INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1336905

Display of arbitrary subgraphs for HPCOM-generated networks

Slipp, Walter Whitfield, M.S.

The University of Arizona, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DISPLAY OF ARBITRARY SUBGRAPHS FOR HPCOM-GENERATED NETWORKS

by

Walter Whitfield Slipp

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1989

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Walter W. Slipp

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

F. J. Hill — 28 April 1989

F. J. Hill
Professor of Electrical and Computer Engineering

Date

ACKNOWLEDGEMENTS

Hallelujah! After months of effort, this research project has finally come to its conclusion. As with all projects, there are a number of people that contributed advice and support for which I am grateful.

In particular, I would like to express my thanks to Dr. Fredrick Hill of the ECE department for his direction during the project and many helpful suggestions. In addition, I have appreciated the instruction of all the ECE professors who have given me a solid background in engineering.

My deepest thanks go to my parents for their unending support. Their encouragement towards my academic interests helped me immensely while in school.

TABLE OF CONTENTS

TABLE OF CONTENTS – Continued

TABLE OF CONTENTS – Continued

LIST OF TABLES

LIST OF ILLUSTRATIONS

ABSTRACT

Hardware description languages provide digital system designers with a convenient, compact method for describing complex circuits. A Hardware Programming Language (AHPL) is a powerful description language based on the APL programming language.

AHPL circuit descriptions can be unambiguously translated into a logic gate network using the HPCOM hardware compiler. The initial discussion section covers the conversion of the VAX version of HPCOM into a version which will run on MS-DOS microcomputers.

The major portion of the research focuses on the development, use, and application of a graphics display tool for HPCOM-generated networks. The display package, SUBGRAPH, allows selected subgraphs of a network to be viewed and/or printed.

The discussion of this research concludes with an extensive example of the complete circuit generation and graphics display sequence. The printed graphics examples feature cases of particular interest for test generation.

Chapter 1. Introduction

1.1. Hardware Description Language.

Many strategies have been developed to approach the problem of digital system design. Hardware description languages (HDL), which describe a digital system by means of a computer language, offer a number of attractive features. An HDL gives the digital system designer a convenient, compact method for describing complex circuits. It also provides convenient documentation which aids in communication between engineers [1]. Design time compared to the conventional schematic capture approach may be significantly less.

Hardware description languages are targeted for specific levels of abstraction with regard to the system description. There are four major levels of design abstraction [4]:

1. System level.
2. Register transfer level.
3. Gate and component level.
4. Detailed wiring level.

Abstraction decreases, while detail increases, from the system level down to the wiring level.

The system level represents the greatest abstraction. While design time is short, system level languages generally offer the least control over the actual hardware produced. The limited control over the architectural details renders system level languages unsuitable for much design work.

At the other extreme, the detailed wiring level generally requires a prohibitive amount of time to provide the necessary details. Combined with a propensity for error resulting from the detailed specifications, designers frequently require a higher level of abstraction.

The gate and component level works well for simple circuits but becomes unsuitable for LSI and VLSI circuits. Thus gate level description is not efficient enough for large circuits.

The register transfer level provides sufficient control over the hardware with an acceptable design time for large circuits. Most computer hardware description languages (CHDL) function at this level. A register transfer language (RTL) describes the system in terms of registers, data paths between registers, and a control sequence to coordinate data flow.

1.2. AHPL.

One of the most widespread academic CHDLs is AHPL, or A Hardware Programming Language, introduced by Hill and Peterson [2]. AHPL is a powerful description language based on APL and falls within the RTL abstraction level. AHPL has been developed and enhanced for almost two decades with the latest version of AHPL designated as Universal AHPL.

Flexibility has been a major goal and element of AHPL. System descriptions may now be written as units which fall into one of the following three categories [3]:

1. Combinational logic unit.
2. Functional register.
3. Module.

A combinational logic unit (CLU) consists exclusively of logic gates. A functional register includes logic gates and memory elements but no internal sequence control. A module is any description which does not fit within the first two groups. Any combination of these units may be used to form a complete system description.

Commonly-used units, or cells, could be developed as part of a library. The design process for application specific integrated circuits (ASIC) would be aided immensely by such a collection.

A major strength of AHPL is that each statement, which resembles an APL command, can be unambiguously translated into hardware. Thus the digital designer can be sure of the actual hardware and connections generated.

The system hardware for each description unit is partitioned into a data and a control section. The data section contains data registers and logic gates for data paths. The control section consists of registers and logic gates to coordinate data flow within the data section.

AHPL includes a broad base of support programs including a functional simulator, a hardware compiler, and a test-generation program. These programs are available in versions for the DEC-10, DEC VAX, CDC CYBER, and IBM 370. The simulator is also available for microcomputers running MS-DOS.

As technology progresses, software must adapt to new situations and incorporate new features to escape obsolescence. As mentioned, the simulator (HPSIM) is available for MS-DOS microcomputers which reflects the desire for AHPL software in a more affordable environment. The translation of existing software and addition of new AHPL software for microcomputers opens up access to millions of personal computers.

1.3. The Microcomputer Environment.

In the past, mainframe and minicomputer systems have been the only viable option for computer aided design (CAD) tools such as AHPL. However, with the advent of high-speed microprocessors and low-cost memory, microcomputers now provide an attractive, flexible environment.

Microcomputers based on the 8086 microprocessor series are currently the most widespread of the available microcomputer systems. MS-DOS is the principal operating system of these machines. The newer OS/2 operating system is available for 80286 and 80386-based machines. OS/2 allows for multi-tasking and, perhaps

more importantly, removes the 640K memory limit which has hindered memory-intensive CAD programs.

Numerous benefits can be gained from the microcomputer environment. Workstations may be easily customized to meet an individual's needs. Best of all, the prices for almost all microcomputers fall well short of the price of a mini-computer. Hence, software developed for a microcomputer has far greater potential for user access.

Of particular note is the graphics capability available on almost all micro-computers. The circuit hardware description produced by the AHPL compiler (HP-COM) is a numbered gate list. The representation of the circuit in terms of standard logic gate and register depictions must be produced by hand from careful study of the gate list. With access to both screen and printer graphics, output from the hardware compiler potentially could be displayed using gate and register symbols.

1.4. The Project Objective.

The intent of this thesis work was to both solidify and enhance the position of AHPL with respect to microcomputer accessibility. Previous work has been done to convert the AHPL simulator, HPSIM, for use on an MS-DOS computer. This project focused on hardware circuitry produced from an AHPL description.

Chapter 2 discusses the changes made to the VAX/VMS version of HPCOM so that it could be successfully compiled and run under MS-DOS. Once this foothold was established, additional software was developed to exploit the advantages of a microcomputer.

The following chapters explain the development, requirements, use, and internal structure of a graphics display package. This program which shall be titled SUBGRAPH is of particular value to the test-generation field. Selected portions

of an HPCOM-generated network may be viewed on the display monitor and/or printed which aids in the analysis of fault locations.

Current limitations and suggestions for possible future enhancements are discussed in the final chapter. The AHPL design environment will no doubt continue to be enhanced and developed as technology advances.

Chapter 2. MS-DOS HPCOM

2.1. HPCOM.

The AHPL compiler, HPCOM, accepts a circuit description written in AHPL and generates a gate list output with complete wire connections. HPCOM, which is written in VAX/VMS FORTRAN, is partitioned into three logical stages which are designated Stage 1, Stage 2, and Stage 3. Masud [4] and Chen [5] present a detailed description of the compiler.

Stage 1 is a distinct program (STAGE01) while Stage 2 and Stage 3 are combined in one executable program (STAGE23). STAGE01 creates an output file which is used as an input file to STAGE23.

Stage 1 produces a tabular representation of the circuit at the functional level. The AHPL source file is decomposed into a 16 separate tables to keep track of variables and provide a fast reference environment.

Stage 2 then develops an internal representation of gate and memory element interconnections from these tables in the form of a two-way linked list network description. It assigns control states and produces control logic for each distinctively controlled data register and bus.

Stage 3 is customized to provide appropriate output for a specific application. The original Stage 3 produces a generalized gate list which is suitable for the gate level analysis done in SUBGRAPH. While SUBGRAPH requires only the simple gate list, other applications might require additional processing. For example, it might be more time-effective for LSI and VLSI mask generation to have additional data produced within Stage 3 as it manipulates the gate list.

Figure 2.1 shows the compilation sequence for an AHPL description. The final gate list output from Stage 3 will be used as input to SUBGRAPH.

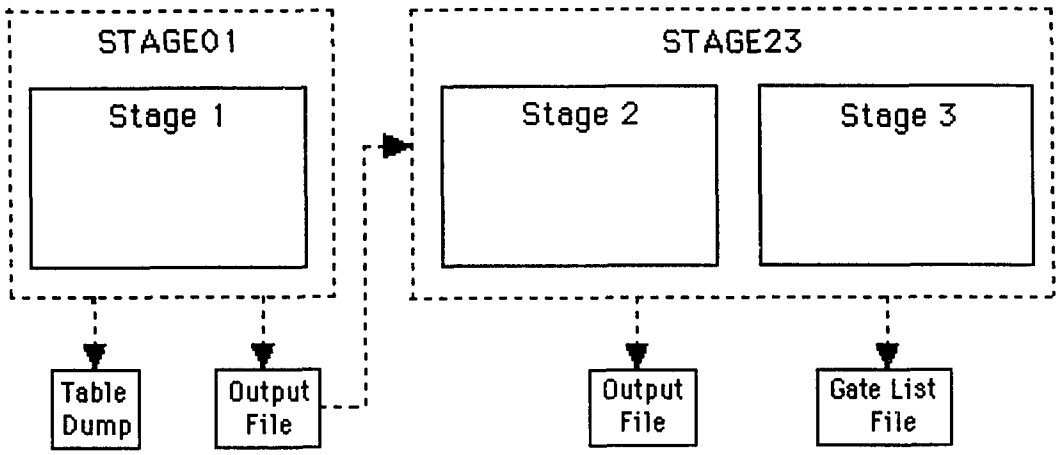


Figure 2.1. AHPL Stages

2.2. Modifications.

Numerous FORTRAN compilers exist which will operate under MS-DOS. Each has its advantages and peculiarities. Microsoft FORTRAN has achieved the greatest widespread popularity which, combined with excellent performance and flexibility, made it the best choice. Therefore, HPCOM was ported from the VAX to MS-DOS using the Microsoft FORTRAN Version 4.01 compiler.

The American National Standards Institute (ANSI) FORTRAN standard [10] is used for virtually all compilers as the basis for the language. However, ANSI standards usually leave a number of constructs and notations undefined to allow for incompatibilities at the time of the standard creation. The standard, then, usually represents a subset of most large-scale compiler versions.

Microsoft FORTRAN and VAX FORTRAN both support extensive, robust language definitions with non-ANSI features. Not all of the enhancements are held in common or defined identically.

The original authors of HPCOM frequently used three non-standard conventions which hindered its porting from VAX FORTRAN to Microsoft FORTRAN:

1. Octal constant notation
2. DATA statements for COMMON initialization
3. Control transfer to CONTINUE statements

An initial attempt was made to standardize VAX HPCOM with respect to the ANSI standard as much as possible. Changes were made to the VAX source code on the VAX so that the downloaded source code would compile under both the VAX and Microsoft versions. The output from the new VAX version was then compared with the output from the original version. This approach aided in verifying that the two versions produced the same results.

The notation for octal constants differs for almost all versions of FORTRAN. VAX FORTRAN allows for 123_8 to be represented as either "123 or '123'O. In Microsoft FORTRAN it must be represented by 8#123. Because no common notation existed, all octal constants were changed to decimal constants.

DATA statements were used to initialize variables in COMMON statements. The ANSI standard requires that a BLOCK DATA subroutine be used for initialization of variables in COMMON statements. Most of the DATA statements were converted to execution-time assignments while the rest were added to the BLOCK DATA subroutine.

GOTO statements frequently transfered control to CONTINUE statements from outside of the associated DO loops. Compilation occured correctly with Microsoft FORTRAN but produced warning messages, so all such transfers were changed to refer to the statement following the original CONTINUE.

After making the stated adjustments the new Stage 1 (STAGE01) source code compiled with both the VAX and Microsoft compilers. The Stage 2 and 3

(STAGE23) source code also compiled successfully with both compilers but a slight change was made to take advantage of Microsoft's intrinsic shift routines.

The VAX version also contained inconsistently-sized arrays within each set of source code. The following three PARAMETER constants were defined to standardize the array sizes within Stage 2 and 3:

1. L$store – size of STORE array
2. L$gate – size of gate arrays
3. L$iolist – size of IOLIST array

The following PARAMETER constant was used within Stage 1:

1. L$store – size of STORE array

Masud [4] give a thorough description of the meaning and use of these arrays. The constant of main concern is L$gate which is the maximum number of logic elements which can be generated. The constants were set to the values given in Table 2.1. Unfortunately, increasing the capacity of HPCOM is more complicated than merely increasing the values of these constants.

Table 2.1. HPCOM Constants.

Constant	Value
L$store	15000
L$gate	1600
L$iolist	6000

The memory requirement for each program after all modifications is given in Table 2.2. Although the STAGE23 requirement is roughly twice that of STAGE01,

the size differential is not simply because two stages are combined. Stage 2 performs the bulk of the manipulation, while Stage 3 does little more than format the output appropriately. If Stage 2 and Stage 3 were split, the overall memory requirement would still be over 300K. Both programs use static memory allocation, so the memory usage is fixed to these limits.

Table 2.2. HPCOM Memory Requirements.

Program	Memory
STAGE01	160K
STAGE23	350K

2.3. Stage 3 Output Tables

Stage 3 of HPCOM produces a complete wire list which is the input file for the graphics program, SUBGRAPH. Additional tables which supplement the gate list are also provided.

One of the functions of Stage 3 is to customize the data for specific applications. The original Stage 3 program from the VAX output several tables along with the gate list which are not used by SUBGRAPH. Although no other microcomputer programs are currently available which use the Stage 3 output, these tables have been left intact to avoid compatibility problems in the event that additional VAX programs are ported for microcomputer use.

The major reason that these tables are discarded is that SUBGRAPH is written in C instead of FORTRAN. C allows for dynamic memory allocation. HPCOM simulates memory allocation by utilizing a large static array, STORE, and then maintaining reference tables to keep track of the allocated segments. Eliminating this accounting eliminates the need for several of the tables. Also only certain

information was considered useful for display purposes, so additional tables were discarded.

The gate list file from stage 3 is composed of seven tables from HPCOM:

1. Symbol Table
2. System Table (STS)
3. Symbol Description Table (SDT)
4. Step QTABLE Relation Table (SQRT)
5. Combinational Logic Unit Table (CLUTAB)
6. Hash Table
7. Gate List

Each table will be briefly described in the following sections.

2.3.1. Symbol Table

The symbol table as shown in Table 2.3 stores the names of symbols from the AHPL description. Symbol names may apply to memory elements, inputs, outputs, CLUnits, functional registers, and modules.

Symbol names of up to 20 characters in length are allowed by the compiler. However, only the first ten are passed to SUBGRAPH. Therefore, all symbol names should be unique in the first ten characters.

2.3.2. System Table (STS)

The system table stores reference information within HPCOM. Entries are made for each combinational logic unit, functional register, and module description.

The table contains the list of steps, master clock, and bounds of allocated STORE segments for each unit. The majority of the data is applicable exclusively within the HPCOM compilation environment.

For purpose of circuit display, none of the information would enhance the circuit schematics. The useful information from the table is incorporated into the

Table 2.3. Symbol Table Content Description

Column	Description
1	First five characters of the symbol
2	Second five characters of the symbol

Table 2.4. Symbol Declaration Table Content Description

Column	Description
1	Symbol table row number for the name of the symbol
2	Code for the type of the symbol
3	Row number of the REF table
4	Number of bit columns of the symbol
5	Number of bit rows of the symbol
6	Row number of THUNK table for columns
7	Row number of THUNK table for rows
8	Unique number assigned to each declared symbol
9	Row number of TOTS
10	Unused

gate list as it is produced. Thus the system table will not be utilized outside of HPCOM.

2.3.3. Symbol Description Table (SDT)

The symbol description table records information describing the symbols in the symbol table. Only some of the columns in Table 2.4 are retained by SUBGRAPH. The columns refering to the REF, THUNK, and TOTS tables are discarded for the same reasons which apply to the system table.

The remaining information describes the nature of each symbol. The code entry distinguishes between memory elements, inputs, and outputs. AHPL description unit names do not have an entry in this table.

2.3.4. Step QTABLE Relation Table (SQRT)

The step QTABLE relation table contains information for the hardware control section. Each AHPL statement receives a slot in the table. A D flip-flop is generated for each clocked statement. The identification number of the flip-flop is placed in that statement's entry. NODELAY steps also receives an entry although no flip-flop is generated.

The decision was made not to import this table into SUBGRAPH. The benefit of including this information would be that each control flip-flop could be labeled as to the step number which it controls. However, the control section was viewed as having less importance for purposes of test-generation. Combined with the fear of over-crowding the display, SQRT was not utilized by SUBGRAPH. Should experience prove these reasons unjustified, future developers of SUBGRAPH may wish to include the control flip-flop labeling.

2.3.5. Combinational Logic Unit Table (CLUTAB)

The combinational logic unit table contains data when combinational logic units are used. CLUTAB is similar in purpose and function to the system table.

None of the information was considered useful for gate list display so it too was discarded.

2.3.6. Hash Table

A hash table with 1000 slots is used within HPCOM for fast access to symbol names and other tokens. The table is provided in the output on the assumption that it will be imported into a similar environment.

Each logic element from the actual gate list, which will be discussed in the next section, includes an index into the symbol table if appropriate. As this is the only referencing to symbols which is performed, the hash table is also discarded.

2.3.7. Gate List

The gate list table is the only essential section of the input file. The actual hardware with connections is described by this data. The symbol table and symbol description table are used to supplement the gate list. The gate list is not an actual table within HPCOM but is produced for the output file.

Each entry in the gate list appears in the form shown in Figure 2.2. Only the first line containing n_1 through n_7 will necessarily appear. The a_1 and a_2 values are present only if the logic element is associated with a symbol, which can be determined by a non-zero value of n_5. An element must have either inputs or outputs, and possibly both. The presence of inputs and outputs is identified by non-zero values of n_3 and n_4 respectively.

$$n_1\ n_2\ n_3\ n_4\ n_5\ n_6\ n_7$$
$$a_1\ a_2$$
$$i_1\ i_2\ \ldots\ i_n$$
$$o_1\ o_2\ \ldots\ o_n$$

Figure 2.2. Gate List Entry.

A description for each value in Figure 2.1 is given in Table 2.5. The values of n_6 and n_7 are meaningful only within HPCOM so they are ignored.

The values in the input and output lists are the numbers of other logic elements to which a connection exists. Because both inputs and outputs are listed for each element, the gate list is a two-way linked list. The order of elements in the input and output lists bears no special significance except in the case of inputs to a memory element. This significance will be discussed in chapter 4.

2.4. Recommendations.

Perhaps the main deficiency of HPCOM is that it lacks dynamic memory allocation. Rather than being limited by the machine's capabilities, limitations are imposed by the HPCOM software. Memory which is not used by a minicomputer user might be assigned to another user, but on a microcomputer, unused memory is simply wasted.

FORTRAN has been and still is one of the best languages for calculation-intensive programs. However, the nature of HPCOM requires only simple integer calculations and string manipulation. FORTRAN lacks the flexibility of PASCAL or C with respect to memory management.

All the required constants and tables could be increased within HPCOM, but if the limits were set to utilize a specific amount of memory, there will always be a user with more memory that might benefit from that extra capacity. The long run solution is to convert HPCOM to a language such as C which can maximize usage of the machine's capabilities. Of course, this would require major effort but it might be in the best interest of HPCOM's future.

A more realistic change could be made to the user interface. One of the friendlier features of the VAX/VMS operating system is that multiple versions of a file may be kept. Therefore, if a file name is entered for use as an output file that already exists, a new version will be created.

Table 2.5. Gate List Entry Explanation.

Number	Description
n_1	Gate Number
n_2	Gate Type
n_3	Number of Inputs
n_4	Number of Outputs
n_5	Symbol Indicator
n_6	Ignored
n_7	Ignored
a_1	Symbol number
a_2	Element position
i_1	Input #1
$\vdots$	$\vdots$
i_n	Input #n
o_1	Output #1
$\vdots$	$\vdots$
o_n	Output #n

MS-DOS allows for only one version. The Microsoft FORTRAN reaction to writing to an existant file is to abort execution. The ability to enter an alternative file name would greatly enhance the user-friendliness of HPCOM.

Chapter 3. SUBGRAPH.

3.1. Goals.

SUBGRAPH is a graphics display package which aids the digital engineer by displaying selected subgraphs of a circuit. Its utility is best realized when used in conjunction with the AHPL test sequence generation program SCIRTSS.

The primary goal which was set for SUBGRAPH was to realize a program which performed the required functions. The main emphasis was placed on display of subgraphs in a printed form which could then be written on. Secondary emphasis was placed on viewing the subgraph on the microcomputer screen prior to printing.

The secondary goal was to present the user with a fast, comprehensible program interface. The AHPL design system originated during a period when merely producing the desired results was a major accomplishment. As computers have grown in power and flexibility, more emphasis has been placed on the ease with which a user may interact with the program. Thus considerable attention was given to the user interface.

The nature of microcomputers allows numerous options with respect to available and implemented hardware. The demands which could be placed on the potential user were evaluated and are discussed in the following section.

3.2. System Hardware Requirements.

One of the benefits of a personal computer is that it may be customized to meet the user's needs. With all the available memory sizes, graphics adapters, microprocessor speeds, peripherals, etc., the number of possible microcomputer hardware configurations is virtually unlimited.

Such potential for configurations can also be viewed as a drawback. The behavior of software is often dependent upon exact hardware configurations. Similar

components from different manufacturers may in fact be incompatible from the software point of view. This problem is particularly applicable to graphics adapters and printers.

Therefore system requirements must be defined to assure that a user can run HPCOM and SUBGRAPH on his system. Ideally, these requirements will be as minimal as possible to allow for a wide range of systems and thus maximize the distribution potential.

The features of concern as requirements are:

1. Minimum Memory.
2. Printer.
3. Graphics Adapter.

Some system feature differences have no impact on the system decision. For example, diskette size is not a concern because HPCOM and SUBGRAPH can be accessed from either the 5.25" or 3.5" size. STAGE01, STAGE23, and SUBGRAPH can all be placed on the same 360K diskette.

The microprocessor speed affects the required compilation time for AHPL descriptions in HPCOM and the rate of screen displays for SUBGRAPH. Print-outs from SUBGRAPH are so restricted by the printer speed that the processor speed makes no difference. No minimum speed is required nor could one be enforced.

3.2.1. Minimum Memory.

STAGE01 and STAGE23 require 160K and 350K of random access memory (RAM) respectively. SUBGRAPH requires a base of 70K and dynamically allocates additional memory as needed. Therefore, the minimum memory size must be at least 350K.

Microcomputers are normally configured with memory in increments of 128K. Memory sizes of 512K and 640K are typical for an MS-DOS machines while OS/2 machines often have several megabytes of memory.

