

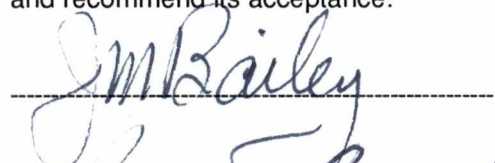
To the Graduate Council:

I am submitting herewith a thesis written by Syed K. Hussain entitled "A Computer Aided Design Software Package for Logic Diagram Generation." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.



Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:




Accepted for the council:



Vice Provost
and Dean of the Graduate School

**A COMPUTER AIDED DESIGN SOFTWARE PACKAGE
FOR LOGIC DIAGRAM GENERATION**

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Syed K. Hussain
December 1987

ACKNOWLEDGEMENTS

The author is very grateful for the invaluable guidance, patience and helpful advice of Dr. Don W. Bouldin, committee chairman, as well as committee members Dr. Robert E. Bodenheimer and Dr. M. Bailey.

Thanks are due to Hal Beck and Maureen Delcroix for proof reading and final preparation of this thesis.

Special appreciation and thanks go to my parents Mr. and Mrs. Mahboob Hussain for their invaluable support and encouragement throughout my education.

To my wife Shampa, a special thank you for having the strength to persevere and for being a constant source of encouragement and inspiration throughout my graduate career.

ABSTRACT

The development of a 'C' language program named LOGDESIGN, which generates a circuit diagram from a list of components is described. The component list, called a net-list, is supplied by the user interactively using the keyboard. Once a net-list is read, the software package manipulates the information and generates the logic diagram on the screen. The package makes some logical decisions about the best placement of modules to enhance clarity. The program operates on the HP-9000 series computer. It employs the HP-9000 advanced graphics package to take care of all graphics tasks. A detailed software development discussion is included along with the source code for future extensions on the project.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
2. RELATED WORK IN LOGIC DIAGRAM GENERATION	3
The ALDGS System	3
CAD System of IIT	5
3. ALGORITHM DEVELOPMENT	7
Design Considerations	7
Data Input	8
Module Selection for Placement	8
Slot Selection on Screen	10
4. SOFTWARE DEVELOPMENT	16
Function Details	16
5. A SAMPLE SESSION	34
Step 1	35
Step 2	35
6. RESULTS	40
7. SUMMARY AND CONCLUSIONS	52
LIST OF REFERENCES	54
APPENDIX	56
VITA	80

LIST OF FIGURES

FIGURE	PAGE
1. Examples of Module Types	12
2. Main Program Flowchart	13
3. Function Mod_num()	17
4. Function read_info()	18
5. Function read_connection()	20
6. Function Get_slot()	21
7. Function sum()	22
8. Function sort()	23
9. Function placing()	24
10. Function connection_matrix()	26
11. Function F_list()	27
12. Function F_list_max()	28
13. Function search_gi()	29
14. Function Global_input()	30
15. Function Prefer_neighbor()	32
16. Function draw_box2()	33
17. Sample Sessions Circuit	39
18. Test Circuit 1	41
19. Test Circuit 2	44
20. Example Circuit 1	48
21. Example Circuit 2	49
22. Example Circuit 3	50

FIGURE	PAGE
23. Example Circuit 4	51

CHAPTER 1

INTRODUCTION

Logic diagrams are one of the most efficient ways to represent digital systems. These pictorial representations may be drawn as a system level block diagram or as a detailed circuit diagram [1]. Such diagrams are necessary to understand the operation of a digital system from a conceptual and an applications point of view. Although the standard logic symbols used in schematics are universal, there are many design criteria which may be implemented to represent a logic diagram.

The preparation of logic diagrams has always been a very tedious and time consuming effort when performed manually. With the increasing use of LSI and VLSI [2] circuits, the number of circuit components in a diagram has increased dramatically. So, automated procedures to generate circuit diagrams have become desirable.

A software package for the automatic generation of schematic diagrams can be very useful in the area of computer aided design (CAD). Schematics may be produced at various levels during the process of synthesizing a detailed circuit design from a high level description [3]. This circuit diagram provides sufficient information to the circuit designer to have at least a conceptual idea of the workings of the circuit. This thesis describes the development of a CAD software package to generate schematic diagrams from an user supplied net-list, i.e., list of components .

A study of other research work in the area of logic diagram generation is presented in Chapter 2. A brief overview of the different techniques are also discussed. Algorithm development and problem overview are explained in Chapter 3. The different design criteria

are also discussed. Chapter 4 presents the software development on the HP 9000. A brief discussion of all the function modules is included. A typical session with LOGDESIGN is presented in Chapter 5 to illustrate the steps involved in generating logic diagrams. Chapter 6 contains some examples of circuits generated by LOGDESIGN. Some test results are included also. Chapter 7 summarizes the work and gives some conclusions. Limitations of LOGDESIGN and future improvements are also discussed. For future expansion of the modules, a source code for the modules is listed in the Appendix.

CHAPTER 2

RELATED WORK IN LOGIC DIAGRAM GENERATION

Several researchers have designed different processes in order to develop schematic diagram generators. Although the primary focus in the electronic industry has been directed toward the physical layout of PC boards, the objectives for PC board layout and logic diagram generation are quite distinct. The main objective of PC board layout is to minimize the length of interconnections or the number of wires or the area of the layout. On the other hand, for logic diagrams the main objective is to have a diagram which illustrates the operation of the circuit clearly. In other words, the user should be able to follow the flow of the circuit easily and quickly without being distracted by a maze of blocks and lines. A brief discussion of two schematic diagram generators is given below.

2-1. The ALDGS system

The ALDGS system is an experimental logic diagram generator. It was developed at the University of Waterloo and used FORTRAN as its carrier language. It takes a pseudo-boolean input description of a logic circuit and generates a logic diagram[1]. The program does four operations simultaneously in order to achieve this. The operations are:

a) Parsing

This section of the program acquires the circuit description from the user. The desired logic diagram size of each logic element is also supplied. Once a basic symbol is defined, it is stored in a library. For different size diagrams, the drawing area differs in size, as does the size of each component. The number of components that can be placed on a page is dependent

upon the user specification of full-size or reduced drawings. The ALDGS offers the option of storing signal names and component data.

b) Partitioning and Placement

In ALDGS, a placement of a particular module is sought so that the logic flows from left to right and reversed connections are minimized [1]. Once a module is selected for placement, the objective is to place the module to the right or to the left of a preferred neighbor module. In this way, signal flow is maintained from left to right. The selected module chooses a "preferred neighbor" which is the rightmost source or the leftmost sink module. A grid location is selected based on the preferred neighbor grid location. If the chosen grid location is already taken, the module selects a secondary preferred neighbor and tries to find a grid location corresponding to the secondary neighbor. The process continues until an empty grid location is found. Once that is done, the module is placed in the location and the program returns to select a second candidate module. If all the grids available on a page are occupied, the system creates another page and starts the process of laying out the unplaced modules. The process continues until all the modules are placed and the drawing is complete.

c) Components Placement Refinement

Once all the modules are placed and the initial drawing is done, a placement refinement algorithm takes over. Now the objective is to place the highly connected modules as close as possible and minimize the interconnection length [1]. This is completed by vertical and horizontal shifting of modules for placement refinement. Pin assignments and signal name assignments are also done at this time.

d) Routing

The ALDGS system employs a channel router to accomplish routing [4]. To minimize wire bends, the router examines each connection and, on the basis of the connection end points, assigns it a connection class. The connections are then broken down into horizontal

and vertical segments. These segments are processed to minimize wire crossings.

The ALDGS system implements most of the design goals of logic diagram generation in a very effective manner. It produces clear and legible diagrams. The page breaking feature makes complex circuit design possible. The authors did not mention the time required for generating such diagrams.

2-2. CAD system of IIT

An automatic schematic diagram generator was developed at the Indian Institute of Technology, Delhi. This package was written in PASCAL and runs on the HP 1000 machine. Input to the schematic generator is a design in structural form, which essentially specifies a set of modules and the interconnections between them [3]. A database contains the graphical information about each module. The input design specifications are given in terms of the following factors:

- a set of modules
- a set of nets
- a set of terminals for each module
- the direction of each terminal; in, out or bi-directional
- terminal orientation; fixed, movable, or semi-movable

Once a circuit is defined, the circuit diagram is drawn in two phases. The first phase, called the logical phase [3], begins with the identification of the nets and how they may be laid out. Then modules are placed logically with the same design criteria in mind. Routing is added to complete a rough sketch. The geometric phase [3] is a refinement of what was done earlier. A detailed description follows.

a) Logical Phase

First the bus nets [3] are analyzed and a decision is made about their layout. The goal

is to have left to right signal flow. If signal loops (i.e. , feedback paths) are present, the nets containing them are set aside to be dealt with later. The next step is to determine a logical placement of the modules. Once a module is selected for placement, it is assigned a location such the bus nets selected in the earlier step have signal flow from left to right. Routing channels are set aside while placing the modules. A channel router is used to accomplish routing.

b) Geometric Phase

The geometrical phase is a schematic refinement step. This step executes manipulations in order to achieve the optimum circuit design. The grids are now transformed to uniform sizes, with each slot being large enough to hold the largest module. The objective now is to minimize interconnection length between modules. If more than one module may be accommodated in one slot, it is done. After placement is redone, routing is invoked once more. The goal now is to minimize wire crossings. A channel router algorithm is employed for the second time. After routing, terminals are put on modules such that neighboring module connections will be on a straight line. The last operation which is performed is compaction [3]. Now, neighboring modules and bus nets are moved closer together without allowing any overlapping. The result is a much more concise drawing which is aesthetically beautiful.

The schematic diagram generator developed at IIT accomplishes most of the goals of schematic diagram generation. The placement improvement algorithm in the geometric phase is extremely effective in making the drawings clear. However it has no option to include signal names and can operate only on one page schematics.

CHAPTER 3

ALGORITHM DEVELOPMENT

There are many design constraints that should be taken into consideration while developing a schematic diagram generator. As observed in chapter two, the most important aspect of a logic diagram is clarity. The following is a list of design considerations which LOGDESIGN attempted to address.

a) Clarity: To get a graphics output which represents the circuit information clearly is the main design objective of LOGDESIGN. In order to accomplish that, the software package goes through an algorithm which determines the placement for each module. A formatted output is generated. Once a slot location is assigned to a module, it is drawn in the slot. The size of each module depends on the number of pins the module has. For a module with a large number of pins, a large one is drawn.

To enhance clarity, a left to right signal flow is desired. A placement algorithm is included which improves the spatial layout of modules on the sheet. The algorithm executes several routines in order to achieve optimal placement of the modules. The goal is to place the heavily connected modules close together without giving up left to right signal flow. The placement routine utilizes the seeded [5] placement technique .

b) User Friendliness: The data entry to the package is made as user friendly as possible. At time of data entry, the package prompts the user for input. The instructions are fairly simple to follow.

c) Portability: The software package LOGDESIGN was developed using the 'C' programming language with portability in mind. Almost all of the main program and all but one of

the function modules are machine independent. The function which draws the circuit modules on the screen utilizing the HP 9000 graphics package is the only instance in which commands specific to the HP 9000 machine have been used. A portion of the main program which calls this function also uses HP commands.

d) Future Enhancement: To have room for future additions and corrections to the package, it was designed employing the structured programming approach. All the different tasks are broken into several functions. In the future, if any changes are to be made or extra features added, the application programmer may make the changes to a function or add independent functions without affecting the existing program.

