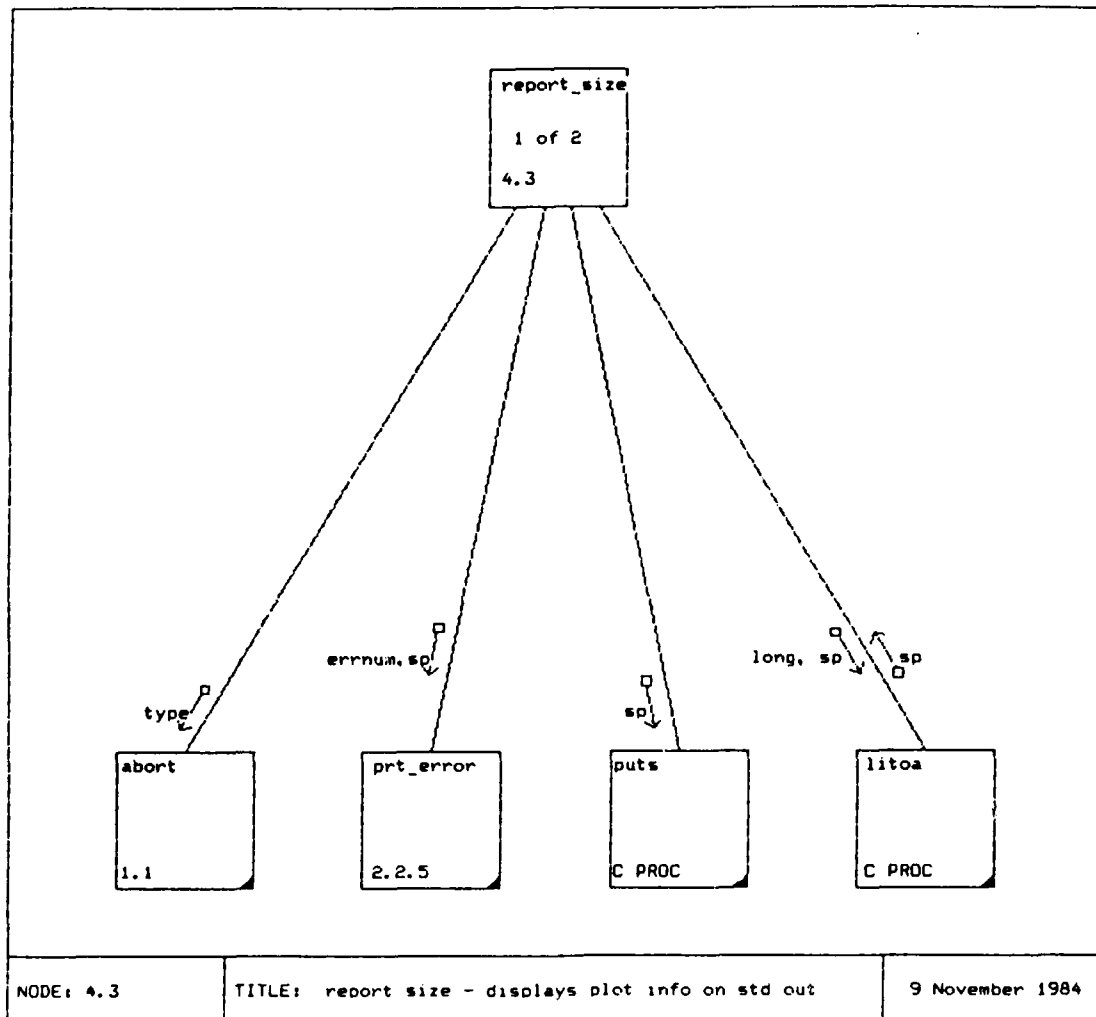


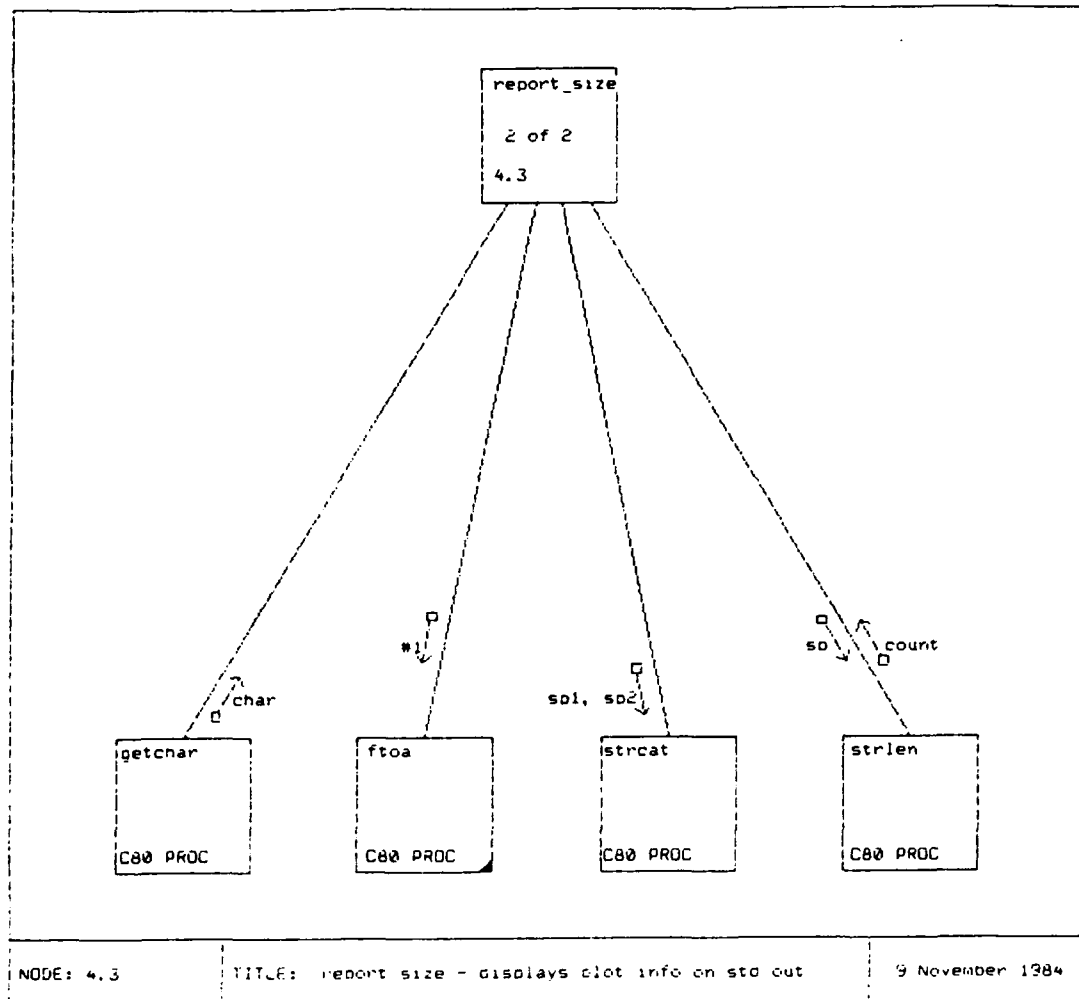




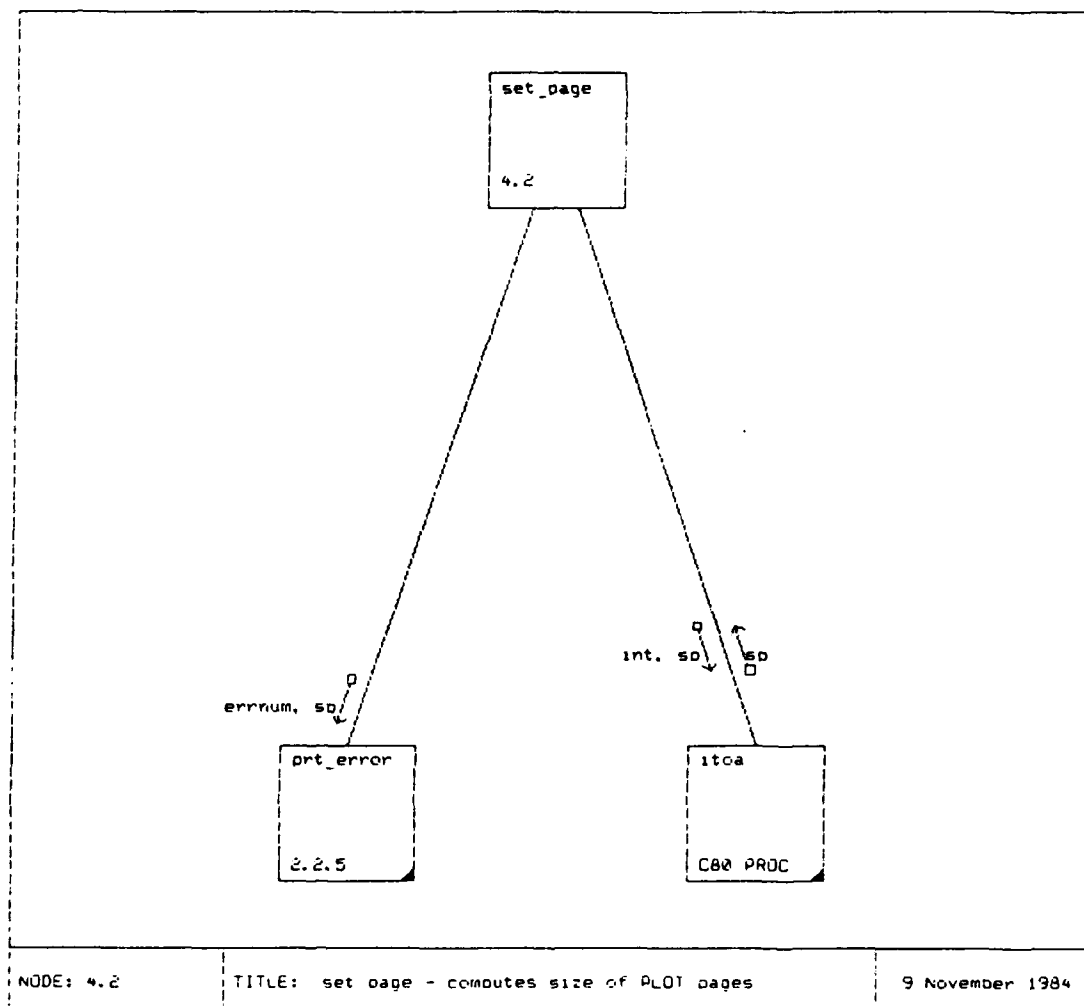


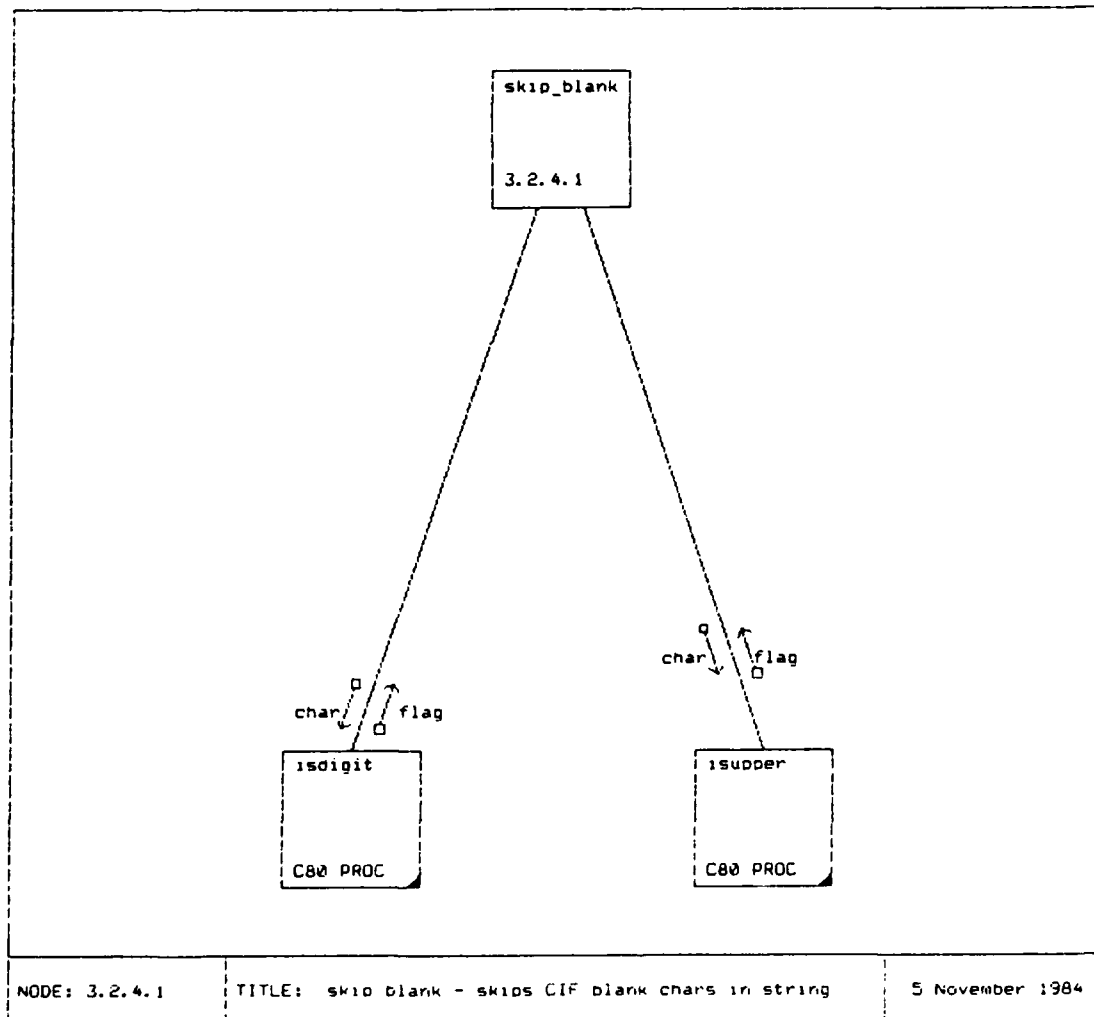


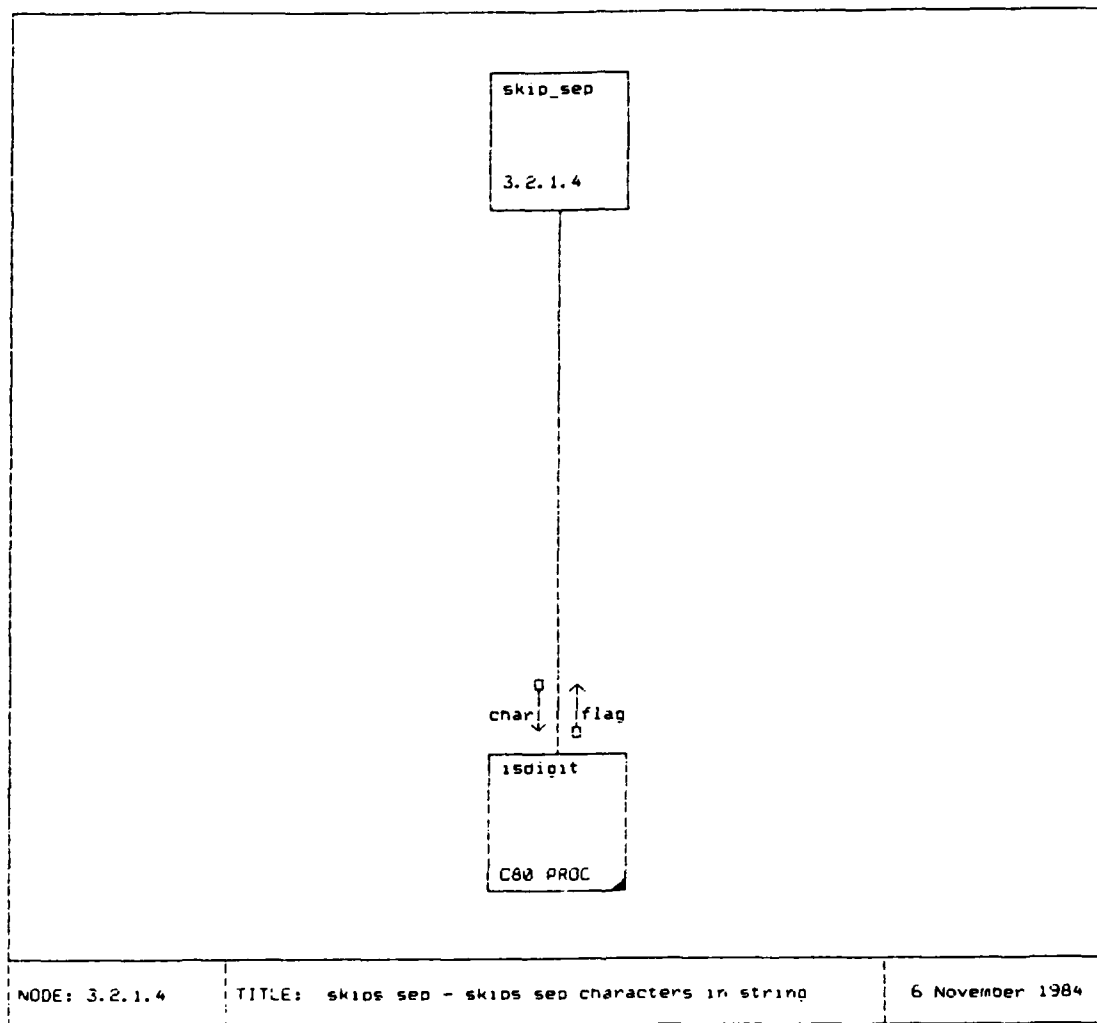


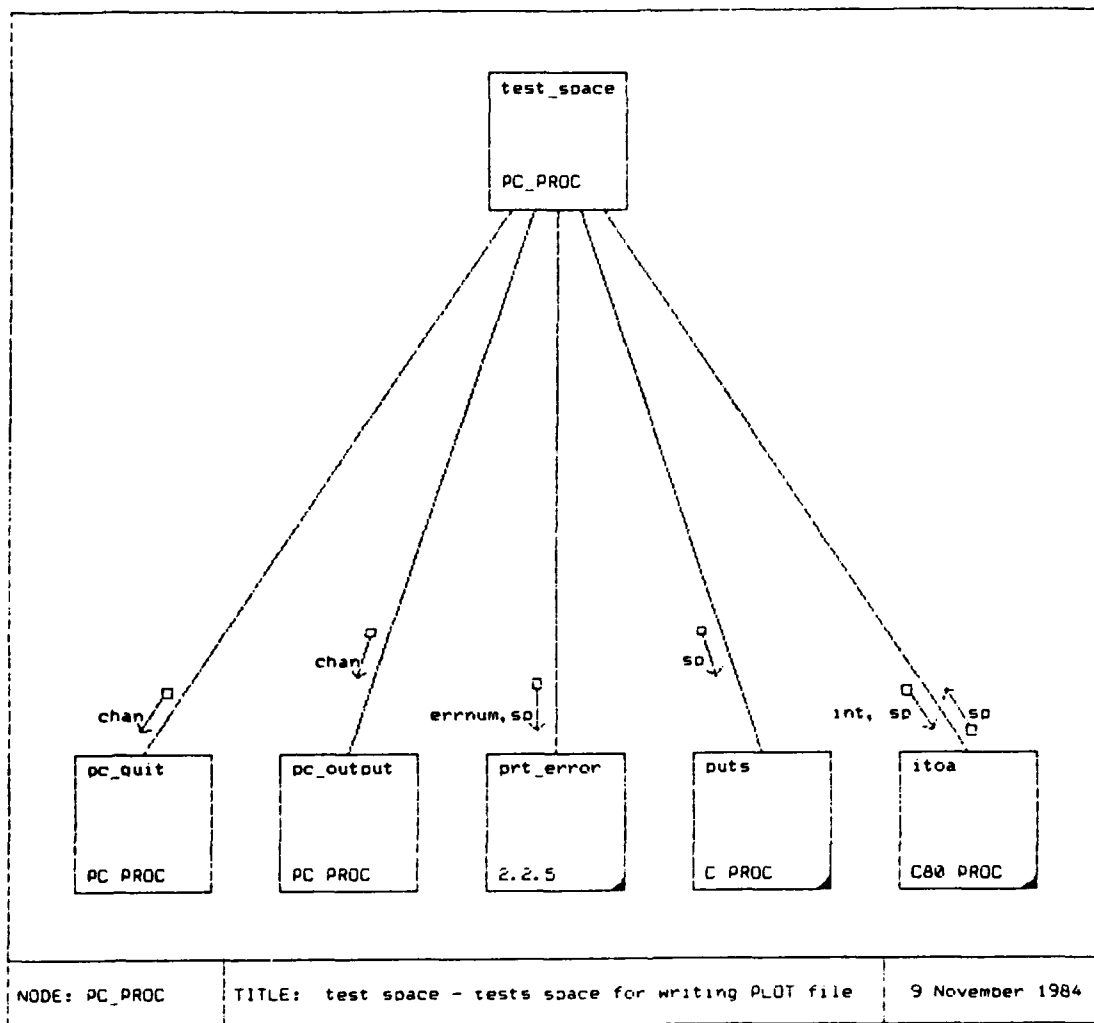


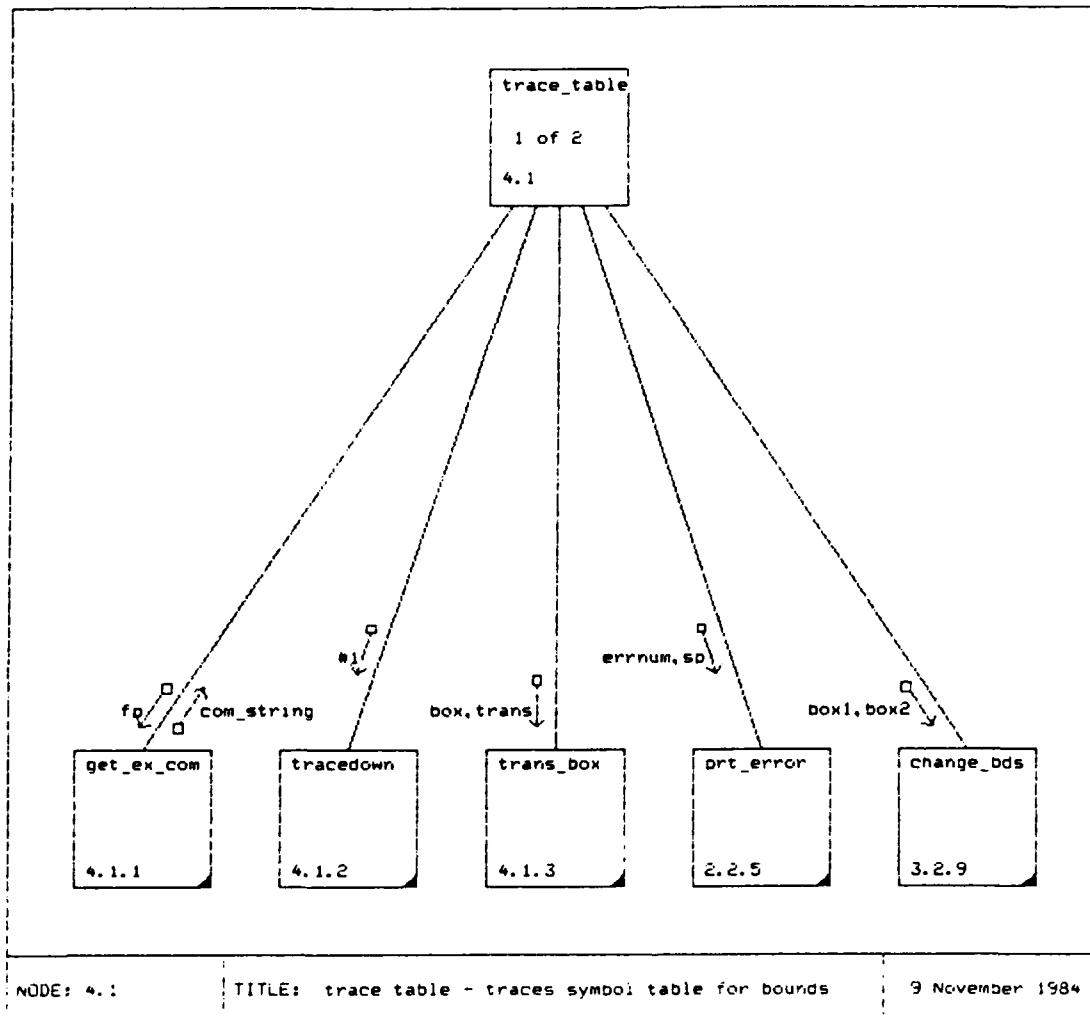
#1: format, digits, float, sp



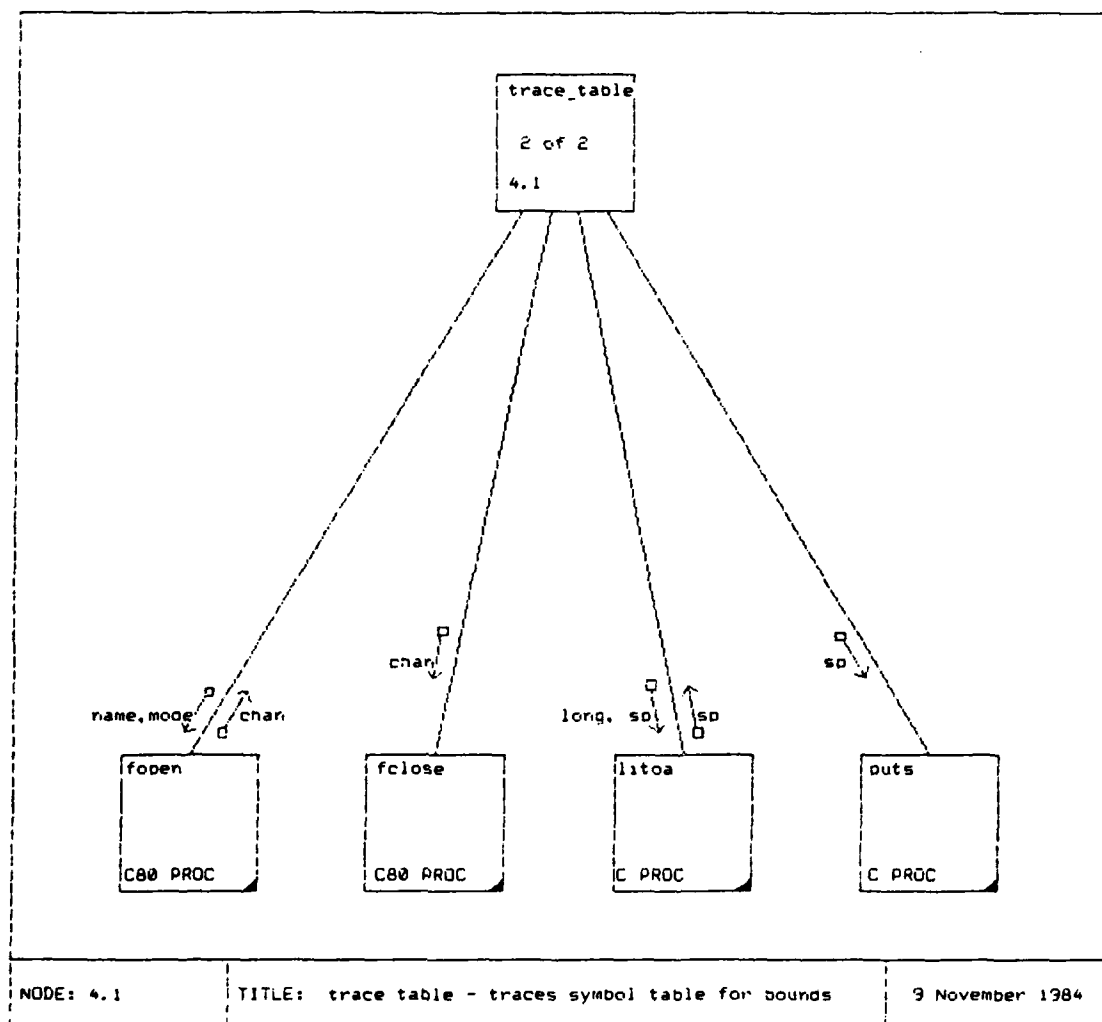


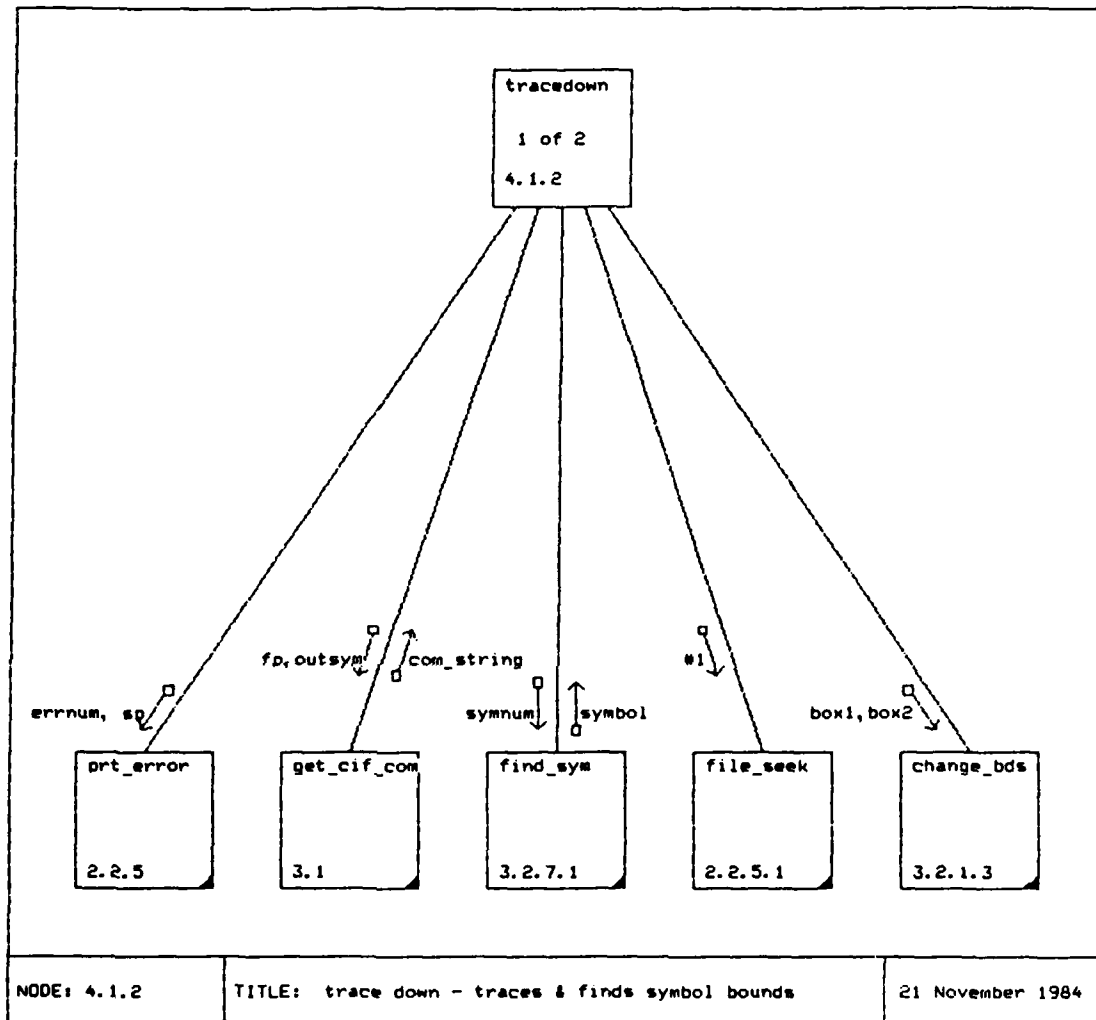




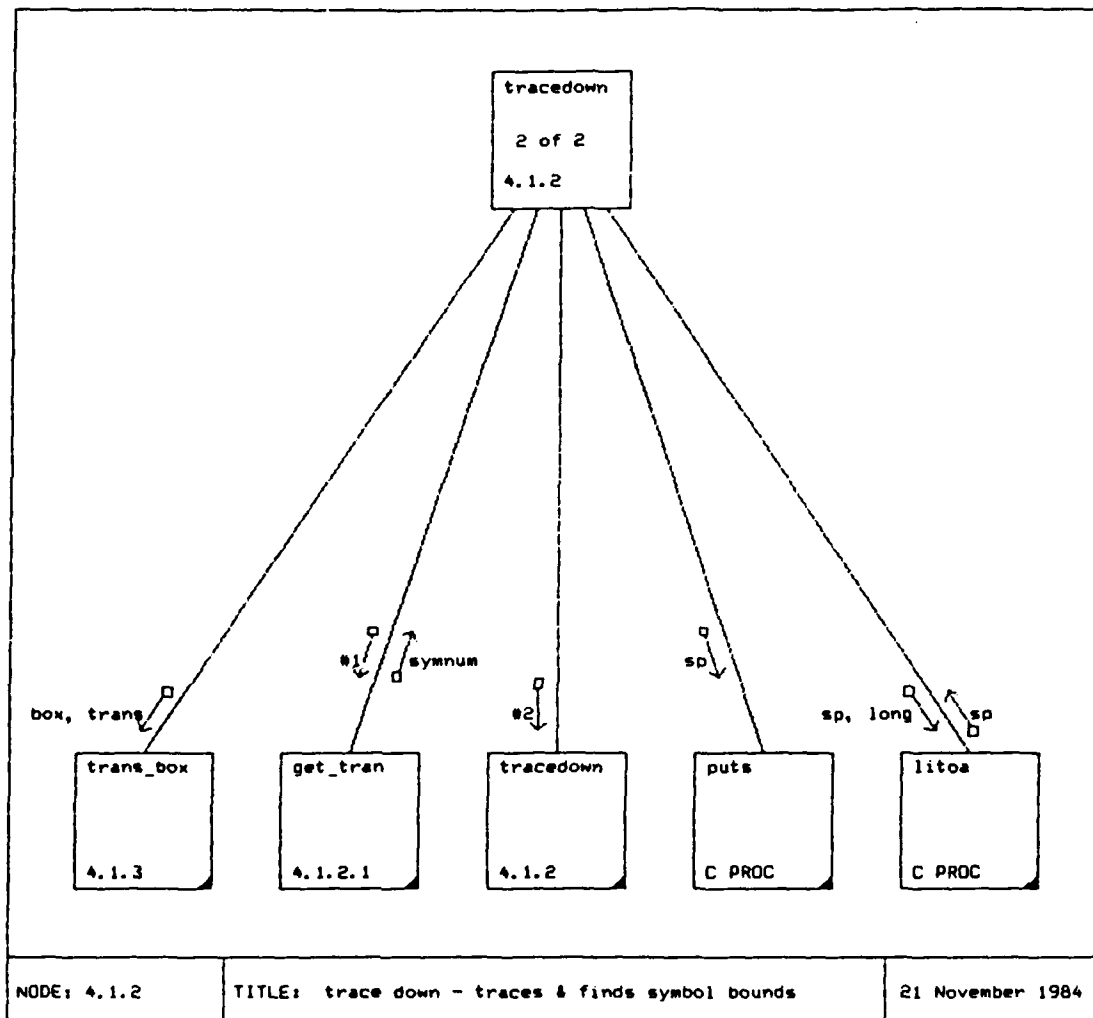


#1: com_string, scale, box

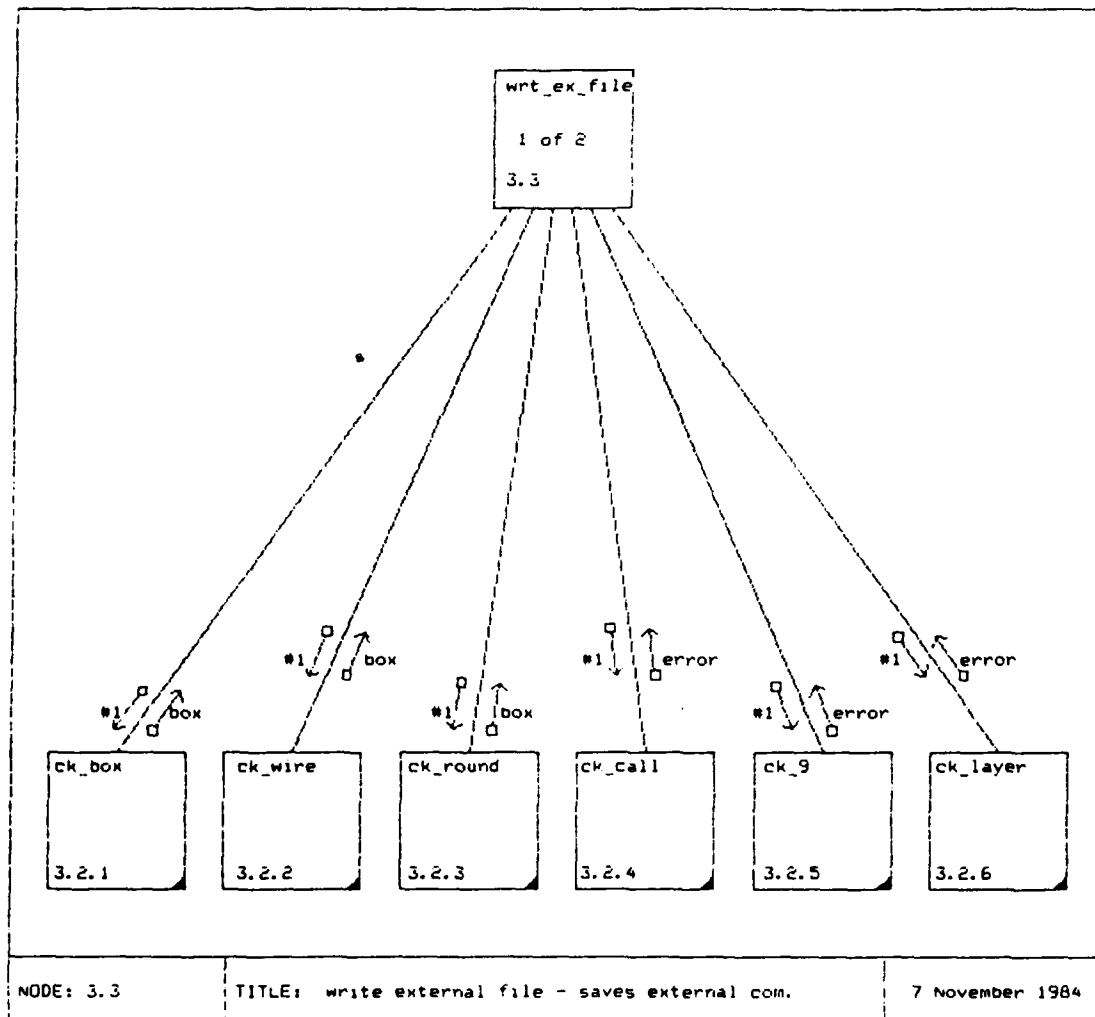




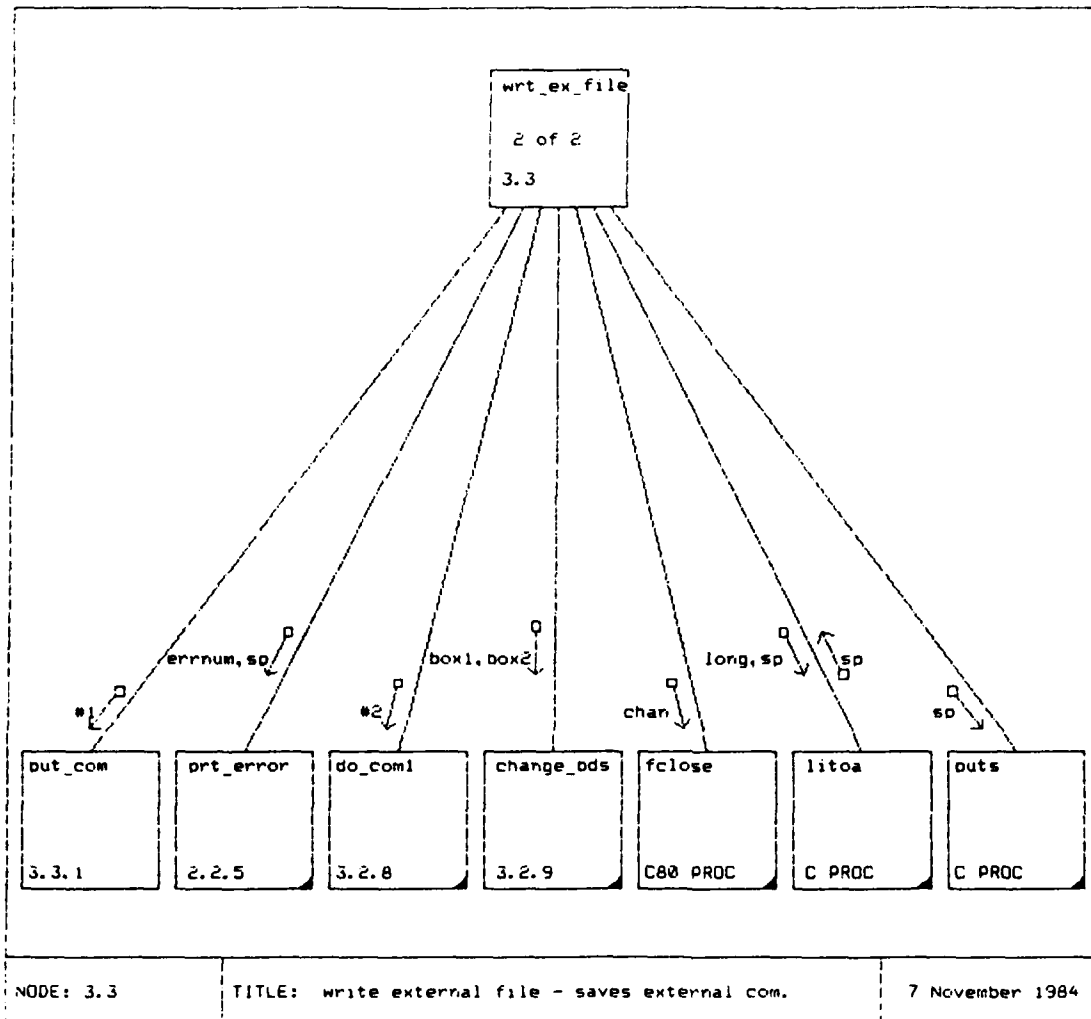
#1: fp, offset, record



#1: sp, scale, transl, trans2
 #2: com_string, scale, box



#1: com_string



#1: fd, com_string
#2: com_string

APPENDIX D. Data Dictionary

Appendix D contains the Data Dictionary for mcifplot. It includes four sections for the four different types of data items: constants, files, structures, and variables. Each section is in alphabetical order and multiple items are listed on each page to conserve space.

CONSTANTS

ANLENGTH
LACK
ADOPTMAX
LOSEROOM
NOTIFY
ONVERT
OF
(NAME
ALSE
EETMICRON
FINITY
AYERMAX
NOTIFY
AXCOMLEN
ESFILE
INLINE
INFILL
ESTLIMIT
EWLINE
LL
)
JLL
IGEMAX
IGERIM
IGEWIDTH
OTCHARS
TCURDISK
VEZONE
LECTDISK
UE
ERSION
ITE
OTIFY
S

AD-A151 715 A MICROCOMPUTER-BASED PROGRAM FOR PRINTING CHECK PLOTS 4/4
OF INTEGRATED CIRCUIT (U) AIR FORCE INST OF TECH

444

A MICROCOMPUTER-BASED PROGRAM FOR PRINTING CIRC. OF INTEGRATED CIRC. (U) AIR FORCE INST OF TECH

WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. K S HORTON

UNCLASSIFIED DEC 84 AFIT/GE/ENG/84D-35 F/G 9/2 NL

DEC 84 AFIT/GE/ENG/84D-35

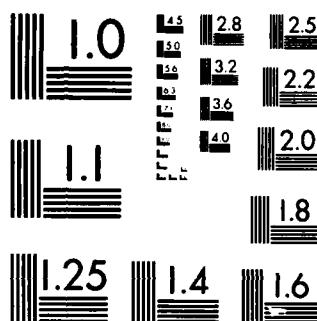
F/G 9/2

NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

NAME: BANLENGTH

TYPE: CONSTANT

DATE: 25 June 1984

DESCRIPTION: maximum length of optional banner = 80

CR = 1

LF = 1

NULL = 1

83 total

this is used as the length of the array for