The closest configurable memory size to 350K is 512K. Therefore the stated requirement for memory will be 512K of which 350K must be available for program use. Conceivably, memory-resident programs might be in memory concurrently with HPCOM or SUBGRAPH. If sufficient memory is not available, HPCOM will simply not run, while SUBGRAPH may be choked and abort execution from lack of memory.

3.2.2. Printer.

Printers use a sequence of numbers known as "control codes" to access special features such as bit-mapped graphics. Choice of the specific numbers for the control codes depends on the manufacturer.

Epson has traditionally been the leader in the printer market. The control code sequences for bit-mapped graphics on an Epson dot-matrix printer are given in Table 3.1.

Table 3.1. Printer Control Codes

Command	Format
Single Density Graphics	27 75 n_1 n_2 v_1 v_2 ... v_n
Variable Graphics Spacing	27 74 n

For the single density graphics command, n_1 and n_2 define the number of graphics columns for the current row. The v_1 to v_n values are the actual graphics

data. One byte must be supplied for each column. For variable graphics spacing, n represents the number of .0046" page advances while in graphics mode. The value of n is normally 24. These commands and their values are transparent to the user.

The stated control codes are recognized by most Epson, IBM, and Panasonic dot-matrix printers. Any printer which recognizes these graphics control sequences should be compatible with SUBGRAPH. These control codes are supported by even low-end dot-matrix printers.

SUBGRAPH was implemented and tested using an IBM Proprinter. The vertical resolution of a Proprinter is 72 dots per inch. At least some Epson printers, such as the LQ 800, support only 60 dots per inch. A printout on an Epson would appear slightly elongated but only in comparison to the Proprinter output.

The requirement for printer use is a dot-matrix printer which recognizes the control code sequences in Table 3.1. From a limited survey, it would appear that a large percentage of dot-matrix printers are supported.

3.2.3. Graphics Adapter.

The original IBM PCs in the early 80s did not have built-in graphics capability but were text-only machines. Bit-mapped graphics required the installation of a special graphics adapter. Since that period the graphics adapters given in Table 3.2 have been widely used.

Variations of the adapters have also been produced. For example, Tandy CGA has the resolution of standard CGA but the color capability of EGA. The cost of the adapter generally increases as the resolution increases.

Screen display of a schematic was considered of secondary importance compared to the printed display. The actual process for screen display is that the print buffer is displayed on the screen instead of being sent to the printer. There is a

one-to-one correspondence between the printed pixels and the video pixels. Therefore, any screen resolution can be used and as screen resolution increases, more of the circuit subgraph can be displayed.

Table 3.2. Graphics Adapters.

Adapter	Resolution
Hercules Graphics Card (HGC)	738 × 348
Color Graphics Adapter (CGA)	320 × 200, 640 × 200
Enhanced Graphics Adapter (EGA)	640 × 340
Multi-Color Graphics Array (MCGA)	640 × 480 (monochrome)
Video Graphics Array (VGA)	640 × 480

With the introduction of the IBM PS/2 line of microcomputers, VGA has become the industry standard. The previous standard was EGA which is still practical for many graphic applications. Any of the graphics adapters in Table 3.2 may be used, but either EGA or VGA would be recommended. A Hercules card provides good resolution but is limited to monochrome display. Future versions of SUBGRAPH may utilize color. Graphics adapters will be discussed further in chapter 5.

SUBGRAPH does not actually require a graphics adapter. If no recognizeable adapter is sensed, the view routine can not be accessed, but all other routines behave normally. This allows for virtually all MS-DOS machines to be used.

3.2.4. Summary.

The hardware requirements for HPCOM and SUBGRAPH are quit minimal. Almost no MS-DOS microcomputers are being produced with less than 512K of

memory. No graphics adapter is required although an EGA or VGA adapter is recommended. The printer presents the major restriction, which presently appears unimposing.

The microcomputer industry is one of the fastest changing sectors of the electronics industry. Future developments will no doubt require modifications to the bit-mapped graphics support due to adapter and printer changes. For the present, the requirements and recommendations for effective use are given below:

1. Minimum Memory of 512K.
2. Dot-matrix printer compatible with the specified control codes.
3. EGA or VGA Graphics Adapter (recommended).

Chapter 4. Data Structures

The tables within SUBGRAPH may be classified into three areas:

1. Dynamic storage tables for the gate list.
2. Static storage tables for graphics data.
3. Tables for logic element layout.

The dynamic tables are blocks of memory allocated when a file is loaded and then accessed by pointers. The static tables are predefined tables which are generally indexed.

The dynamic storage tables consist of the element table, the symbol table, and the system table. Of these three only the element table is required. The other two tables provide label information to identify flip-flops, inputs, and outputs by name when displayed.

The static storage tables for the graphics data consist of bit-mapped graphic representations of the logic elements and alphanumeric characters. As discussed in chapter 4, the graphic representations were designed for the IBM Proprinter. The graphic representations appear identical when displayed on an EGA (640×340) monitor. Display on a printer or screen with slightly greater or lesser resolution will be acceptable, but a major increase in resolution would require the addition of new graphic representations.

The tables for logic element layout consist of a dynamic layout table and a static position offset table. The algorithm for layout and use of these tables will be discussed in detail in the location table and layout tables sections.

SUBGRAPH was written in Microsoft C so the tables discussed are formed from C structures. Each table is generally referenced by a structure pointer.

4.1. Element Table

The most important table within SUBGRAPH is the element table. All logic elements are stored in this table. Each element slot contains a pointer to the list of input connections and a pointer to the list of output connections. Throughout this paper the term "gate" and the term "element" are used interchangeably, although inputs, outputs, and memory elements are not actually gates.

The table is allocated dynamically as a continuous segment of memory at the time that an input file is loaded. The table size is allocated based upon the number of elements which is given at the start of the gate list in the input file.

Each slot in the table is defined as a structure of type "element". The element table is then referenced by an "element" structure pointer "gateptr".

HPCOM generates the network and then removes unnecessary gates. The number of elements applies to the pre-optimized network, so the actual number of elements will be less. Space is wasted because slots within the table are left empty. To eliminate this waste, the element numbers should be resequenced before the output tables are written by Stage 3. Stage 3 was not changed to avoid creating any incompatibilities with other software.

Each element in the gate list from the Stage 3 output is converted into the slot form shown in Table 4.1. The element slot contains pointers to the input and output connections, each of which are stored in a block of allocated memory. Each element slot requires 21 bytes plus any space allocated for input and output connections.

An approximate maximum for the number of elements can be determined based on the following assumptions:

1. SUBGRAPH requires 100K for the executable code.
2. SUBGRAPH is run on a machine with 640K.

Table 4.1. Element Slot Field Description

Name	Type	Bytes	Description
type	int	2	Element type
sym	int	2	Symbol number
elem	int	2	Element position
placed	char	1	Flag for placement in location table
xpos	char	1	X coordinate in location table
ypos	char	1	Y coordinate in location table
numin	int	2	Number of inputs
cin	int *	4	Pointer to input connections
numout	int	2	Number of outputs
cout	int *	4	Pointer to output connections

3. The average element has 4 inputs.
4. The average element has 2 outputs.
5. All other dynamic tables are of negligible size.

Based on these assumptions, the upper limit would be approximately 16,500 elements. Of course, because SUBGRAPH allocates memory dynamically, it can be run on a machine with less memory or, if running under OS/2, could use substantially more memory. At present HPCOM can produce a maximum of 1600 elements, so SUBGRAPH will permit substantial expansion.

4.2. Symbol Table

The symbol table is a table of character structures. A ten-character character string is defined as the structure "symtab". At the time an input file is loaded, space for the required number of symbol structures is allocated.

Table 4.2 shows the table structure. The symbol table is referenced through the "symtab" structure pointer "symptr".

4.3. Symbol Description Table

The symbol description table stores information regarding the number of column and row bits associated with a symbol. The main purpose of this table is to recognize when a symbol truly has columns and rows. The symbol description table is referenced through the "sdttab" structure pointer "sdtptr".

From Table 4.1, each element has a "sym" and an "elem" field. If the element is an input, an output, or a data flip-flop, it is associated with a symbol. The "elem" field will contain a number in the form shown in Figure 4.1.

The "elem" field can be checked to determine the row and column position of the element. However, this field does not indicate if the symbol actually includes rows and columns for certain cases. The element for a dimensionless symbol can

not be distinguished from elements in row 0 of a multiple-row symbol or column 0 of a multiple-column symbol by the "elem" field alone. For example, the element in row 0 and column 0 of A⟨4⟩[4] has the same "elem" value as the element for B (no dimensions).

position = row*4000 + column

e.g. 12004

row = 12/4 = 3 column = 4

Figure 4.1. Element Position

Therefore, the symbol description table must be checked for each symbol to assure the correct dimensions are indicated.

4.4. Location Table

The location table is a dynamically allocated block of memory which is used for coordinating the placement of logic elements for display. The table may be logically conceived of as a rectangular matrix.

The number of rows is equal to the maximum number of elements which may be displayed horizontally on an 8.5″ × 11″ piece of paper. The value is defined as a constant, GWIDTH, and is currently set at 11.

The number of columns represents the number of elements which may be displayed vertically. Because tractor-feed computer paper is continuous, there is no upper limit for the number of columns. An initial value for the number of columns is chosen, but if the required number of columns surpasses that value, a new, larger block of memory is allocated and the old table is copied into the new table.

The location table is a memory block of integers which is pointed to by the integer pointer "locptr". Each vacant location contains a zero. Each non-vacant

location contains the number of the element which resides there. Figure 4.2 shows an example location table.

4.5. Layout Tables

The layout tables are used in conjunction with the location table. The "first-pos" table is statically allocated and contains column numbers for the location table. The "gatpos" table is also statically allocated and contains character data which are actually single byte numbers. In actual use, the value OFFSET is always subtracted from the "gatpos" table numbers to produce a signed offset.

The layout algorithm begins by placing a logic element in the center of the first column of the location table. The elements with connections to the inputs of that element are then placed in the second column according to the row values in "firstpos". The row subscript is the number of inputs up to eight. The layout algorithm supports up to eight inputs. Any additional inputs are ignored.

Beginning from the second column, each column of the location table is searched for non-vacant locations. When a placed-element is found, the "gatpos" offsets are used to find open locations for the elements with connections to the inputs of the element.

The next column is searched for open locations by adding the offsets to the row location of the specified element from the previous column. Depending on the number of input elements which must be placed, there are several combinations of positions into which the new elements may be placed. The location possibilities for three input elements are shown in Figure 4.3.

The location of the AND gate in the first row is fixed. The three AND gates below it are tried in the different locations until a successful fit is found.

If the element can not be placed in the immediately following column, then the column beyond that is tried and so on. The layout algorithm based on offsets

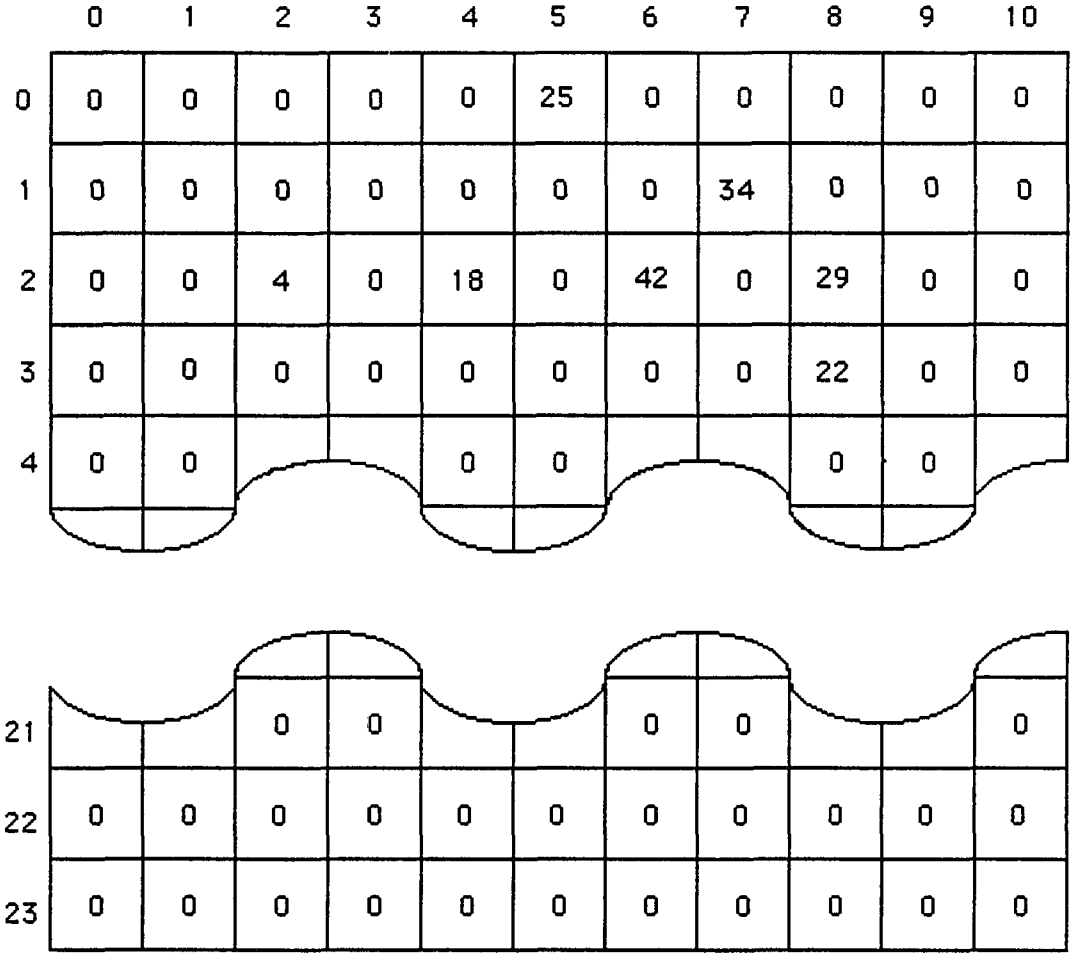


Figure 4.2. Example Location Table.

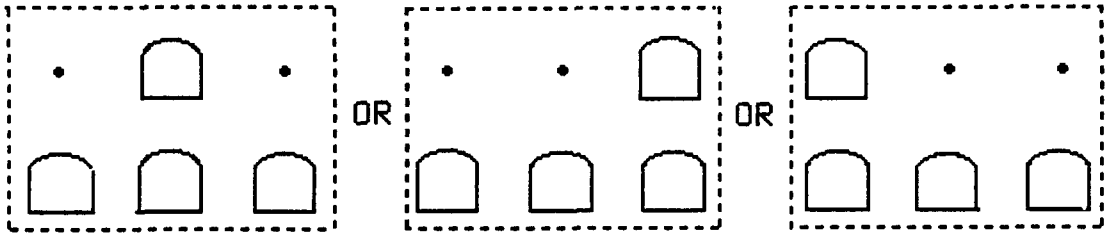


Figure 4.3. Offset Example.

allows for a logical grouping of related elements. If a large number of logic elements are involved in the subgraph (e.g. more than twenty) the display should still retain a structured appearance.

A maximum of MAXIN, which is currently eight, inputs may be displayed for a gate. A maximum was set due to manipulation problems which occur as the number of inputs increases. Commercial logic gates with more than eight inputs do exist. The SN74133 is a 13-input NAND gate [8]. However, the normal maximum for most gates is eight inputs, such as the SN7430 8-input NAND gate.

4.6. Print Buffer

The print buffer is a block of character bytes which forms a table with RBUFSIZE rows and CBUFSIZE columns. RBUFSIZE and CBUFSIZE are currently 12 and 480 respectively. Each byte in the buffer produces eight pixels of output. Therefore, each row of the buffer provides eight rows of pixels across the entire page or screen.

Individual pixel locations within the buffer may be addressed which is done for the case of alphanumeric characters. Logic element representations are always placed in the same row locations.

The print buffer has been sized to accomodate one column of logic elements. Because no upper limit can be placed on the number of columns of logic elements, the display must be done a section at a time.

Each time a display is required, the print buffer is cleared (i.e. set to all zeros). The bit-mapped element representations are then OR-d into place along with applicable labels and numbers.

The print buffer is then sent to the printer or screen one byte at a time. The print routines contain algorithms to speed up display by not sending blank data to the printer.

4.7. Graphic Element Representations

Each graphic element representation is associated with a table of bits which when printed or viewed produce the pictorial element. The tables are LRSIZE by LCSIZE in size. LRSIZE and LCSIZE are currently 8 and 31 respectively. Each byte in the table supplies eight pixels of graphic information.

Figure 4.4 shows the bit representation of the number three. Seven bytes are used to store the required bit maps. The byte values are displayed underneath each column of the map.

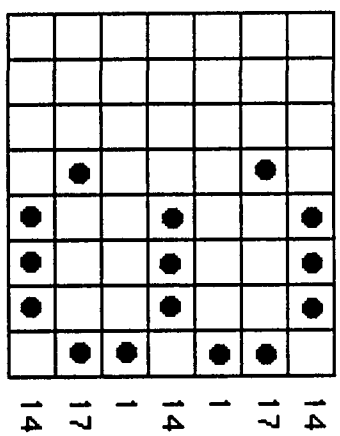


Figure 4.4. Character Bit Map.

The maximum number of logic elements which may be displayed in a column of printed output is limited to the floor of the row width in columns divided by LCSIZE. The current limit is 12 which is reduced to 11 because some space between elements is required for connection wires. Figure 4.5 shows the EOR gate bit map assignments.

Logic elements and characters are displayed by ORing the graphic bytes into the print buffer. When all the information has been placed into the buffer, the buffer is then printed or viewed.

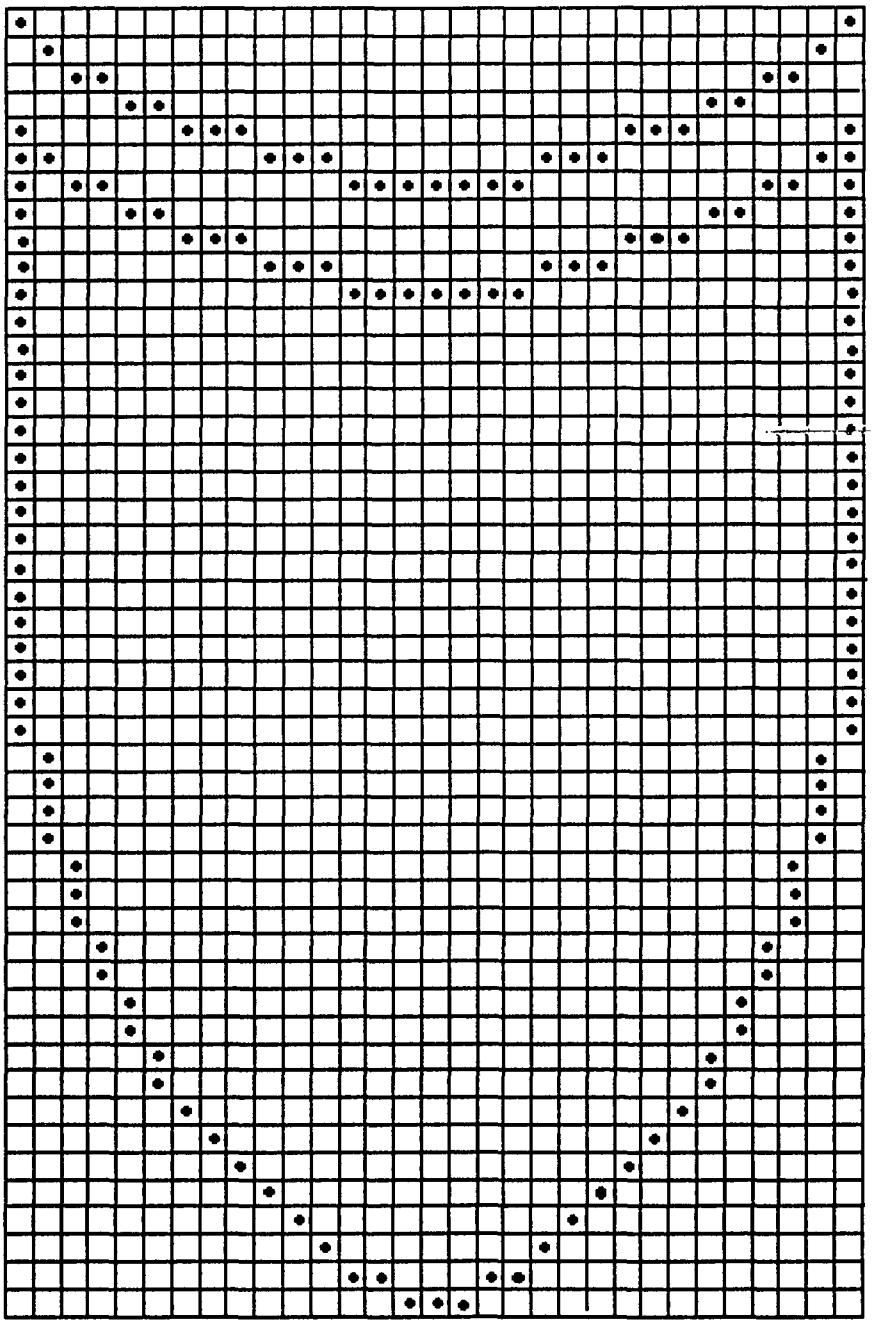


Figure 4.5. EOR Bit Map.

Chapter 5. SUBGRAPH Options

Over the past decade more and more attention has been given to the user interface of a program. Programs which merely do the required job but present the user with cryptic prompts or promote a general feeling of confusion are no longer tolerable. Much of the success of Apple's Macintosh computer line may be attributed to the extreme emphasis placed on its powerful, user-friendly interface.

Thus careful attention was paid to maximize the "useability" of SUBGRAPH. The main realization of this attempt was the development and use of a menu system.

Menu-driven interfaces eliminate the need to remember the possible options at a given level. If the option names are descriptive enough, the need for a user guide can almost be eliminated.

The menus are organized as a tree structure. A root, or main, menu provides access to options which in turn may branch into sub-options.

Each menu displays a list of options. A cursor in the form of a highlight bar may be moved up and down to the desired option. The list is circular in that moving down from the bottom option moves the cursor to the top option and vice versa.

The UP and DOWN arrow keys are used to move the highlight bar to the respective adjacent option. The ENTER key selects the highlighted option. The selection method could eventually be implemented using a mouse. Simple mouse movement would correspond to the UP and DOWN keys, while a mouse-click would emulate pressing the ENTER key.

The first option of each menu is "Quit" or "Exit" which quits back to the previous menu or exits the program respectively. The ESC key produces the same

effect and may be used to exit from any menu or prompt. The principal reason for including the quit option is for the eventual inclusion of a mouse driver.

5.1. Load Gate List

Prior to any other action, a file containing the gate list and associated tables must be loaded. The input file must rigidly conform to the input table structures described in chapter 2. If any irregularities are observed, the load is aborted. Therefore, modification of the output stage of MS-DOS HPCOM must be justified with the input routines for SUBGRAPH.

Good file organization practice recommends that files for a specific application have a descriptive file extension for easy recognition. SUBGRAPH recognizes the .GLF (Gate List File) extension as a potential input file. Any file name or extension may be used, but the specified extension aids in file management.

Upon accessing the load option, an intrinsic Microsoft C routine is invoked to obtain the current working directory. The names of all files with a .GLF extension within this directory are then obtained using another routine and are then displayed. Displayed file names are for the user's benefit only. A complete file name and extension must still be entered at the file name prompt.