With these design goals in mind, the algorithm for LOGDESIGN was developed. The development process may be divided into three distinct parts. A discussion about each section is given below.

i) Data Input

Data entry to LOGDESIGN is done on the HP 9000 terminal. During the running of the program, the user is prompted for the different types of input, i.e., module names, connection details, connection quantity, etcetera. The details are discussed in chapter five (sample session). Correct data entry is a requirement for the proper usage of LOGDESIGN.

ii) Module Selection For Placement

The placement algorithm for LOGDESIGN is the key to a clear circuit diagram. The algorithm employs the "seed" placement algorithm similar to the one employed by the MIMOLA software package developed by Honeywell [5]. The module which has the most connections to other modules is selected first as the "initial seed". The other modules are then selected one by one based on their connectivity to the initial seed. These modules attempt to build a cluster around the selected module.

During the development of LOGDESIGN, the order of module selection was

determined by the weight of connectivity of modules. The sequence of modules selected for placement is a logical decision based upon the notion of how "strongly" [6] unplaced modules are bound to the modules already placed. Let us define some terminology which will be used to develop the placement algorithm.

S	the weight (number of modules) associated with pin number "a" of module
k	total number of modules in system (maximum 9)
i	module selected for placement
j	iteration cycle
A	set of modules selected after j th iteration
B	set of modules not selected after j th iteration
ip	input pin number
op	output pin number

The initial seed is determined by taking the summation of the modules connected to all the output pins of each module and then choosing the maximum. We denote the total number of modules connected to module l by

$$P(l) = \begin{cases} S, & \text{for all } l \\ 0, & \text{for all } op = 0 \end{cases} \quad (\text{Eq. 3 - 1})$$

The summation value matrix is called the P matrix.

After the formation of the P matrix (Eq. 3-1), the next step is to determine the maximum entry in the matrix. The program sorts each value to find the maximum. In case of a tie, the first module with the maximum P matrix entry is chosen arbitrarily. The module chosen is the initial seed. The seed module will be placed on the middle of the drawing area and all the other

modules will be placed to build a cluster around it.

Once the selection and placement of the first module is complete, the selection of the next module depends upon its connectivity to the initial module. The algorithm tries to find the measure of connectivity between modules which are placed and the ones which are unplaced. Before a new module "I" is selected for placement, a number $F(I)$ is selected for each of the unplaced modules. If c is the number of connections between unplaced module "I" and placed module p , then we have

$$F(I) = \begin{cases} c & \text{where } I \text{ not equal to } p \\ 0, & \text{where } I \text{ equal } p \end{cases} \quad (\text{Eq. 3-2})$$

Once the computation of the F list for each module is complete, the task now is to find the maximum entry. The module with the maximum F list value will be the next module selected for placement. To resolve a tie situation, LOGDESIGN takes the modules with the maximum F list values and sorts their P lists (Eq. 3 - 1) values to compute a maximum. If a tie situation still persists, module selection is done arbitrarily. After the selection of the second module, the program selects another unplaced module. This time the F list (Eq. 3 - 2) is formed with respect to both placed modules. The algorithm applied is the same as before. The selection process is determined using the same algorithm applied earlier. The process iterates until all the modules are selected for placement. For a system of eight modules, in addition to the initial seed module, six iterations are required.

iii) Slot Selection on Screen

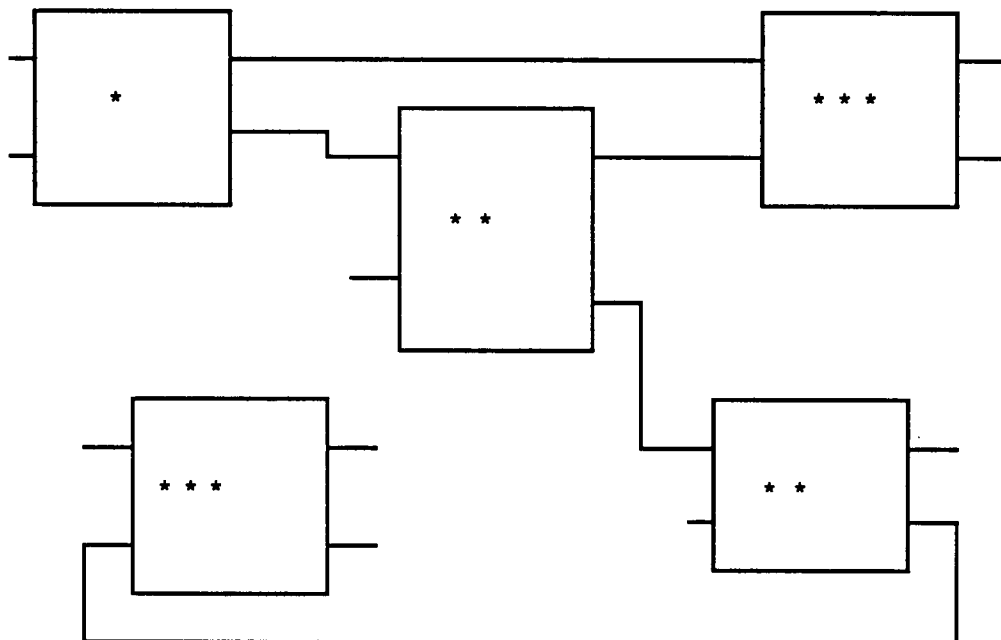
The display area on the screen is divided into nine distinct slots. The space separating each slot is the same as the width of a module. This allows for adequate routing space for interconnections.

Once a module is selected for placement, LOGDESIGN proceeds to classify it as one

of three types. The first type is called the "Global Input" type. A global input module may be defined as the source module to the schematics from an outside circuit. It may also be defined as the input module to the circuit from an external environment. If the module selected is of the global input type, LOGDESIGN proceeds to place it to the leftmost slot available to insure left to right signal flow. A "Global Output" module is the opposite of a global input type. It is the type of module which has only output connections to the external environment. All its output connections are to devices outside the schematic created by LOGDESIGN. Figure 1 explains graphically the different types of modules. The goal of LOGDESIGN now is to place a global output module to the rightmost location possible to insure left to right signal flow.

When a module is identified as neither the Global Input nor the Global Output type, it is defined as a general type module. The algorithm used to find a slot location for this type of module is quite different from what was employed earlier. Now the slot location of the unplaced module depends upon the location of its "preferred neighbor". The "preferred neighbor" module is the unplaced module which has the maximum number of connections with a placed module. If a preferred neighbor module exists, LOGDESIGN tries to determine whether the selected unplaced module is a source or a sink to the preferred neighbor module. If it is a source, LOGDESIGN searches to find an empty slot to the left of the preferred neighbor. For a sink, the search is to the right. If no empty module is available, an arbitrary search is initiated to find the first empty slot.

After a module is assigned a slot, the program selects another module for placement. This selection procedure was discussed earlier. It then proceeds to select a slot location for the module. The process continues until all the modules are placed. The last step is the graphics phase, in which the modules are drawn on the screen in their respective slot locations. Figure 2 gives the flowchart of the main program of LOGDESIGN.