storing it

DATA CHARACTERISTICS: integer

VALUES: 83

NAME: BLACK

TYPE: CONSTANT

DATE: 5 SEP 84

DESCRIPTION: PLOT.COM color for Black

DATA CHARACTERISTICS: integer

VALUES: 127

NAME: CADOPTMAX

TYPE: CONSTANT

DATE: 2 July 1984

DESCRIPTION: Constant which specifies the maximum number of
options that can be obtained from the CADRC
File.

DATA CHARACTERISTICS: integer

VALUES: 30

NAME: CIF BUFFER

TYPE: CONSTANT

DATE: 31 May 1984

DESCRIPTION: Length of buffer for reading in CIF file

DATA CHARACTERISTICS: integer

VALUES: 1024

NAME: CLOSEROOM

TYPE: CONSTANT

DATE: 5 SEP 84

DESCRIPTION: The number of characters required to close an
output vector file. This space must be
subtracted from the available space to write
other PLOT.COM commands.

DATA CHARACTERISTICS: long

VALUES: 1 'O' + 1 'Q' + 3 safety zone = 5

NAME: CNOTIFY
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: Call notify - the number of times to write a status message to the standard out when finding plot bounds. A message is written every CNOTIFY calls.
DATA CHARACTERISTICS: long
VALUES: 10

NAME: CONVERT
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: Used to convert the number returned by get_space to characters. Thus the total number of characters that can be written to a new file on disk can be determined.
DATA CHARACTERISTICS: long
VALUES: 1024 (number of bytes in one 1 K)

NAME: EOF
TYPE: CONSTANT
DATE: 4 July 1984
DESCRIPTION: Flag which signals end of file
DATA CHARACTERISTICS: integer
VALUES: -1

NAME: EXNAME
TYPE: CONSTANT
DATE: 1 NOV 84
DESCRIPTION: The name of the EXECUTABLE file on disk.
DATA CHARACTERISTICS: char array (string)
VALUES: EXCOM.TMP

NAME: FALSE
TYPE: CONSTANT
DATE: 22 May 1984
DESCRIPTION: Boolean value for false.
DATA CHARACTERISTICS: integer
VALUES: 0

NAME: FEETMICRON

TYPE: CONSTANT

DATE: 5 SEP 84

DESCRIPTION: The number of feet in one micron