Besides the mentioned Microsoft C DOS interface routine, numerous other Microsoft C dependent routines were utilized for screen management. Porting the source code to another C compiler would require major reworking of certain sections of code. However, such usage was unavoidable, and also does not appear to currently impose any problems.

5.2. Gate Information

Stage 3 of HPCOM produces an output file suitable for reading. In order to complement this output, SUBGRAPH provides a feature which will list all the logic elements of a particular type based on one of the following three groups:

1. All logic elements
2. All elements of a specified type (e.g. NOR gates)
3. An individual element

Examples for each of the three cases are given in chapter 6.

The gate information menu lists each logic element type as well as an "all", and a "find gate" option. Choosing an element type performs a search of order O(n) through the complete element table.

Each successful match is displayed with the element number, element type, number of inputs, and number of outputs. If no elements of the specified type are present in the circuit "** none **" is displayed. Choosing "all" displays all logic elements in the same format.

The find gate option requests an element number which may be obtained from the other options. The element number is also its index into the element table so no search is performed.

For the find gate feature, the first line lists the element number, element type, number of inputs, and the number of outputs. If the element is a flip-flop, input, or output, its symbol name and bit row and column follow if applicable. The gate numbers of any elements which are connected to the inputs of the specified gate are displayed on the next line. Similarly, element numbers for output connections follow the inputs.

5.3. Subgraph Manipulation

5.3.1. Subgraph Definition

One of the motivations for displaying only a selected portion of a digital circuit comes from the field of fault detection. If a stuck-at-one (SA1) or stuck-at-zero

(SA0) fault propagates through to a specific point, the stuck-at fault will be located within the subset of gates and registers which will be defined as a "subgraph".

From an initial logic element, the subgraph consists of all gates and registers encountered as paths are traced back from the inputs of the initial and preceeding logic elements. Each trace-back path is terminated when a primary input or register is encountered. If the initial logic element is a flip-flop, either the D input, the enable input, or both, may be chosen as the input for trace-back.

Figure 5.1 exhibits several subgraph examples. The entire figure is the subgraph of gate 14. All inputs paths can be traced back to a primary input or register. The sections labeled A and B represent the subgraphs of gates 33 and 29 respectively.

5.3.2. Selection

Gate selection requests the number of a logic element. The gate information option or the Stage 3 output may be consulted for gate numbers. After successfully selecting a gate number, SUBGRAPH internally organizes the layout pattern.

If the initial element is not a flip-flop, then no further prompting occurs. If the initial element is a flip-flop, then the next prompt requests the input line for trace-back.

The only memory elements currently supported by HPCOM are D flip-flops which are modeled with five inputs. The set, reset, and clock inputs are ignored for subgraph purposes. The D input, enable input, or both, may be selected for trace-back of the subgraph. The D input and enable input often will share gates, so the resulting subgraph may have crossing lines.

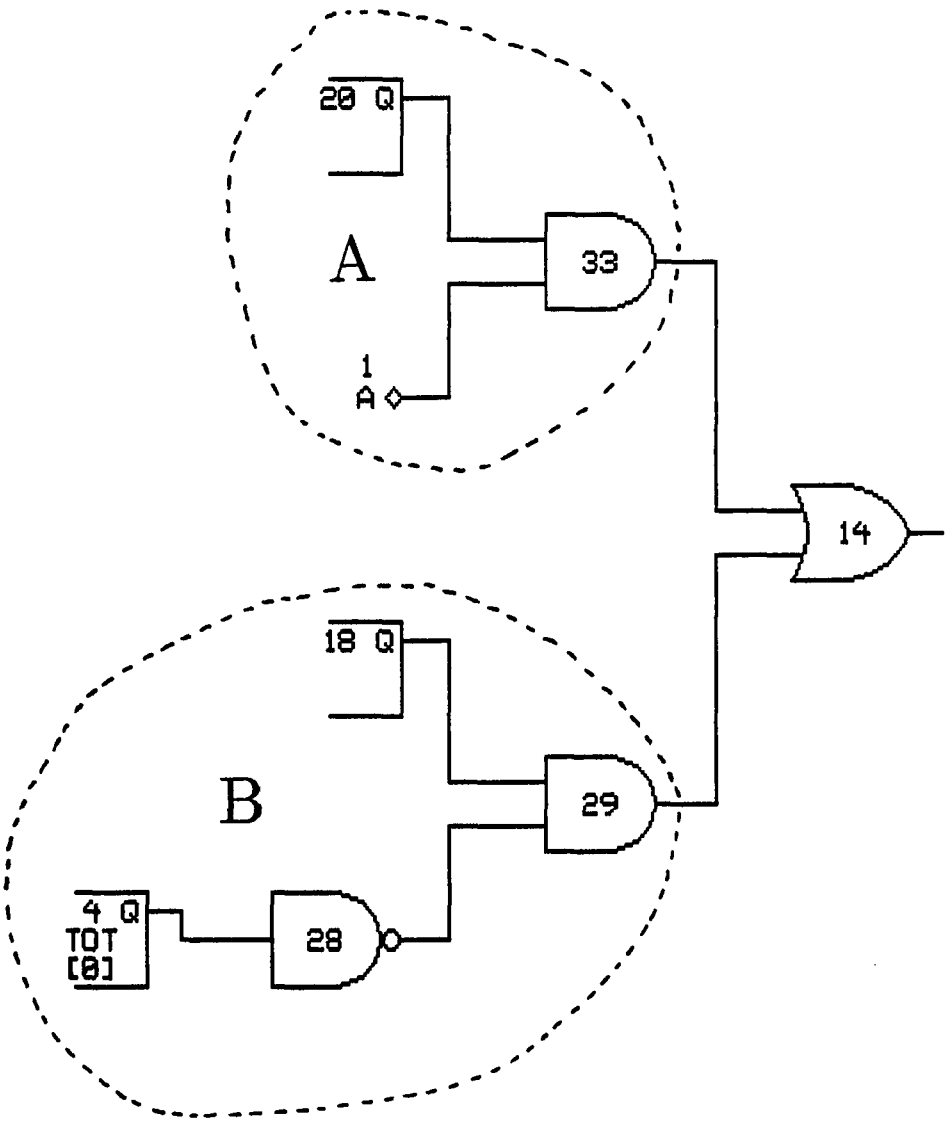


Figure 5.1. Subgraph Examples.

5.3.3. Print

The print option requires that a subgraph has already been selected. The subgraph does not need to be viewed before it is printed. Because the print data structures were discussed in chapter 4, no further comments are deemed necessary.

5.3.4. View

The view option requires that a subgraph has already been selected and that a recognizeable graphics adapter is present. If both conditions are met the same procedures as the print option are followed except the output is bit-mapped to the screen.

When a subgraph is viewed, the UP, DOWN, LEFT and RIGHT keys are used to move throughout the display, if the circuit can not be displayed on one screen. A border is placed along the edges of the subgraph so the perimeter of the circuit can be identified. The ESC key is used to return to the subgraph manipulation menu.

The bit-mapped graphics were designed to look "normal" on an IBM Proprinter which has a horizontal resolution of 60 dots per inch and a vertical resolution of 72 dots per inch. The display looks proportionally identical to the IBM Proprinter print-out on a 640 $\times$ 340 resolution EGA monitor.

Video display utilizes a one-to-one mapping of the print display. Therefore, the same bit map that is used for the print utility is used for the view utility.

VGA has a higher resolution (640 $\times$ 480) than EGA, so the graphic display looks somewhat small but is still acceptable. Future higher resolution displays would require higher-resolution bit-mapped logic element representations for acceptable display.

A notable characteristic of VGA use is that the vertical resolution (480 pixels) is identical to the horizontal width (480 dots) of the printer. Therefore, the complete span of the page may be seen on a VGA monitor. As a result, only the LEFT and RIGHT keys for screen movement. MCGA resolution in monochrome mode is identical to VGA.

Hercules Graphics Card (HGC) adapters produce a resolution of (748×340). The logic elements appear slightly compressed in the horizontal direction but are recognizeable. A Hercules card has excellent graphics capability but is limited to monochrome display.

Color graphic adapters (CGA) operate in either low resolution (320×200) or high resolution (640×200) graphic mode. CGA is the least desireable of the graphics adapter especially for the low resolution version. To traverse the full 480 pixel vertical width, five screens of display must be used.

One of the strengths of SUBGRAPH is that any of these adapters may be used. Some variant adapters are not recognized by the Microsoft C drivers. As was mentioned before, Tandy CGA is a mix of regular CGA and EGA. Although it has good resolution, it is not recognized by the C graphics routines, and is therefore not useable.

Chapter 6. SUBGRAPH Example.

The first five chapters have covered the philosophy, development, and internal structure of SUBGRAPH. All the descriptive pieces from these chapters will be put together in the form of an in-depth example covering the complete execution sequence.

6.1. HPCOM Execution.

The first step in the process is to provide HPCOM with an AHPL description. In actual practice, the AHPL source file would probably have been debugged and tested using the AHPL simulator before invoking HPCOM.

All the information of importance from HPCOM is placed in the output files. No run-time messages of consequence are issued. Thus only commands and prompts for both stages are shown in Figure 6.1. The AHPL source file which was used may be found in Appendix A. The circuit RANDPRO is a random process controller, but the description's function makes no difference for the examples.

```
C:\>stage01
 ENTER AHPL SEQUENCE INPUT FILE NAME....:rnd.scr
 ENTER THE SEQUENCE OUTPUT FILE NAME....:rnd.sq1
 ENTER STAGE1 OUTPUT FILE (STAGE2 INPUT):rnd.st1
 RESPOND (Y) TO DUMP THE TABLES:y
 ENTER A FILE NAME FOR STAGE1 TABLE DUMP:rnd.dmp
Stop - Program terminated.

C:\>stage23
INPUT  FILE NAME = rnd.st1
OUTPUT FILE NAME = rnd.out
INPUT FILE RESTORE!
START MODULE  1 HARDWARE GENERATION
START MODULE  1 OPTIMIZATION
START MODULE  1 OUTPUT
GATE LIST FILE (STAGE3 INPUT) = (TYPE CR TO IGNORE) rnd.glf
Stop - Program terminated.
```

Figure 6.1. HPCOM Compilation Sequence.

6.2. SUBGRAPH Load Execution.

SUBGRAPH is executed by entering its name at the DOS prompt. The main menu is displayed which consists of three options. The options within each menu are generally ordered beginning with the most important or most frequently-used. Because a gate list must be loaded before anything else can be done, it appears first.

SUBGRAPH was invoked from the TEST subdirectory on the C drive. Figure 6.2 shows the load screen after a gate list has been loaded. The current working directory is featured at the top of the screen.

All the files in the current working directory that have the .GLF extension are displayed. Once again, any file name may be entered at the file prompt. The Microsoft C routine which queries DOS for file names apparently receives the names in the order in which they appear in the file access table (FAT). Therefore, no alphabetical or other ordering of the file names occurs.

The last point of interest is that the actual number of gate elements that are in the file is presented at the bottom of the screen. As will be evidenced in the next section, the gate numbers are not sequential.

6.3. SUBGRAPH Gate Information.

Once a gate has been loaded, information about all or individual logic elements may be found by accessing the third option of the main menu. The gate information feature is intended to eliminate or diminish the user's dependence on the HPCOM user output file.

The initial gate information screen shows the options on the left side of the screen. The loaded file name is exhibited in the single border box in the lower left corner. The gate display portion of the screen is initially blank.

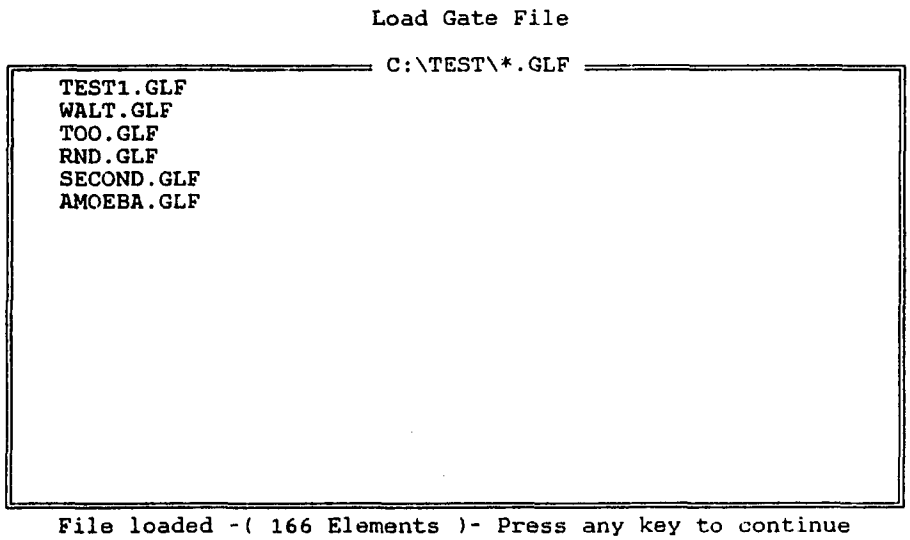


Figure 6.2. Load Screen

The five screens of Figure 6.3 show the complete gate list (166 elements) which were produced by selecting "All" from the menu. The menu options, for the most part, are self-explanatory. All logic elements of a specified type may be exhibited by selecting the appropriate option. Figure 6.4 contains the list of all EOR gates found in the network list.

The only option which requires some explanation is the "find gate" option. As can be seen from Figures 6.3 and 6.4, each logic element has an associated gate number which is actually its index into the element table. When the "find gate" option is provided with a gate number, all stored information about the element is presented.

Figure 6.5 shows the response for gate number 30. The specified logic element is a D flip-flop which stores bit 7 of the symbol NUM. Because it is a flip-flop, there are five inputs. The D input comes from gate 66. The clock is controlled by gate

Gate Information

Gate #	Type	In	Out	Gate #	Type	In	Out
1	Input	0	3	20	Dff	5	4
2	Input	0	3	21	Dff	5	4
3	Input	0	3	22	Dff	5	4
4	Input	0	26	23	Dff	5	4
5	Output	2	0	24	Dff	5	5
6	Output	1	0	25	Dff	5	5
7	Output	1	0	26	Dff	5	5
8	Output	1	0	27	Dff	5	5
9	Output	1	0	28	Dff	5	5
10	Output	1	0	29	Dff	5	5
11	Output	1	0	30	Dff	5	5
12	Output	1	0	31	Dff	5	4
13	Output	1	0	32	Dff	5	2
14	Output	1	0	33	Dff	5	2
15	Dff	5	2	34	Dff	5	2
16	Dff	5	4	35	Dff	5	2
17	Dff	5	4	36	Or	2	1
18	Dff	5	4	37	Or	2	8
19	Dff	5	4	38	Or	2	1

Quit
All
Find Gate
And
CS-And
Nand
Or
Xor
Nor
Dff
CS-Dff
Input
Output

rnd.glf

All

MORE -- Press any key to continue

Gate Information

Gate #	Type	In	Out	Gate #	Type	In	Out
40	Or	2	1	80	CS-Dff	5	3
42	Or	2	1	82	CS-Dff	5	3
44	Or	2	1	84	CS-Dff	5	8
46	Or	2	1	85	Or	3	1
48	Or	2	1	86	CS-Dff	5	9
50	Or	2	1	88	Nand	1	2
52	Or	2	1	89	CS-And	2	1
53	Or	2	8	90	Nand	1	1
54	Or	2	1	91	CS-And	2	1
56	Or	2	1	127	Xor	2	1
58	Or	2	1	128	Xor	2	1
60	Or	2	1	129	Xor	2	1
62	Or	2	1	130	Xor	2	1
64	Or	2	1	131	And	2	2
66	Or	2	1	132	And	2	2
69	Or	3	4	133	And	2	1
77	Or	2	1	134	CS-And	2	1
78	CS-Dff	5	2	135	CS-And	2	1
79	Or	2	1	136	CS-And	2	1

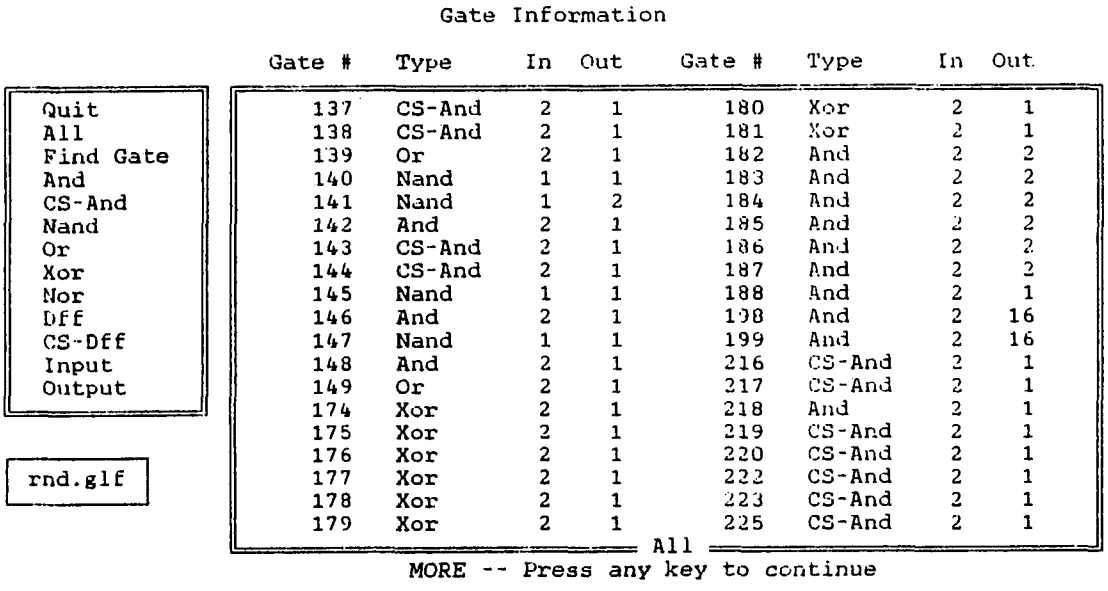
Quit
All
Find Gate
And
CS-And
Nand
Or
Xor
Nor
Dff
CS-Dff
Input
Output