* - Type Global Input module

** - Type regular module

*** - Type Global Output module

Figure 1. Examples of Module Types

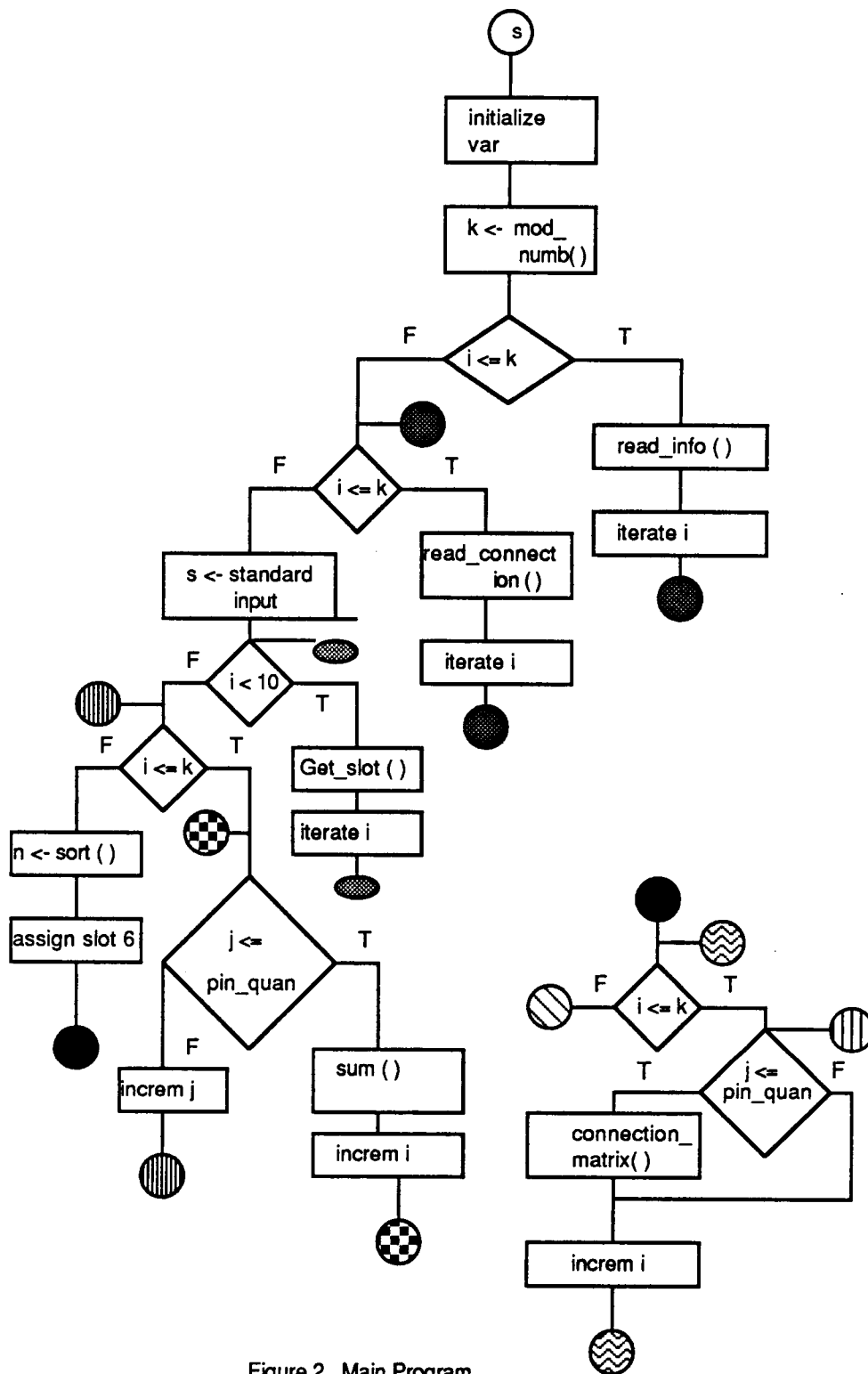


Figure 2. Main Program

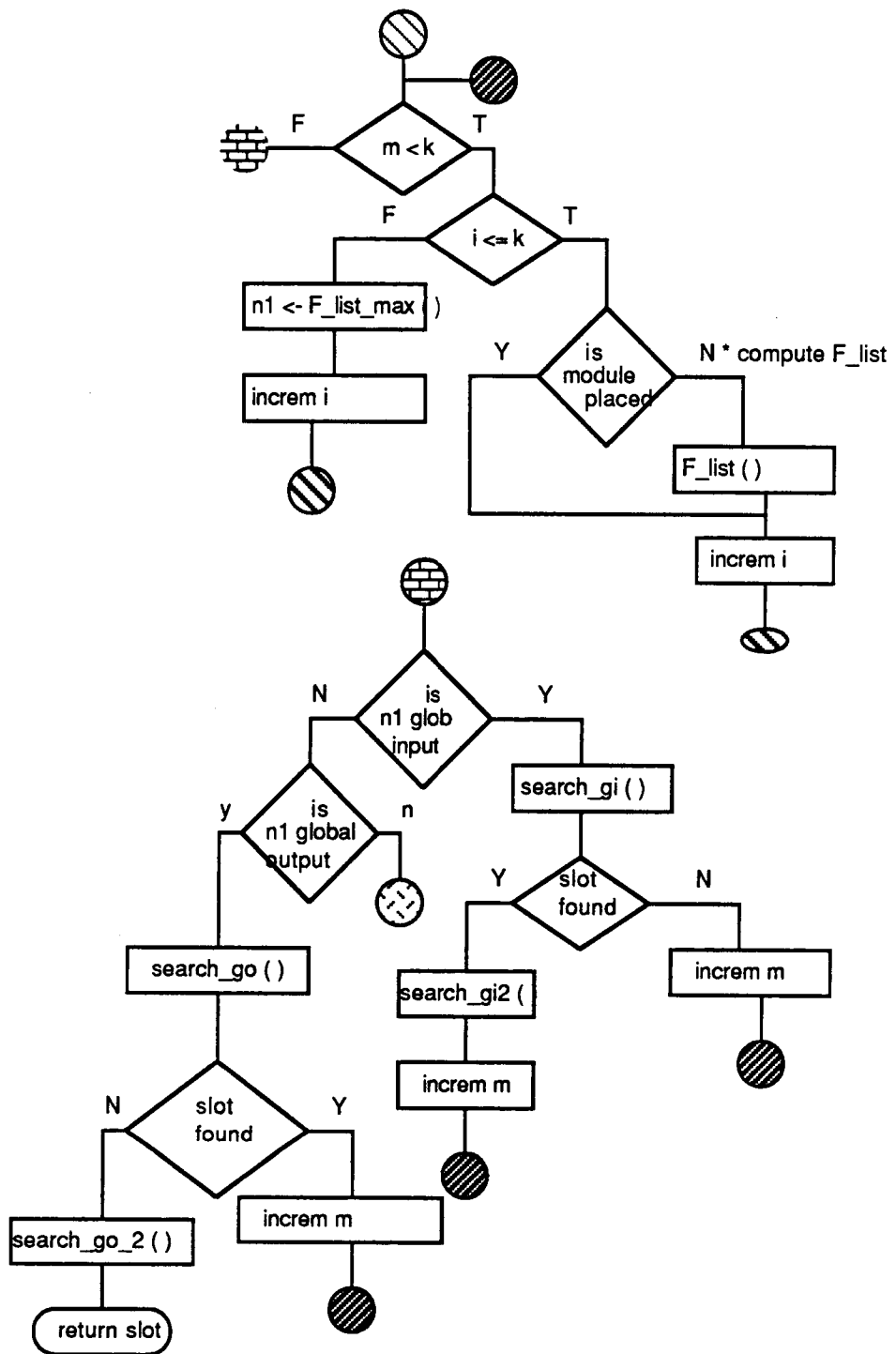


Figure 2. Continued

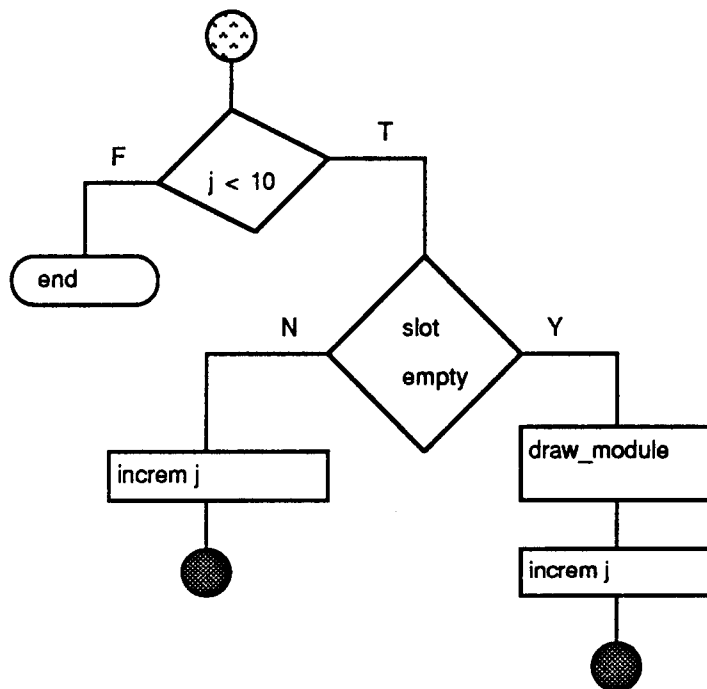
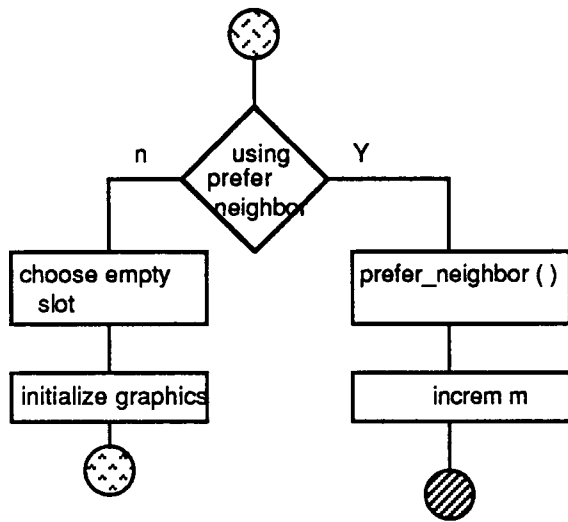


Figure 2. Continued

CHAPTER 4

SOFTWARE DEVELOPMENT

The software package LOGDESIGN is designed using the 'C' programming language, since 'C' is highly portable. Also 'C' allows the applications programmer to design software using structured programming techniques [7]. A large program may be broken down into several distinct modules. This gives the main program the option to call on an individual function when it needs to use it. If an error occurs, it is limited to that function only and the other modules remain unaffected. Modular design makes the task of debugging program flow simple.

The different data structures employed by LOGDESIGN are included in the header file "local2.h". Data for each individual module is stored as a structure [7]. A variable of type structure may be defined as a group of variables stored as one. This is an attribute of the 'C' language. This type of variable is used to store different information about the same variable together. Hence they may be accessed one at a time and the total number of variables is reduced in the system.

Given below is a detailed description of each of the function modules. Their source codes are listed in the appendix.

a) function mod_numb()

This function prompts the user for the total number of modules in the system. Once it is entered, The value is returned to the invocation environment. The program then clears the standard input buffer [7]. Figure 3 shows the flowchart of these operations.

b) function read_info()

The function gets the name of a module and the input and output pin quantity. A variable of type structure module is passed to the function to store the information. There is no return value. Figure 4 shows the flowchart of these operations.

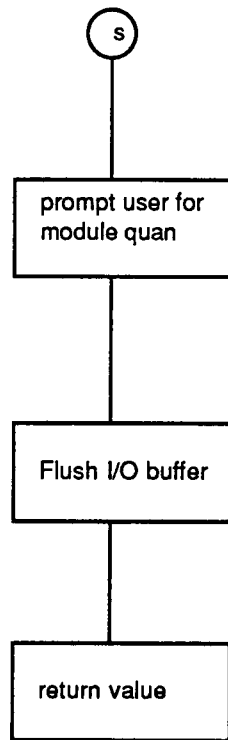


Figure 3. Function `mod_numb()`

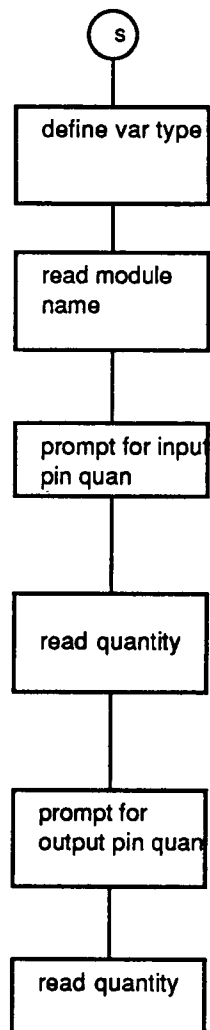


Figure 4. Function `read_info()`

c) function read_connection()

This function gets the net-list of the system from the user. For each output pin (load pin), it prompts the user for the number of modules it is connected to, the names of the connected modules and the signal names associated with them.

The function accepts as an input the chip type module, the module number and an array to store the quantity of modules that are connected to. Figure 5 contains the flowchart.

d) function Get_slot()

Function Get_slot() determines the slot location on the screen for each grid. It stores the x and y coordinate values in a structure of type slot1. There are nine slots for module placement on the screen. Figure 6 gives a flowchart of these operations.

e) function sum()

The function is used to find the summation of the number of modules each chip is connected to on the load side. This value is used later to choose the initial module for placement. Ties are also resolved using this value. An array is passed to the function which stores the computed value for each module. Figure 7 depicts the flowchart.

f) function sort()

This function determines the first module to be chosen. This sorts the values calculated earlier by the sum() function (P_matrix) and tries to locate the maximum. In case of a tie, module selection is arbitrary. The module chosen is the initial seed. Figure 8 is a flowchart of this operation.

g) function placing()

The function placing() keeps track of the modules that are placed. When called upon by the main program, it checks a matrix with the placement information for each module and returns a zero (0) value for a placed matrix and a one (1) value for an unplaced one. Figure 9 is a flowchart of this operation.