DATA CHARACTERISTICS: float (in exponential form)

VALUES:

2.54 mils	*	1 inch	*	1 foot	
-----		-----		-----	= 2.1167e-4
1 micron		1000 mils		12 inches	

NAME: INFINITY

TYPE: CONSTANT

DATE: 30 May 1984

DESCRIPTION: Maximum legal CIF number (see Hon).

DATA CHARACTERISTICS: long integer

VALUES: 16777215

NAME: LAYERMAX

TYPE: CONSTANT

DATE: 5 SEP 84

DESCRIPTION: Maximum number of different layers that can be specified.

DATA CHARACTERISTICS: integer

VALUES: 15

NAME: LNOTIFY

TYPE: CONSTANT

DATE: 5 SEP 84

DESCRIPTION: Line notify - specifies how often to display a status message on the standard out when parsing the CIF file. A status message is written every LNOTIFY lines parsed.

DATA CHARACTERISTICS: long

VALUES: 100

NAME: MAXCOMLEN

TYPE: CONSTANT

DATE: 4 July 1984

DESCRIPTION: The maximum number of characters allowed in one command. Specified to limit the size of buffer required.

DATA CHARACTERISTICS: integer

VALUES: 161 (two 80 char lines)

NAME: MESFILE
TYPE: CONSTANT
DATE: 1 NOV 84
DESCRIPTION: The name of the MESSAGE file on disk.
DATA CHARACTERISTICS: char array (string)
VALUES: MESSAGE.BIN

NAME: MINLINE
TYPE: CONSTANT
DATE: 1 NOV 84
DESCRIPTION: The minimum length of a line to be written to the output PLOT file. Lines in geometric elements which are smaller than this are not written.
DATA CHARACTERISTICS: float
VALUES: .001 (1.0 / 1000)
The printer would require a resolution greater than 250 dpi for a line which is smaller than .001 to require more than one dot. As a comparison, the Versetec printer has a resolution of 200 dpi.

NAME: MINFILL
TYPE: CONSTANT
DATE: 1 NOV 84
DESCRIPTION: The minimum width of a FILL command to be written to the output PLOT file. FILLS which are smaller than this are not written.
DATA CHARACTERISTICS: float
VALUES: .002 (2 * MINLINE)
FILL's require at least a 2 dot width to be distinguished from a line.