rnd.glf

All

MORE -- Press any key to continue

Figure 6.3. Gate Information Example



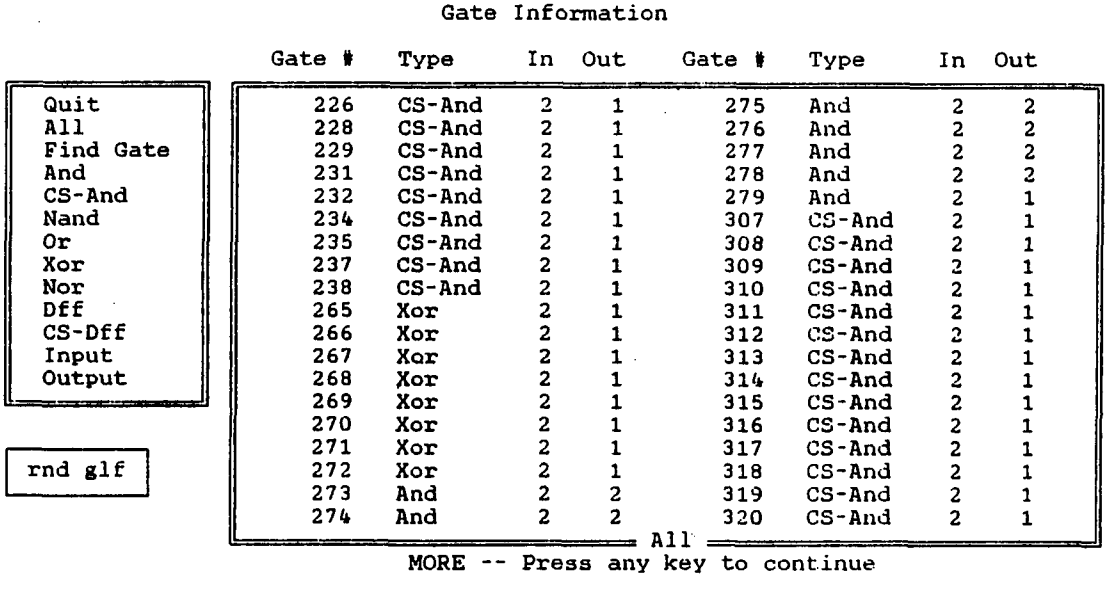


Figure 6.3. Continued

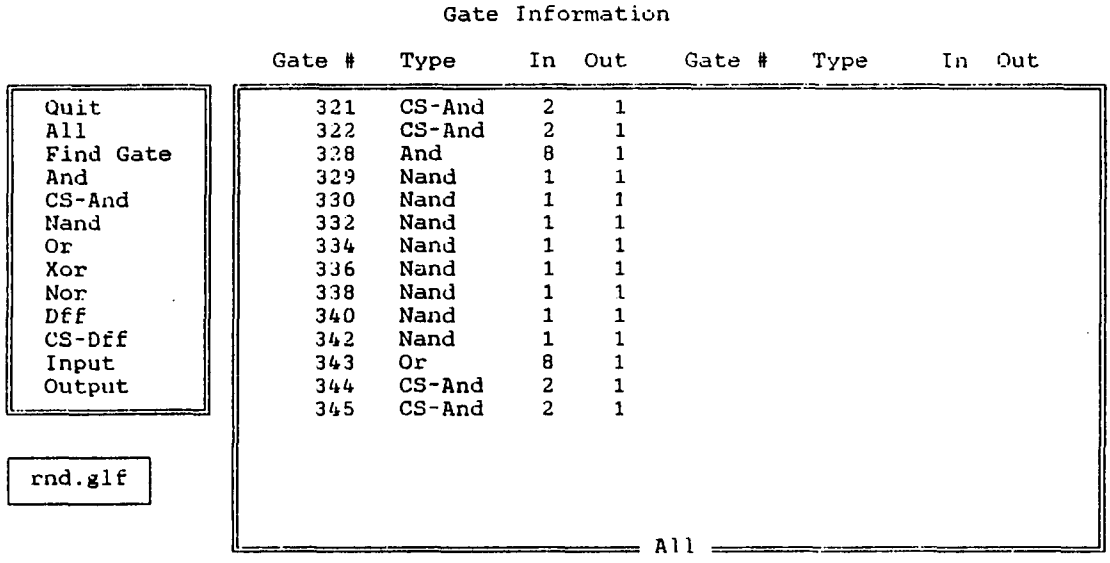


Figure 6.3. Continued

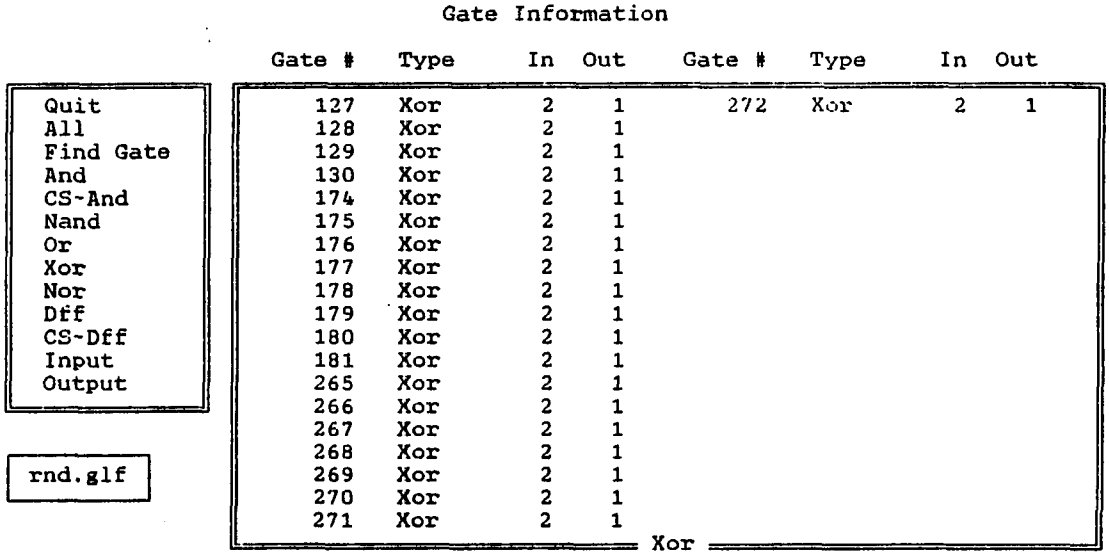


Figure 6.4. Exclusive-OR Gate Information Example

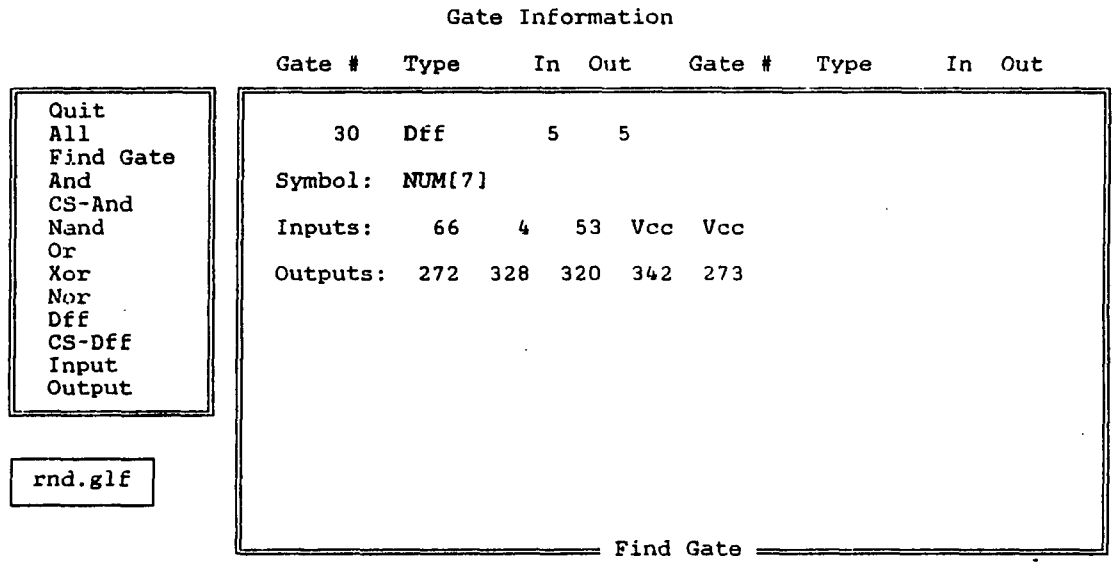


Figure 6.5. Find Gate Example

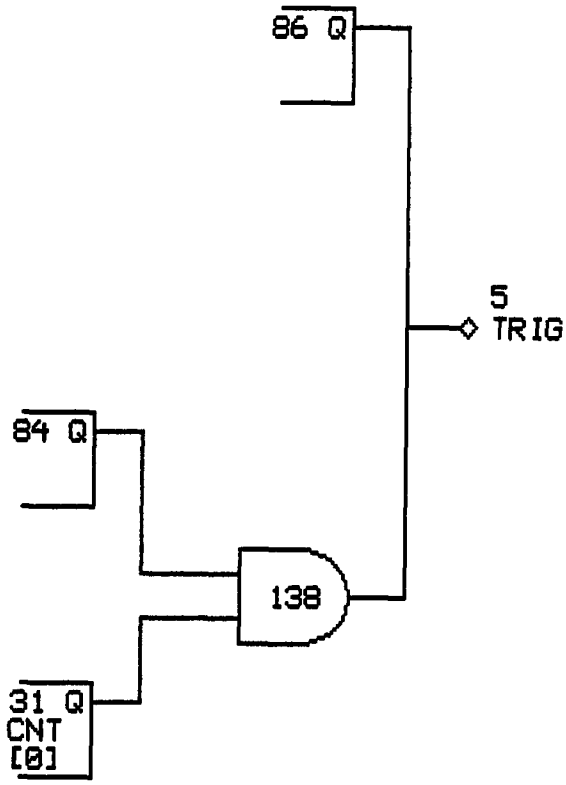


Figure 6.6. Output Example

4 and the enable comes from gate 53. Both the set and reset lines are wired high. The flip-flop output fans out to five gates whose numbers are shown in the figure.

6.4. Subgraph Manipulation.

The important feature of SUBGRAPH is the actual display of a subgraph. The three options under the main menu subgraph manipulation option allow for the selection, viewing, and printing of a subgraph. The selection feature merely queries for a gate number and then internally organizes the subgraph. The view features behaves identically to the print feature except that output is sent to the screen. The print feature is the only option of the three that produces captureable output, so its results will be reproduced here.

Several examples are given in this section to illustrate cases of general and particular interest. The number of special cases that can occur is too large for reasonable examination of all possible situations. For 166 logic elements with three cases for each flip-flop (one subgraph for each D input, enable input, and both) the number of unique subgraphs is over 200.

For fault-propagation analysis, outputs are of particular interest. The logic elements which can propagate a fault to a primary output during a given clock cycle are assigned to the subgraph of the output. Figure 6.6 shows the subgraph for the output TRIG (gate 5). If a fault is detected at TRIG, then the fault is either stored in one of the flip-flops or occurs in one of the logic gates.

The flips numbered 84 and 86 are control flip-flops. All data flip-flops are labeled, as is 31, with their associated symbol and bit position. The other outputs in the circuit are simple wires taken directly from a flip-flop and thus of no great interest.

The majority of the possible subgraphs consisted of a low number of logic elements. The typical subgraph had from five to ten logic elements. The subgraphs

of elements in the combinational logic unit were composed of considerably more gates due to the nature of the boolean computations.

Figure 6.7 shows the subgraph of an EOR gate from the combinational logic unit. The AND gate numbered 131 actually has two inputs, one of which is V_{cc}. Because $a \cdot 1$ is a, it would seem that the HPCOM optimization routines have room for improvement.

The maximum number of inputs which can be displayed for a logic element is eight. One gate did reach the maximum but none surpassed the limit. The subgraph for an eight-input OR gate (gate 343) is shown in Figure 6.8. The use of the previously described layout tables assured that the display maintained a symmetric look.

All of the NAND gates act as inverter gates for the flip-flop outputs. A $\bar{Q}$ output such as in a SN74175 D flip-flop would have eliminated the need for the NAND gates. HPCOM does not produce inverter gates.

Selecting a memory element as the root of a subgraph requires choosing either the D input, enable input, or both as the source for trace-back. For fault-propagation purposes, examining the D inputs shows the possible sources from which a fault might be received and stored. A fault propagated to the enable input would cause improper behavior of the flip-flop and therefore invalidate the stored value.

The D flip-flop (gate 32) in Figure 6.9 was selected with the D input as the subgraph root. A number of the gates in this figure are shared with gate 127 of Figure 6.7. Most gates have multiple fan-out, but no indication of this can be detected from the subgraph picture. The gate information feature or HPCOM output must be consulted for such information.

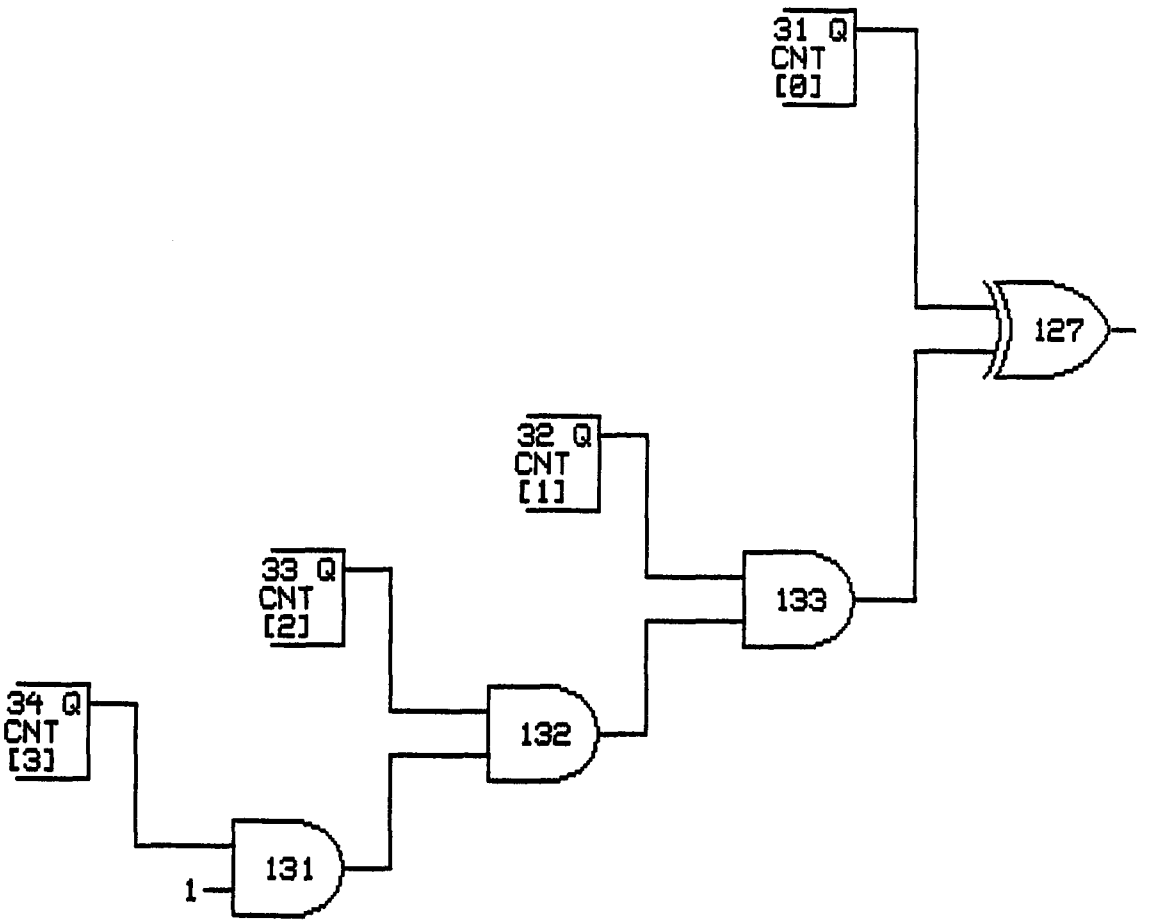


Figure 6.7. Exclusive-OR Example

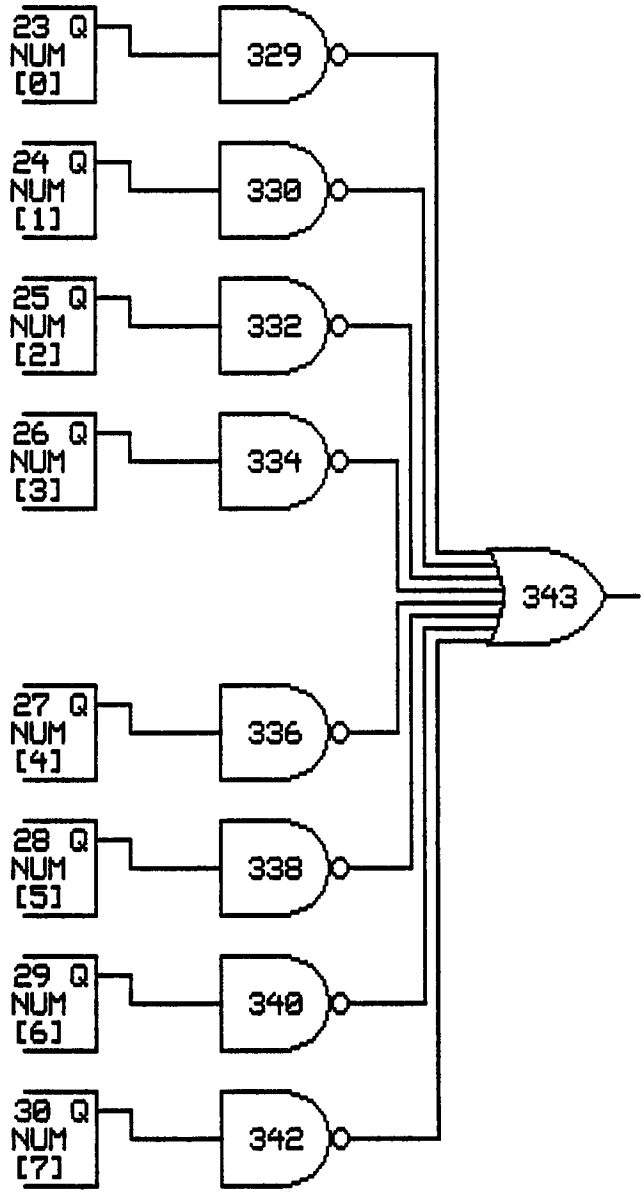


Figure 6.8. Eight-Input Example

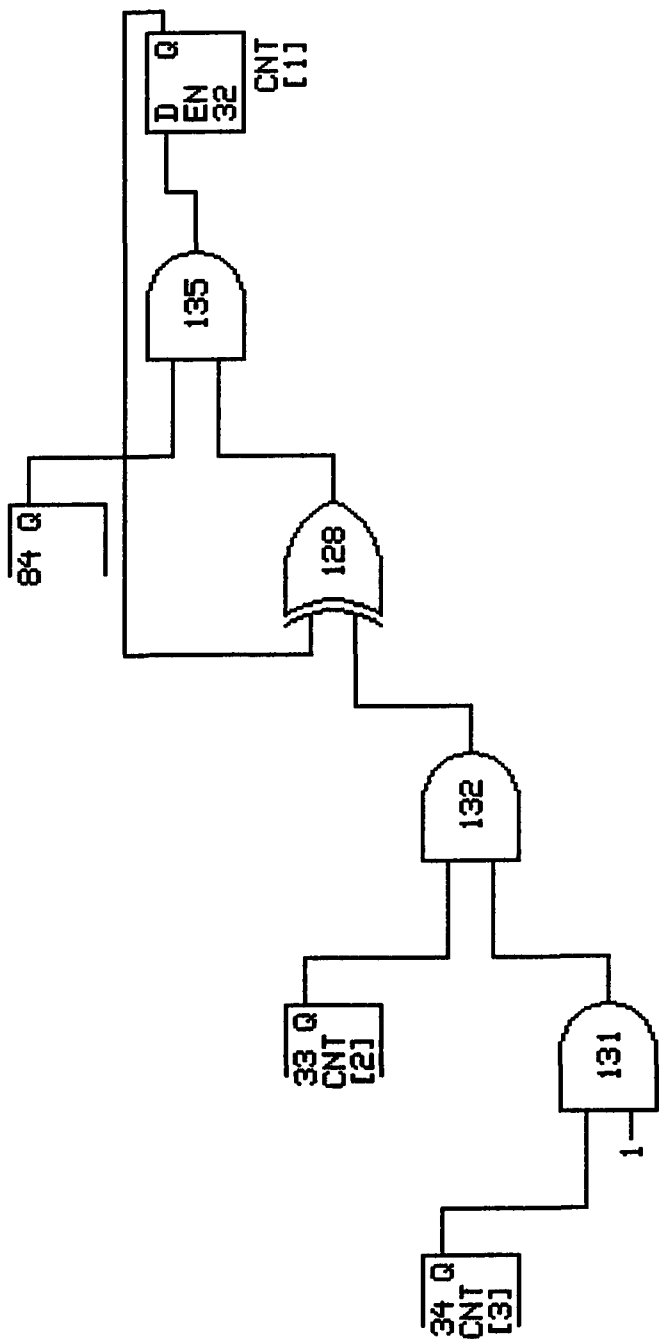


Figure 6.9. D Input Example

The concluding examples in Figure 6.10 and Figure 6.11 demonstrate selection of the enable input and both inputs for subgraph generation. No other characteristics of interest are exhibited in these figures.

Dozens of more subgraphs could be included, but these should be sufficient for an overview of the program's capability. If the subgraph consisted of perhaps a hundred elements, the gate layout would probably appear more cluttered. No extremely large subgraphs were found for the RANDPRO AHPL description.

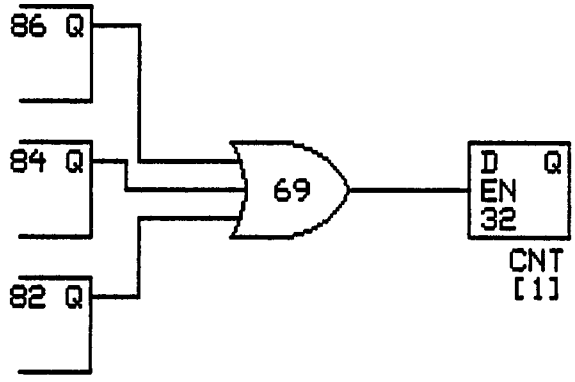


Figure 6.10. Enable Input Example

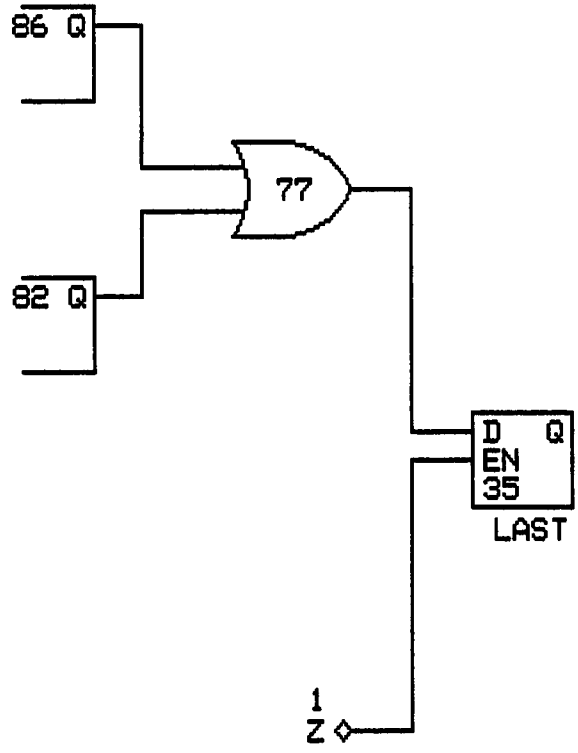


Figure 6.11. D and Enable Input Example

Chapter 7. Conclusion.

7.1. Achievements.

The end result of this research is the expansion of the AHPL design environment to more extensively cover MS-DOS microcomputers. HPCOM was successfully converted and should offer all the features of the VAX version.

SUBGRAPH was developed to aid test generation development through display of selected subgraphs. The hardware requirements for microcomputer use were minimized to the extent that almost any available MS-DOS microcomputers with an appropriate dot-matrix printer should be compatible.

Chapter 6 sufficiently exhibits the capabilities of SUBGRAPH. Any logic gate network that is generated by HPCOM may be loaded and manipulated. Subgraphs may consist of any number of logic elements which are organized for display in a logical, orderly arrangement.

7.2. Improvements.

The combined utility of HPCOM and SUBGRAPH offers valuable analysis and design information. However, as with all software, there remains room for improvement. The basic suggestions for HPCOM were presented at the end of chapter 2, so this section will deal solely with SUBGRAPH.

One of the strengths of SUBGRAPH is that all memory size limitations are set by the available hardware configuration. Prior to actual software implementation and subsequent experimentation, the author assumed that there should be a reasonable balance between program speed and memory use. However, after implementation it was found that, even with major enhancements, the program size would probably not exceed 100K. HPCOM limits the number of gates which can

be produced to roughly 1600, which would require at most 50K for full element and symbol description storage.

Thus memory conservation is a mute issue. Major emphasis should then be placed on execution speed. Printing is severely limited by the speed of the printer so little can be done to accelerate the print speed. Noticeable gains can be achieved with respect to viewing the selected subgraph on the display screen.

Two points within the design appear as the most likely bottlenecks. The size of the print buffer and the size of each field within the print buffer might be modified for better results.

The current print buffer requires about 6K of memory and holds one row of logic gates. The print buffer is filled and displayed as needed to fill the available screen space. Because the buffer holds only one row of gates, each time the cursor is repositioned the entire screen must be regenerated. If the print buffer was enlarged to hold perhaps 12 rows (i.e. a 72K buffer), recreation of the graphic display could be minimized or eliminated in many cases.

The widest display for the RANDPRO example was eight columns. Except for extremely complex cases, most subgraphs could fit in the prescribed buffer. An initial increase in creation time would occur but the subsequent savings would be significant.

There is a one-to-one bit mapping between the print buffer and the screen. Therefore, each bit in the print buffer must be checked. If the bit is set, then the corresponding screen pixel is turned on.

The percentage of "on" pixels for the screen in a typical subgraph is low (e.g. 2%). The view algorithm was accelerated by ignoring zero-valued bytes in the print buffer. Compared to checking each bit in each byte, this improvement provided about a five-fold increase in speed.

As a further extension, if the data type size of the buffer was increased from "char" to "integer" or "double", the rate of pixel checks would increase. In other words, because most of the values in the buffer are zero, more zero-value bits could be ignored with each check by increasing the data type size. Hence view time could be cut significantly.

For the most part, SUBGRAPH was implemented in a fashion which allows for easy modification and enhancement. Hopefully, this foresight will ensure its longevity.