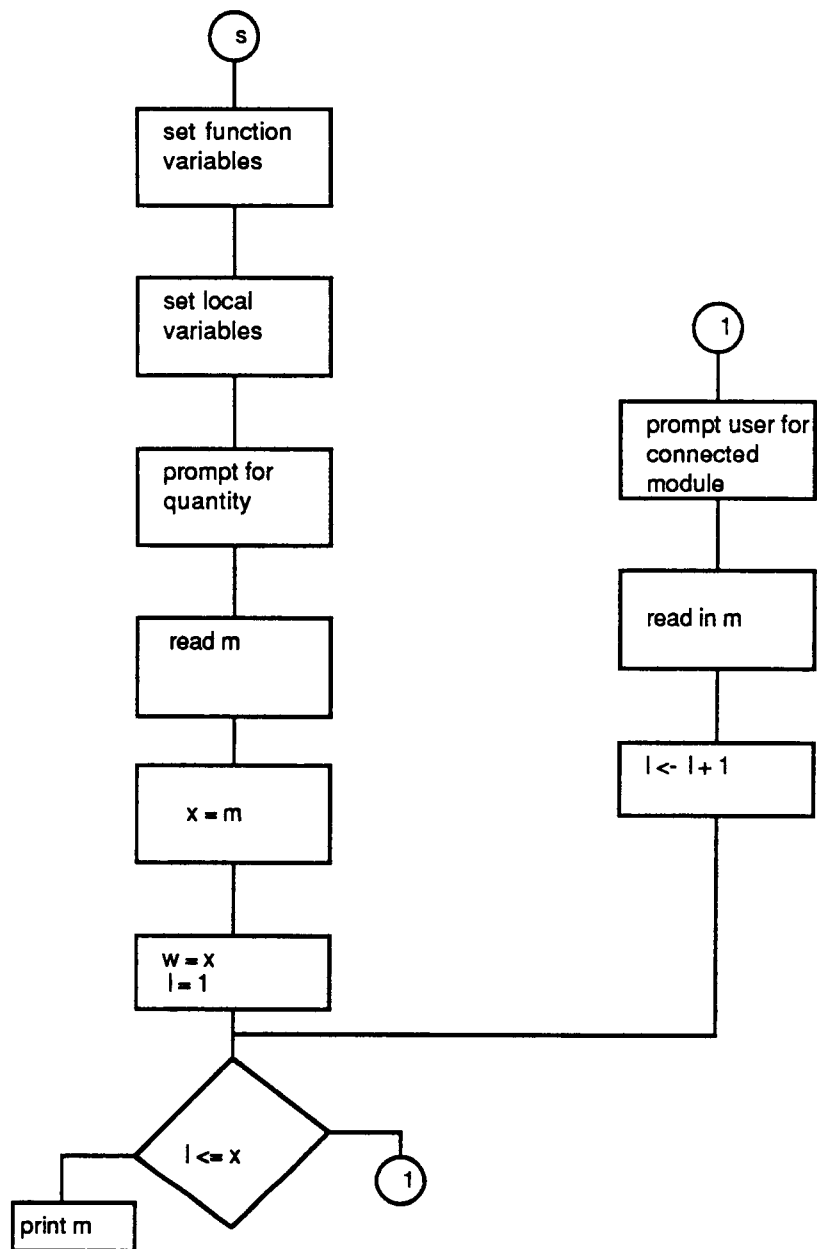


Figure 5. Function read_connection()

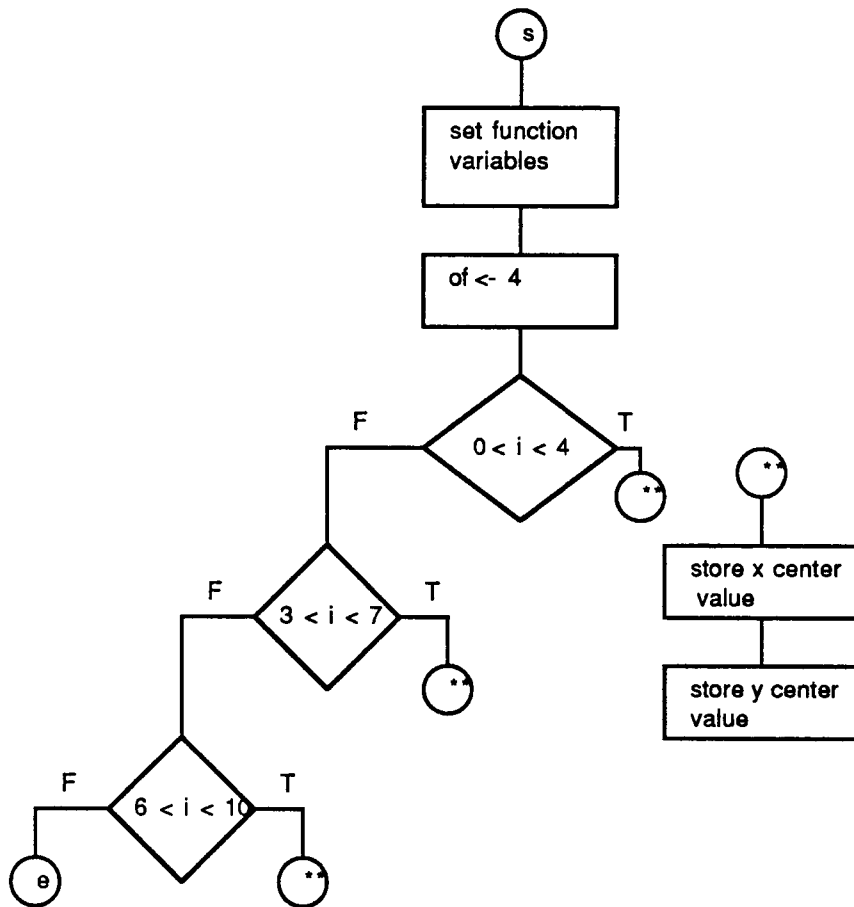


Figure 6. Function Get_slot()

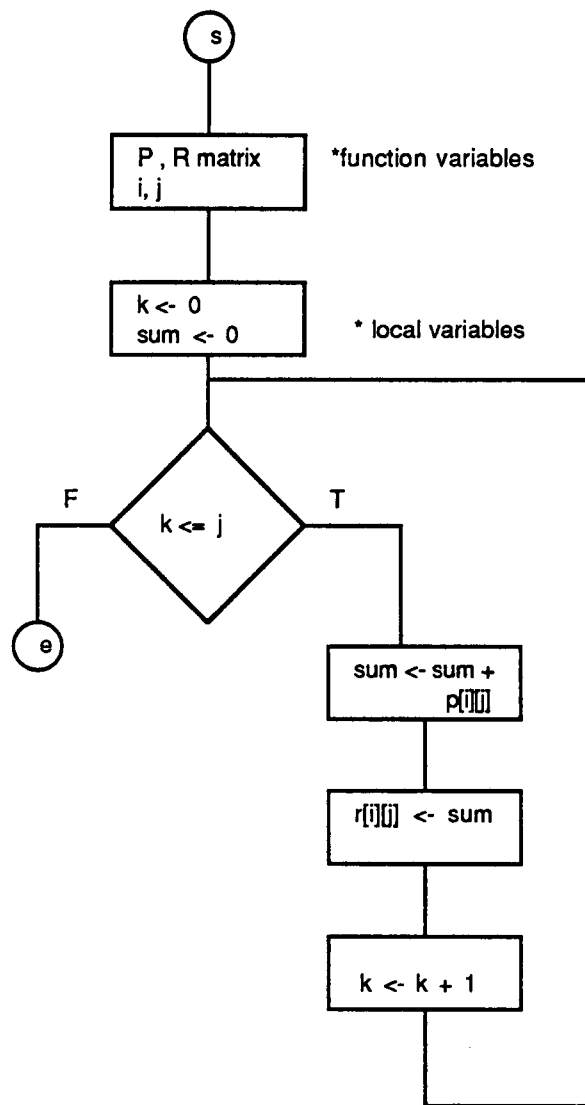


Figure 7. Function `sum()`

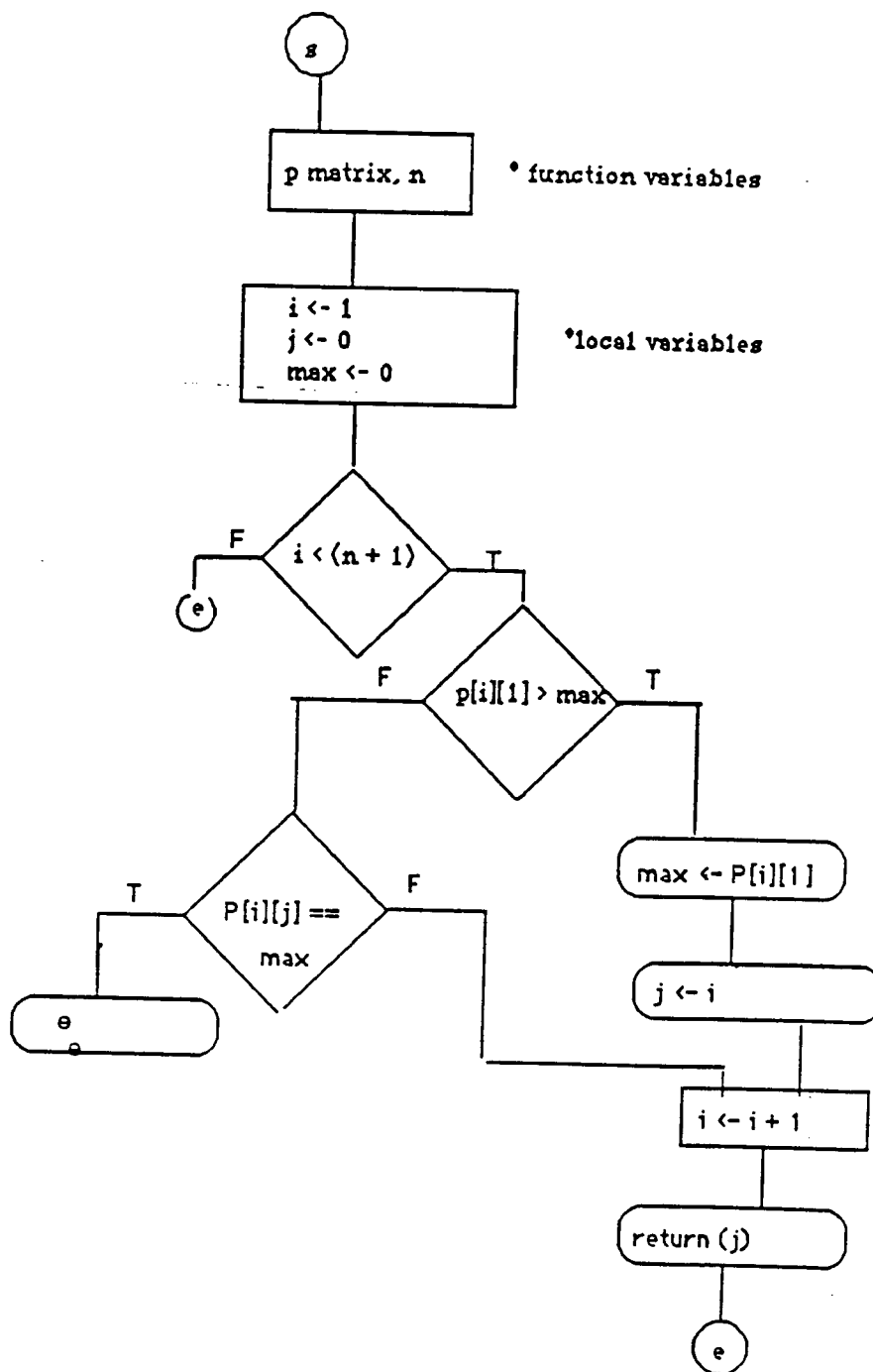


Figure 8. Function sort()