NAME: NESTLIMIT
TYPE: CONSTANT
DATE: 7 SEP 84
DESCRIPTION: The maximum depth that symbol nesting will be traced. This is the default value for depth.
DATA CHARACTERISTICS: integer
VALUES: 20

NAME: NEWLINE
TYPE: CONSTANT
DATE: 30 June 1984
DESCRIPTION: Carriage return
DATA CHARACTERISTICS: char
VALUES: '\n'

NAME: NILL
TYPE: CONSTANT
DATE: 4 July 1984
DESCRIPTION: Empty pointer value, signals pointer to
nothing.
DATA CHARACTERISTICS: integer
VALUES: 0

NAME: NO
TYPE: CONSTANT
DATE: 4 July 1984
DESCRIPTION: Used to signify false for a character variable
(like the the flags in the symbol table).
DATA CHARACTERISTICS: character
VALUES: '0'

NAME: NULL
TYPE: CONSTANT
DATE: 23 June 1984
DESCRIPTION: character to signal end of string
DATA CHARACTERISTICS: character
VALUES: '\0'

NAME: PAGEMAX
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: The maximum number of pages allowed for output.
Prevents the start of a plot that could not
possibly fit on disk (or would take several
hours to plot).
DATA CHARACTERISTICS: integer
VALUES: 20

NAME: PAGERIM
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: The percent of white space to leave around the
outside of a plot specified with the makepage
option.
DATA CHARACTERISTICS: int
VALUES: 10

NAME: PAGEWIDTH
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: Width of page in inches - used to determine the
 inches per micron value of plot.
DATA CHARACTERISTICS: float
VALUES: 8.0

NAME: PLOTCHARS
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: The number of ASCII characters that can be
 printed across one page. Used to clip text to
 the edge of the page. Set to a minimum possible
 PLOT.COM installation value to insure
 compatibility with all versions.
DATA CHARACTERISTICS: long
VALUES: 80L

NAME: RETCURDISK
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: The number of the CPM BDOS call that returns
 the number of the current disk.
DATA CHARACTERISTICS: int
VALUES: 25

NAME: SAVEZONE
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: The number of bytes of memory used to separate
 the top of the Symbol Table from the stack as
 it grows downward.
DATA CHARACTERISTICS: int
VALUES: 200

NAME: SELECTDISK
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: The number of the CPM BDOS call that selects
 the input disk number.
DATA CHARACTERISTICS: int
VALUES: 14

NAME: TRUE
TYPE: CONSTANT
DATE: 22 May 1984
DESCRIPTION: Boolean value for true
DATA CHARACTERISTICS: integer
VALUES: 1

NAME: WHITE
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: PLOT.COM color for White
DATA CHARACTERISTICS: integer
VALUES: 0

NAME: VERSION
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: A character which represents the current
version of mcifplot.
DATA CHARACTERISTICS: array of characters (string)
VALUES: Version 1.0\t15 November 1984

NAME: WNOTIFY
TYPE: CONSTANT
DATE: 5 SEP 84
DESCRIPTION: Write notify - the number of times to display a
status message on the standard out when writing
the PLOT file. A message is written every
WNOTIFY characters.
DATA CHARACTERISTICS: long
VALUES: 2048 (2 K)

NAME: YES
TYPE: CONSTANT
DATE: 4 July 1984
DESCRIPTION: Used to signify true for a character variable
(like the the flags in the symbol table).
DATA CHARACTERISTICS: character
VALUES: '1'

STRUCTURES

box
clirec
file
point
symbol

NAME: box

TYPE: STRUCTURE

DATE: 12 JUL 1984

DESCRIPTION: Used to pass and store the xmin,ymin,xmax,ymax of a box. For passing bounds of a particular box or bounding box of a symbol.

COMPOSITION:

```
struct box {  
    long    xmin;  
    long    xmax;  
    long    ymin;  
    long    ymax;  
};
```

NAME: cliprec

TYPE: STRUCTURE

DATE: 5 NOV 1984

DESCRIPTION: Used to store "boolean" values which record the status of a clipped box. If the box has been clipped on the top, bottom, left, or right, then the value is set to TRUE, else the value is set to FALSE. A box outline is drawn only on the unclipped side of the box.

COMPOSITION:

```
struct    cliprec {  
    int    top;  
    int    bottom;  
    int    left;  
    int    right;  
};
```

NAME: file
TYPE: STRUCTURE
DATE: 30 June 1984
DESCRIPTION: for control of file I/O information
COMPOSITION:

name - CPM file name storage
mode - mode of file I/O = r / w / u / rb / rw / ru
channel - C80 channel number for file when open
next_char - pointer to next buffer character
buff_start - pointer to buffer start in memory
buff_size - size of file I/O buffer

```
struct file {  
    char    name[15];  
    char    mode[2];  
    int     channel;  
    char    *next_char;  
    char    *buff_start;  
    int     buff_size;  
};
```

NAME: point
TYPE: STRUCTURE
DATE: 12 JUL 1984
DESCRIPTION: Used to pass x,y coordinate of a point.
COMPOSITION:

```
struct point {  
    long    x;  
    long    y;  
};
```

NAME: symbol
TYPE: STRUCTURE
DATE: 5 SEP 84
DESCRIPTION: One symbol entry is created for each symbol in the CIF file and added to the linked list which forms the symbol table. A few words of justification on the size and composition of the symbol structure: Some data elements of course had to be at least a certain size like the offset and record (the types of these are very! unclear in the C/80 documentation - I've taken a good guess). Limiting symbol numbers to 0 - 65K should present no practical problems, but it certainly falls under the category of an "arbitrary" restriction. Originally, the scale was to be stored in the table to allow representation of the bounds as 4 short integers. It appears to be most economical to store the

scale and to store true (scale included) bounds values in 4 long integers. The bounds will be used to determine the window of the entire plot and to quickly decide if a symbol is in the output window. Originally, the bounds were to be integers which when multiplied together with the scale could represent any legal CIF number. However, if the bounds of a symbol happen to exceed the range of short integers, then the scale values and bounds values will have to be "rescaled" (one or the other increased so that the bounds can be represented by short integers). The code to accomplish the check and rescaling will probably exceed the savings in space gained by using short integer values. The average CIF file (from my experience) contains from 5-40 symbols. This means that the increase in size for the table will run from about 30-240 bytes. I think this choice is a reasonable compromise and will also allow for simplification of the code in other areas. The same justification can be applied to the decision to use a character for each flag vs. setting and checking bits in one character.

COMPOSITION:

number - number of symbol
 offset - offset of start of symbol from start of file
 record - record number of start of symbol,
 if > than 255 then offset relative to record
 scale - A/B value of symbol (division complete)
 sxmin - minimum x of symbol bounding box
 symin - minimum y of symbol bounding box
 sxmax - maximum x of symbol bounding box
 symax - maximum y of symbol bounding box
 calls - YES if symbol has calls to another symbol
 inuse - YES if symbol is being traced
 next - pointer to next symbol (no more = NULL)

```
struct symbol
{
    long    number;
    int     offset;
    int     record;
    float   scale;
    long    sxmin;
    long    symin;
    long    sxmax;
    long    symax;
    char    calls;
    char    inuse;
    struct symbol *next;
}
```

FILES

CADRC
CIF
EXECUTABLE
MESSAGE
PATTERN
PLOT

NAME: idenity
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: 3 X 3 matrix which is the identity matrix
(diagonal = 1.0)
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: array[9] of floats
VALUES: pointer to array of float with the identity
values
STORAGE TYPE: global to mcifplot

NAME: inmestab
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: "Boolean" variable, used to indicate whether or
not the table of message numbers and offsets
has been read from the MESSAGE file into
memory.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE or FALSE
STORAGE TYPE: global to main

NAME: last_symbol
TYPE: VARIABLE
DATE: 4 July 1984
DESCRIPTION: Pointer to the the last symbol added to the
symbol table. Used to allow linking of symbols
via next variable. Initially set to NIL.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: pointer to symbol structure
VALUES: NIL to pointer
STORAGE TYPE: global to parse_cif

NAME: exlayer
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: "boolean" variable which indicates that a layer
has been specified for the Executable commands.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE or FALSE
STORAGE TYPE: global to parse_cif

NAME: flag
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: A "boolean" variable used to indicate if a
condition is true or not.
PASSED FROM: isdigit / isspace / isalpha
PASSED TO: bld_st / ck_start / ck_9 / ck_box / ck_wire /
do_9xcom / ck_layer / do_box
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE or FALSE
STORAGE TYPE: passed

NAME: fp
TYPE: VARIABLE
DATE: 26 June 1984
DESCRIPTION: pointer to file structure which contains
information on the file
PASSED FROM: par_options / bld_st
PASSED TO: file_open / get_cif_com
COMPOSITION: N/A
DATA TYPE: pointer to a file structure
VALUES: pointer values
STORAGE TYPE: passed

NAME: error
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: A "boolean" variable used to signal success or failure when returned from many different processes.
 PASSED FROM: ck_call / get_integer / ck_9 / get_point / ck_start / box_bounds / ck_layer / ck_bounds / ck_range
 PASSED TO: bld_st / ck_start / box_bounds / ck_wire / ck_bounds / do_box / ck_box / ck_call / ck_round
 COMPOSITION: N/A
 DATA TYPE: integer
 VALUES: TRUS or FALSE
 STORAGE TYPE: passed

NAME: exbox
 TYPE: VARIABLE
 DATE: 12 July 1984
 DESCRIPTION: The bounds of all the executable commands in the CIF file. It also has an additional meaning - it is used to determine if the bld_file procedure should be executed or not. If it is equal to NILL, then build_file is not called. It is set from within parse_cif. The pointer is set to NILL if for example a symbol number cannot be read. This would cause misleading errors and information when the symbol table is traced and scaled.
 PASSED FROM: parse_cif / main
 PASSED TO: main / bld_file
 COMPOSITION: N/A
 DATA TYPE: struct box *exbox
 VALUES: NILL or pointer value
 STORAGE TYPE: passed and global to parse_cif

NAME: exec_file
 TYPE: VARIABLE
 DATE: 4 July 1984
 DESCRIPTION: Pointer to the EXECUTABLE file structure
 PASSED FROM: N/A
 PASSED TO: N/A
 COMPOSITION: N/A
 DATA TYPE: pointer to file (structure)
 VALUES: pointer
 STORAGE TYPE: global to main

NAME: end
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: "boolean" variable which indicates (if TRUE)
that the CIF End Command has been found.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE or FALSE
STORAGE TYPE: global to parse_cif

NAME: er_file
TYPE: VARIABLE
DATE: 30 June 1984
DESCRIPTION: Pointer to the MESSAGE file structure.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: pointer to file (structure)
VALUES: pointer
STORAGE TYPE: global to main

NAME: errnum
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: Number passed to prt_error which signals that a
particular error has occurred.
PASSED FROM: bld_file / bld_st / ck_9 / ck_box / ck_call
ck_layer / ck_wire / ck_round / do_grid
ck_start
PASSED TO: prt_error
COMPOSITION: N/A
DATA TYPE: integer
VALUES: legal error number
STORAGE TYPE: passed

NAME: err_flag
TYPE: VARIABLE
DATE: 4 July 1984
DESCRIPTION: flag which signifies that at least one runtime error
has been encountered. If TRUE then no output
PLOT file is written.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE or FALSE
STORAGE TYPE: global to main

NAME: com_string
 TYPE: VARIABLE
 DATE: 4 July 1984
 DESCRIPTION: com_string is a C string (terminated with NULL)
 which contains one CIF command like "DF;" in
 the form: X-----;\0
 PASSED FROM: get_cif_com / parse_cif / bld_st / ck_start
 ck_wire / ck_9 / do_9xcom / ck_box / do_box
 ck_call / ck_round
 PASSED TO: parse_cif / bld_st / wrt_ex_file / ck_box
 do_lcom / ck_wire / ck_start / ck_round
 ck_call / get_point / get_integer / ck_9
 ck_layer
 COMPOSITION: N/A
 DATA TYPE: *char
 VALUES: pointer
 STORAGE TYPE: passed

NAME: comstore
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: Global storage location for CIF commands and
 other strings which require a large character
 array.
 PASSED FROM: N/A
 PASSED TO: N/A
 COMPOSITION: N/A
 DATA TYPE: array of characters
 VALUES: comstore[MAXCOMLEN]
 STORAGE TYPE: global to main

NAME: count
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: Integer count returned from several different
 types of procedures.
 PASSED FROM: bld_file / set_page / bld_st / strlen
 PASSED TO: ble_file / do_9com / plot_page / sbrk
 COMPOSITION: N/A
 DATA TYPE: integer
 VALUES: integer
 STORAGE TYPE: passed

NAME: color
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: The PLOT.COM color which corresponds to a CIF
layer name.
PASSED FROM: do_box / do_wire / do_grid / do_round
PASSED TO: put_box / pc_color
COMPOSITION: N/A
DATA TYPE: integer
VALUES: 0 ~ 127
STORAGE TYPE: passed

NAME: com_offset
TYPE: VARIABLE
DATE: 7 SEP 84
DESCRIPTION: The number returned by C80 ftell procedure
which is the number of bytes to the current
read pointer of a file. Get_cif_com stores the
number here for every DS command found. Bld_st
uses this number to fill in the Symbol Table.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: integer
STORAGE TYPE: global to main

NAME: com_record
TYPE: VARIABLE
DATE: 7 SEP 84
DESCRIPTION: The number returned by C80 ftellr which is the
number of bytes to the current read pointer of
a file mod 256. Get_cif_com stores the number
here for every DS command found. Bld_st uses
this number to fill in the Symbol Table.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: integer
STORAGE TYPE: global to main

NAME: chan
TYPE: VARIABLE
DATE: 30 June 1984
DESCRIPTION: C/80 channel assigned to each file as it is
opened. All C/80 file I/O references this
channel number.
PASSED FROM: file_open / get_pattern / abort / bld_file
PASSED TO: C80 file I/O procedures and all PC Procedures
COMPOSITION: N/A
DATA TYPE: integer
VALUES: 0 - 6 for C/80
STORAGE TYPE: passed

NAME: charcount
TYPE: VARIABLE
DATE: 7 SEP 84
DESCRIPTION: Number of characters that can be written to a
new file on disk before the disk is full.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: long
VALUES: long
STORAGE TYPE: global to main

NAME: cif_file
TYPE: VARIABLE
DATE: 30 June 1984
DESCRIPTION: Pointer to the CIF input file structure.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: pointer to file (structure)
VALUES: pointer
STORAGE TYPE: global to main

NAME: cliprec
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: This is the cliprec structure being passed
between procedures.
PASSED FROM: do_9xcom / do_box / do_wire / do_round
PASSED TO: clip_box / put_box
COMPOSITION: see cliprec structure
DATA TYPE: cliprec structure
VALUES: pointer to cliprec structure
STORAGE TYPE: passed

NAME: box data
 TYPE: VARIABLE
 DATE: 30 July 1984
 DESCRIPTION: A collection of values which describe a
 geometric box in CIF terms.
 PASSED FROM: ck_box / do_box
 PASSED TO: box_bounds
 COMPOSITION:
 long length: length of box
 long width: width of box
 long cx: center of box x value
 long cy: center of box y value
 long dx: directional point x value
 long dy: directional point y value
 STORAGE TYPE: passed

NAME: build
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: "boolean" variable which signifies the status
 of the attempt to write the PLOT file. If build
 is TRUE, then the PLOT file can be built, else
 a runtime error has occurred, and the PLOT file
 will not be built.
 PASSED FROM: N/A
 PASSED TO: N/A
 COMPOSITION: N/A
 DATA TYPE: integer
 VALUES: TRUE or FALSE
 STORAGE TYPE: global to parse_cif

NAME: cadrc
 TYPE: VARIABLE
 DATE: 23 June 1984
 DESCRIPTION: signals to par_options that the options being
 parsed are from the CADRC file, necessary
 because their are only options (no filename) to
 parse in the CADRC file.
 PASSED FROM: int_command / get_cad_opt
 PASSED TO: par_options
 COMPOSITION: N/A
 DATA TYPE: integer
 VALUES: TRUE or FALSE
 STORAGE TYPE: passed

NAME: argc
TYPE: VARIABLE
DATE: 23 June 1984
DESCRIPTION: standard C parameter passed to main procedure,
 it is the number of words in the command line.
PASSED FROM: Operating system / int_command / get_cad_opt
PASSED TO: main / par_options
COMPOSITION: N/A
DATA TYPE: integer
VALUES: integer > 0
STORAGE TYPE: passed

NAME: argv
TYPE: VARIABLE
DATE: 23 June 1984
DESCRIPTION: standard C parameter passed to main procedure,
 it is an array of pointers which point to the
 strings in the command line.
PASSED FROM: Operating system / int_command / get_cad_opt
PASSED TO: main / par_options
COMPOSITION: N/A
DATA TYPE: array of pointers to characters
VALUES: pointer to characters, array size = argc
STORAGE TYPE: passed

NAME: box
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: This is a box structure which contains the
 same elements as in the definition for the box
 structure.
PASSED FROM: bld_file / ck_bounds / trace_table / clip_rec
 put_box / bld_st / trantoplot / change_bds
 trans_box
PASSED TO: bld_file / box_bounds / trace_table / ck_wire
 set_page / do_9xcom / ck_box / ck_wire
 ck_round / bld_st / do_box
COMPOSITION: N/A
DATA TYPE: box structure
VALUES: see box structure
STORAGE TYPE: passed

Control Variables (included at the end of APPENDIX D):

banner
bantext[BANLENGTH]
bbox
color[LAYERMAX]
comments
depth
extension
grid
gridspace
interactive
layer[LAYERMAX][5]
makepage
notify
outfile
outfname
outline
pattern
pointname
quiet
rotate
scale
scalenum
symbolname
text
transfer
trname
window
xmin
xmax
ymin
ymax

VARIABLES

argc	last_symbol
argv	line_number
box_data	mesfiletest
build	mode
cadrc	name
chan	outsym
charcount	plot_file
cif_file	point
cliprec	realxmax
color	rotmatrix
com_offset	sp
com_record	sploc
com_string	symnum
comstore	syn_ascale,syn_bscale
count	table_top
end	totbox
er_file	totcall
errnum	totccomment
err_flag	totcommand
error	totextension
exbox	totlayer
exec_file	totround
exlayer	totsymbol
flag	totwire
fp	trans
identity	upchar
inmestab	

NOT defined in the data dictionary are Structure Chart parameters which are identified as standard C variable names. These have the characteristics of the type name and are passed over and over between modules. They are:

float
char
long
int

TYPE: FILE

DATE: 30 May 1984

NAME: PLOT

DESCRIPTION: This file is the output file of mcifplot. It contains elementary plot commands in the format required for PLOT.COM. The file is named filename.VEC, where filename is the name of the input CIF file. Check plots are produced by using this file as input to PLOT.COM.