Appendix A. AHPL Source File Listing

```
MODULE:RANDPRO.
        EXINPUTS:Z;START;TEST;CLOCK.
        EXOUTPUTS:TRIG;AVEOUT[4];FRACT[4];TOUT.
        MEMORY:TOT[8];NUM[8];CNT[4];LAST.
        CLUNITS:INC1[4]<:INCTST{4}.
        CLUNITS:INC2[8]<:INCTST{8}.
        PULSES:CLOCK.
BODY SEQUENCE:CLOCK.
1       =>(^START)/(1).

2       TOT <= 8$0; NUM <= 8$0.

3       CNT <= 4$0; LAST<=Z.

4       CNT<=INC1(CNT); TRIG=CNT[0];

        =>(CNT[0]+TEST,^CNT[0]&^TEST)/(5,4).

5       TOT*(^Z&LAST + Z&^LAST) <= (INC2(TOT)!(TOT[1:7],START))*(^TEST,TEST);
        LAST<=Z; NUM<= (INC2(NUM)!(NUM[1:7],START))*(^TEST,TEST);
        TRIG=\1\; CNT <= 4$0;
   =>(&/NUM,
    (^NUM[0]+^NUM[1]+^NUM[2]+^NUM[3]+^NUM[4]+^NUM[5]+^NUM[6]+^NUM[7]))/(1,4).

ENDSEQUENCE
CONTROLRESET(1); AVEOUT=TOT[0:3]; FRACT=TOT[4:7]; TOUT=NUM[0].
END.

CLU:INCTST(OP) (I).
INPUTS:OP[I].
OUTPUTS:TERMOUT[I].
CTERMS:TA[I].
BODY
        FOR J = 0 TO I-1 CONSTRUCT
                TERMOUT[J] = OP[J]@ TA[J]
        ROF;
        FOR J = I-1 TO 0 CONSTRUCT
                IF J=I-1 THEN
                        TA[J]=\1\
                ELSE
                        TA[J]=OP[J+1]&TA[J+1]
                FI
        ROF.

END.
```

Appendix B. SUBGRAPH Source Listing

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

        Programmer:     Walter Slipp
                        Spring 1989

        Advisor:        Dr. Fredrick J. Hill

        Thesis Project

        Graphics display package for AHPL circuits


        Source code was compiled using  Microsoft C  Version 5.10

        This program has been tested on the following machines:

        Zenith Z-386    (80386, 16MHz, EGA)
        PS/2 25         (8086, 8MHz, MCGA)
        IBM AT          (80286, 6MHz, EGA)
        PS/2 70         (80386, 16MHz, VGA)

*************************************************************************/

#include <graph.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <bios.h>

#include "gfx.h"

extern keyhit();
extern load();
extern demo();
extern adpt_ck();
extern display();
extern p_tables();
extern box();
extern kb_readln();
```

```
int main(void);
int init(void);
int options(void);


/*****************************************************************/
/*                  SUBGRAPH Display Program                     */
/*****************************************************************/

main()
{
  init();
  options();
  exit(0);
}


/*-----  init  ----------------------------------------*/
/*  Initializes the graphics mode and screen parameters.  */

init()
{
  int i, test;

  mm_acc = 0;

  strcpy(curfile, "None");

  _clearscreen(_GCLEARSCREEN);

  _getvideoconfig(&vc);
  txtmode = vc.mode;
  gphmode = set_mode();

  _getvideoconfig(&vc);
  _setvideomode(_DEFAULTMODE);

  xlim = vc.numxpixels;
  ylim = vc.numypixels;

  if (gphmode == _NOGRAPH)
    {
      center(3, "No recognizeable graphics adaptor was detected.");
      center(4, "The VIEW option can not be used.");
    }

  center( 8, "SUBGRAPH");
  center(10, "Author:  Walter Slipp");
  center(12, "(c) 1989");
```

```
  box(6, 26, 7, 26, DOUBLE);

  cursor(TCURSOROFF);
  press(C, 16, 0);

  _clearscreen(_GCLEARSCREEN);
}


/*-----  set_mode  ------------------------------------------------*/
/*  Determines the highest possible screen resolution which can be  */
/*  used.  */

int set_mode()
{
  if (_setvideomode(_VRES16COLOR))  return(_VRES16COLOR);
  if (_setvideomode(_VRES2COLOR))   return(_VRES2COLOR);
  if (_setvideomode(_ERESCOLOR))    return(_ERESCOLOR);
  if (_setvideomode(_ERESNOCOLOR))  return(_ERESNOCOLOR);
  if (_setvideomode(_HRES16COLOR))  return(_HRES16COLOR);
  if (_setvideomode(_HRESBW))       return(_HRESBW);
  if (_setvideomode(_MRES256COLOR)) return(_MRES256COLOR);
  if (_setvideomode(_MRES16COLOR))  return(_MRES16COLOR);
  if (_setvideomode(_MRES4COLOR))   return(_MRES4COLOR);
  if (_setvideomode(_MRESNOCOLOR))  return(_MRESNOCOLOR);
  if (_setvideomode(_HERCMONO))     return(_HERCMONO);
  return(_NOGRAPH);
}


/*-----  options  -------------------------------------------------*/
/*  Prompts for the main options and executes the appropriate  */
/*  function.  */

options()
{
  int choice;
  int menu();
  static char *mnopt[] = {
        "Exit",
        "Load Gate List",
        "Subgraph Manipulation",
        "Gate Information",
        NULL };


  choice = 0;

  while (TRUE)
```

```
{
  _clearscreen(_GCLEARSCREEN);
  header("Main Menu");
  cursor(TCURSOROFF);
  choice = menu(mnopt, C, 0, 0, choice);

  switch(choice)
    {
      case 0:
                    _clearscreen(_GCLEARSCREEN);
                    exit(0);
                    break;
      case 1:
                    load();
                    break;
      case 2:
                    subgraph();
                    break;
      case 3:
                    gate_info();
                    break;
      default:
                    break;
    }
  }
}
```

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

        gate.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains routines for the gate information option.

*********************************************************************/

#include <dos.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "gfx.h"


/*-----  clear_gates  ------------------------------------------------*/
/*  Clears screen area where gate information appears.  */

clear_gates()
{
  int row;
  char buff[60];

  _settextwindow(5, 19, 23, 75);
  _clearscreen(_GWINDOW);
  _settextwindow(1, 1, 25, 80);
}


/*-----  curgate  ----------------------------------------------------*/
/*  Displays the type of gate currently being viewed.  */

curgate(p)
char *p;
{
  int col;
  char buff[42];

  memset(buff, 205, 41);
  buff[40] = 0;
  _settextposition(24, 30);
  _outtext(buff);
```

```
  col = ((60 - strlen(p)) / 2) + 19;
  _settextposition(24, col);
  _outtext(" ");
  _outtext(p);
  _outtext(" ");
}


/*-----  find_gate  ------------------------------------------------*/
/*  Looks for a specific gate and displays information.  */

int find_gate()
{
  int num, n, x, i;
  int r, c;
  char buff[10];
  gatelst *p;

  _settextposition(6, 21);
  _outtext("Enter gate number: ");
  if (!kb_readln(buff, 8))
    {
      _settextposition(6, 21);
      _outtext("                          ");
      return(0);
    }
  num = atoi(buff);
  x = 6;

  if (num < 1 || num > numgate)
    {
      _settextposition(6, 21);
      _outtext("Gate number out of range  ");
      return(0);
    }

  p = gateptr + num;
  _settextposition(6, 21);
  if (p->type > 0)
    {
      printf("%6d   ", num);
      printf("%-6s ", lgcsym[p->type]);
      printf("%4d %4d", p->numin, p->numout);

      if (p->sym > 1)
        {
          x = x + 2;
          _settextposition(x, 21);
          printf("Symbol:  %s", (symptr+p->sym)->symname);
          n = rowcol(num, &r, &c);
```

```
        if (n == 2) printf("<%d>", r);
        if (n) printf("[%d]", c);
      }

    if (p->numin)
      {
        x = x + 2;
       _settextposition(x, 21);
       _outtext("Inputs:  ");
       for (i=0; i < p->numin; i++)
         {
           if (i % 8 == 0 && i != 0) { x++; _settextposition(x, 30); }
           n = *(p->cin + i);
           switch (n)
             {
               case GND:
                         printf(" GND ");
                         break;
               case VCC:
                         printf(" Vcc ");
                         break;
               case FLT:
                         printf(" FLT ");
                         break;
               default:
                         printf("%4d ", n);
                         break;
             }
         }
      }

    if (p->numout)
      {
        x = x + 2;
        _settextposition(x, 21);
        _outtext("Outputs: ");
        for (i=0; i < p->numout; i++)
          {
            if (i % 8 == 0 && i != 0) { x++; _settextposition(x, 30); }
            printf("%4d ",*(p->cout + i));
          }
      }
    return(num);
  }
else
  {
    printf("Gate %d not found            ", num);
    return(0);
  }
}
```

```
/*-----  gate_info  ----------------------------------------------*/
/*  Handles gate information.  */

gate_info()
{
  int choice;
  int row, col;
  int i, none;
  gatelst *p;

  static char *mnginfo[] = {
        "Quit",
        "All",
        "Find Gate",
        "And",
        "CS-And",
        "Nand",
        "Or",
        "Xor",
        "Nor",
        "Dff",
        "CS-Dff",
        "Input",
        "Output",
        NULL};

  static int ginfovl[] = {
        GQUIT,
        ALL,
        FNDGATE,
        AND,
        CND,
        NAND,
        OR,
        XOR,
        NOR,
        DFF,
        DFCS,
        INPUT,
        OUTPUT,
        NULL};


  _clearscreen(_GCLEARSCREEN);

  header("Gate Information");
  _settextposition(3, 21);
  printf("Gate #   Type     In  Out    Gate #   Type     In  Out");
```

```
box(4, 18, 19, 60, DOUBLE);
curgate("None");

box(20, 2, 1, strlen(curfile)+2, SINGLE);
_settextposition(21, 4);
_outtext(curfile);

choice = 0;
for (;;)
  {
    choice = menu(mnginfo, PICK, 4, 2, choice);
    curgate(mnginfo[choice]);
    clear_gates();
    none = 1;

    switch (ginfovl[choice])
      {
        case GQUIT:
                    _clearscreen(_GCLEARSCREEN);
                    return;
        case FNDGATE:
                    find_gate();
                    break;
        default:
              row = 5;
              col = 21;
              for (i=1; i<=numgate; i++)
                {
                  p = gateptr + i;
                  if (p->type > 0)
                  if (p->type == ginfovl[choice] || ginfovl[choice] == ALL)
                    {
                      none = 0;
                      if (row == 24)
                        {
                          row = 5;
                          if (col == 50)
                            {
                              pause(25, 30);
                              col = 21;
                              clear_gates();
                            }
                          else
                            col = 50;
                        }
                      _settextposition(row++, col);
                      printf("%6d   ", i);
                      printf("%-6s ", lgcsym[p->type]);
                      printf("%4d %4d", p->numin, p->numout);
                    }
```

```
            }
          if (none)
            {
              _settextposition(5,22);
              _outtext("** none **");
            }
          break;
      }
    }
}


/*-----  pause  ------------------------------------------------*/
/*  Waits for user response if screen becomes full.  */

pause(r, c)
short r, c;
{
  int tcolor, bcolor;

  bcolor = _getbkcolor();
  tcolor = _gettextcolor();

  _settextposition(r, c);
  _settextcolor(bcolor);
  _setbkcolor( (long)tcolor);
  _outtext(" MORE -- Press any key to continue ");
  _settextcolor(tcolor);
  _setbkcolor( (long)bcolor);
  while (!kbhit());
  do { getch(); }
    while (kbhit());
  _settextposition(r, c);
  _outtext("                                  ");
}
```

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

        global.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains global variables

*************************************************************************/

#include        <graph.h>
#include        <stdio.h>
#include        <stdlib.h>
#include        "gfx.h"


/*  location table variables  */

int     xlim;           /* x pixel maximum */
int     ylim;           /* y pixel maximum */

int     xlevel;         /* current print column */
int     xmap;           /* maximum print column */
int     *locptr;        /* pointer to location table */
int     xlast;          /* last view x coordinate */
int     ylast;          /* last view y coordinate */


/*  table pointers and counters  */

symtab  *symptr;        /* pointer to symbol table */
int     numsym;         /* number of symbols */
sdt     *sdtptr;        /* pointer to symbol description table */
int     numsdt;         /* number of symbol descriptions */
sqrtb   *sqrtptr;       /* pointer to SQRT table */
int     numsqrt;        /* number of SQRT entries */
gatelst *gateptr;       /* pointer to gate list */
int     numgate;        /* number of gate list entries */
int     numread;        /* number of actual gate list entries */


/*  print buffer  */

char    p_buffer[RBUFSIZE+1][CBUFSIZE];


/*  Others  */
```

```
char    curfile[14];    /*  active file name  */

int mm_acc;             /* number of times main menu is accessed */

int frow;               /* first column from left with gates */
int scr_lim;            /* number of print buffers per screen */

int din;                /* DFF input selected */

struct videoconfig vc;  /* video configuration */

int gphmode;            /* best mode for graphics */
int txtmode;            /* best mode for text */

char *lgcsym[] = {
        "Pass",
        "And",
        "CS-And",
        "Nand",
        "Or",
        "Xor",
        "Nor",
        "Dff",
        "CS-Dff",
        "Input",
        "Output" };
```

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

        Graphic Aids

        Programmer:     Walter Slipp
                        Spring 1989

        Contains bit-map representations for letters, numbers,
        and symbols.  Includes routines for printing out the
        forementioned.

************************************************************/

#include  <graph.h>

#include  "gfx.h"

extern digits();


char p_lett[26][7] = {
        33, 33, 63, 33, 33, 18, 12,  /* A */
        62, 17, 17, 30, 17, 17, 62,  /* B */
        30, 33, 32, 32, 32, 33, 30,  /* C */
        62, 17, 17, 17, 17, 17, 62,  /* D */
        63, 32, 32, 62, 32, 32, 63,  /* E */
        32, 32, 32, 62, 32, 32, 63,  /* F */
        30, 33, 33, 39, 32, 33, 30,  /* G */
        33, 33, 33, 63, 33, 33, 33,  /* H */
        14,  4,  4,  4,  4,  4, 14,  /* I */
        28, 34,  2,  2,  2,  2, 15,  /* J */
        33, 34, 36, 56, 36, 34, 33,  /* K */
        63, 32, 32, 32, 32, 32, 32,  /* L */
        33, 33, 33, 33, 45, 51, 33,  /* M */
        33, 33, 35, 37, 41, 49, 33,  /* N */
        30, 33, 33, 33, 33, 33, 30,  /* O */
        32, 32, 32, 62, 33, 33, 62,  /* P */
        29, 34, 37, 33, 33, 33, 30,  /* Q */
        33, 34, 36, 62, 33, 33, 62,  /* R */
        30, 33,  1, 30, 32, 33, 30,  /* S */
         4,  4,  4,  4,  4,  4, 31,  /* T */
        30, 33, 33, 33, 33, 33, 33,  /* U */
        12, 12, 18, 18, 33, 33, 33,  /* V */
        33, 51, 45, 45, 33, 33, 33,  /* W */
        33, 33, 18, 12, 18, 33, 33,  /* X */
         4,  4,  4,  4, 10, 17, 17,  /* Y */
        63, 32, 16,  8,  4,  2, 63   /* Z */
};
```

```
char p_numb[10][7] = {
        30, 33, 49, 45, 35, 33, 30,  /* 0 */
        14,  4,  4,  4,  4, 12,  4,  /* 1 */
        63, 32, 32, 30,  1, 33, 30,  /* 2 */
        30, 33,  1, 14,  1, 33, 30,  /* 3 */
         2,  2,  2, 63, 34, 18,  8,  /* 4 */
        30, 33,  1,  1, 62, 32, 63,  /* 5 */
        30, 33, 33, 62, 32, 16, 12,  /* 6 */
         8,  8,  8,  4,  2, 33, 63,  /* 7 */
        30, 33, 33, 30, 33, 33, 30,  /* 8 */
        12,  2,  1, 31, 33, 33, 30   /* 9 */
};


char p_symb[5][7] = {
         0, 18, 12, 63, 12, 18,  0,  /* * */
         2,  4,  8, 16,  8,  4,  2,  /* < */
        16,  8,  4,  2,  4,  8, 16,  /* > */
        14,  8,  8,  8,  8,  8, 14,  /* [ */
        28,  4,  4,  4,  4,  4, 28   /* ] */
};


/*-----  cp_number  ------------------------------------------*/
/*  Prints a number centered around (x, y).  */

cp_number(n, x, y)
unsigned int n, x, y;
{
  short int digit, i;

  if (n < 10000 && n > 999) y -= 4;
  if (n <  1000 && n >  99) y -= 7;
  if (n <   100 && n >   9) y -= 11;
  if (n <    10)            y -= 14;

  for (i=0; i<5; i++)
    {
      digit = n % 10;
      n /= 10;
      p_number(digit, x, y);
      if (n == 0) break;
      y -= 7;
    }
}


/*-----  rp_number  ------------------------------------------*/
/*  Prints a number right-justified.  */
```

```
rp_number(n, x, y)
unsigned int n, x, y;
{
  short int digit, i, j, k;

  if (n == VCC)
    {
      p_vcc(x, y);
      return;
    }

  if (n == GND)
    {
      p_gnd(x, y);
      return;
    }

  if (n == FLT)
    {
      p_flt(x, y);
      return;
    }

  for (k=0; k<5; k++)
    {
      digit = n % 10;
      n /= 10;
      p_number(digit, x, y);
      if (n == 0) break;
      y -= 7;
    }
}


/*-----  lp_number  ------------------------------------------*/
/*  Prints a number left-justified.  */

lp_number(n, x, y)
unsigned int n, x, y;
{
  short int digit, i, j, k;

  if (n == VCC)
    {
      p_vcc(x, y);
      return;
    }

  if (n == GND)
    {
```

```
      p_gnd(x, y);
      return;
    }

  if (n == FLT)
    {
      p_flt(x, y);
      return;
    }

  if (            n > 9999) y += 28;
  if (n < 10000 && n >  999) y += 21;
  if (n <  1000 && n >   99) y += 14;
  if (n <   100 && n >    9) y +=  7;

  for (k=0; k<5; k++)
    {
      digit = n % 10;
      n /= 10;
      p_number(digit, x, y);
      if (n == 0) break;
      y -= 7;
    }
}

/*-----  pr_name  ------------------------------------------------*/
/*  Prints out a string right-justified.  */

pr_name(s, x, y)
char *s;
int x, y;
{
  int i, n;

  n = strlen(s);
  y = y - 7 * (n - 1);

  for (i = 0; i < n; i++, s++, y = y + 7)
    p_letter(*s, x, y);
}


/*-----  pl_name  ------------------------------------------------*/
/*  Prints out a string left-justified.  */

pl_name(s, x, y)
char *s;
int x, y;
{
  int i, n;
```

```
  n = strlen(s);

  for (i = 0; i < n; i++, s++, y = y + 7)
    p_letter(*s, x, y);
}



/*-----  p_letter  ------------------------------------------------*/
/*  Places a single character in the print buffer.  */

p_letter(c, x, y)
int x, y;
char c;
{
  int i, n;
  int msk, row;

  if (x < 0 || x > 472) return;
  if (y < 0 || y > RBUFSIZE*8) return;

  if (c < 'A' || c > 'Z') return;
  n = c - 'A';

  msk = y % 8;
  row = y / 8;

  for (i=0; i<7; i++)
    {
      if (msk < 3)
        {
          p_buffer[row][x+i] |= (p_lett[n][i] << (2-msk));
        }
      else
        {
          p_buffer[row][x+i]   |= (p_lett[n][i] >> (msk-2));
          p_buffer[row+1][x+i] |= (p_lett[n][i] << (10-msk));
        }
    }

}


/*-----  p_symbol  ------------------------------------------------*/
/*  Places a single symbol in the print buffer.  */

p_symbol(c, x, y)
int x, y;
char c;
```

```
{
  int i, n, s;
  int msk, row;

  if (x < 0 || x > 472) return;
  if (y < 0 || y > RBUFSIZE*8) return;

  switch (c)
    {
      case '*':
                s = 0;
                break;
      case '<':
                s = 1;
                break;
      case '>':
                s = 2;
                break;
      case '[':
                s = 3;
                break;
      case ']':
                s = 4;
                break;
      default:
                return;
                break;
    }

  msk = y % 8;
  row = y / 8;

  for (i=0; i<7; i++)
    {
      if (msk < 3)
        {
          p_buffer[row][x+i] |= (p_symb[s][i] << (2-msk));
        }
      else
        {
          p_buffer[row][x+i]   |= (p_symb[s][i] >> (msk-2));
          p_buffer[row+1][x+i] |= (p_symb[s][i] << (10-msk));
        }
    }

}


/*-----  p_number  ------------------------------------------*/
/*  Places a single digit in the print buffer.  */
```

```
p_number(n, x, y)
int n, x, y;
{
  int i;
  int msk, row;

  if (x < 0 || x > 472) return;
  if (y < 0 || y > RBUFSIZE*8) return;

  msk = y % 8;
  row = y / 8;

  for (i=0; i<7; i++)
    {
      if (msk < 3)
        {
          p_buffer[row][x+i] |= (p_numb[n][i] << (2-msk));
        }
      else
        {
          p_buffer[row][x+i]   |= (p_numb[n][i] >> (msk-2));
          p_buffer[row+1][x+i] |= (p_numb[n][i] << (10-msk));
        }
    }
}


/*-----  p_gnd  ------------------------------------------------*/
/*  Places the string 'GND' in the print buffer.  */

p_gnd(x, y)
int x, y;
{
  p_letter('G', x, y);
  p_letter('N', x+8, y);
  p_letter('D', x+16, y);
}


/*-----  p_vcc  ------------------------------------------------*/
/*  Places the string 'VCC' in the print buffer.  */

p_vcc(x, y)
int x, y;
{
  p_letter('V', x, y);
  p_letter('C', x+8, y);
  p_letter('C', x+16, y);
}
```

```
/*-----  p_flt  ------------------------------------------------*/
/*  Places the string 'FLT' in the print buffer.  */

p_flt(x, y)
int x, y;
{
  p_letter('F', x, y);
  p_letter('L', x+8, y);
  p_letter('T', x+16, y);
}
```

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

        layout.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains routines to layout a gate list map

*************************************************************************/

#include <dos.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <malloc.h>
#include <bios.h>

#include "gfx.h"

extern header();
extern center();
extern cursor();
extern press();

extern find_gate();
extern print_layout();
extern view_layout();

/*  These are the row positions for elements in the second column.  */
/*  e.g.  the initial gate has three inputs so the three elements in  */
/*  the second column are placed in rows 2, 5, and 8.  */

static char firstpos[8][8] = {
        5, 11, 11, 11, 11, 11, 11, 11,          /* 1 input  */
        3,  7, 11, 11, 11, 11, 11, 11,          /* 2 inputs */
        2,  5,  8, 11, 11, 11, 11, 11,          /* 3 inputs */
        1,  4,  6,  9, 11, 11, 11, 11,          /* 4 inputs */
        1,  3,  5,  7,  9, 11, 11, 11,          /* 5 inputs */
        0,  2,  4,  6,  8, 10, 11, 11,          /* 6 inputs */
        0,  2,  3,  5,  7,  8, 10, 11,          /* 7 inputs */
        1,  2,  3,  4,  6,  7,  8,  9  };       /* 8 inputs */


typedef struct psarray  {
        int     qpos[MAXIN];
};
```

```
#define TRIES   8

static struct psarray gatpos[MAXIN][TRIES] = {
        { { 0, 0, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0},
          { 1, 0, 0, 0, 0, 0, 0, 0},
          { 0, 0, 0, 0, 0, 0, 0, 0},
          { 0, 0, 0, 0, 0, 0, 0, 0},
          { 0, 0, 0, 0, 0, 0, 0, 0},
          { 0, 0, 0, 0, 0, 0, 0, 0},
          { 0, 0, 0, 0, 0, 0, 0, 0} },  /* 1 input */

        { {-1, 1, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0},
          { 0, 1, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0},
          {-1, 0, 0, 0, 0, 0, 0, 0} },  /* 2 inputs */

        { {-1, 0, 1, 0, 0, 0, 0, 0},
          {-2,-1, 0, 0, 0, 0, 0, 0},
          { 0, 1, 2, 0, 0, 0, 0, 0},
          {-1, 0, 1, 0, 0, 0, 0, 0},
          {-1, 0, 1, 0, 0, 0, 0, 0},
          {-1, 0, 1, 0, 0, 0, 0, 0},
          {-1, 0, 1, 0, 0, 0, 0, 0},
          {-1, 0, 1, 0, 0, 0, 0, 0} },  /* 3 inputs */