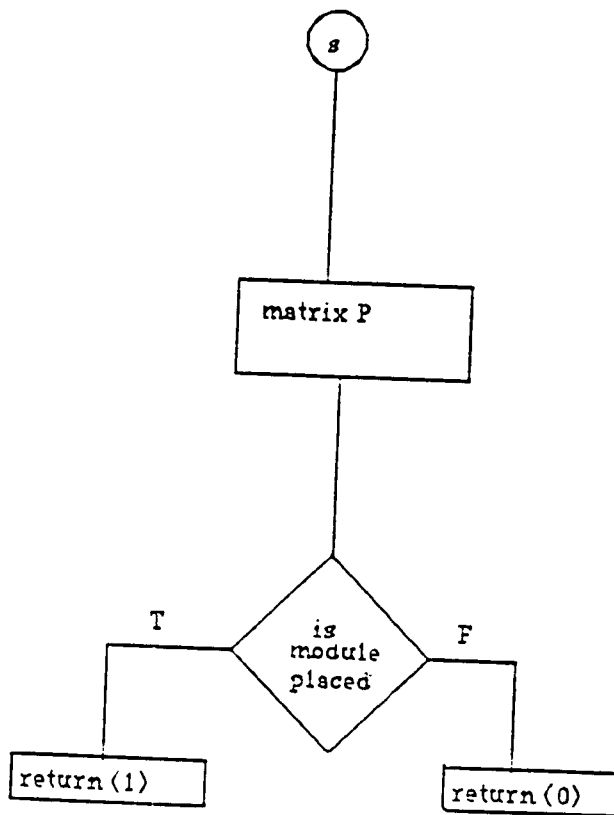


Figure 9. Function Placing()

h) function connection_matrix()

This function computes a matrix from the interconnection between the different modules called the connection matrix. This matrix is the primary source where all the decision about placement and slot selection are based. For N number of modules a N x N matrix is formed. The function also generates a matrix called the output matrix which is used in the preferred neighbor algorithm. Figure 10 gives a flowchart of the function.

i) function F_list()

This function determines the F_list (Eq. 3.2) for module selection. It accepts as an input the connection matrix and determines the F_list entry for each unplaced module. Figure 11 gives the flow diagram of these operations.

j) function F_list_max()

This function module is used to find the module with the biggest F_list value. In case a tie exists, an unplaced module is chosen by referring back to the P_list computed before. The module with the larger P_list value is chosen. Figure 12 gives the flow diagram.

k) functions search_gi(), search_gi_2(), search_go(), search_go_2()

These functions implement the search algorithms to find empty slot locations for global input and output modules. The functions with the _2 extensions employ the same algorithm, but give the secondary choice of slot locations. Once an empty slot location is found, it is returned to the calling environment. Figure 13 gives the flowchart of operations of function search_gi(). The other functions implement a similar search algorithm.

l) functions global_input() and global_output()

These two functions determine the classification of a module, whether it is a global_input or a global_output module. The type is determined by manipulating the output matrix. If either case is true a one(1) value is returned. Otherwise a zero(0) is returned to indicate it is a regular module. Figure 14 gives the flowchart.

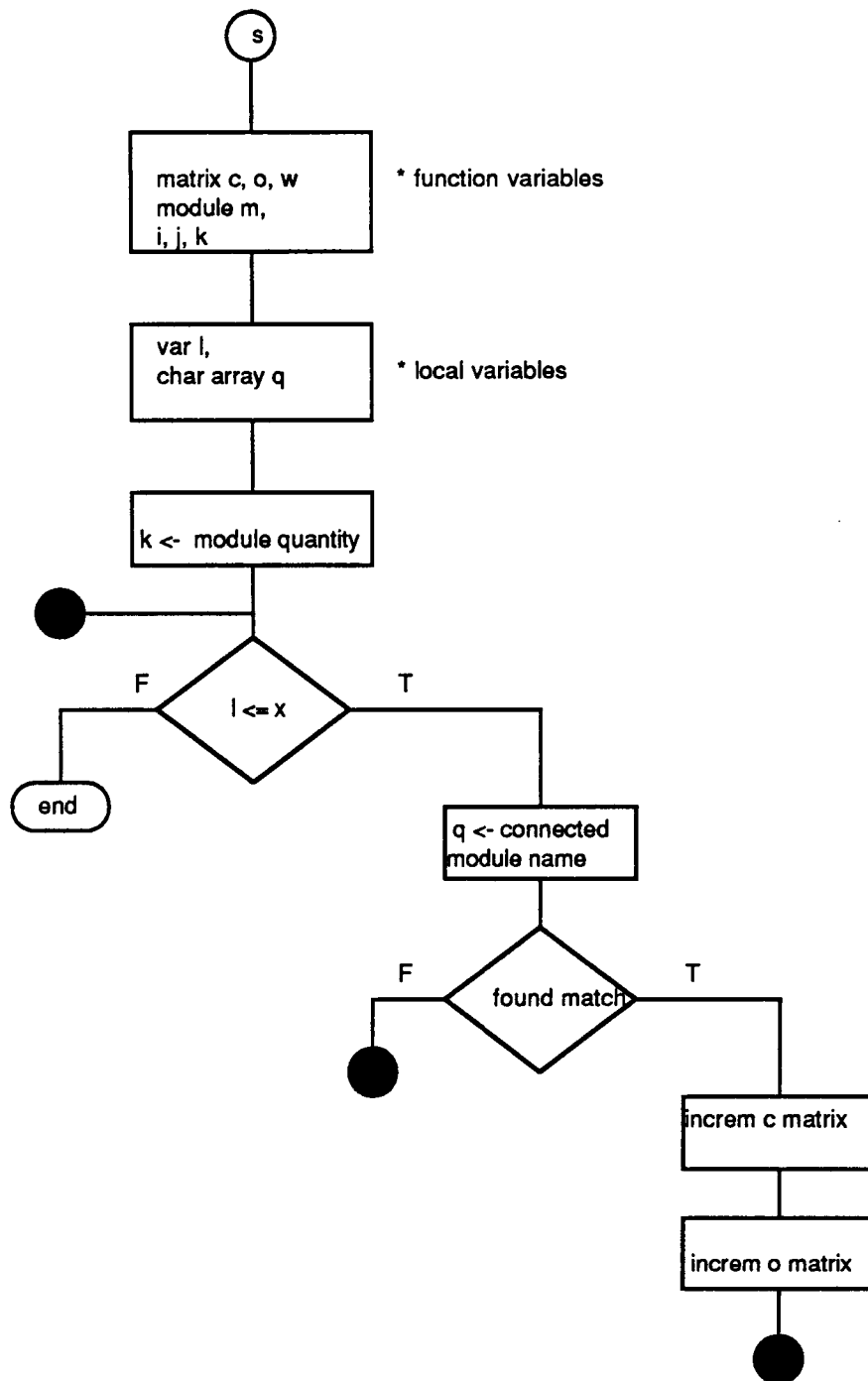


Figure 10. Function connection_matrix ()