LOCATION: The same drive as the input CIF file.

DATA CHARACTERISTICS: The file is a mixture a text and binary data as required by PLOT.COM

TYPE: FILE

DATE: 30 May 1984

NAME: MESSAGE

DESCRIPTION: This file contains all messages which are required by mcifplot. These include all errors and warnings. The program code will only contain message numbers which reference a particular message in this file.

LOCATION: The same drive as for program execution.

DATA CHARACTERISTICS: This file is constructed using another program and it has the following format:

First Element in File->	Number of messages	integer
Table of Message Numbers and Offsets	-> Message Number	integer
of message from	Message Offset	integer
the start of file		
Message strings	-> First Message	char array
	...	ends with '\0'
	...	
	Last Message	

TYPE: FILE

DATE: 26 June 1984

NAME: PATTERN

DESCRIPTION: File which contains a new optional set of PLOT.COM colors and layer names to be used instead of the default layer names and colors in mcifplot. A maximum of 15 layers can be specified. CIF layer names and PLOT.COM colors must be legal for their respective environments, else fatal errors will be produced. Default PLOT.COM colors which closely correspond to the standard NMOS stipple patterns of cifplot will be built into mcifplot.

LOCATION: Always on drive specified in the command line.

DATA CHARACTERISTICS:

The pattern_file format is signified by the character "1" as the first character in the file on the first line. It is followed by a list of layer names and a corresponding PLOT.COM color with a decimal value from 0 to 127. One layer is specified per line and lines that begin with ';' are ignored. For example:

```
1
; example pattern file of format 1
NP 27
NC 13
NB 127
```

TYPE: FILE
DATE: 30 May 1984
NAME: CADRC
DESCRIPTION: The CADRC file contains additional options which are included in the with the command line options as input to VLSI design tools. The options for a particular program are on a single line which begins with the name of that program. These options are always executed. For example:

mcifplot -l bbox -p CMOSPAT.STI

LOCATION: Always on the same drive as the program executed.
DATA CHARACTERISTICS: text file

TYPE: FILE
DATE: 31 May 1984
NAME: CIF
DESCRIPTION: The CIF file is the main input to mcifplot. It contains the CIF commands that are read to produce the output PLOT file for producing check plots. The default extension is .CIF and the file should not have an extension of .VEC (this extension is used for the output PLOT file).

LOCATION: Default drive or drive specified in command line.
DATA CHARACTERISTICS: text file in CIF 2.0 syntax.

TYPE: FILE
DATE: 30 May 1984
NAME: EXECUTABLE
DESCRIPTION: This a temporary file which contains all CIF commands from the input CIF files that are not in contained in symbol definitions. It will always contain the End command and should contain at least one call if symbols are used.
LOCATION: Drive from which program executes.
DATA CHARACTERISTICS: text file in CIF 2.0 format.

NAME: line_number
TYPE: VARIABLE
DATE: 4 July 1984
DESCRIPTION: The number of Carriage Returns found by the
get_cif_com procedure. This is used by the
prt_error routine to print errors containing
line numbers.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: long integer
VALUES: long > 0
STORAGE TYPE: global to main

NAME: mesfiletest
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: "Boolean" variable which if set to TRUE if an
attempt has already been made to read in the
MESSAGE file table.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE of FALSE
STORAGE TYPE: global to main

NAME: mode
TYPE: VARIABLE
DATE: 26 June 1984
DESCRIPTION: The mode used to open a file which defines the
access as read, write or append.
PASSED FROM: par_options / open_pfile
PASSED TO: file_open
COMPOSITION: N/A
DATA TYPE: array of characters
VALUES: K&R standard C80 equivalent
read "r" "r"
write "w" "w"
append "a" "u" (update)
 --- "rb" (binary mode)
 --- "wb"
 --- "ub"
STORAGE TYPE: passed

NAME: name
TYPE: VARIABLE
DATE: 26 June 1984
DESCRIPTION: file name of file to be operated on. Must be a
 legal CPM file name
PASSED FROM: par_options / abort
PASSED TO: file_open / get_pattern / unlink / exec
COMPOSITION: N/A
DATA TYPE: array of characters terminated by NULL
VALUES: CPM file name
STORAGE TYPE: passed

NAME: outsym
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: A "boolean" variable used to indicate if a a
 symbol is being used (during parsing or
 tracing). TRUE indicates out of a symbol.
 get_cif_com will not execute ftell and ftellr
 if this is TRUE.
PASSED FROM: bld_st / parse_cif
PASSED TO: get_cif_com
COMPOSITION: N/A
DATA TYPE: integer
VALUES: TRUE or FALSE
STORAGE TYPE: passed

NAME: plot_file
TYPE: VARIABLE
DATE: 7 SEP 84
DESCRIPTION: Pointer to the file structure for the PLOT
 file.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: struct file *plot_file
VALUES: pointer
STORAGE TYPE: global to main

NAME: point
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: This is the point structure being passed
 between procedures.
 PASSED FROM: do_9xcom / ck_9 / ck_box / ck_round / do_box /
 ck_wire
 PASSED TO: get_point
 COMPOSITION: see point structure
 DATA TYPE: point structure
 VALUES: pointer to point structure
 STORAGE TYPE: passed

NAME: realxmax
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: This is the real maximum x bounds of the plot
 in CIF units. This is sometimes different from
 xmax of the current window.
 PASSED FROM: set_page / report_size
 PASSED TO: bld_file
 COMPOSITION: N/A
 DATA TYPE: long integer
 VALUES: long integer
 STORAGE TYPE: passed

NAME: rotmatrix
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: 3 X 3 matrix which contains values to translate
 2 dimensional coordinates in a 90 degrees
 rotation.
 PASSED FROM: N/A
 PASSED TO: N/A
 COMPOSITION: N/A
 DATA TYPE: array[9] of float
 VALUES:

0.0	1.0	0.0
-1.0	0.0	0.0
0.0	0.0	1.0

 STORAGE TYPE: gloabal to mcifplot

NAME: sp
TYPE: VARIABLE
DATE: 26 June 1984
DESCRIPTION: array of characters which forms a string and is
 terminated by NULL
PASSED FROM: generally used: par_options / abort / add_line
PASSED TO: generally used: get_layer, get_pattern, latoi, etc.
COMPOSITION: N/A
DATA TYPE: array of characters
VALUES: char
STORAGE TYPE: passed

NAME: sploc
TYPE: VARIABLE
DATE: 7 NOV 84
DESCRIPTION: The location of a string (array of characters).
 Allows called processes to modify the location
 of the string pointer.
PASSED FROM: ck_9 / ck_box / ck_round / do_box / ck_start /
 ck_call / ck_wire / do_9xcom / ck_layer
PASSED TO: get_point / get_integer / skip_sep / skip_blank
COMPOSITION: N/A
DATA TYPE: pointer to pointer to character
VALUES: pointer
STORAGE TYPE: passed

NAME: symnum
TYPE: VARIABLE
DATE: 5 SEP 84
DESCRIPTION: Variable whose value represents a symbol
 number. It contains the symbol number from the
 DS command. Used to modify the number in a
 symbol entry.
PASSED FROM: blt_st / ck_start
PASSED TO: ck_start / find_sym
COMPOSITION: N/A
DATA TYPE: long
VALUES: long > 0
STORAGE TYPE: passed

NAME: sym_ascale,sym_bscale
TYPE: VARIABLE
DATE: 5 SEP 84
DESCRIPTION: The two scaling ratios in a DS command. Found
by ck_start and used to determine the true
bounds of a symbol.
PASSED FROM: bld_st
PASSED TO: ck_start
COMPOSITION: N/A
DATA TYPE: long integer
VALUES: legal CIF-numbers > 0
STORAGE TYPE: passed

NAME: table_top
TYPE: VARIABLE
DATE: 4 July 1984
DESCRIPTION: Pointer to the top of the symbol table.
Initially set to NIL.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: pointer to symbol structure
VALUES: NIL to pointer
STORAGE TYPE: global to main

NAME: totbox, totcall, totccomment, totcommand, totextension,
totlayer, totround, totsymbal, totwire
TYPE: VARIABLE
DATE: 4 NOV 1984
DESCRIPTION: Storage locations for the counting of CIF
command totals. Only used if notify is TRUE.
PASSED FROM: N/A
PASSED TO: N/A
COMPOSITION: N/A
DATA TYPE: long
VALUES: long >= 0
STORAGE TYPE: global to main

NAME: trans
 TYPE: VARIABLE
 DATE: 7 NOV 84
 DESCRIPTION: Transformation which is passed between procedures. This is a 3 X 3 matrix (really an array of floats with a dimension of 9) used to transform CIF coordinates.
 PASSED FROM: do_box / do_9xcom
 PASSED TO: trans_box
 COMPOSITION: N/A
 DATA TYPE: pointer to array[9] of floats
 VALUES: pointer (no restrictions on floats)
 STORAGE TYPE: passed

NAME: upchar
 TYPE: VARIABLE
 DATE: 26 June 1984
 DESCRIPTION: upper case ASCII character
 PASSED FROM: toupper
 PASSED TO: par_options
 COMPOSITION: N/A
 DATA TYPE: char
 VALUES: upper case ASCII characters (A-Z)
 STORAGE TYPE: passed

NAME: Control Variables (related variables NOT defined separately)
 TYPE: VARIABLE
 DATE: 5 SEP 84
 DESCRIPTION: Variables which control the function and operation the program. They are modified by the command line and CADRC options.
 PASSED FROM: N/A
 PASSED TO: N/A
 DATA TYPE: see below
 VALUES: see below
 STORAGE TYPE: global to main

Name	Type	Default Val	Range
banner	int	FALSE	TRUE or FALSE
bantext	char[BL]	NULL	NULL to BANLENGTH (BL)
bbox	int	TRUE	TRUE or FALSE
color[0]	int	49	range of colors for PLOT
color[1]	int	50	range of colors for PLOT
color[2]	int	51	range of colors for PLOT
color[3]	int	52	range of colors for PLOT
color[4]	int	53	range of colors for PLOT
color[5]	int	54	range of colors for PLOT
color[6]	int	55	range of colors for PLOT
color[7]	int	56	range of colors for PLOT

Name	Type	Default Val	Range
color[8]	int	57	range of colors for PLOT
color[9]	int	58	range of colors for PLOT
color[10]	int	59	range of colors for PLOT
color[11]	int	60	range of colors for PLOT
color[12]	int	61	range of colors for PLOT
color[13]	int	62	range of colors for PLOT
color[14]	int	63	range of colors for PLOT
comments	int	FALSE	TRUE or FALSE
depth	long	NESTLIMIT	0 < long > INFINITY
extension	int	TRUE	TRUE or FALSE
grid	int	FALSE	TRUE or FALSE
gridspace	long	0	0 < long > INFINITY
interactive	int	TRUE	TRUE or FALSE
layer[0]	char[5]	"ND"	char[5] (4 chars + '\0')
layer[1]	char[5]	"NI"	char[5]
layer[2]	char[5]	"NP"	char[5]
layer[3]	char[5]	"NC"	char[5]
layer[4]	char[5]	"NM"	char[5]
layer[5]	char[5]	"NM"	char[5]
layer[6]	char[5]	"NG"	char[5]
layer[7]	char[5]	"CW"	char[5]
layer[8]	char[5]	"CD"	char[5]
layer[9]	char[5]	"CP"	char[5]
layer[10]	char[5]	"CS"	char[5]
layer[11]	char[5]	"CC"	char[5]
layer[12]	char[5]	"CM"	char[5]
layer[13]	char[5]	"CG"	char[5]
layer[14]	char[5]	"CE"	char[5]
makepage	int	FALSE	TRUE or FALSE
notify	int	TRUE	TRUE or FALSE
outfile	int	TRUE	TRUE or FALSE
outfname	char[15]	NULL	CPM file name
outline	int	TRUE	TRUE or FALSE
pattern	int	FALSE	TRUE or FALSE
patstring	char[tbd]	NULL	as required for pattern
pointname	int	TRUE	TRUE or FALSE
quiet	int	FALSE	TRUE or FALSE
rotate	int	FALSE	TRUE or FALSE
scale	int	FALSE	TRUE or FALSE
scalenum	float	0	float > 0
symbolname	int	TRUE	TRUE or FALSE
text	int	TRUE	TRUE or FALSE
window	int	FALSE	TRUE or FALSE
wxmin	long	INFINITY	long
wxmax	long	-INFINITY	long
wymin	long	INFINITY	long
wymax	long	-INFINITY	long

APPENDIX E: C / PC PROCEDURES

```

/*****
* DATE: 17 OCT 84
* VERSION: 1.0
*
* TITLE: newlib - New library of C functions
* OWNER: Kirk S. Horton
* OPERATING SYSTEM: CPM 2.2
* LANGUAGE: Software Toolworks C/80
* USE: linked with application programs
* CONTENTS:
*     get_space - gets free disk space in K bytes
*     latoi     - converts ascii string to long integer
*     litoa     - converts long integer to ascii string
*     puts      - puts string on standard output
* FUNCTION: General C functions, either new or modified C/80
*     functions.
*****/

```

```

/*****
* DATE: 18 SEP 84
* VERSION: 1.0
* NAME: get_space
* FUNCTION: Gets the number of 1K blocks left on disk.
* INPUTS: none
* OUTPUTS: blocks
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: reads disk directory info from disk
* FILES WRITTEN:
* PROCEDURES CALLED: uses SYSLIB Disk Utilities
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

get_space()

```

{
#asm
; THIS PROCEDURE USES THREE MODIFIED SYSLIB FUNCTIONS.
; THE BDOS VARIABLE IS CHANGED TO BDOSLO SO AS NOT TO CONFLICT
; WITH THE C/80 FUNCTION bdos AND IS ADDED TO SDIRBF.MAC.
; THE REGISTER SAVES ARE DELETED AT THE START AND END OF DPARAMS
; AND DFREE (NOT REQUIRED WITH C/80). REFERENCES TO CPM1.4 ARE
; DELETED FROM DPARAPMS AND THE FINAL ANSWER IS NOT MOVED TO DE
; BUT LEFT IN HL WHERE C/80 EXPECTS IT.
;
; SYSLIB Module Name:  SDIRn
; Author:   Richard Conn
; Part of SYSLIB3 SDIR Series
; SYSLIB Version Number:  3.0
; Module Version Number:  1.4
; Module Entry Points:
;     BLKSHF  BLKMSK  BLKMAX  DIRMAX  SELFLG  ORDER  DIRBUF

```

```

; Module External References:
;*
;* BUFFERS
;*
BDOSLO EQU      5
BLKSHF::
    DB          0          ; BLOCK SHIFT FACTOR
BLKMSK::
    DB          0          ; BLOCK MASK
BLKMAX::
    DW          0          ; MAX NUMBER OF BLOCKS
DIRMAX::
    DW          0          ; MAX NUMBER OF DIRECTORY ENTRIES
SELFLG::
    DB          0          ; FILE ATTRIBUTE FLAG
ORDER::
    DW          0          ; POINTER TO ORDER TABLE
DIRBUF::
    DW          0          ; POINTER TO DIRECTORY
;
; SYSLIB Module Name: SDIRO2
; Author: Richard Conn
; Part of SYSLIB3 SDIR Series
; SYSLIB Version Number: 3.0
; Module Version Number: 1.4
; Module Entry Points:
;     DPARAMS
; Module External References:
;
;*
;* THIS ROUTINE EXTRACTS DISK PARAMETER INFORMATION FROM THE DPB AND
;* STORES THIS INFORMATION IN:
;*     BLKSHF  <-- BLOCK SHIFT FACTOR (1 BYTE)
;*     BLKMSK  <-- BLOCK MASK (1 BYTE)
;*     EXTENT  <-- EXTENT MASK (1 BYTE) (NOT ANY MORE)
;*     BLKMAX  <-- MAX NUMBER OF BLOCKS ON DISK (2 BYTES)
;*     DIRMAX  <-- MAX NUMBER OF DIRECTORY ENTRIES (2 BYTES)
;*
DPARAMS::
;*
;* VERSION 2.x OR MP/M
;*
    MVI         C,31      ; 2.x OR MP/M...REQUEST DPB
    CALL        BDOSLO
    INX         H
    INX         H
    MOV         A,M        ; GET BLOCK SHIFT
    STA         BLKSHF     ; BLOCK SHIFT FACTOR
    INX         H          ; GET BLOCK MASK
    MOV         A,M
    STA         BLKMSK     ; BLOCK MASK
    INX         H
    INX         H
    MOV         E,M        ; GET MAX BLOCK NUMBER

```