        { {-2,-1, 0, 1, 0, 0, 0, 0},
          {-1, 0, 1, 2, 0, 0, 0, 0},
          { 0, 1, 2, 3, 0, 0, 0, 0},
          {-3,-2,-1, 0, 0, 0, 0, 0},
          { 0, 1, 2, 3, 0, 0, 0, 0},
          {-3,-2,-1, 0, 0, 0, 0, 0},
          { 0, 1, 2, 3, 0, 0, 0, 0},
          {-3,-2,-1, 0, 0, 0, 0, 0} },  /* 4 inputs */

        { {-2,-1, 0, 1, 2, 0, 0, 0},
          {-3,-2,-1, 0, 1, 0, 0, 0},
          {-1, 0, 1, 2, 3, 0, 0, 0},
          {-4,-3,-2,-1, 0, 0, 0, 0},
          { 0, 1, 2, 3, 4, 0, 0, 0},
          {-4,-3,-2,-1, 0, 0, 0, 0},
          { 0, 1, 2, 3, 4, 0, 0, 0},
          {-5,-4,-3,-2,-1, 0, 0, 0} },  /* 5 inputs */

        { {-2,-1, 0, 1, 2, 3, 0, 0},
```

```
          {-3,-2,-1, 0, 1, 2, 0, 0},
          {-1, 0, 1, 2, 3, 4, 0, 0},
          {-4,-3,-2,-1, 0, 1, 0, 0},
          { 0, 1, 2, 3, 4, 5, 0, 0},
          {-4,-3,-2,-1, 0, 1, 0, 0},
          { 0, 1, 2, 3, 4, 5, 0, 0},
          {-5,-4,-3,-2,-1, 0, 0, 0} },  /* 6 inputs */

        { {-3,-2,-1, 0, 1, 2, 3, 0},
          {-4,-3,-2,-1, 0, 1, 2, 0},
          {-2,-1, 0, 1, 2, 3, 4, 0},
          {-5,-4,-3,-2,-1, 0, 1, 0},
          {-1, 0, 1, 2, 3, 4, 5, 0},
          {-6,-5,-4,-3,-2,-1, 0, 0},
          { 0, 1, 2, 3, 4, 5, 6, 0},
          {-7,-6,-5,-4,-3,-2,-1, 0} },  /* 7 inputs */

        { {-3,-2,-1, 0, 1, 2, 3, 4},
          {-4,-3,-2,-1, 0, 1, 2, 3},
          {-2,-1, 0, 1, 2, 3, 4, 5},
          {-5,-4,-3,-2,-1, 0, 1, 2},
          {-1, 0, 1, 2, 3, 4, 5, 6},
          {-6,-5,-4,-3,-2,-1, 0, 1},
          { 0, 1, 2, 3, 4, 5, 6, 7},
          {-7,-6,-5,-4,-3,-2,-1, 0} }   /* 8 inputs */
};


/*-----  subgraph  ------------------------------------------------*/
/*  Handles selection of the subgraph options.  */

subgraph()
{
  int choice;
  int i;
  gatelst *p;

  static char *dl[] = {
        "Quit",
        "Select Subgraph",
        "View Layout",
        "Print Layout",
        NULL
  };

  _setlogorg(0,0);
  choice = 0;

  for (;;)
    {
```

```
        _clearscreen(_GCLEARSCREEN);
        header("Subgraph Manipulation");
        cursor(TCURSOROFF);
        choice = menu(dl, C, 10, 0, choice);

        switch(choice)
          {
            case 0:
                        return;
                        break;
            case 1:
                        sel_sub();
                        break;
            case 2:
                        view_layout();
                        break;
            case 3:
                        print_layout();
                        break;
            default:
                        break;
          }
    }

}


/*-----  sel_sub  ------------------------------------------------*/
/*  Requests the number of the desired element.  */

sel_sub()
{
  int i, j, n, num;
  gatelst *p, *p2;

  _clearscreen(_GCLEARSCREEN);
  header("Select Subgraph");

  if (!strcmp(curfile, "None"))
    {
      center(10, "No gate list has been loaded.");
      press(C, 14, 0);
      return;
    }

  num = find_gate();

  if (!num)
    {
      press(PICK, 11, 21);
```

```
      return;
    }

  _settextposition(4, 21);
  printf("Gate #   Type     In  Out");

  p = gateptr + num;
  if (p->type > 0)
    {
      make_map();
      for (i=0, p2 = gateptr + 1; i<numgate; i++, p2++)
        {
          p2->placed = FALSE;
          p2->xpos = 0;
          p2->ypos = 0;
        }

      set_pos(num, 0, 5);

      xlevel = 1;
      first_level(p);
      while (next_level(xlevel) && (xlevel < xmap))
        xlevel++;
    }

  press(PICK, 23, 21);
}


/*-----  clr_dquiz  ------------------------------------------*/
/*  Clears text area around the element number prompt.  */

clr_dquiz()
{
  _settextwindow(4, 3, 15, 15);
  _clearscreen(_GWINDOW);
  _settextwindow(1, 1, 25, 80);
}


/*-----  first_level  ----------------------------------------*/
/*  Handles option for the initial element.  */

first_level(p)
gatelst *p;
{
  int i, n, num;

  static char *dffchoice[] = {
        "D input",
```

```
        "Enable",
        "Both",
        NULL
    };

    if (p->type == INPUT) return(0);

    if (p->type == DFF || p->type == DFCS)
      {
        _settextposition(5,3);
        _outtext("Select an");
        _settextposition(6,3);
        _outtext("input choice");

        din = menu(dffchoice, PICK, 7, 3, 0);
        clr_dquiz();
        switch (din)
          {
            case 0:
            default:
                        n = *(p->cin);
                        if (n > 0) set_pos(n, 1, 5);
                        break;
            case 1:
                        n = *(p->cin+2);
                        if (n > 0) set_pos(n, 1, 5);
                        break;
            case 2:
                        n = *(p->cin);
                        if (n > 0) set_pos(n, 1, 3);
                        n = *(p->cin+2);
                        if (n > 0) set_pos(n, 1, 7);
                        break;
          }
      }
    else
      {
        num = (int) min(p->numin, MAXIN);
        for (i=0; i < num; i++)
          {
            n = *(p->cin + i);
            if (n > 0) set_pos(n, 1, firstpos[num-1][i]);
          }
      }
}

/*-----  next_level  ----------------------------------------*/
/*  Handles the next column of logic elements.  */
```

```
int next_level(x)
int x;
{
  int i, n, cnt;
  gatelst *p;

  cnt = 0;
  for (i=0; i<GWIDTH; i++)
    if (n = chk_loc(x, i))
      {
        cnt++;
        p = gateptr + n;
        if (p->type != DFF   &&
            p->type != DFCS  &&
            p->type != INPUT  ) seek(n);
      }
  return(cnt);
}


/*-----  look  ------------------------------------------------*/
/*  Checks for space for the next input group.  */

int look(num, px, py)
int num, px, py;
{
  int i, j, flag, yc;

  for (i=0; i<TRIES; i++)
    {
      flag = 1;
      for (j=0; j<num; j++)
        {
          yc = py + gatpos[num-1][i].gpos[j];
          if (chk_loc(px, yc)) flag = 0;
        }
      if (flag) return(i+1);
    }
  return(0);
}


/*-----  seek  ------------------------------------------------*/
/*  Handles placement of next input group.  */

seek(num)
int num;
{
  gatelst *p;
  int i, j, n;
```

```
  int py;

  p = gateptr + num;
  n = min(MAXIN, p->numin);
  if (n == 0) return;

  for (i = p->xpos+1, py = p->ypos; i < xmap; i++)
    if (j = look(n, i, py))
      {
        if (n == 2 && j ==1 && chk_loc(p->xpos,py+1) > 0) j = 2;
        gcopy(num, n, j-1, i, py);
        return;
      }
}


/*-----  gcopy  ------------------------------------------------*/
/*  Places next group of input elements into location table.  */

gcopy(node, n, i, x, y)
int node, n, i, x, y;
{
  gatelst *p;
  int k, cn;

  p = gateptr + node;

  for (k=0; k<n; k++)
    {
      cn = *(p->cin+k);
      if (cn > 0) set_pos(cn, x, y + gatpos[n-1][i].gpos[k]);
    }
}


/*-----  set_pos  ----------------------------------------------*/
/*  Assigns logic element to a specific space in the location table.  */

set_pos(n, x, y)
int n, x, y;
{
  gatelst *p;

  p = gateptr + n;

  if (p->placed == TRUE) return;

  p->xpos = x;
  p->ypos = y;
  p->placed = TRUE;
```

```
  set_loc(n, x, y);
}


/*-----  make_map  ----------------------------------------------*/
/*  Creates the location table.  */

make_map()
{
  int i, size;
  int *s;

  if (locptr != NULL) free(locptr);

  xmap = 24;

  size = xmap*GWIDTH;
  locptr = (int *)malloc(size*sizeof(int));
  for (i=0, s = locptr; i < size; i++, s++)
    *s = 0;
}


/*-----  set_loc  -----------------------------------------------*/
/*  Sets location in the location table to an element number.  */

set_loc(n, x, y)
int n, x, y;
{
  int i;

  if (x < 0 || x >= xmap) return;
  if (y < 0 || y >= GWIDTH) return;

  i = y * xmap + x;

  *(locptr + i) = n;
}


/*-----  chk_loc  -----------------------------------------------*/
/*  Checks if a location in the location table is free.  */

int chk_loc(x, y)
int x, y;
{
  int i;

  if (x < 0 || x >= xmap) return(-1);
  if (y < 0 || y >= GWIDTH) return(-1);
```

```
  i = y * xmap + x;

  return(*(locptr + i));
}
```

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

        load.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains the routines to load a gate list file

*********************************************************************/

#include <dos.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "gfx.h"

extern header();
extern center();
extern press();


/*-----  load  ------------------------------------------------*/
/*  Obtains a directory of .GLF files, queries for input file,  */
/*  and then loads the requested file.  */

load()
{
  FILE *fp;
  char buff[80];
  char *p;
  struct find_t flinfo, *flptr;
  int col, row;

  _clearscreen(_GCLEARSCREEN);
  header("Load Gate List");
  box(3, 8, 18, 62, DOUBLE);

  getcwd(buff, 80);
  if (buff[strlen(buff)-1] == '\\')
    strcat(buff, "*.GLF");
  else
    strcat(buff, "\\*.GLF");
  curdir(buff);

  row = 4;
```

```
col = 12;
flptr = &flinfo;
if (!_dos_findfirst(buff, _A_NORMAL, flptr))
  {
    _settextposition(row++, col);
    _outtext(flptr->name);
  }

while (!_dos_findnext(flptr))
  {
    if (row == 20) {row = 4; col = col + 16;}
    _settextposition(row++, col);
    _outtext(flptr->name);
  }

buff[0] = 0;
_settextposition(23, 12);
_outtext("Enter input file name: ");

if (!kb_readln(buff, 75)) return;
p = buff;
while (*p != 0 && isspace(*p)) p++;

if ((fp = fopen(p, "r")) == NULL)
  {
    _settextposition(23, 12);
    printf("File %s not found - Press any key to continue", buff);
    while (!kbhit());
    do { getch(); }
      while (kbhit());
    return;
  }

if (!strcmp(curfile, "none")) free_list();

memset(curfile, 0, 13);
strncpy(curfile, buff, 12);

if (!rd_symtb(fp))
  {
    _settextposition(23, 12);
    _outtext("                ");
    center(21, "File does not contain the proper format");
    press(C, 23, 0);
    return;
  }
if (!rd_ststab(fp))
  {
    _settextposition(23, 12);
    _outtext("                ");
```

```
      center(21, "File does not contain the proper format");
      press(C, 23, 0);
      return;
    }
  if (!rd_sdt(fp))
    {
      _settextposition(23, 12);
      _outtext("                ");
      center(21, "File does not contain the proper format");
      press(C, 23, 0);
      return;
    }
  rd_sqrt(fp);
  rd_clutab(fp);
  rd_hash(fp);
  rd_glist(fp);

  normalize();

  _settextposition(23, 12);
  printf("File loaded -( %d Elements )- Press any key to continue", numread);
  while (!kbhit());
  do { getch(); }
    while (kbhit());
}


/*-----  readln  ------------------------------------------------*/
/*  Reads a line of input from a file and places it into a buffer.  */

int readln(fp, bp)
FILE *fp;
char *bp;
{
  char ch;

  while ((ch = fgetc(fp)) != EOF)
    {
      if (ch == '\n')
        {
          *bp = '\0';
          return(1);
        }
      *bp++ = ch;
    }
  *bp = '\0';
  return(0);
}
```

```
/*-----  get_symtb  ------------------------------------------*/
/*  Reads in the symbol table values.  */

get_symtb(fp, num)
FILE *fp;
int num;
{
  char buff[LBUFSZ];
  int i, j;
  symtab *p;

  _settextposition(2,1);
  for (i=0, p = symptr+1; i<num; i++, p++)
    {
      readln(fp, buff);
      buff[11] = 0;
      for (j=1; j<11; j++)
        if (isspace(buff[j]))
          {
            buff[j] = 0;
            break;
          }
      strcpy(p->symname, buff+1);
    }
}


/*-----  rd_symtb  -------------------------------------------*/
/*  Looks for symbol table section in file and then initiates read.  */

int rd_symtb(fp)
FILE *fp;
{
  char buff[LBUFSZ];
  char *bp;
  int i, num;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " SYMTB", 6) != 0) return(0);
  bp = buff;
  bp = strchr(bp, ':')+1;
  num = atoi(bp);
  numsym = num;
  symptr = (symtab *)malloc((num+1)*sizeof(symtab));
  get_symtb(fp, num);
  return(1);
}


/*-----  rd_ststab  ------------------------------------------*/
```

```
/*  Reads and discards the system table information.  */

int rd_ststab(fp)
FILE *fp;
{
  char buff[LBUFSZ];
  char *bp;
  int i, ln;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " STSTAB", 7) != 0) return(0);
  bp = buff;
  bp = strchr(bp, ':')+1;
  ln = atoi(bp);
  for (i=0; i<ln; i++)
    readln(fp, buff);
  return(1);
}


/*-----  get_sdt  ------------------------------------------------*/
/*  Reads and stores relevant symbol description table data.  */

get_sdt(fp, num)
FILE *fp;
int num;
{
  char buff[LBUFSZ], *bp;
  int i;
  sdt *p;

  for (i=0, p = sdtptr+1; i<num; i++, p++)
    {
      readln(fp, buff);
      bp = buff+1;
      p->strwnum = get_five(&bp);
      p->code    = get_five(&bp);
      get_five(&bp);
      p->bcols   = get_five(&bp);
      p->brows   = get_five(&bp);
      get_five(&bp);
      get_five(&bp);
      p->uid     = get_five(&bp);
    }
}


/*-----  rd_sdt  -------------------------------------------------*/
/*  Looks for symbol description table section and begins read.  */
```

```
int rd_sdt(fp)
FILE *fp;
{
  char buff[LBUFSZ];
  char *bp;
  int num;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " SDT", 4) != 0) return(0);
  bp = buff;
  bp = strchr(bp, ':')+1;
  num = atoi(bp);
  numsdt = num;
  sdtptr = (sdt *)malloc((num+1)*sizeof(sdt));
  get_sdt(fp, num);
  return(1);
}


/*-----  get_sqrt  ------------------------------------------*/
/*  Reads and stores the relevant SQRT information.  */

get_sqrt(fp, num)
FILE *fp;
int num;
{
  char buff[LBUFSZ], *bp;
  int i;
  sqrtb *p;

  for (i=0, p = sqrtptr+1; i<num; i++, p++)
    {
      readln(fp, buff);
      bp = buff+1;
      get_five(&bp);
      p->step = get_five(&bp);
      get_five(&bp);
      get_five(&bp);
      get_five(&bp);
      p->csff = get_five(&bp);
    }
}


/*-----  rd_sqrt  ------------------------------------------*/
/*  Looks for SQRT section and begins read.  */

int rd_sqrt(fp)
FILE *fp;
{
```

```
  char buff[LBUFSZ];
  char *bp;
  int num;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " SQRT", 5) != 0) return(0);
  bp = buff;
  bp = strchr(bp, ':')+1;
  num = atoi(bp);
  numsqrt = num;
  sqrtptr = (sqrtb *)malloc((num+1)*sizeof(sqrtb));
  get_sqrt(fp, num);
  return(1);
}


/*-----  rd_clutab  ----------------------------------------------*/
/*  Reads and discard the combinational logic unit table data.  */

int rd_clutab(fp)
FILE *fp;
{
  char buff[LBUFSZ];
  char *bp;
  int i, ln;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " CLUTAB", 7) != 0) return(0);
  bp = buff;
  bp = strchr(bp, ':')+1;
  ln = atoi(bp);
  for (i=0; i<ln; i++)
    readln(fp, buff);
  return(1);
}


/*-----  rd_hash  ------------------------------------------------*/
/*  Reads and discards the hash table data.  */

int rd_hash(fp)
FILE *fp;
{
  char buff[LBUFSZ];
  char *bp;
  int i, ln;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " HASH", 5) != 0) return(0);
  bp = buff;
```

```
  bp = strchr(bp, ':')+1;
  ln = atoi(bp) / NHASHLN;
  for (i=0; i<ln; i++)
    readln(fp, buff);
  return(1);
}


/*-----  get_glist  ------------------------------------------*/
/*  Reads and stores the relevant gate list data.  */

get_glist(fp, num)
FILE *fp;
int num;
{
  char buff[LBUFSZ], *bp;
  int i, k, s, n, slot;
  gatelst *p;

  for (i=0; i<num; i++)
    {
      if (!readln(fp, buff)) return;
      numread++;
      bp = buff+1;
      slot = get_five(&bp);
      p = gateptr + slot;
      p->type = get_five(&bp);
      s = get_five(&bp);
      p->numin = get_five(&bp);
      p->numout = get_five(&bp);

      if (s)
        {
          bp = buff+1;
          readln(fp, buff);
          p->sym = get_five(&bp);
          p->elem = get_five(&bp);
        }
      else
        {
          p->sym = EMPTY;
          p->elem = EMPTY;
        }

      p->cin = NULL;
      n = p->numin;
      if (n)
        {
          p->cin = (int *)malloc((n+1)*sizeof(int));
          bp = buff+1;
```

```
            readln(fp, buff);
            for (k=0; k < n; k++)
              {
                if (k == 20) { bp = buff+1; readln(fp, buff); }
                *(p->cin + k) = get_five(&bp);
              }
          }

        p->cout = NULL;
        n = p->numout;
        if (n)
          {
            p->cout = (int *)malloc((n+1)*sizeof(int));
            bp = buff+1;
            readln(fp, buff);
            for (k=0; k < n; k++)
              {
                if (k == 20) { bp = buff+1; readln(fp, buff); }
                *(p->cout + k) = get_five(&bp);
              }
          }
      }
}


/*----- rd_glist ----------------------------------------------*/
/*  Looks for gate list section and initiates read.  */

int rd_glist(fp)
FILE *fp;
{
  char buff[LBUFSZ];
  char *bp;
  int i, num;
  int sym;
  gatelst *p;

  numread = 0;

  if (!readln(fp, buff)) return(0);
  if (strncmp(buff, " GATE LIST", 10) != 0) return(0);
  bp = buff;
  bp = strchr(bp, ':')+1;
  num = atoi(bp);
  numgate = num;
  gateptr = (gatelst *)malloc((num+1)*sizeof(gatelst));
  for (i=0, p = gateptr+1; i<num; i++, p++)
    p->type = EMPTY;
  get_glist(fp, num);
  return(1);
```

```
}


/*----- free_list ------------------------------------------*/
/*  Frees all memory allocated to a gate list.  */

free_list()
{
  int i;
  gatelst *p;

  free(symptr);
  free(sdtptr);
  free(sqrtptr);
  for (i=0, p = gateptr+1; i<numgate; i++, p++)
    {
      if (p->cin  != NULL) free(p->cin);
      if (p->cout != NULL) free(p->cout);
    }
  free(gateptr);
}


/*----- get_val ------------------------------------------*/
/*  Gets a numeric value from a string buffer and advances the  */
/*  buffer pointer.  */

int get_val(p)
char **p;
{
  int n;

  while (isspace(**p)) (*p)++;
  n = atoi(*p);
  while (!isspace(**p)) (*p)++;
  return(n);
}


/*----- get_five ------------------------------------------*/
/*  Gets a numeric value from a five-char space in a string buffer.  */

int get_five(p)
char **p;
{
  char buff[6];

  strncpy(buff, *p, 5);
  buff[5] = 0;
  *p = *p + 5;
```

```
  return(atoi(buff));
}


/*-----  normalize  ------------------------------------------*/
/*  Converts unimplemented element types to implemented types.  */

normalize()
{
  int i;
  gatelst *p;

  for (i=0, p = gateptr+1; i<numgate; i++, p++)
    {
      switch(p->type)
        {
          case 4001:
          case 4031:
                      p->type = AND;
                      break;
          case 4029:
                      p->type = CND;
                      break;
          case 4002:
          case 4032:
                      p->type = NAND;
                      break;
          case 4003:
          case 4030:
          case 4033:
                      p->type = OR;
                      break;
          case 4004:
          case 4034:
                      p->type = XOR;
                      break;
          case 4005:
                      p->type = NOR;
                      break;
          case 4006:
                      p->type = DFCS;
                      break;
          case 4007:
          case 4009:
          case 4010:
          case 4011:
                      p->type = DFF;
                      break;
          case 4012:
```

```
            case 4018:
                        p->type = INPUT;
                        break;
            case 4013:
            case 4019:
                        p->type = OUTPUT;
                        break;
            case 3998:
            case 3999:
            case 4008:
            case 4014:
            case 4015:
            case 4016:
            case 4017:
            case 4020:
            case 4021:
            case 4022:
            case 4023:
            case 4024:
            case 4025:
            case 4026:
            case 4027:
            case 4028:
            default:
                        p->type = PASS;
                        break;
        }
    }
}


/*----- curdir ----------------------------------------------*/
/* Displays current working directory on the screen. */

curdir(s)
char *s;
{
  int col;
  char buff[42];

  memset(buff, 205, 41);
  buff[40] = 0;
  _settextposition(3, 25);
  _outtext(buff);

  col = 40 - strlen(s) / 2;
  _settextposition(3, col);
  _outtext(" ");
  _outtext(s);
  _outtext(" ");
}
```

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

        lreps.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains the bit-map representations of the various
        logic elements.