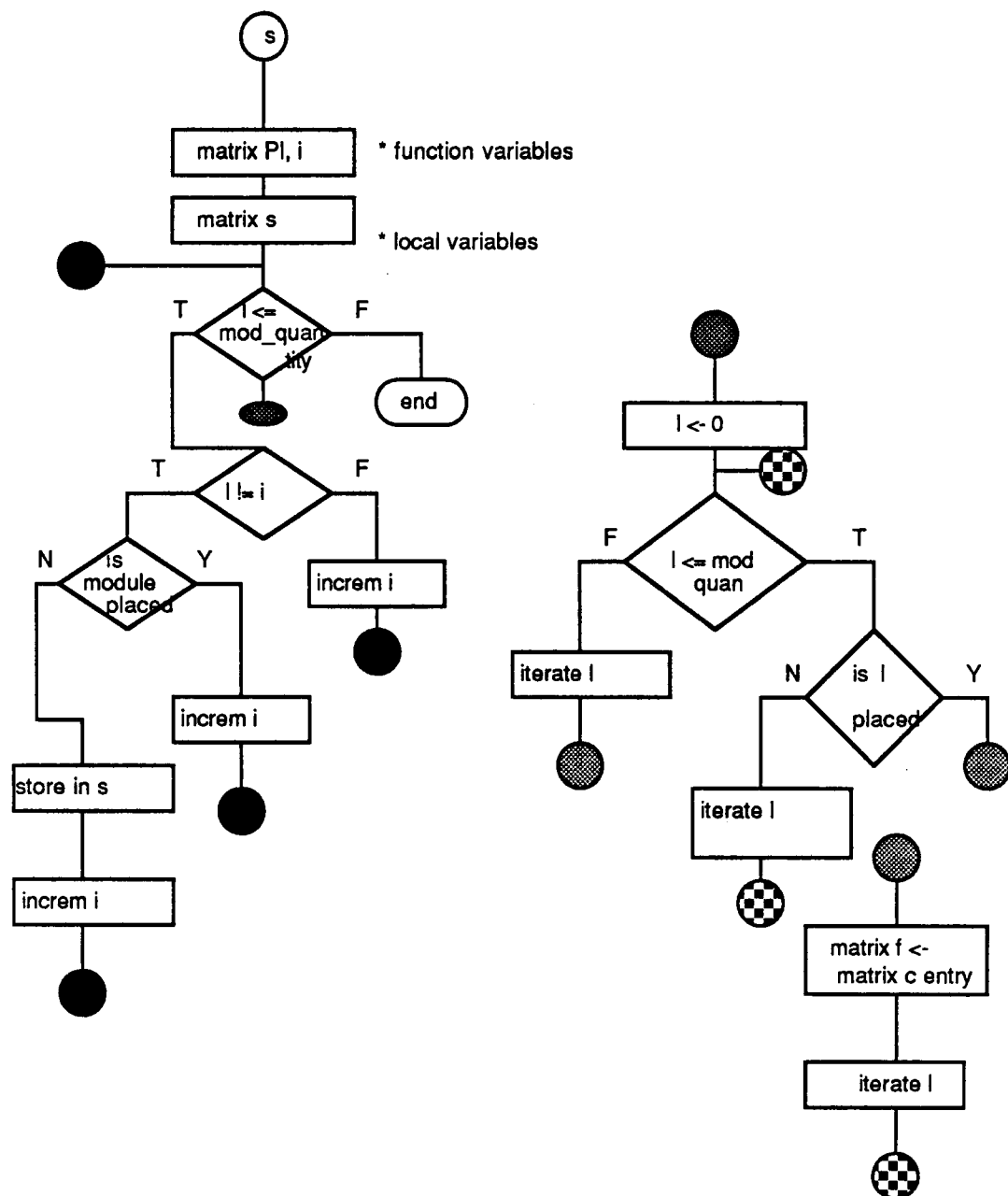


Figure 11. Function `F_list ( )`

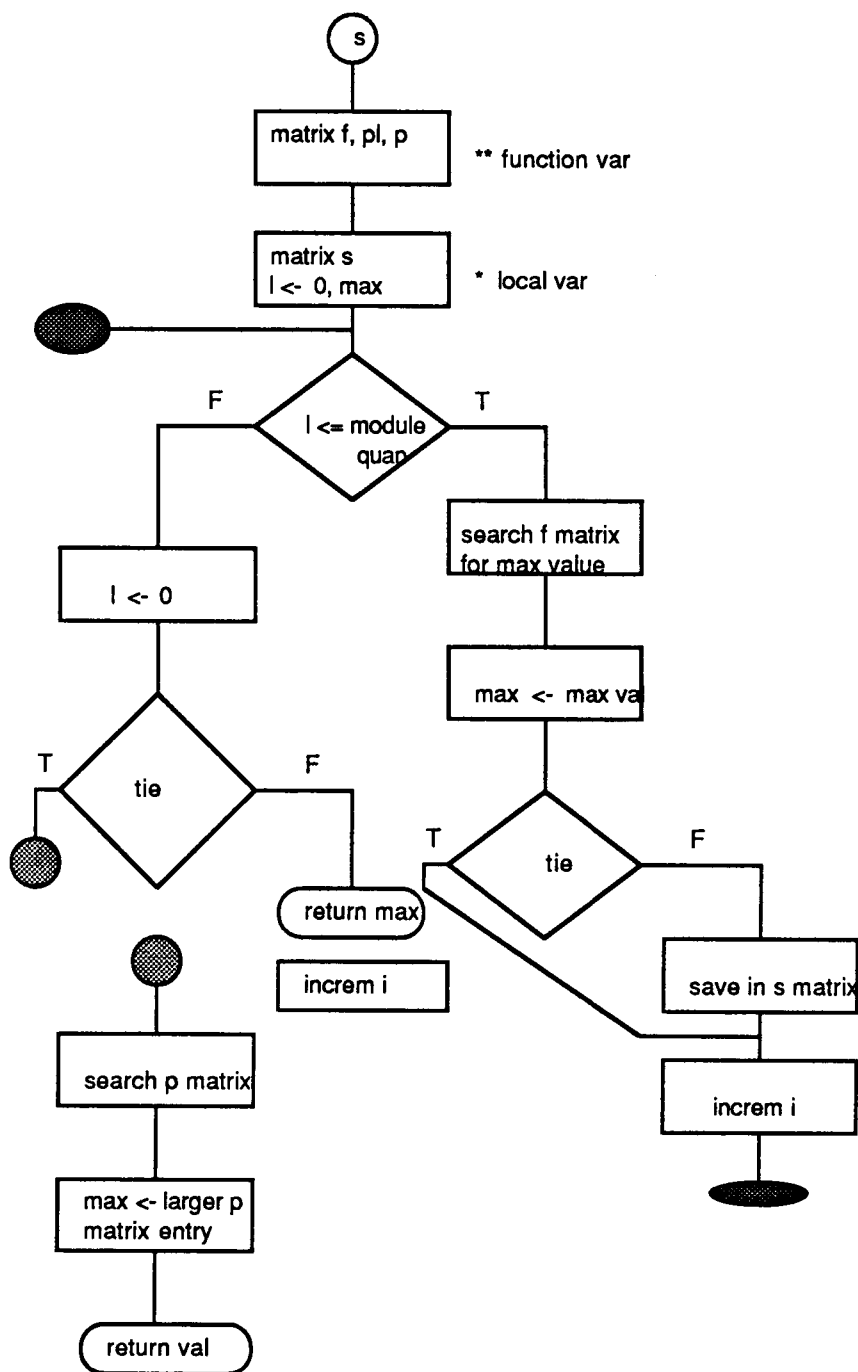


Figure 12. Function F_list_max()

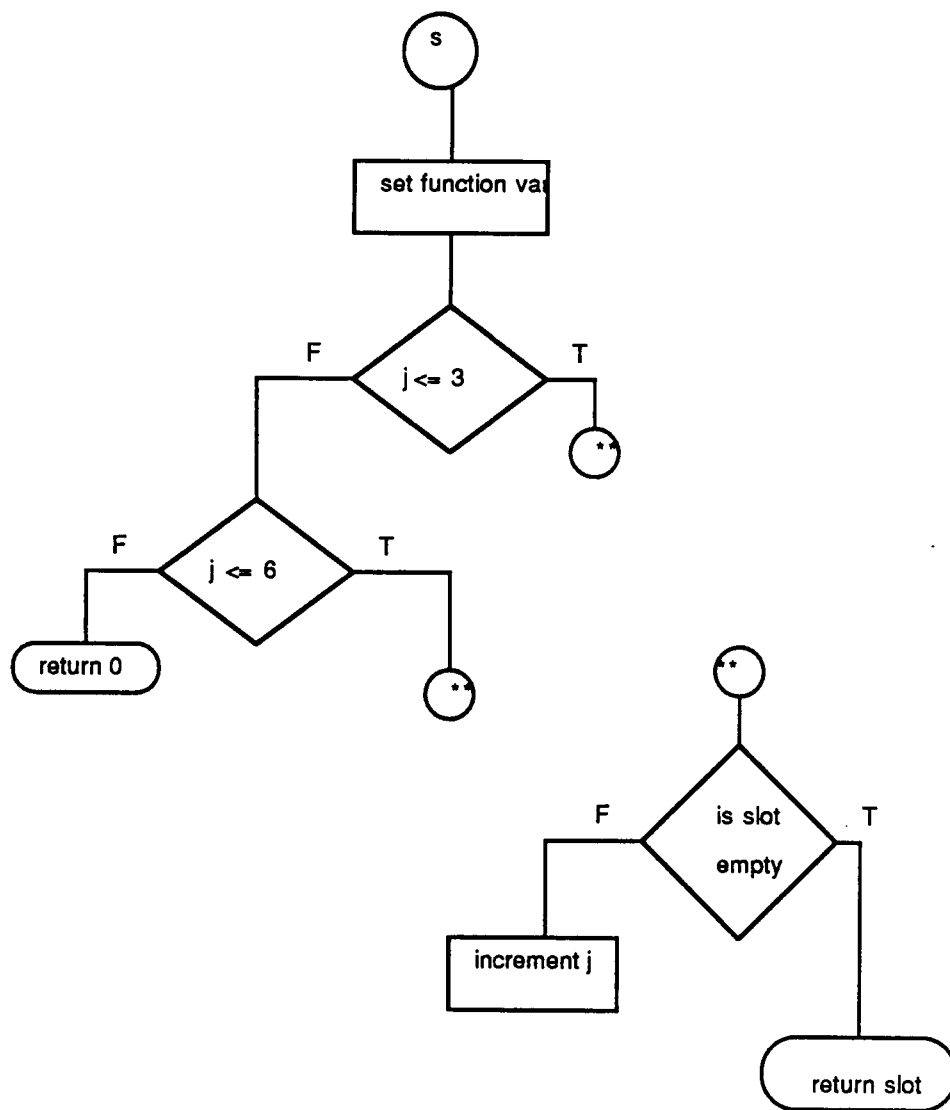


Figure 13. Function `search_gi()`

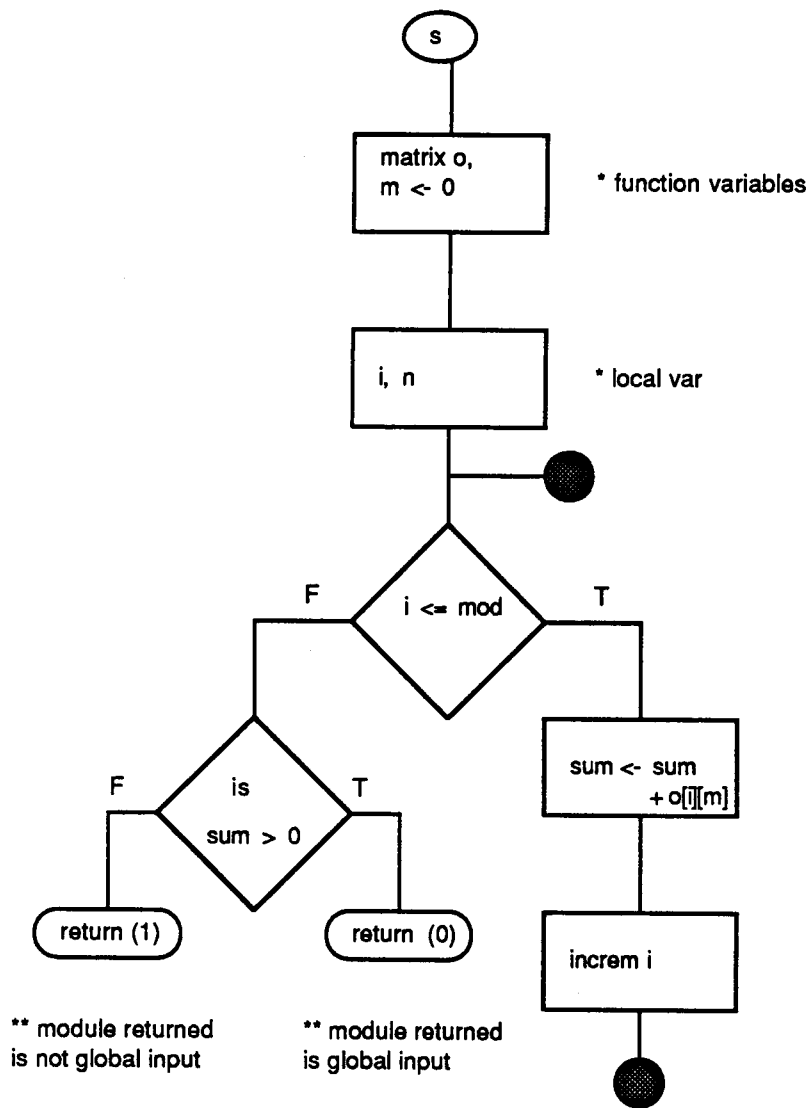


Figure 14. Function Global_input ()

m) function prefer_neigh()

The function is used to find the preferred neighbour for a selected module. The prefer neighbour selection is critical because the module placement is determined by it. The function attempts to find a slot location for the selected module . The choice is to find a location next to the preferred neighbour. If it is successful, the slot location is returned. Otherwise the module is placed in the first empty slot. Figure 15 gives the flowchart of this operation.

n) function draw_box2()

This function employs the HP 9000 Advanced Graphics Package [8] to draw each module on the screen. The function draws each module based on their pin quantity. It uses some commands specific to the HP. To transfer LOGDESIGN in a different machine , this function have to be modified to exclude the commands of the graphics package. Figure 16 gives the flowchart.