```

    INX     H
    MOV     D,M
    XCHG
    INX     H      ; ADD 1 FOR MAX NUMBER OF BLOCKS
    SHLD    BLKMAX ; MAXIMUM NUMBER OF BLOCKS
    XCHG
    INX     H
    MOV     E,M      ; GET DIRECTORY SIZE
    INX     H
    MOV     D,M
    XCHG
    INX     H      ; ADD 1 FOR NUMBER OF ENTRIES
    SHLD    DIRMAX ; MAXIMUM NUMBER OF DIRECTORY ENTRIES
;
;
; * ALL PARAMETERS EXTRACTED
;
;
; SYSLIB Module Name:  SDIR03
; Author:  Richard Conn
; Part of SYSLIB3 SDIR Series
; SYSLIB Version Number:  3.0
; Module Version Number:  1.4
; Module Entry Points:
;     DFREE
; Module External References:
;
;
; *
; * COMPUTE AMOUNT OF FREE SPACE LEFT ON DISK
; * ON EXIT, HL=AMOUNT OF FREE SPACE ON DISK IN K
; * THE DPARAMS ROUTINE MUST BE CALLED BEFORE THIS ROUTINE IS USED
; *
DFREE::
    MVI     C,27      ; GET ADDRESS OF ALLOCATION VECTOR
    CALL    BDOSLO
    XCHG
    LHL     BLKMAX     ; GET LENGTH OF ALLOCATION VECTOR
    LXI     B,0        ; INIT BLOCK COUNT TO 0
;
;
; * BC IS ACCUMULATOR FOR SPACE
; *
FREE1:
    PUSH    D          ; SAVE ALLOC ADDRESS
    LDAX    D          ; GET BIT PATTERN OF ALLOCATION BYTE
    MVI     E,8        ; SET TO PROCESS 8 BLOCKS
FREE2:
    RAL              ; ROTATE ALLOCATED BLOCK BIT INTO CARRY FLAG
    JC      FREE3      ; IF SET (BIT=1), BLOCK IS ALLOCATED
    INX     B          ; IF NOT SET, BLOCK IS NOT ALLOCATED, SO INCREMENT
                    ; FREE BLOCK COUNT
FREE3:
    MOV     D,A        ; SAVE REMAINING ALLOCATION BITS IN D

```



```

DCX      H          ; COUNT DOWN NUMBER OF BLOCKS ON DISK
MOV      A,L
ORA      H
JZ       FREE4      ; DONE IF NO MORE BLOCKS LEFT
MOV      A,D        ; A=CURRENT ALLOCATION BIT PATTERN
DCR      E          ; HAVE ALL 8 BITS BEEN EXAMINED?
JNZ      FREE2      ; CONTINUE IF NOT
POP      D          ; GET POINTER TO ALLOCATION VECTOR
INX      D          ; POINT TO NEXT ALLOCATION BYTE
JMP      FREE1      ; CONTINUE BY PROCESSING NEXT ALLOCATION BYTE

```

```

;*
;* BC = TOTAL AMOUNT OF FREE SPACE IN TERMS OF BLOCKS
;*

```

```

FREE4:

```

```

POP      D          ; CLEAR DE FROM STACK
MOV      L,C        ; HL=BC=NUMBER OF FREE BLOCKS
MOV      H,B
LDA      BLKSHF     ; GET BLOCK SHIFT FACTOR
SUI      3          ; CONVERT NUMBER OF BLOCKS TO K
JZ       FREE6      ; DONE IF SINGLE DENSITY (1K PER BLOCK)

```

```

;*
;* WE ARE AT A MORE ADVANCED DENSITY LEVEL; MULTIPLY THE NUMBER OF BLOCKS
;* BY THE SIZE OF A BLOCK IN K
;*

```

```

FREE5:

```

```

DAD      H          ; 2, 4, 8, 16, ETC K/BLK, SO BLOCK SHIFT FACTOR
DCR      A          ; IS A POWER-OF-TWO MULTIPLE
JNZ      FREE5

```

```

;*
;* AT THIS POINT, HL=AMOUNT OF FREE SPACE ON DISK IN K
;*

```

```

FREE6:

```

```

NOP

```

```

#endasm
)

```

```

/*****
* DATE: 26 JUL 84
* VERSION: 1.0
* NAME: latoi
* FUNCTION: Converts ascii string to long integer. Returns long.
* Modified C/80 function.
* INPUTS: *char
* OUTPUTS: long integer
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: none
* CALLING PROCEDURES: as required
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/
long latoi(s)
char *s;
{
    static int    sign;
    static long   n;

    sign = 1;
    n = 0;
    switch (*s)
    {
        case '-': sign = -1;
        case '+': ++s;
    }
    while (*s >= '0' && *s <= '9') n = (long) 10 * n + *s++ - '0';
    return( (long) sign * n);
}

/*****
* DATE: 17 OCT 84
* VERSION: 1.0
* NAME: litoa
* FUNCTION: Converts long integer to ascii string. Returns the
* the ascii string. The only change made to the corrected
* C/80 function itoa is the change of the input type and
* internal integers to long integer.
* INPUTS: long integer, string pointer
* OUTPUTS: ascii string (in location of string pointer)
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: none
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

```

char *ltoa(n, s)
char s[];
long n;
{
    static int c;
    static long k;
    static char *p, *q;

    if ((k = n) < 0L) n = -n;
    q = p = s;
    do {
        *p++ = n % 10L + '0';
    } while ((n /= 10L) > 0L);
    if (k < 0L) *p++ = '-';
    *p = 0;
    while (q < --p) {
        c = *q; *q++ = *p; *p = c; }
    return (s);
}

```

```

/*****
* DATE: 1 OCT 84
* VERSION: 1.0
* NAME: puts
* FUNCTION: Writes string to standard output. Used putchar
*           rather than the smaller (& faster?) putc to allow
*           redirected output. From K&R.
* INPUTS: *char (string terminated by '\0')
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: none
* PROCEDURES CALLED: putchar - outputs one char to standard out.
* CALLING PROCEDURES: as required
* AUTHOR: Kirk S. Horton
* HISTORY: N/A
*****/

```

```

puts(s)
char *s;
{
    while (*s) putchar(*s++);
}

```

```

/*****
* DATE: 30 SEP 84
* VERSION: 1.0
*
* TITLE: pc_lib.c
* OWNER: Kirk S. Horton
* OPERATING SYSTEM: CPM 2.2
* LANGUAGE: Software Toolworks C/80
* USE: Compile and assemble to form relocatable module. Can be
*      linked with applications when plot commands are required
* CONTENTS:
*      all pc_XXXX functions write PLOT.33 commands to the file
*      opened for binary writing on channel chan. See PLOT.##
*      documentation for additional info on the format of PLOT
*      commands. Each plot command calls test_space to insure
*      there is room for the command before writing, and puts to
*      write the required characters to the file. Test_space
*      should gracefully close the output VECTOR file if enough
*      space is not available.
*      IMPORTANT: test_space must be defined external when
*      using these procedures

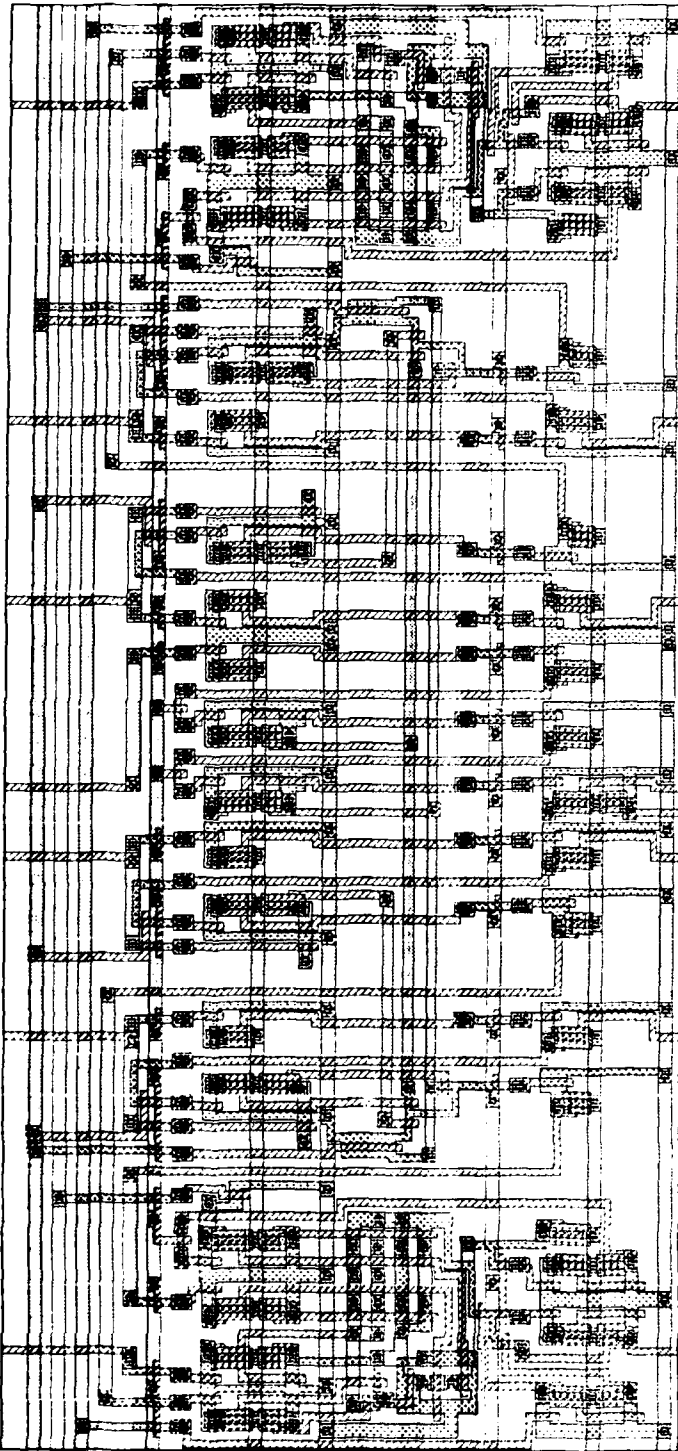
pc_color(color,chan) - writes Color command to change color
                      of plot
pc_draw(x1,y1,x2,y2,chan) - writes Draw command to draw line
                          from x1,y1 to x2,y2.
pc_erase(chan) - writes Erase command to erase memory to most
               recently defined color.
pc_fill(x1,y1,x2,y2,yf,chan) - writes Fill command to fill
                              from the line defined by x1,y1 and x2,y2 down to the
                              horizontal line defined by yf with the most recently
                              defined color.
pc_increment(x1,y1,chan) - writes Increment command to draw
                          line from the most recent point plotted to x1,y1.
pc_output(chan) - writes Output command to force output of plot
pc_point(x1,y1,chan) - writes Point command to draw point at
                      x1,y1.
pc_quit(chan) - writes Quit command to stop execution of
               PLOT33.COM.
pc_string(x1,y1,string,chan) - writes String command to
                              write string at x1,y1 on plot.
pc_text(string,chan) - writes Text command to immediately
                     output string to the printer.
pc_upload(n,pointer,chan) - writes Upload command to change
                           color values as defined by n and the values in the
                           array pointed to by pointer.

* Two pc functions also call the standard C functions strlen.
*
* FUNCTION: Writes plot commands to VECTOR file.
*****/
/*                                  DEFINES                                  */

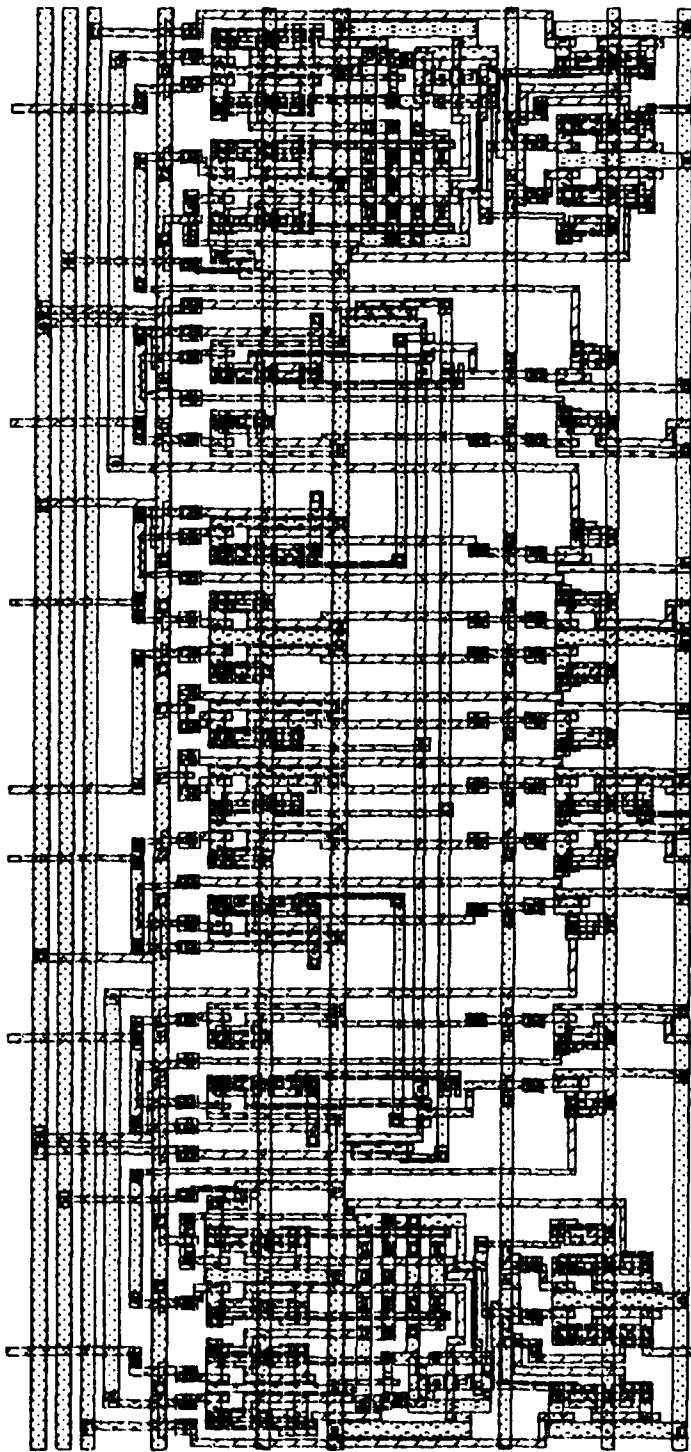
#define PLOTMAX          32767

```

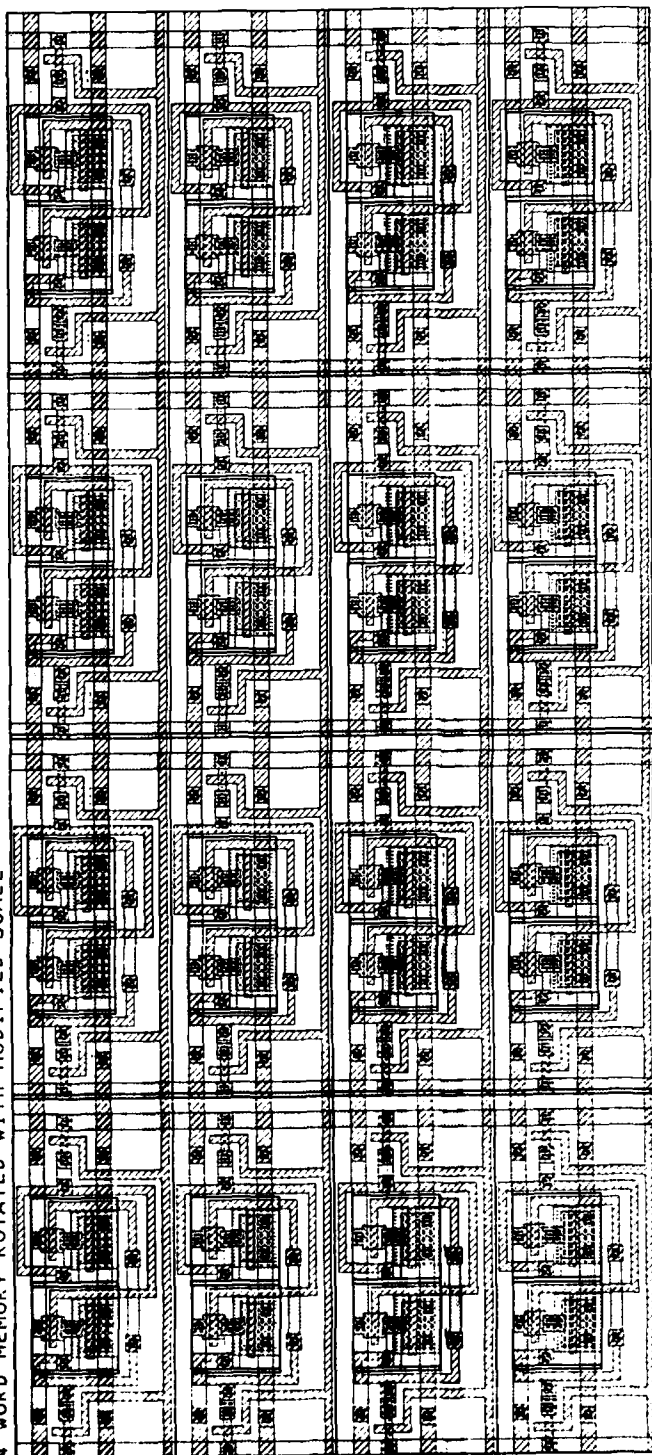
:Tue Nov 13 13:27:32 1984
cifplot* Window: 0 400 0 836 --- Scale: 1 micron is 0.956938 inches (24306x)
DBLMULT.CIF WITH MODIFIED SCALE