*************************************************************************/

#include "gfx.h"

/*  Bit representation for an AND gate.  */
char prn_and[LRSIZE][LCSIZE] = {

          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,

          3,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,
          2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,
          2,   2,   2,   2,   2,   2,   3,

        255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0, 255,

        255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0, 255,

        255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0, 255,

        128, 112,  14,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   1,  14, 112, 128,

          0,   0,   0, 128,  64,  32,  16,   8,   4,   4,   2,   2,
          2,   1,   1,   1,   1,   1,   2,   2,   2,   4,   4,   8,
         16,  32,  64, 128,   0,   0,   0
};


/*  Bit representation for a NAND gate.  */
char prn_nand[LRSIZE][LCSIZE] = {
```

```
        3,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  3,

      255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,255,

      255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,255,

      255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,255,

      128,112, 14,  1,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  1, 14,112,128,

        0,  0,  0,128, 64, 32, 16,  8,  4,  4,  2,  2,
        2,  1,  1,  1,  1,  1,  2,  2,  2,  4,  4,  8,
       16, 32, 64,128,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       56, 68,130,131,130, 68, 56,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0
};


/*  Bit representation for an OR gate.  */
char prn_or[LRSIZE][LCSIZE] = {

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

       31,  8,  4,  4,  2,  2,  1,  1,  1,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,
        1,  2,  2,  4,  4,  8, 31,

      255,  0,  0,  0,  0,  0,  0,  0,  0,128,128,128,
       64, 64, 64, 64, 64, 64, 64,128,128,128,  0,  0,
        0,  0,  0,  0,  0,  0,255,

      255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,255,
```

```
        224,  30,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   1,  30, 224,

          0,   0, 192,  48,  12,   3,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   3,  12,  48, 192,   0,   0,

          0,   0,   0,   0,   0,   0, 128,  64,  32,  16,   8,   4,
          2,   2,   1,   1,   1,   2,   2,   4,   8,  16,  32,  64,
        128,   0,   0,   0,   0,   0,   0
};


/*  Bit representation for a NOR gate.  */
char prn_nor[LRSIZE][LCSIZE] = {

         31,   8,   4,   4,   2,   2,   1,   1,   1,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,
          1,   2,   2,   4,   4,   8,  31,

        255,   0,   0,   0,   0,   0,   0,   0,   0, 128, 128, 128,
         64,  64,  64,  64,  64,  64,  64, 128, 128, 128,   0,   0,
          0,   0,   0,   0,   0,   0, 255,

        255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0, 255,

        224,  30,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   1,  30, 224,

          0,   0, 192,  48,  12,   3,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   3,  12,  48, 192,   0,   0,

          0,   0,   0,   0,   0,   0, 128,  64,  32,  16,   8,   4,
          2,   2,   1,   1,   1,   2,   2,   4,   8,  16,  32,  64,
        128,   0,   0,   0,   0,   0,   0,

          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         56,  68, 130, 131, 130,  68,  56,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0
};


/*  Bit representation for an XOR gate.  */
char prn_xor[LRSIZE][LCSIZE] = {
```

```
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,

      143,  68,  34,  34,  17,  17,   8,   8,   8,   4,   4,   4,
        2,   2,   2,   2,   2,   2,   2,   4,   4,   4,   8,   8,
        8,  17,  17,  34,  34,  68, 143,

      255,   0,   0,   0,   0,   0, 128, 128, 128,  64,  64,  64,
       32,  32,  32,  32,  32,  32,  32,  64,  64,  64, 128, 128,
      128,   0,   0,   0,   0,   0, 255,

      255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0, 255,

      224,  30,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   1,  30, 224,

        0,   0, 192,  48,  12,   3,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   3,  12,  48, 192,   0,   0,

        0,   0,   0,   0,   0,   0, 128,  64,  32,  16,   8,   4,
        2,   2,   1,   1,   1,   2,   2,   4,   8,  16,  32,  64,
      128,   0,   0,   0,   0,   0,   0
};


/*  Bit representation for a DFF (output side).  */
char prn_dff[LRSIZE][LCSIZE] = {

        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,

        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,

        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,

       15,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,  15,

      255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 255,

       255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 255,

       255,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1, 255
};

/*  Bit representation for a DFF (input and output sides).  */
char prn_frn[LRSIZE][LCSIZE] = {

         7,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,
         4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,
         4,   4,   4,   4,   4,   4,   7,

       255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 255,

       255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 255,

       255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 255,

       255,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 255,

       240,  16,  16,  16,  16,  16,  16,  16,  16,  16,  16,  16,
        16,  16,  16,  16,  16,  16,  16,  16,  16,  16,  16,  16,
        16,  16,  16,  16,  16,  16, 240,

         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,
};

/*  Bit representation for an INPUT.  */
char prn_inp[LRSIZE][LCSIZE] = {

         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        8, 20, 34, 65, 34, 20,  8,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0
};

/* Bit representation for an OUTPUT. */
char prn_out[LRSIZE][LCSIZE] = {

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       16, 40, 68,130, 68, 40, 16,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
```

```
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,

        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0
};
```

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

        print.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains routines for printing out a logic gate display.

*************************************************************************/

#include <bios.h>
#include <dos.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <malloc.h>

#include "gfx.h"

extern header();
extern center();
extern cursor();

extern cp_number();

#define DELTA   10

/*  These are pixel offset for horizontal path connections.  */
int hchnoff[3] = { 6, 3, 9 };

/*  There are eight vertical paths which can be used for connections.  */
/*  These values assign an initial guess for which path to used.  */
char vchan[DELTA+1] = { 2, 3, 4, 5, 6, 7, 0, 1, 0, 1, 0 };

/*  These are the pixel offsets for the vertical channels.  */
int vchnoff[8] = { 2, 7, 12, 17, 22, 27, 32, 37 };

/*  These are pixel offsets for input connection locations.  */
char pos_out[8][8] = {
       15,  0,  0,  0,  0,  0,  0,  0,        /* 1 input  */
        8, 22,  0,  0,  0,  0,  0,  0,        /* 2 inputs */
        6, 15, 24,  0,  0,  0,  0,  0,        /* 3 inputs */
        3, 11, 19, 27,  0,  0,  0,  0,        /* 4 inputs */
        3,  9, 15, 21, 27,  0,  0,  0,        /* 5 inputs */
        2,  7, 12, 18, 23, 28,  0,  0,        /* 6 inputs */
        3,  7, 11, 15, 19, 23, 27,  0,        /* 7 inputs */
```

```
        1,  5,  9, 13, 17, 21, 25, 29  };      /* 8 inputs */


extern char prn_and[LRSIZE][LCSIZE];
extern char prn_nand[LRSIZE][LCSIZE];
extern char prn_or[LRSIZE][LCSIZE];
extern char prn_nor[LRSIZE][LCSIZE];
extern char prn_xor[LRSIZE][LCSIZE];
extern char prn_dff[LRSIZE][LCSIZE];
extern char prn_frn[LRSIZE][LCSIZE];
extern char prn_inp[LRSIZE][LCSIZE];
extern char prn_out[LRSIZE][LCSIZE];


extern int din;

struct con_slot con_tab[CONMAX];


/*-----  print_layout  ------------------------------------------*/
/*  Organizes print sequence.  Each column of logic elements is  */
/*  placed in the print buffer and printed.  */

print_layout()
{
  int n, x, y, res;
  con_slot *cs;

  _clearscreen(_GCLEARSCREEN);
  header("Print Subgraph");

  if (locptr == NULL)
    {
      center(10, "No subgraph has been selected.");
      press(C, 14, 0);
      return;
    }

  _settextposition(10, 22);
  _outtext("Printing subgraph...");

  for (n=0, cs = &con_tab[0]; n<CONMAX; n++, cs++)
    cs->cfree = TRUE;

  for (x = first_col(); x>0; x--)
    {
      clr_buff();
      for (y=0; y<GWIDTH; y++)
        if (n = chk_loc(x, y))
          p_logic(n, y*GSLOTW+9, x);
```

```
      old_paths(x);
      new_paths(x);
      prn_buff(RBUFSIZE);
    }
  clr_buff();
  res = first_elem();
  prn_buff(RBUFSIZE);
  if (res)
    {
      clr_buff();
      p_line(5*GSLOTW+24, 12, 5*GSLOTW+24, 0);
      prn_buff(2);
    }

  fputc(10, stdprn);
  fputc(10, stdprn);
  fputc(13, stdprn);
  fflush(stdprn);

  printf(" Print complete");
  press(C, 23, 0);
}


/*-----  first_col  ------------------------------------------------*/
/*  Returns the number of the first column from the left which  */
/*  has logic elements.  */

int first_col()
{
  int i;

  for (i=xmap-1; i>0; i--)
    if (active_col(i)) return(i);
  return(0);
}


/*-----  active_col  -----------------------------------------------*/
/*  Checks if a column has any logic elements in it.  */

int active_col(x)
int x;
{
  int y;

  for (y=0; y<GWIDTH; y++)
    if (chk_loc(x, y)) return(1);
  return(0);
}
```

```
/*-----  first_elem  ----------------------------------------------*/
/*  Handles placement of the initial logic element.  */

int first_elem()
{
  gatelst *p, *p2;
  int x, n, i, r, c;
  int num;
  char *ptr, *s;

  n = chk_loc(0, 5);
  p = gateptr + n;
  x = 5*GSLOTW+9;

  switch (p->type)
    {
      case DFF:
                        s = (symptr+p->sym)->symname;
                        pr_name(s, x-10, 75);
                        i = rowcol(n, &r, &c);
                        p_rgt_rc(i, r, c, x-19, 75);
      case DFCS:
                        lp_number(n, x+3, 50);
                        p_letter('Q', x+21, 74);
                        p_letter('D', x+21, 50);
                        p_letter('E', x+12, 50);
                        p_letter('N', x+12, 57);
                        ptr = &prn_frm[0][0];
                        place_gate(ptr, x);
                        switch (din)
                          {
                            case 0:
                                    n = chk_loc(1, 5);
                                    if (n > 0)
                                      do_dff(x+24, PAD+5, gateptr+n);
                                    break;
                            case 1:
                                    n = chk_loc(1, 5);
                                    if (n > 0)
                                      do_dff(x+15, PAD+5, gateptr+n);
                                    break;
                            case 2:
                                    n = chk_loc(1, 7);
                                    if (n > 0)
                                      do_dff(x+24, PAD+5, gateptr+n);
                                    n = chk_loc(1, 3);
                                    if (n > 0)
                                      do_dff(x+15, PAD+5, gateptr+n);
```

```
                        break;
                    default:
                        break;
                  }
                fin_path();
                break;
    case OUTPUT:
                s = (symptr+p->sym)->symname;
                pl_name(s, x+12, 52);
                lp_number(n, x+22, 52);
                i = rowcol(n, &r, &c);
                p_lft_rc(i, r, c, x+2, 52);
                ptr = &prn_out[0][0];
                place_gate(ptr, x);
                for (i=0; i<GWIDTH; i++)
                  {
                    n = chk_loc(1, i);
                    if (n > 0) do_dff(x+15, PAD, gateptr+n);
                  }
                break;
    default:
                p_logic(n, 5*GSLOTW+9, 0);
                old_paths(0);
                return(1);
                break;
   }
  return(0);
}


/*-----  do_dff  ------------------------------------------------*/
/*  Sets up connections between the initial logic element and the  */
/*  next column of elements.  */

do_dff(x, y, p)
int x, y;
gatelst *p;
{
  int yc, x2, d;

  x2 = p->ypos*GSLOTW+24;

  yc = y / 2;

  d = (p->type == DFF || p->type == DFCS) ? 9 : 0;

  p_line(x,    yc,    x,  y);
  p_line(x,    yc, x2+d, yc);
  p_line(x2+d, yc, x2+d, -1);
}
```

```
/*-----  get_cslot  ------------------------------------------------*/
/*  Allocates a slot in the connection table.  */

con_slot *get_cslot()
{
  int i;
  con_slot *cs;

  for (i=0, cs=&con_tab[0]; i<CONMAX; i++, cs++)
    if (cs->cfree)
      {
        cs->cfree = FALSE;
        return(cs);
      }
  return(NULL);
}


/*-----  old_paths  ------------------------------------------------*/
/*  Continues drawing connections begun in previous columns.  */

old_paths(level)
int level;
{
  int i;
  gatelst *p;
  con_slot *cs;

  for (i=0, cs = &con_tab[0]; i<CONMAX; i++, cs++)
    if (cs->cfree == FALSE)
      {
        p = gateptr + cs->cn_to;
        if (p->xpos < level)
          cont_path(cs);
        else
          if (cs->cdir == IPATH)
            old_ipath(cs);
          else
            if (p->xpos == level)
              cont_path(cs);
            else
              old_opath(cs);
      }
}


/*-----  old_ipath  ------------------------------------------------*/
/*  Draws paths from previous columns to gate inputs.  */
```

```
old_ipath(cs)
con_slot *cs;
{
  int i, j, k;
  int n1, n2;
  gatelst *p, *p2;
  int x, y;
  int delta;

  p = gateptr + cs->cn_to;
  p2 = gateptr + cs->cn_from;
  n1 = cin_num(cs->cn_from, p);
  if (n1)
    {
      n2 = (int) min(p->numin, MAXIN) - 1;
      x = p->ypos*GSLOTW + 9 + pos_out[n2][n1-1];
      delta = min( abs(p2->ypos - p->ypos), DELTA);
      j = vchan[delta];
      for (k=0; k<8; k++, j = inc8(j))
        {
          y = vchnoff[j];
          if (!taken(cs->cx, y, x, y)) break;
        }
      p_line(cs->cx,  0, cs->cx, y);
      p_line(cs->cx,  y,      x, y);
      p_line(     x,  y,      x, PAD);
      if (chk_pix(cs->cx, y+1))
        joint(cs->cx, 0, cs->cx, y);
      else
        joint(cs->cx, 0, cs->cx, y-1);
      extend(x, PAD);
    }
  cs->cfree = TRUE;
}


/*-----  old_opath  ------------------------------------------------*/
/*  Draws paths from previous columns to gate outputs.  */

old_opath(cs)
con_slot *cs;
{
  int i, j, k;
  gatelst *p;
  int x, y;

  p = gateptr + cs->cn_to;
  x = p->ypos*GSLOTW+24;
  if (p->type == DFF || p->type == DFCS)
```

```
    x += 9;
  j = 1;
  for (k=0; k<8; k++, j = inc8(j))
    {
      y = vchnoff[j];
      if (!taken(cs->cx, y, x, y)) break;
    }
  p_line(cs->cx, 0, cs->cx, y);
  p_line(cs->cx, y,      x, y);
  p_line(     x, 0,      x, y);
  if (chk_pix(x, y+1))
    joint(x, 0, x, y);
  else
    joint(x, 0, x, y-1);
  cs->cfree = TRUE;
}


/*-----  fin_path  ----------------------------------------------*/
/*  Handles paths from previous columns to the output of the initial  */
/*  logic element.  */

fin_path()
{
  con_slot *cs;
  int i, cnt=0;
  int x1, x2, y;

  for (i=0, cs = &con_tab[0]; i<CONMAX; i++, cs++)
    if (cs->cfree == FALSE)
      if (cs->cdir == OPATH)
        {
          p_line(cs->cx,  0,      cs->cx, 91);
          p_line(cs->cx, 91, 5*GSLOTW+34, 91);
          p_line(5*GSLOTW+34, 91, 5*GSLOTW+34, 83);
          cs->cfree = TRUE;
          cnt++;
        }
  if (cnt > 1) dot(5*GSLOTW+33, 80);
}


/*-----  cont_path  ---------------------------------------------*/
/*  Draws paths from previous columns to succeeding columns.  */

cont_path(cs)
con_slot *cs;
{
  p_line(cs->cx, 0, cs->cx, BUFW+1);
}
```

```
/*-----  new_paths  ------------------------------------------------*/
/*  Handles new connections between logic elements.  */

new_paths(level)
int level;
{
  int i, n;

  for (i=0; i<GWIDTH; i++)
    if (n = chk_loc(level, i))
      {
        path_in(n);
        path_out(n);
      }
}


/*-----  path_in  --------------------------------------------------*/
/*  Draws paths from the current output to succeeding gate inputs.  */

path_in(n)
int n;
{
  int i, num;
  gatelst *p, *gp;
  con_slot *cs;

  p = gateptr + n;
  for (i=0; i<p->numout; i++)
    {
      num = *(p->cout + i);
      if (num > 0)
        {
          gp = gateptr + num;
          if (gp->placed)
            if (gp->xpos < p->xpos)
              {
                cs = get_cslot();
                cs->cdir = IPATH;
                cs->cn_from = n;
                cs->cn_to = num;
                cs->cx = p->ypos * GSLOTW + 24;
                if (p->type == DFF || p->type == DFCS)
                  cs->cx += 9;
              }
        }
    }
}
```

```
/*-----  path_out  ------------------------------------------------*/
/*  Draws paths from the current inputs to previous gate outputs.  */

path_out(n)
int n;
{
  int i, j, k;
  int m, num, in, delta;
  int xtemp;
  gatelst *p, *gp;
  con_slot *cs;
  int x1, x2, y;

  p = gateptr + n;
  if (p->type == DFF || p->type == DFCS || p->type == INPUT) return;

  m = min(MAXIN, p->numin);

  for (i=0; i<m; i++)
    {
      num = *(p->cin + i);
      if (num > 0)
        {
          gp = gateptr + num;
          if (gp->placed)
            if (gp->xpos <= p->xpos)
              {
                delta = (gp->ypos >= p->ypos) ? 0 : -1;
                if (p->ypos+delta == 4 && chk_loc(0, 5) == num) delta++;
                x1 = p->ypos*GSLOTW+9 + pos_out[m-1][i];
                y = PAD;
                xtemp = (p->ypos+delta)*GSLOTW+LCSIZE+9;
                x2 = xtemp+10;
                j = 7;
                for (k=0; k<8; k++, j = dec8(j))
                  {
                    y = vchnoff[j];
                    if (!taken(x1, y, x2, y)) break;
                  }
                for (k=0; k<3; k++)
                  {
                    x2 = xtemp+hchnoff[k];
                    if (!taken(x2, PAD, x2, PAD+5)) break;
                  }
                p_line(x1, PAD, x1, y);
                p_line(x1,   y, x2, y);
                p_line(x2,   y, x2, BUFW+1);
                extend(x1, PAD+1);
```

```
                cs = get_cslot();
                cs->cdir = OPATH;
                cs->cn_from = n;
                cs->cn_to = num;
                cs->cx = x2;
              }
        }
      else
        {
          x1 = p->ypos*GSLOTW+9 + pos_out[m-1][i];
          extend(x1, PAD+3);
          switch(num)
            {
              case FLT:
                       p_letter('F', x1-3, PAD-5);
                       break;
              case GND:
                       p_number(0, x1-3, PAD-5);
                       break;
              case VCC:
                       p_number(1, x1-3, PAD-5);
                       break;
              default:
                       break;
            }
        }
    }
}


/*-----  cin_num  ------------------------------------------------*/
/*  Returns index in the input list for a given gate number.  */

int cin_num(n, p)
int n;
gatelst *p;
{
  int i;

  for (i=0; i<MAXIN; i++)
    if (n == *(p->cin+i)) return(i+1);
  return(0);
}


/*-----  inc8  ---------------------------------------------------*/
/*  Increments the value using modulo 8 arithmetic.  */

int inc8(n)
int n;
```

```
{
  return(++n % 8);
}


/*-----  dec8  ------------------------------------------------*/
/*  Decrements the value using modulo 8 arithmetic.  */

int dec8(n)
int n;
{
  return(--n % 8);
}


/*-----  dot  -------------------------------------------------*/
/*  Marks a connection (square) on crossing wires.  */

dot(x, y)
int x, y;
{
  p_point(x-1, y-1);
  p_point(  x, y-1);
  p_point(x+1, y-1);
  p_point(x-1, y);
  p_point(x+1, y);
  p_point(x-1, y+1);
  p_point(  x, y+1);
  p_point(x+1, y+1);
}


/*-----  extend  ----------------------------------------------*/
/*  Draws input lines until they touch the input face of a gate.  */

extend(x, y)
int x, y;
{
  int i;

  for (i=0; i<20; i++)
    if (chk_pix(x, y+i))
      return;
    else
      p_point(x, y+i);
}


/*-----  p_line  ----------------------------------------------*/
/*  Draws a line from (x1, y1) to (x2, y2).  */
```

```
p_line(x1, y1, x2, y2)
int x1, y1, x2, y2;
{
  int i, dx, dy;

  dx = (x1 > x2) ? -1 : 1;
  dy = (y1 > y2) ? -1 : 1;

  if (x1 == x2)
    {
      for (i=y1; i!=y2; i = i+dy)
        p_point(x1, i);
      return;
    }

  if (y1 == y2)
    for (i=x1; i!=x2; i = i+dx)
      p_point(i, y1);
}


/*-----  p_point  ------------------------------------------------*/
/*  Draws a point at (x, y).  */

p_point(x, y)
int x, y;
{
  static char p_m[8] = { 128, 64, 32, 16, 8, 4, 2, 1 };
  int i;

  if (x < 0 || x > PWIDTH) return;
  if (y < 0 || y > RBUFSIZE*8) return;

  i = y % 8;
  y = y / 8;
  p_buffer[y][x] |= p_m[i];
}


/*-----  chk_pix  ------------------------------------------------*/
/*  Checks if point (x, y) is set in the print buffer.  */

int chk_pix(x, y)
int x, y;
{
  static char p_m[8] = { 128, 64, 32, 16, 8, 4, 2, 1 };
  int i;

  if (x < 0 || x > PWIDTH-1) return(0);
```

```
  if (y < 0 || y > RBUFSIZE*8-1) return(0);

  i = y % 8;
  y = y / 8;
  if (p_buffer[y][x] & p_m[i]) return(1);
  return(0);
}


/*----- taken ------------------------------------------------*/
/*  Determines if a potential connection path has been used already.  */

int taken(x1, y1, x2, y2)
int x1, y1, x2, y2;
{
  int i;

  if (x1 == x2)
    {
      if (y1 > y2)
        {
          for (i=y2; i<y1; i=i+2)
            if (chk_pix(x1, i) && chk_pix(x1, i+1)) return(1);
          return(0);
        }
      else
        {
          for (i=y1; i<y2; i=i+2)
            if (chk_pix(x1, i) && chk_pix(x1, i+1)) return(1);
          return(0);
        }
    }

  if (y1 == y2)
    {
      if (x1 > x2)
        {
          for (i=x2; i<x1; i=i+2)
            if (chk_pix(i, y1) && chk_pix(i+1, y1)) return(1);
          return(0);
        }
      else
        {
          for (i=x1; i<x2; i=i+2)
            if (chk_pix(i, y1) && chk_pix(i+1, y1)) return(1);
          return(0);
        }
    }

  return(1);
```

```
}


/*-----  joint  ------------------------------------------------*/
/*  Determines (and marks if appropriate) whether two lines crossing  */
/*  are connected.  */

joint(x1, y1, x2, y2)
int x1, y1, x2, y2;
{
  int i;

  if (x1 == x2)
    {
      if (y1 > y2)
        {
          for (i=y2; i<=y1; i++)
            if (chk_pix(x1-1, i) ^ chk_pix(x1+1, i)) dot(x1, i);
        }
      else
        {
          for (i=y1; i<=y2; i++)
            if (chk_pix(x1-1, i) ^ chk_pix(x1+1, i)) dot(x1, i);
        }
    }

  if (y1 == y2)
    {
      if (x1 > x2)
        {
          for (i=x2; i<=x1; i++)
            if (chk_pix(i, y1-1) ^ chk_pix(i, y1+1)) dot(i, y1);
        }
      else
        {
          for (i=x1; i<=x2; i++)
            if (chk_pix(i, y1-1) ^ chk_pix(i, y1+1)) dot(i, y1);
        }
    }

  return(1);
}



/*-----  clr_buff  ---------------------------------------------*/
/*  Clears the print buffer.  */

clr_buff()
{
```

```
  int i, j;

  for (i=0; i<RBUFSIZE; i++)
    for (j=0; j<CBUFSIZE; j++)
      p_buffer[i][j] = 0;
}


/*-----  prn_buff  ------------------------------------------------*/
/*  Prints out the print buffer.  */

prn_buff(amount)
int amount;
{
  int i, j;
  int n1, n2;
  int width;

  for (i=0; i<amount; i++)
    {
      fputc(27, stdprn);
      fputc(74, stdprn);
      fputc(24, stdprn);

      fputc(27, stdprn);
      fputc(SDENSITY, stdprn);

      width = PWIDTH;

      for (j = PWIDTH-1; j>=0; j--)
        if (p_buffer[i][j])
          break;
        else
          width--;

      n1 = width % 256;
      n2 = width / 256;

      fputc(n1, stdprn);
      fputc(n2, stdprn);

      for (j=0; j<width; j++)
        fputc(p_buffer[i][j], stdprn);

      fputc(13, stdprn);
    }
}


/*-----  p_logic  -------------------------------------------------*/
```

```
/*  Handles placing the specified gate bit representation into  */
/*  the print buffer.  */

p_logic(n, x, y)
int n, x, y;
{
  int i, j, r, c;
  int msk, row;
  int num, pos;
  gatelst *p;
  char *ptr, *s;

  p = gateptr + n;

  switch (p->type)
    {
      case AND:
      case CND:
                        cp_number(n, x+12, 86);
                        ptr = &prn_and[0][0];
                        break;
      case NAND:
                        cp_number(n, x+12, 78);
                        ptr = &prn_nand[0][0];
                        break;
      case OR:
                        cp_number(n, x+12, 86);
                        ptr = &prn_or[0][0];
                        break;
      case NOR:
                        cp_number(n, x+12, 79);
                        ptr = &prn_nor[0][0];
                        break;
      case XOR:
                        cp_number(n, x+12, 87);
                        ptr = &prn_xor[0][0];
                        break;
      case DFF:
                        s = (symptr+p->sym)->symname;
                        pr_name(s, x+12, 78);
                        i = rowcol(n, &r, &c);
                        p_rgt_rc(i, r, c, x+3, 78);
      case DFCS:
                        rp_number(n, x+21, 72);
                        p_letter('Q', x+21, 86);
                        ptr = &prn_dff[0][0];
                        break;
      case INPUT:
                        s = (symptr+p->sym)->symname;
                        pr_name(s, x+12, 79);
```

```
                    rp_number(n, x+22, 79);
                    i = rowcol(n, &r, &c);
                    p_rgt_rc(i, r, c, x+3, 79);
                    ptr = &prn_inp[0][0];
                    break;
      default:
                    break;
    }

  place_gate(ptr, x);
};


/*-----  place_gate  ------------------------------------------*/
/*  Places bit representation into print buffer.  */

place_gate(p, x)
char *p;
int x;
{
  int i, j;

  for (i=0; i<LRSIZE; i++)
    for (j=0; j<LCSIZE; j++)
      p_buffer[RBUFSIZE-LRSIZE+i][x+j] |= *p++;
}


/*-----  p_rgt_rc  --------------------------------------*/
/*  Prints the column and row number of a symbol right-justified.  */

p_rgt_rc(n, r, c, x, y)
int n, r, c, x, y;
{
  if (n == 2)
    {
      p_symbol('>', x, y);
      rp_number(r, x, y-7);
      y = y - 7 * digits(r) - 14;
      p_symbol('<', x, y+7);
    }

  if (n)
    {
      p_symbol(']', x, y);
      rp_number(c, x, y-7);
      y = y - 7 * digits(c) - 14;
      p_symbol('[', x, y+7);
    }
}
```

```
/*-----  p_lft_rc  ----------------------------------------*/
/*  Print the column and row number of a symbol left-justified.  */

p_lft_rc(n, r, c, x, y)
int n, r, c, x, y;
{
  if (n == 2)
    {
      p_symbol('<', x, y);
      lp_number(r, x, y+7);
      y = y + 7 * digits(r) + 7;
      p_symbol('>', x, y);
    }

  if (n)
    {
      p_symbol('[', x, y);
      lp_number(c, x, y+7);
      y = y + 7 * digits(c) + 7;
      p_symbol(']', x, y);
    }
}
```