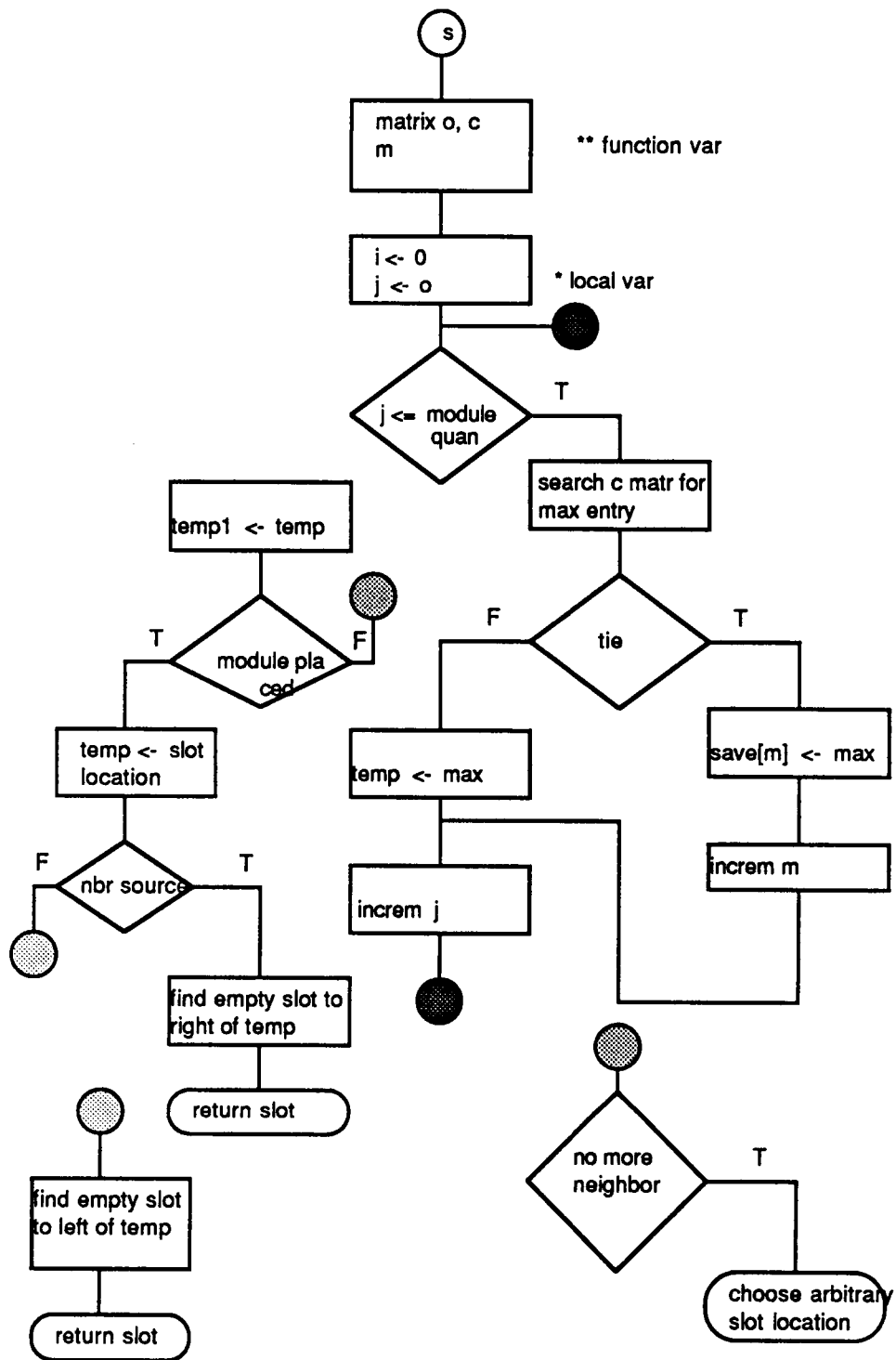


Figure 15. Function Prefer_neighbor()

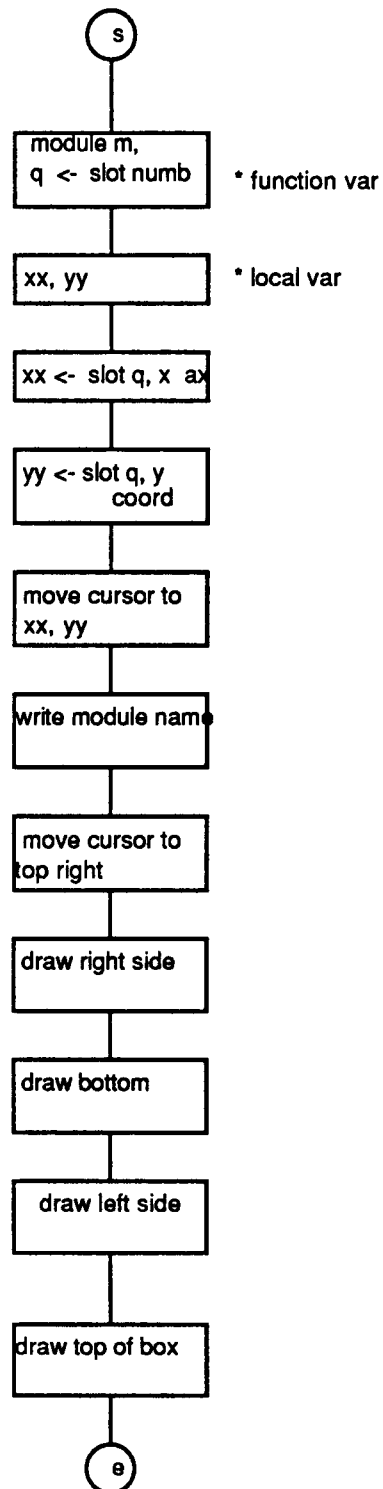


Figure 16. Function draw_box2()

CHAPTER 5

A SAMPLE SESSION

To demonstrate a session with LOGDESIGN, a simple circuit is chosen. The circuit has four modules. The net-list is given below:

<u>Module name</u>	<u>source pins</u>	<u>load pins</u>	<u>load pin no.</u>	<u>connected</u>	<u>signature</u>
	<u>quan</u>	<u>quan</u>		<u>module</u>	
a	2	3	1	c	ac_1
			2	c	ac_2
			3	b	ab_3
b	2	3	1	d	bd_1
			2	d	bd_2
			3	d	bd_3
c	2	3	1		co_1
			2		co_2
			3		co_3
d	2	2	1		do_1
			2		do_2

The signal names associated with the source pins are as follows:

module a:

source pin num: 1

signal name: a1

source pin num: 2

signal name: a2

module b:

source pin num: 1

signal name: ab3

source pin num: 2

signal name: b2

module c:

source pin num: 1 signal name: ac_1

source pin num: 2 signal name: ac_2

module d:

source pin num: 1 signal name: bd_1

source pin num: 2 signal name: bd_2

To generate the logic diagram from this net-list employing LOGDESIGN, a step by step approach is given below:

note: The boldfaced letters are what the user has to type in. The dollar(\$) sign denotes the HP prompt. Hit enter (return key) after data input is done.

STEP 1

To initiate LOGDESIGN type:

\$./LOGDESIGN

STEP 2

\$ enter total number of modules in system: 4

\$ enter chipname(string) & hit return a

\$ enter number of input pins (int) & hit return 2

\$ enter number of output pins (int) & hit return 3

\$ enter chipname(string) & hit return b

\$ enter number of input pins (int) & hit return 2

\$ enter number of output pins (int) & hit return 3

\$ enter chipname(string) & hit return **c**
\$ enter number of input pins (int) & hit return **2**
\$ enter number of output pins (int) & hit return **3**

\$ enter chipname(string) & hit return **d**
\$ enter number of input pins (int) & hit return **2**
\$ enter number of output pins (int) & hit return **3**

***** input pin info module a*****

\$ enter signal name of pin 1: **a1**
\$ enter signal name of pin 2: **a2**

***** input pin info module b*****

\$ enter signal name of pin 1: **ab_3**
\$ enter signal name of pin 2: **b2**

***** input pin info module c*****

\$ enter signal name of pin 1: **ac_1**
\$ enter signal name of pin 2: **ac_2**

***** input pin info module d*****

\$ enter signal name of pin 1: **bd_1**
\$ enter signal name of pin 2: **bd_2**

***** output pin info module a *****

\$ enter quantity of modules pin 1 is connected to: 1

*** note: DO NOT leave any space between the chipname and signal

name, the comma(,) is the separator *****

\$ enter connected chipname(string) a comma(',') signal name (string) and hit enter:

c,ac_1

\$ enter quantity of modules pin 2 is connected to: 1

\$ enter connected chipname(string) a comma(',') signal name (string) and hit enter: **c,ac_2**

\$ enter quantity of modules pin 3 is connected to: 1

\$ enter connected chipname(string) a comma(',') signal name (string) and hit enter:

b,ab_3

***** output pin info module b *****

\$ enter quantity of modules pin 1 is connected to: 1

\$ enter connected chipname(string) a comma(',') signal name (string) and hit enter:

d,bd_1

\$ enter quantity of modules pin 2 is connected to: 1

\$ enter connected chipname(string) a comma(',') signal name (string) and hit enter: **d,bd_2**

\$ enter quantity of modules pin 3 is connected to: 1

\$ enter connected chipname(string) a comma(',') signal name (string)

and hit enter: **d,bd_3**

***** output pin info module c*****

\$ enter quantity of modules pin 1 is connected to: 0

\$ if no modules, enter connected signal name: **co_1**

\$ enter quantity of modules pin 2 is connected to: 0

\$ if no modules, enter connected signal name: **co-2**

\$ enter quantity of modules pin 3 is connected to: 0

\$ if no modules, enter connected signal name: co_3

***** output pin info module d*****

\$ enter quantity of modules pin 1 is connected to: 0

\$ if no modules, enter connected signal name: do_1

\$ enter quantity of modules pin 2 is connected to: 0

\$ if no modules, enter connected signal name: do_2

\$ press 'y' and then hit enter if you want to employ the preferred neighbor algorithm: y

The package LOGDESIGN is ready to be executed. In the first iteration, the seed module is selected by examining the number of connections of each module to other modules and selecting the maximum among them. Module 'a' is selected first and placed on the middle of the screen in slot number six. Then the selection procedure is implemented by the F_list algorithm explained in Chapter 3. Module 'c' is selected next and placed in slot seven. The system recognizes 'c' as a global output module and places it to the right. The next module selected is 'b'. The system attempts to find a preferred neighbor for 'b'. Module 'd' is identified as the preferred neighbor module. Since 'd' is not placed yet, slot selection for 'b' is arbitrary. Slot one is selected. The only module left to be selected is 'd'. This is a global output module and the system selects slot location eight to place it. Figure 17 gives the schematic generated by the sample session.