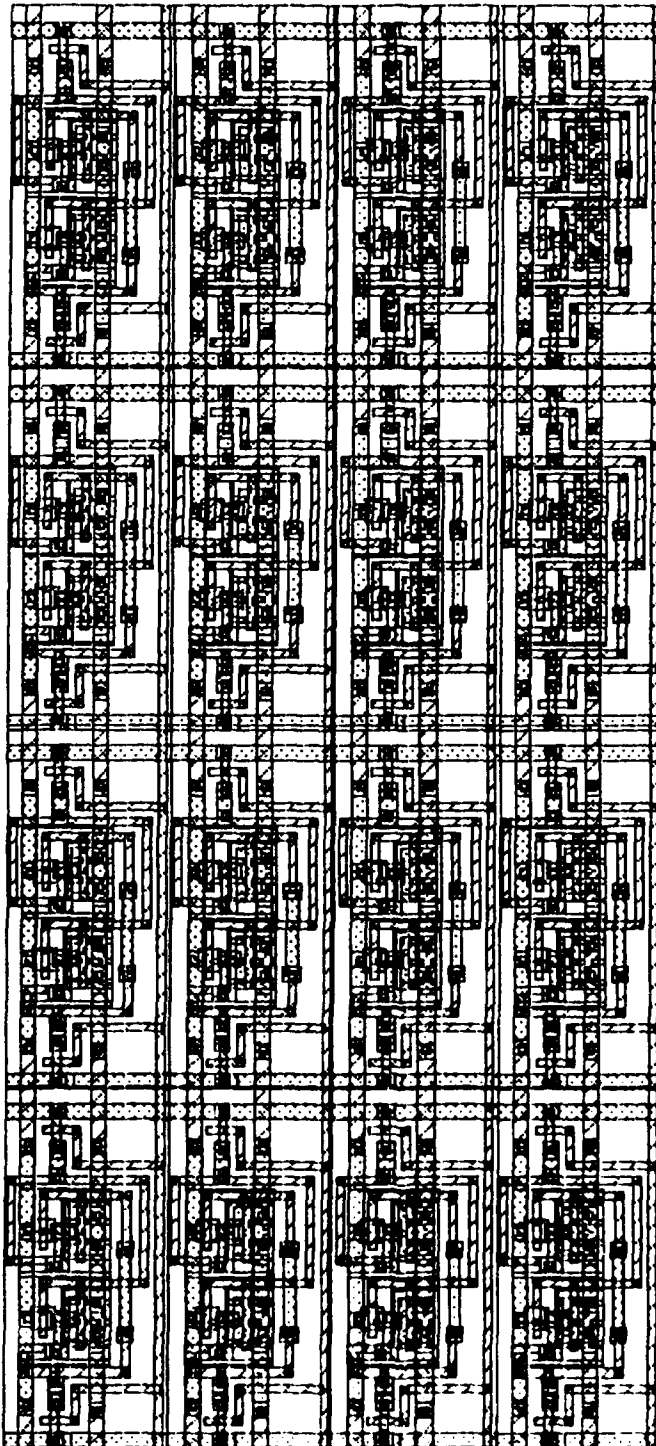
mcifplot* of DELMULT.CIF
Window: 0 400 0 836
Scale: 1 micron is .9569379 inches



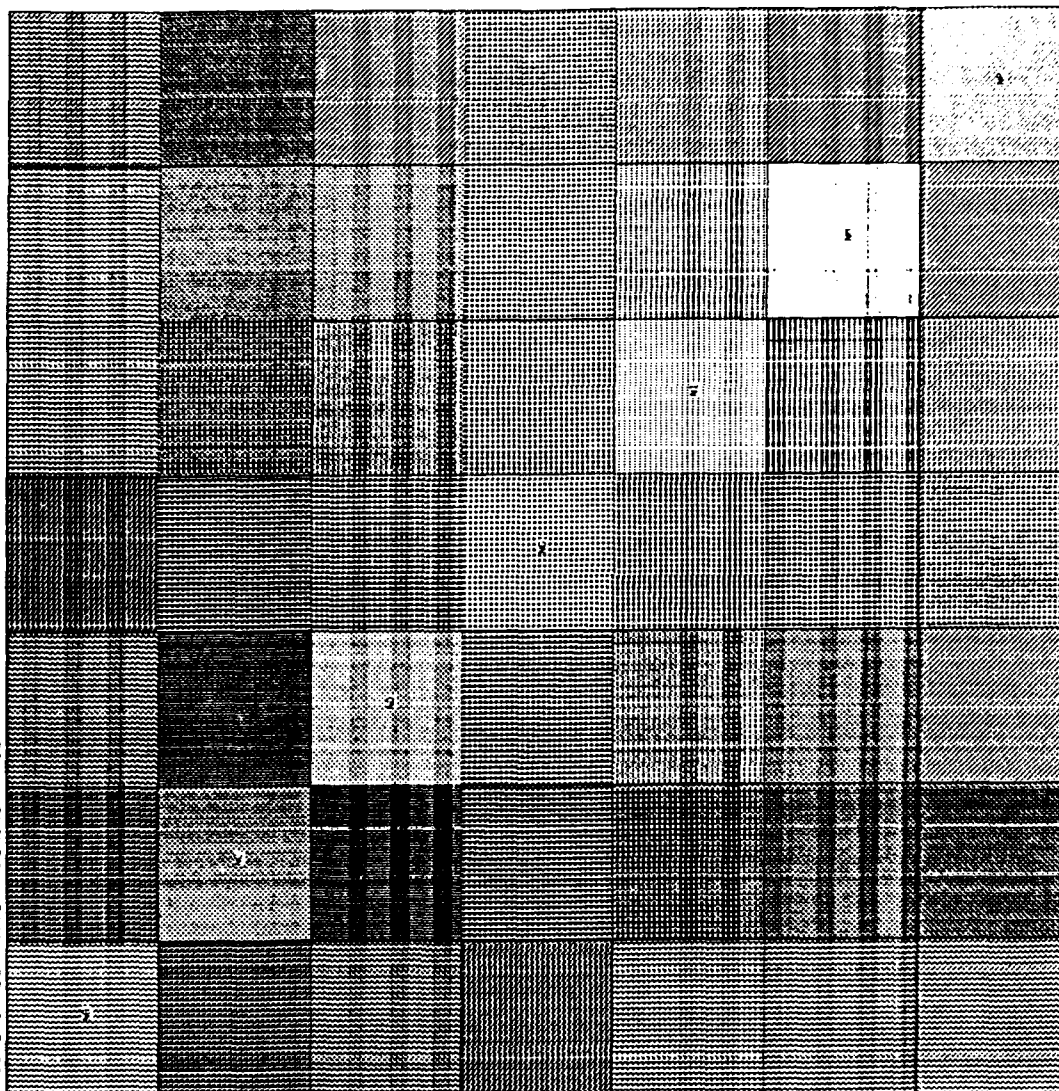
:Tue Nov 13 13:28:29 1984
 c:\plot* Window: -43000 @ 0 94250 --- Scale: 1 micron is 0.0084881 inches (216x)
 4 WORD MEMORY ROTATED WITH MODIFIED SCALE



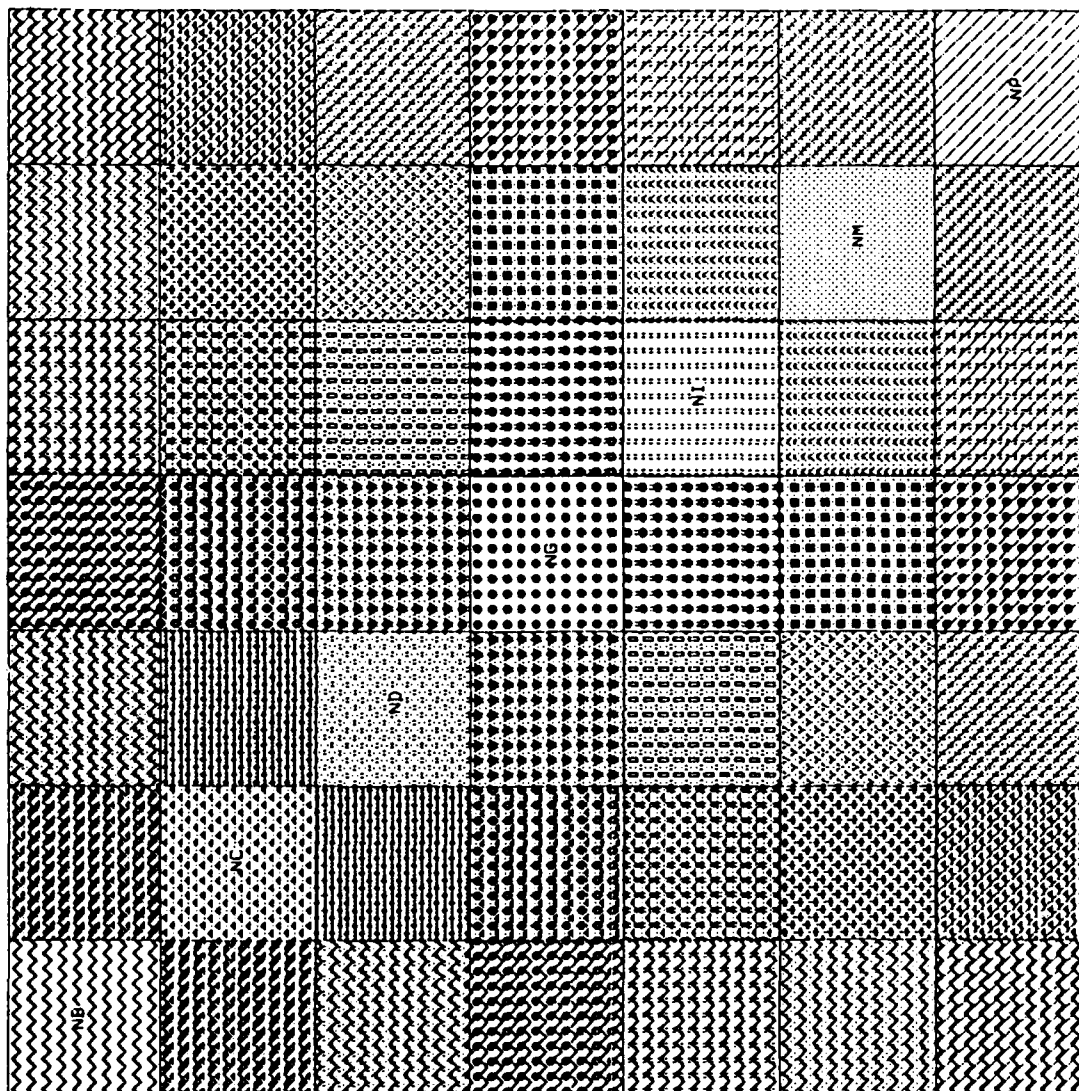
mcifplot* of 4WORDMEM.CIF
Window: -43000 0 0 94250
Scale: 1 micron is .0084881 inches
4 WORD MEMORY ROTATED



Tue Nov 13 13:26:25 1984
cifplot - Window: 0 7000 0 7000 --- Scale: 1 micron is 0.114286 inches (2903x)
NMOS STIPPLE PATTERN DEMONSTRATION



mcifplot of NMOS.CIF
 Window: 0 7000 0 7000
 Scale: 1 micron is .1142857 inches
 NMOS STIPPLE PATTERN DEMONSTRATION



itoa(i,s) - integer to ASCII convert
 converts the integer i into the ASCII string in the
 char array s[7]; returns s.

putc(c,chan) - put character to file
 writes the character c on the file or device opened for
 writing or update on channel chan.

putchar(c) - put character
 writes the character c on the standard output and
 returns c.

sbrk(n) - allocates memory block
 Allocates a block of n bytes of memory, returning
 the address of the first byte, or -1 if that much
 memory is not available.

strcat(s,t) - string cat
 Copies string t onto the end of string s.

strcpy(s,t) - string copy
 Copies string t into memory starting at the pointer
 s.

strcmp(s,t) - string compare
 Compares string s to string t, returning -1, 0 or 1
 if s is less than, equal to, or greater than t.
 Inequality is computed by numerical ASCII value. In
 particular, strings containing only upper or only
 lower case letters are compared in alphabetical
 order, but all upper case letters are less than any
 lower case letters.

strlen(s) - string length
 Returns the number of bytes in the string s, exclu-
 sive of the terminating 0 byte. Note that it takes
 strlen(s) + 1 bytes to hold string s.

toupper(c) - to upper case
 Returns the character c, but if c is an lower case
 letter (a - z) it is converted to the corresponding
 upper case letter.

unlink(s) - unlink file
 Deletes the file s, if it exists.

APPENDIX F. C80 Procedures

`atof(s)` - ASCII string to float conversion (from MATHPAK)
s is a string containing the representation of a floating point constant. `atof` returns the float value of the constant.

`exec(prog,args)` - execute another program
chain to another program. `prog` is a string containing the name of a program. `args` is a string containing any arguments.

`fclose(chan)` - file close
closes the file or device open on channel `chan`. An end of file (^Z) is written to the current position unless the file was opened in the binary mode.

`fopen(fname,mode)` - file open
opens the named file and returns the channel number of the file. `Fname` is a string constant or pointer to a string. Mode may be `r`, `w`, `u`, `rb`, `wb`, `ub`. `Fopen` returns 0 if the file cannot be opened.

`ftell(chan)` - tell file pointer
returns the current read/write pointer for the file open on channel `chan`. This pointer is the number of bytes before the current position in the file. Returns mod 256 if the number is greater than 65K.

`ftellr(chan)` - tell file record
returns the current read/write pointer for the file open on channel `chan`, divided by 256.

`ftoa(fmt,digits,f,where)` - float to ASCII convert (MATHPAC)
converts the floating point number `f` to a string in the character array `where`. `fmt` is for exponential or floating conversion. `digits` is the number of digits after the decimal place.

`getc(chan)` - get character from file
returns the next character from the file or device open for reading on channel `chan`. Returns -1 to signify end of file.

`isalpha(c)` - is character alphabetic?
returns 1 if `c` is an ASCII alphabetic character (A-Z or a-z), or otherwise 0.

`index(s,t)` - is s substring of t?
checks to see if `s` is a substring of `t`. If so, returns starting position in `s` of `t`; if not returns -1.

`isdigit(c)` - is character digit?
returns 1 if `c` is an ASCII digit (0-9), otherwise 0.

`isupper(c)` - is character upper case?
returns 1 if `c` is an upper case alphabetic character (A-Z), otherwise 0.