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

        utility.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains utility functions

************************************************************/

#include        <stdio.h>
#include        <bios.h>
#include        <string.h>
#include        <graph.h>

#include        "gfx.h"


/*  Character codes for lines  */
char boxes[2][6] = {
        196, 179, 218, 191, 217, 192,
        205, 186, 201, 187, 188, 200,
};


/*-----  box  ----------------------------------------*/
/*  Draws a box on the screen.  Does not require graphics adapter.  */

box(row, col, hi, wd, btype)
int row, col, hi, wd, btype;
{
  int i;
  char temp[80];

  _settextposition(row, col);
  temp[0] = boxes[btype][NW];
  memset(temp+1,boxes[btype][HZ],wd);
  temp[wd+1] = boxes[btype][NE];
  temp[wd+2] = 0;
  _outtext(temp);
  for (i=1; i<=hi; ++i)
    {
      _settextposition(row+i,col);
      putch(boxes[btype][VT]);
      _settextposition(row+i,col+wd+1);
      putch(boxes[btype][VT]);
    }
  _settextposition(row+hi+1,col);
```

```
  temp[0] = boxes[btype][SW];
  memset(temp+1,boxes[btype][HZ],wd);
  temp[wd+1] = boxes[btype][SE];
  temp[wd+2] = 0;
  _outtext(temp);

  return(0);
}


/*----- header ----------------------------------------*/
/* Writes centered string on the top line of the screen. */

header(p)
char *p;
{
  int bcolor, tcolor;
  char buff[40];
  int i;

  bcolor = _getbkcolor();
  tcolor = _gettextcolor();

  _settextcolor(bcolor);
  _setbkcolor((long)tcolor);

  i = (80 - strlen(p)) / 2;
  memset(buff, ' ', i);
  buff[i] = 0;
  _settextposition(1,1);
  _outtext(buff);
  _outtext(p);
  _outtext(buff);

  _settextcolor(tcolor);
  _setbkcolor((long)bcolor);
}


/*----- out_item ----------------------------------------*/
/* Writes out a string a (r:c). */

out_item(r, c, p, l)
int r, c;
char *p;
int l;
{
  char buff[80];

  _settextposition(r, c);
```

```
  _outtext(" ");
  _outtext(p);
  memset(buff, ' ', l);
  buff[l] = 0;
  _outtext(buff);
}


/*-----  menu  ----------------------------------------*/
/*  Displays a menu and returns position of selected item.  */

int menu(p, type, r, c, first)
char *p[];
int type, r, c, first;
{
  int i, num, mxwidth;
  int row, col, srow;
  int curr, prev;
  short tcolor, bcolor;
  int len[MAXITEM];

  num = 0;
  mxwidth = 0;
  curr = first;
  prev = first;

  for (i=0; i<MAXITEM; i++)
    {
      if (p[i] == NULL) break;
      len[i] = strlen(p[i]);
      mxwidth = (len[i] > mxwidth) ? len[i] : mxwidth;
      num++;
    }
  mxwidth = mxwidth + 2;

  bcolor = _getbkcolor();
  tcolor = _gettextcolor();

  row = (24 - num) / 2;
  col = (80 - mxwidth - 4) / 2;
  if (type == PICK)
    {
      row = r;
      col = c;
    }
  srow = row+1;
  box(row++, col++, num, mxwidth+2, DOUBLE);

  for (i=0; i<num; i++)
    if (i == first)
```

```
  _outtext(s);
}


/*----- panic ------------------------------------------*/
/*  Prints a fatal error message and terminates execution.  */

panic(n)
int n;
{
  _clearscreen(_GCLEARSCREEN);
  _settextposition(2,2);
  printf("Program termination: ");
  switch (n)
    {
      case  ERR_MEM:
                      printf("Memory Allocation Failure\n");
                      break;
      default:
                      break;
    }
  exit(0);
}


/*----- digits -----------------------------------------*/
/*  Returns the number of digits in a number.  */

int digits(n)
int n;
{
  if (n < 10) return(1);
  if (n < 100) return(2);
  if (n < 1000) return(3);
  if (n < 10000) return(4);
  return(5);
}



/*----- kb_readln --------------------------------------*/
/*  Reads a line of data from the keyboard.  */

int kb_readln(p, max)
char *p;
int max;
{
  char ch[2];
  char *cp = ch;
  int i=0;
```

```
      {
        _settextcolor(bcolor);
        _setbkcolor((long)tcolor);
        out_item(row++, col, p[i], mxwidth-len[i]);
        _settextcolor(tcolor);
        _setbkcolor((long)bcolor);
      }
    else
      out_item(row++, col, p[i], mxwidth-len[i]);

  _settextposition(srow,col);
  for (;;)
    {
      switch ((_bios_keybrd(_KEYBRD_READ) & 0xff00) >> 8)
        {
          case UP:
                curr = (curr > 0) ? --curr : num-1;
                break;
          case DOWN:
                curr = (curr < num - 1) ? ++curr : 0;
                break;
          case ESC:
                return(0);
          case ENTER:
                return(curr);
          default:
                continue;
        }
      out_item(srow+prev, col, p[prev], mxwidth-len[prev]);
      _settextcolor(bcolor);
      _setbkcolor((long)tcolor);
      out_item(srow+curr, col, p[curr], mxwidth-len[curr]);
      _settextcolor(tcolor);
      _setbkcolor((long)bcolor);
      prev = curr;
    }
}


/*-----  cursor  ----------------------------------------*/
/*  Turns off cursor.  */

unsigned cursor(value)
unsigned value;
{
  union REGS inregs, outregs;
  int ret;

  inregs.h.ah = 3;
  inregs.h.bh = 0;
```

```
  int86(0x10,&inregs,&outregs);
  ret = outregs.x.cx;

  inregs.h.ah = 1;
  inregs.x.cx = value;
  int86(0x10,&inregs,&outregs);

  return(ret);
}


/*-----  keyhit  ------------------------------------------------*/
/*  Discards any keys that have been pressed.  */

keyhit()
{
  char ch;

  while (!kbhit());
  do { ch = getch(); }
    while (kbhit());
}


/*-----  press  -------------------------------------------------*/
/*  Displays a message and waits for a key to be pressed.  */

press(type, row, col)
int type, row, col;
{
  if (type == PICK)
    _settextposition(row, col);
  else
    _settextposition(row, 28);
  _outtext("Press any key to continue");
  keyhit();
}


/*-----  center  ------------------------------------------------*/
/*  Writes a string centered on line 'pos'.  */

center(pos, s)
int pos;
char *s;
{
  int col;

  col = (80 - strlen(s)) / 2;
  _settextposition(pos, col);
```

```
  *(cp+1) = 0;
  *p = 0;
  for (;;)
    {
      switch (*cp = getch())
        {
          case KBRUB:
                        if (i>0)
                          {
                            i--;
                            *(--p) = 0;
                            mvback();
                          }
                        break;
          case KBESC:
                        return(0);
                        break;
          case KBENTER:
                        return(i);
                        break;
          case KBZERO:
                        *cp = getch();
                        if (*cp == KBDEL || *cp == KBLEFT)
                          if (i>0)
                            {
                              i--;
                              *(--p) = 0;
                              mvback();
                            }
                        break;
          default:
                        if (i<max)
                          {
                            _outtext(cp);
                            *p++ = *cp;
                            *p = 0;
                            i++;
                          }
                        break;
        }
    }
}


/*-----  mvback  ----------------------------------------*/
/*  Moves back the cursor one space on the screen.  */

mvback()
{
```

```
  struct rccoord crd;

  crd = _gettextposition();
  _settextposition(crd.row, crd.col-1);
  _outtext(" ");
  _settextposition(crd.row, crd.col-1);
}


/*-----  highlt  ------------------------------------------------*/
/*  Writes a highlighted a string.  */

highlt(row, col, p, len)
int row, col, len;
char *p;
{
  short tcolor, bcolor;

  bcolor = _getbkcolor();
  tcolor = _gettextcolor();

  _settextcolor(bcolor);
  _setbkcolor((long)tcolor);
  out_item(row, col, p, len);
  _settextcolor(tcolor);
  _setbkcolor((long)bcolor);
}


/*-----  rowcol  ------------------------------------------------*/
/*  Returns the row and column number of a symbol element.  */

int rowcol(n, row, col)
int n;
int *row, *col;
{
  gatelst *p;
  sdt *s;

  p = gateptr + n;

  if (p->sym < 2) return(0);

  *row = p->elem / 4000;
  *col = p->elem % 1000;

  s = sdtptr + p->sym - 1;

  if (s->brows > 1) return(2);
  if (s->bcols > 1) return(1);
```

```
  return(0);
}

/*************************************************************************

        view.c

        Programmer:     Walter Slipp
                        Spring 1989

        Contains routines for screen display of subgraphs

*************************************************************************/

#include <bios.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "gfx.h"

extern header();
extern center();
extern cursor();
extern press();

extern int chk_pixel();

extern struct con_slot con_tab[];


/*-----  draw  ------------------------------------------*/
/*  Cycles through the print buffers until screen is full.  */

draw(a, b)
int a, b;
{
  int i, n, x, y;
  con_slot *cs;
  int res;

  if (a == xlast && b == ylast) return;

  for (n=0, cs = &con_tab[0]; n<CONMAX; n++, cs++)
    cs->cfree = TRUE;

  _clearscreen(_GVIEWPORT);
```

```
  edge(a, b);

  i = 0;
  for (x=frow; x>0; x--)
    {
      if (x < a - scr_lim - 1) return;
      clr_buff();
      for (y=0; y<GWIDTH; y++)
        if (n = chk_loc(x, y))
          p_logic(n, y*43+9, x);
      old_paths(x);
      new_paths(x);
      if (x <= a) paint(i++, b);
    }
  clr_buff();
  res = first_elem();
  paint(i, b);
  if (res && (x < scr_lim))
    {
      clr_buff();
      p_line(5*GSLOTW+24, 0, 5*GSLOTW+24, 9);
      paint(i+1, b);
    }
}


/*-----  edge ----------------------------------------------*/
/*  Draws line around boundaries of subgraph if visible.  */

edge(x, y)
int x, y;
{
  int xmax, n;

  xmax = (int) min((x+1)*RBUFSIZE*8+22, xlim-1);

  if (y == 0)
    {
      _moveto(0, ylim-1);
      _lineto(xmax, ylim-1);
    }

  if (y == PWIDTH-ylim)
    {
      _moveto(0, 0);
      _lineto(xmax, 0);
    }
```

```
  if (x == frow)
    {
      _moveto(0, 0);
      _lineto(0, ylim-1);
    }

  if (x <= scr_lim)
    {
      _moveto(xmax, 0);
      _lineto(xmax, ylim-1);
    }
}


/*-----  paint  ------------------------------------------------*/
/*  Sets pixels on screen corresponding to print buffer.  */

paint(lev, b)
int lev, b;
{
  int i, j, k, m, n, c;
  int top;

  c = lev*RBUFSIZE*8;
  for (i=0; i<RBUFSIZE; i++)
    for (j=b, m=ylim-1; m; j++, m--)
      if (p_buffer[i][j])
        for (k=0, n=8*i; k<8; k++, n++)
          if (chk_pix(j, n)) _setpixel(c+n, m);
}


/*-----  view_layout  ------------------------------------------*/
/*  Initializes screen display and handles cursor movement.  */

view_layout()
{
  gatelst *p;
  int i, x, y;
  int xskip, yskip, xhskip, yhskip;
  int pchar, shift;
  int num;

  _clearscreen(_GCLEARSCREEN);

  if (gphmode == _NOGRAPH)
    {
      header("View Subgraph");
      center(10, "You may not access the display routines.");
```

```
    press(C, 14, 0);
    return;
  }

if (locptr == NULL)
  {
    header("View Subgraph");
    center(10, "No subgraph has been selected.");
    press(C, 14, 0);
    return;
  }

scr_lim = xlim / (RBUFSIZE*8) - 1;

_setvideomode(gphmode);

frow = first_col();
x = frow;
y = 0;
xlast = -1;
ylast = -1;

xskip = scr_lim - 1;
yskip = ylim - GSLOTW;
xhskip = 1;
yhskip = GSLOTW;

draw(x, y);
xlast = x;
ylast = y;

for (;;)
  {
    pchar = (_bios_keybrd(_KEYBRD_READ) & 0xff00) >> 8;
    shift = _bios_keybrd(_KEYBRD_SHIFTSTATUS) & 0003;
    switch (pchar)
      {
        case UP:
              if (shift)
                y = (y+yhskip > PWIDTH-ylim) ? PWIDTH-ylim : y+yhskip;
              else
                y = (y+yskip > PWIDTH-ylim) ? PWIDTH-ylim : y+yskip;
              draw(x, y);
              break;
        case DOWN:
              if (shift)
                y = (y-yhskip < 0) ? 0 : y-yhskip;
              else
                y = (y-yskip < 0) ? 0 : y-yskip;
              draw(x, y);
```

```
            break;
        case LEFT:
            if (shift)
              x = (x+xhskip > frow) ? frow : x+xhskip;
            else
              x = (x+xskip > frow) ? frow : x+xskip;
            draw(x, y);
            break;
        case RIGHT:
            if (shift)
              {
                x = (x-xhskip < scr_lim) ? scr_lim : x-xhskip;
                if (x > frow) x = frow;
              }
            else
              {
                x = (x-xskip < scr_lim) ? scr_lim : x-xskip;
                if (x > frow) x = frow;
              }
            draw(x, y);
            break;
        case ESC:
            _clearscreen(_GVIEWPORT);
            _setvideomode(_DEFAULTMODE);
            return;
            break;
        default:
            break;
      }
    xlast = x;
    ylast = y;
  }
}
```

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

        gfx.h

        Programmer:     Walter Slipp
                        Spring 1989

        Contains all defined constants within SUBGRAPH

*********************************************************************/


/* Constants */

#define SINGLE          0       /* single line box */
#define DOUBLE          1       /* double line box */

#define DEFAULT         -1

#define RBUFSIZE        12      /* number of rows in print buffer */
#define CBUFSIZE        480     /* number of columns in print buffer */

#define GWIDTH          11      /* maximum number of gates in a column */

#define LRSIZE          7       /* number of rows for a logic rep */
#define LCSIZE          31      /* number of cols for a logic rep */

#define BUFW            RBUFSIZE*8-1        /* max pixel in print buff */
#define PAD             (RBUFSIZE-LRSIZE)*8  /* empty pixels in p buff */

#define IPATH           0       /* input connection path */
#define OPATH           1       /* output connection path */

#define LBUFSZ          200     /* input buffer character limit */
#define NHASHLN         10      /* ten numbers per line in hash table */
#define EMPTY           -1

#define VCC             -1
#define GND             -2
#define FLT             -4

#define R               0       /* right justification */
#define L               1       /* left justification */
#define C               2       /* center justification */
#define PICK            3       /* choose location */

#define SDTINP          123     /* input symbol */
#define SDTOUT          128     /* output symbol */
#define SDTMEM          125     /* memory symbol */
```

```
#define _NOGRAPH        -3      /* no graphics adapter present */
#define MAXITEM         24      /* max items in a menu */

#define TRUE            1
#define FALSE           0

#define TCURSOROFF      0x2020


/* Scan codes */

#define UP      72
#define DOWN    80
#define LEFT    75
#define RIGHT   77
#define ENTER   28
#define SPACE   57
#define ESC     1
#define RUB     14
#define DEL     83


#define KBZERO  0
#define KBLEFT  75
#define KBRUB   8
#define KBDEL   83
#define KBESC   27
#define KBENTER 13


/*  box line values */

#define HZ      0
#define VT      1
#define NW      2
#define NE      3
#define SE      4
#define SW      5


/*  logic constants  */

#define PASS    0
#define AND     1
#define CND     2
#define NAND    3
#define OR      4
#define XOR     5
#define NOR     6
#define DFF     7
```

```
#define DFCS    8
#define INPUT   9
#define OUTPUT  10

#define GQUIT   11
#define FNDGATE 12
#define ALL     13


/* Other defines */

#define LINELN  80

#define DTYPE   6

#define MAXIN   8               /* maximum number of inputs displayed */

#define SDENSITY        75      /* single density printing */
#define DDENSITY        76      /* double density printing */

#define PWIDTH          480     /* maximum column print width */

#define CONMAX  30

#define GSLOTW  43      /*  pixel width for each row */


#define ERR_MEM 0


/*  Structures  */

typedef struct  gatelst {
        int     type;
        int     sym;
        int     elem;
        char    placed;
        char    xpos;
        char    ypos;
        int     numin;
        int     *cin;
        int     numout;
        int     *cout;
} gatelst;


typedef struct symtab {
        char    symname[11];
} symtab;
```

```
typedef struct sdt {
        int     strwnum;
        int     code;
        int     bcols;
        int     brows;
        int     uid;
} sdt;


typedef struct sqrtb {
        int     step;
        int     csff;
} sqrtb;


typedef struct con_slot {
        int     cn_from;
        int     cx;
        int     cn_to;
        char    cdir;
        char    cfree;
} con_slot;


/*  global variables  */

extern struct videoconfig vc;

extern int      gphmode;
extern int      txtmode;

extern char     curfile[];

extern char     *lgcsym[];

extern symtab   *symptr;
extern int      numsym;
extern sdt      *sdtptr;
extern int      numsdt;
extern sqrtb    *sqrtptr;
extern int      numsqrt;
extern gatelst  *gateptr;
extern int      numgate;
extern int      numread;

extern int      *locptr;
extern int      xmap;
extern int      xlevel;
extern int      xlast;
extern int      ylast;
```

```
extern char     p_buffer[RBUFSIZE+1][CBUFSIZE];

extern int      din;

extern int      xlim;
extern int      ylim;
extern int      scr_lim;
extern int      frow;

extern int      nn_acc;
```

REFERENCES

[1] Chu, Y. H., "Why Do We Need Hardware Languages?", Computer, Dec. 1974, pp. 18–22.

[2] Hill, F. J., and Peterson, G. R., Digital Systems: Hardware Organization and Design, John Wiley and Sons, Inc. 3rd Edition, 1987.

[3] Hill, F. J., and Peterson, G. R., Introduction to Switching Theory and Logical Design, John Wiley and Sons, Inc. 3rd Edition, 1981.

[4] Masud, M., "Modular Implementation of a Digital Hardware Design Automation System", PhD Dissertation, University of Arizona, 1981.

[5] Chen, Duan-Ping, "Compilation of Combinational Logic Units for Universal AHPL", Master's Thesis, University of Arizona, 1981.

[6] Digital Equipment Corporation, "Programming in VAX FORTRAN", Maynard, Massachusetts, September 1984.

[7] Microsoft Corporation, "Microsoft FORTRAN Compiler Language Reference", 1987.

[8] Texas Instruments, "The TTL Data Book Volume 1", 1984.

[9] Kohavi, Z., Switching and Finite Automata Theory, McGraw-Hill, Inc., 2nd Edition, 1978.

[10] American National Standards Institute (ANSI), American National Standard FORTRAN X3.9-1978, 1978.

[11] Su, S. Y. H., "Hardware Description Language Applications", Computer, June 1977, pp. 10–13.