`isspace(c)` - is character space?
returns 1 if `c` is an space, tab, or newline, otherwise 0.

```

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_text
* FUNCTION: Writes PLOT text command to input channel.
* STRUCTURED ENGLISH:
* INPUTS: chan, text
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc / strlen
* HISTORY: N/A
*****/

```

```

pc_text(text,chan)
int      chan;
char     *text;
{
    test_space((1 + (strlen(text)) + 1),chan);
    putc('T',chan);
    while (*text) putc(*text++,chan);
    putc('\0',chan);
}

```

```

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_upload
* FUNCTION: Writes PLOT upload command to input channel.
* INPUTS: n, point, chan
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/

```

```

pc_upload(n,point,chan)
int      n,chan;
char     *point;
{
    int      i;
    char     *pt;

    test_space((3*n),chan);
    putc('U',chan);
    pt = &n;
    putc(*pt,chan);
    putc(*(pt+1),chan);
    for(i=1; i<=n; ++i) putc((*(point++)),chan);
}

```

```

pc_quit(chan)
int    chan;
(
    test_space(1,chan);
    putc('Q',chan);
)

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_string
* FUNCTION: Writes PLOT string command to input channel.
* STRUCTURED ENGLISH:
* INPUTS: chan, x1, y1, string
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc / strlen
* HISTORY: N/A
*****/
pc_string(x1,y1,string,chan)
int    chan;
float  x1,y1;
char   *string;
(
    int    i, outint[2];
    char   *pt;

    test_space((5 + (strlen(string)) + 1),chan);
    putc('S',chan);
    outint[0] = (int) (PLOTMAX * x1);
    outint[1] = (int) (PLOTMAX * y1);
    pt = &outint[0];
    for (i=0; i<=3; ++i) putc(*(pt+i),chan);
    while (*string) putc(*string++,chan);
    putc('\015',chan);
)

```

```

pc_output(chan)
int    chan;
{
    test_space(1,chan);
    putc('O',chan);
}

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_point
* FUNCTION: Writes PLOT point command to input channel.
* INPUTS: chan, x1, y1,
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/
pc_point(x1,y1,chan)
int    chan;
float  x1,y1;
{
    int    i, outint[2];
    char    *pt;

    test_space(5,chan);
    putc('P',chan);
    outint[0] = (int) (PLOTMAX * x1);
    outint[1] = (int) (PLOTMAX * y1);
    pt = &outint[0];
    for (i=0; i<=3; ++i) putc(*(pt+i),chan);
}

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_quit
* FUNCTION: Writes PLOT quit command to input channel.
* STRUCTURED ENGLISH:
* INPUTS: chan
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/

```

```

        outint[3] = (int) (PLOTMAX * y2);
        outint[4] = (int) (PLOTMAX * yf);
        pt = &outint[0];
        for (i=0; i<=9; ++i) putc(*(pt+i),chan);
    }

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_increment
* FUNCTION: Writes PLOT increment command to input channel.
* INPUTS: chan, x1, y1,
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/
pc_increment(x1,y1,chan)
int    chan;
float  x1,y1;
{
    int    i, outint[2];
    char    *pt;

    test_space(5,chan);
    putc('I',chan);
    outint[0] = (int) (PLOTMAX * x1);
    outint[1] = (int) (PLOTMAX * y1);
    pt = &outint[0];
    for (i=0; i<=3; ++i) putc(*(pt+i),chan);
}

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_output
* FUNCTION: Writes PLOT output command to input channel.
* INPUTS: chan
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/

```



```

        outint[3] = (int) (PLOTMAX * y2);
        pt = &outint[0];
        for (i=0; i<=7; ++i) putc(*(pt+i),chan);
    }

    /*****
    * DATE: 11 SEP 1984
    * VERSION: 1.0
    * NAME: pc_erase
    * FUNCTION: Writes PLOT erase command to input channel.
    * INPUTS: chan
    * OUTPUTS: none
    * GLOBAL VARIABLES USED: none
    * GLOBAL VARIABLES CHANGED: none
    * FILES READ: none
    * FILES WRITTEN: file on channel chan
    * MODULES CALLED: test_space / putc
    * HISTORY: N/A
    *****/
    pc_erase(chan)
    int    chan;
    {
        test_space(1,chan);
        putc('E',chan);
    }

    /*****
    * DATE: 11 SEP 1984
    * VERSION: 1.0
    * NAME: pc_fill
    * FUNCTION: Writes PLOT fill command to input channel.
    * INPUTS: chan, x1, y1, x2, y2, yf
    * OUTPUTS: none
    * GLOBAL VARIABLES USED: none
    * GLOBAL VARIABLES CHANGED: none
    * FILES READ: none
    * FILES WRITTEN: file on channel chan
    * MODULES CALLED: test_space / putc
    * HISTORY: N/A
    *****/
    pc_fill(x1,y1,x2,y2,yf,chan)
    int    chan;
    float  x1,x2,y1,y2,yf;
    {
        int    i, outint[5];
        char    *pt;

        test_space(11,chan);
        putc('F',chan);
        outint[0] = (int) (PLOTMAX * x1);
        outint[1] = (int) (PLOTMAX * y1);
        outint[2] = (int) (PLOTMAX * x2);

```

```

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_color
* FUNCTION: Writes PLOT color command to input channel.
* STRUCTURED ENGLISH:
* INPUTS: chan, color
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/

```

```

pc_color(color,chan)
int      color,chan;
(
    char      c;

    test_space(2,chan);
    putc('C',chan);
    c = color;
    putc(c,chan);
)

```

```

/*****
* DATE: 11 SEP 1984
* VERSION: 1.0
* NAME: pc_draw
* FUNCTION: Writes PLOT draw command to input channel.
* INPUTS: chan, x1, y1, x2, y2,
* OUTPUTS: none
* GLOBAL VARIABLES USED: none
* GLOBAL VARIABLES CHANGED: none
* FILES READ: none
* FILES WRITTEN: file on channel chan
* MODULES CALLED: test_space / putc
* HISTORY: N/A
*****/

```

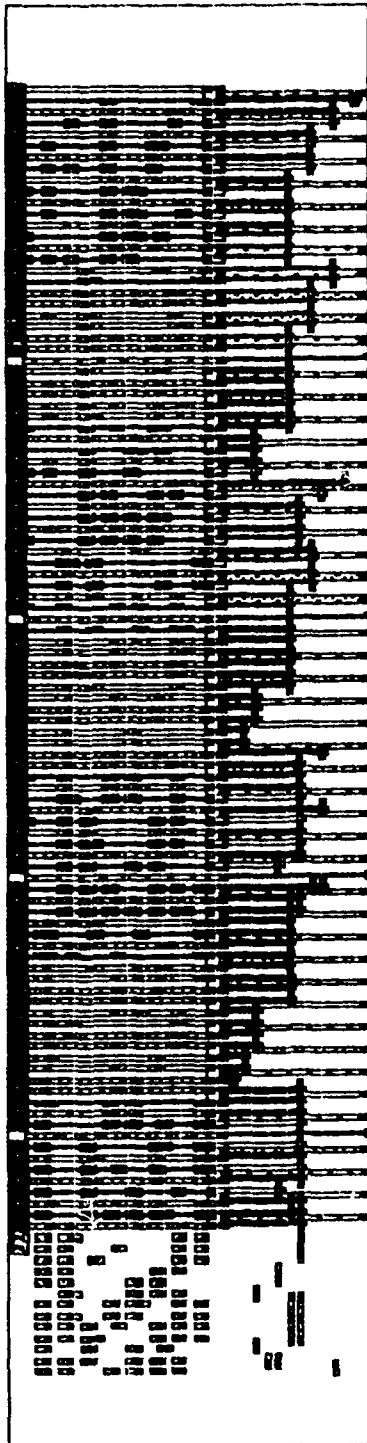
```

pc_draw(x1,y1,x2,y2,chan)
int      chan;
float    x1,x2,y1,y2;
(
    int      i, outint[4];
    char      *pt;

    test_space(9,chan);
    putc('D',chan);
    outint[0] = (int) (PLOTMAX * x1);
    outint[1] = (int) (PLOTMAX * y1);
    outint[2] = (int) (PLOTMAX * x2);

```

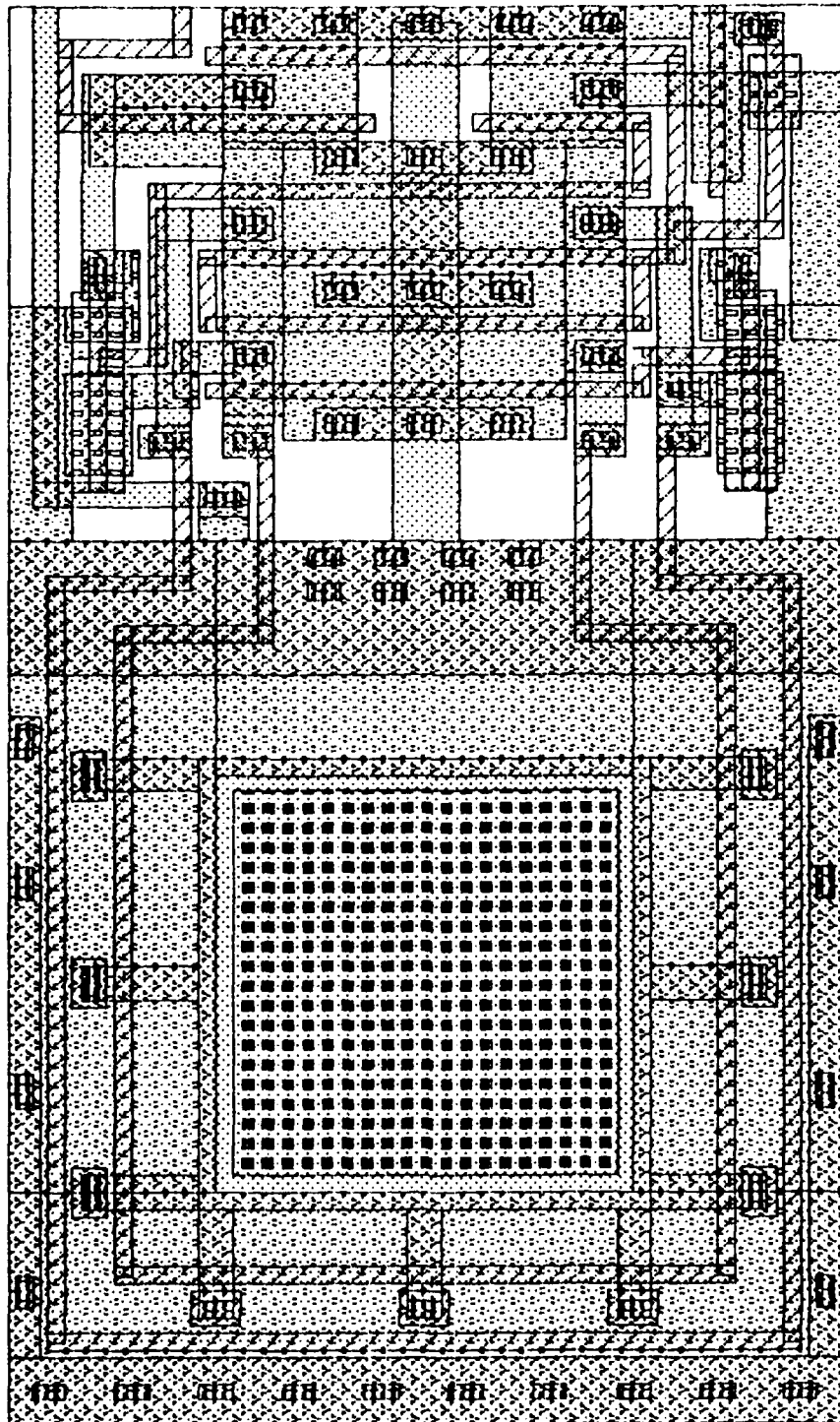
mcifplot* of MKPLA44.CIF
 Window: 0 64750 -246000 10500
 Scale: 1 micron is .0031189 inches



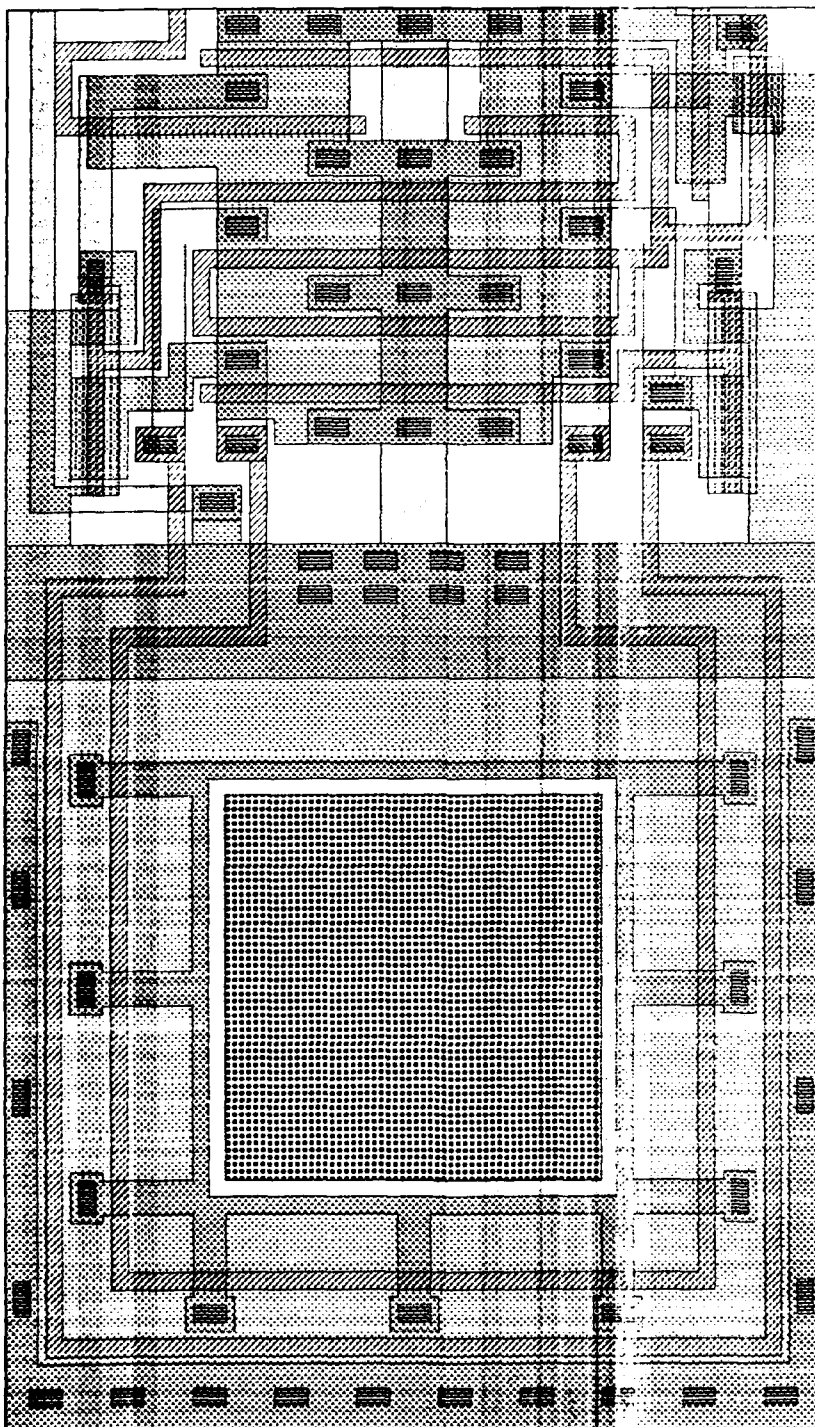
Wed Nov 14 22:45:31 1984
 cifplot* Window: -41 64791 -246000 10500 --- Scale: 1 micron is 0.0031189 inches (79x)
 mkpla 4 x 4 multiplier - mkpla44.cif with modified scale



mcifplot* of PADTRIST.CIF
 Window: 0 25000 0 42500
 Scale: 1 micron is .0188235 inches
 PADTRISTATE WITHOUT SPECIAL OPTIONS



Wed Nov 14 22:08:18 1984
 ctfplot Window: 0 230 8 340 --- Scale: 1 micron is 2.35294 inches (59765x)
 PADTRISTATE WITH MODIFIED SCALE



VITA

Kirk Stephen Horton was born on February 26, 1958 in Savannah, Georgia. He graduated from Lexington High School in 1976 and attended Clemson University where he received the degree of Bachelor of Science in Electrical Engineering in May 1980. Upon graduation, he was commissioned a second lieutenant in the USAF through the ROTC program. He was then stationed at Nellis AFB where he served as a Unit Electronics Engineer from August 1980 until May 1983. In June 1983, he entered the Air Force Institute of Technology to pursue a graduate degree in Electrical Engineering.

Permanent Address: 3508 Perliter Avenue

North Las Vegas, Nevada 89030

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/84D-35		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Office of Scientific Research	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code) Air Force Office of Scientific Research Bolling AFB, DC 20332		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See box 19		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT NO.
12. PERSONAL AUTHOR(S) Kirk S. Horton, B.S., Captain, USAF			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Yr., Mo., Day) 1984 December	15. PAGE COUNT 347
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB GR.	
09	02		
		Computer Aided Design; Integrated Circuits; Computer Programs; Computer Graphics; Microcomputer	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
Title: A MICROCOMPUTER-BASED PROGRAM FOR PRINTING CHECK PLOTS OF INTEGRATED CIRCUITS SPECIFIED IN CALTECH INTERMEDIATE FORM			
Thesis Chairman: Harold W. Carter, Lt. Col., USAF			
Approved for public release: IAW AFR 190-17. LYNN E. V. CLAVER Deputy for Research and Professional Development Air Force Institute of Technology (ATC) Wright-Patterson AFB OH 45433			
20. DISTRIBUTION AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED SAME AS REPORT DISCOVERIES		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Harold W. Carter, Lt. Col., USAF		22b. TELEPHONE NUMBER (Include Area Code) 512-245-6912	22c. OFFICE SYMBOL AFIT/ENG

mcifplot, a microcomputer-based program which prints check plots of integrated circuits on a dot-matrix printer, was designed, implemented, and tested. The SADT Design, Detailed Design, Data Dictionary, algorithms, Structure Charts, and program code are presented. mcifplot interprets a geometric description file written in Caltech Intermediate Form (CIF) and produces an output file which can be printed using the public domain program PLOT.COM. mcifplot is written in C for the CPM operating system and closely matches the function of the Unix based VLSI design tool cifplot.

mcifplot interprets all standard CIF 2.0 commands with the exception of Poly Commands, Delete Definition Commands, and commands which define geometric elements in non-Manhattan directions. Program execution is primarily limited by disk access delays and depends greatly on the format and structure of the input CIF file.

END

FILMED

5-85

DTIC