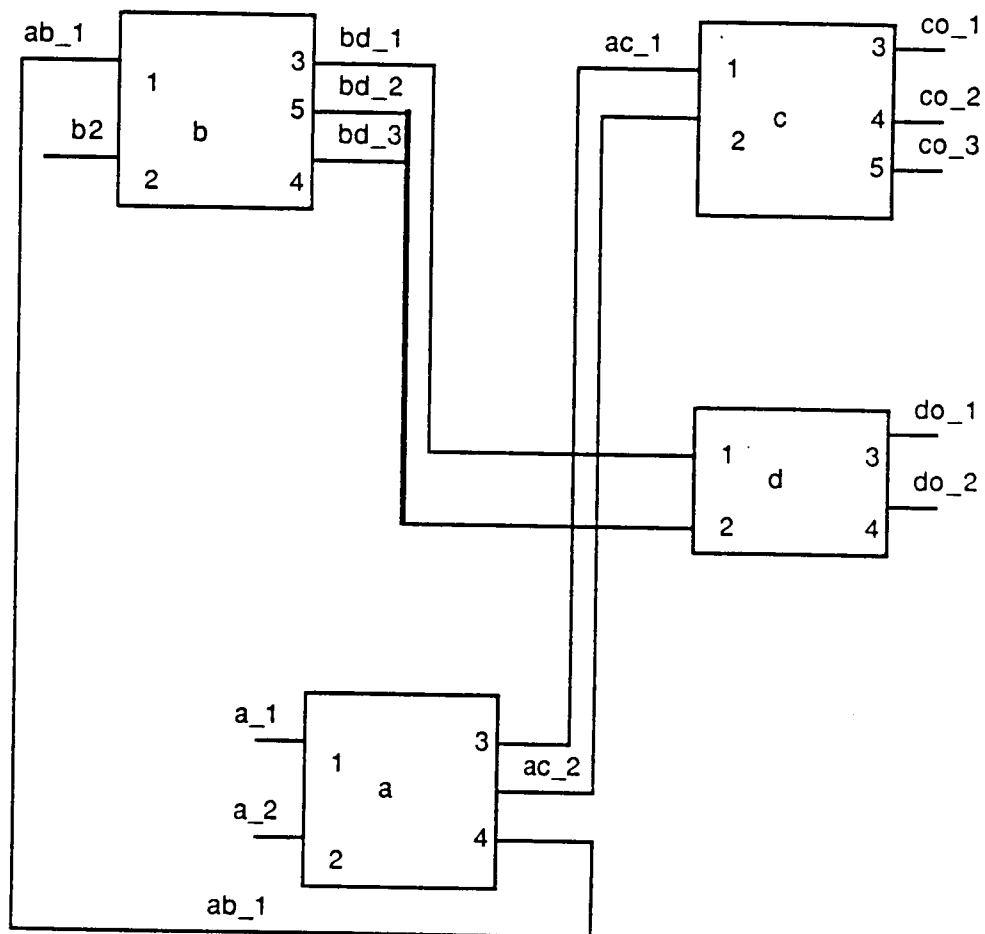


Figure 17. Sample Session Circuit

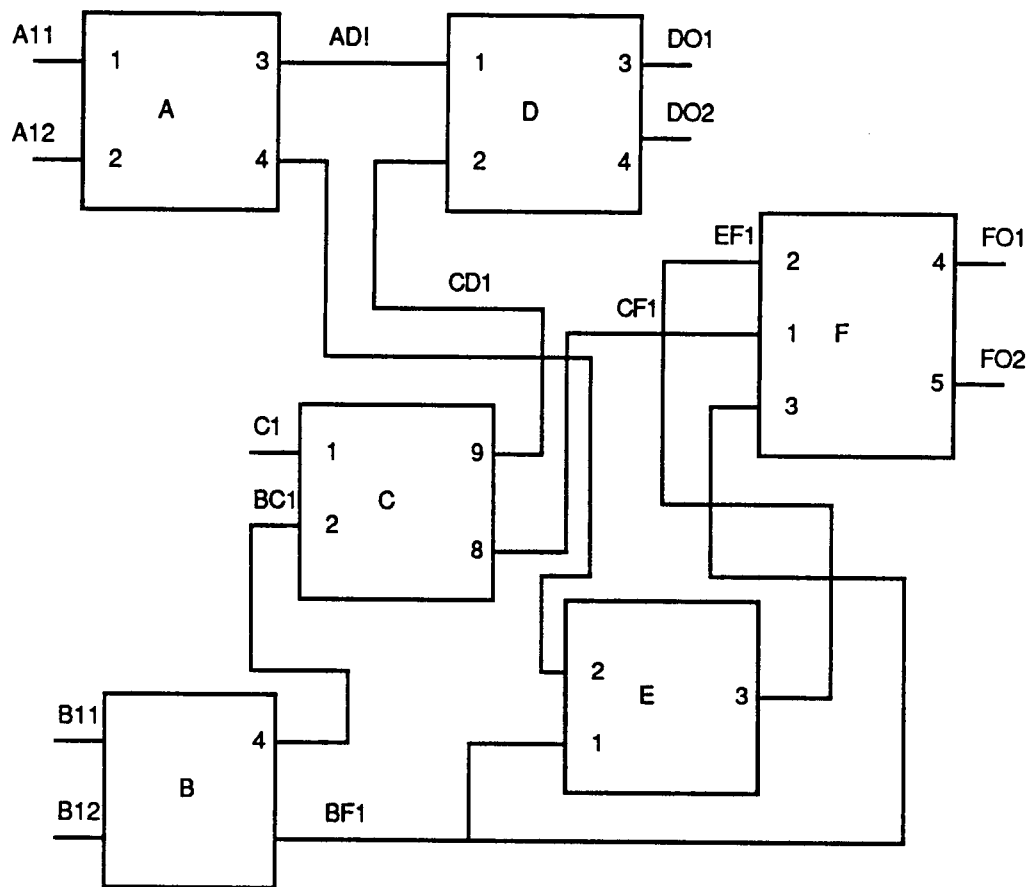
CHAPTER 6

RESULTS

The graphics package LOGDESIGN implements most of the design goals discussed in Chapter 3 quite effectively. The package generates a clear and concise circuit diagram, given the net-list. The diagram which is generated is a formatted schematic with each module occupying a slot determined by the placement algorithm.

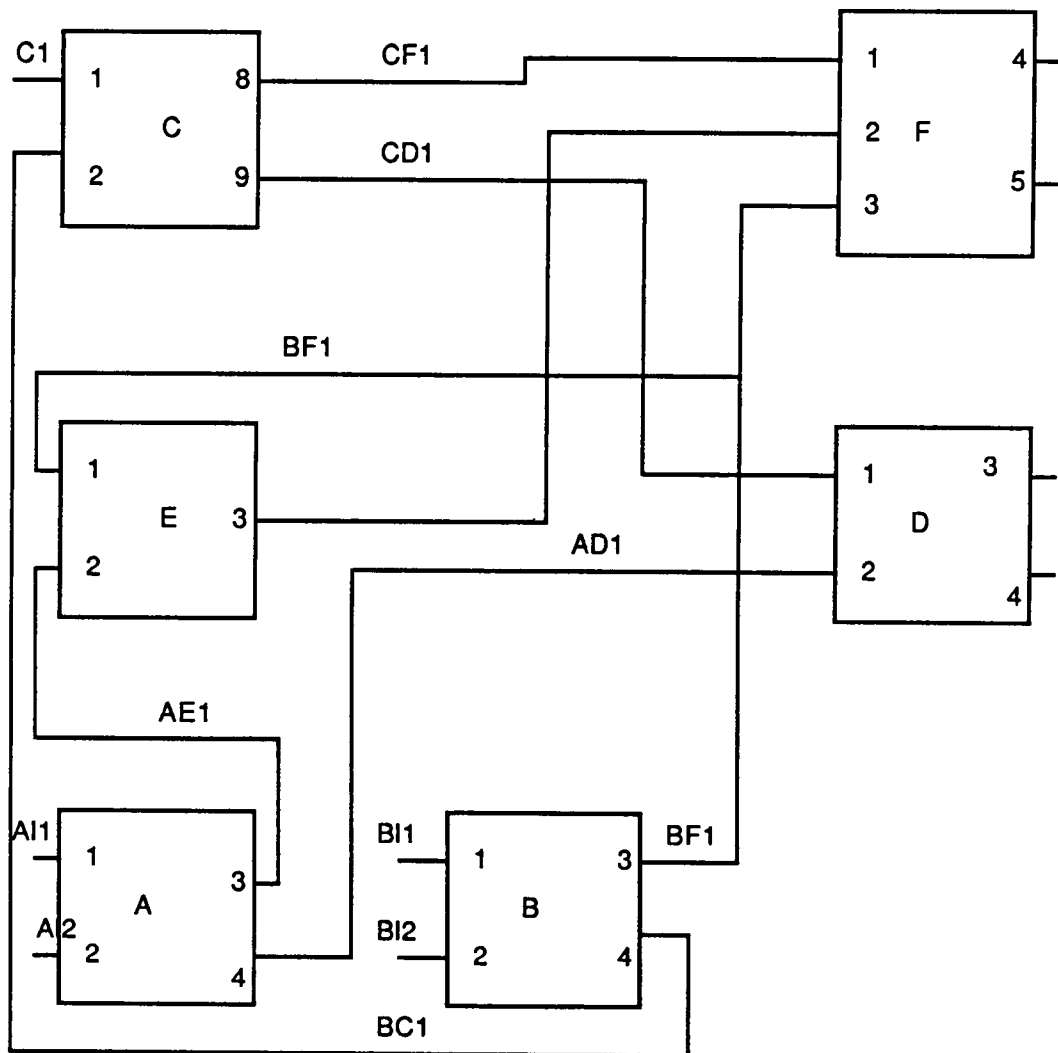
The schematic is generated by placing the modules to have signal flow from left to right. Modules which are input to the system are attempted to be placed in slots to the left side of the screen. The system output modules are placed in slots to the right side of screen. Each module is drawn based on it's pin quantity. Modules with larger number of connections between them are placed together. This reduces the total wire length of connections. Figures 18 and 19 gives two examples of circuits generated by LOGDESIGN. The routing and signal names were added manually to enhance clarity.

Figures 18a and 19a are the graphical representations of two net-lists for the test circuits. They are shown graphically to allow visual comparison with the schematics generated later. Figures 18b and 19b are defined as being the intermediate placement of the modules. In these two cases the preferred neighbor algorithm was not employed. The placement of global input and output modules remains unaffected. But when slot selection for a normal module is called, the first empty slot is chosen. In most cases by not employing the preferred neighbor algorithm, optimum module placement cannot be achieved. Figures 18c and 19c give two examples of optimum circuits generated by LOGDESIGN. In both tests, the initial module selected is the one with the most number of connections to other modules and it is placed in the middle row of the screen. For Figure 18c the initial module is module b. The selection of the remaining modules are based on the connection between the placed and the unplaced



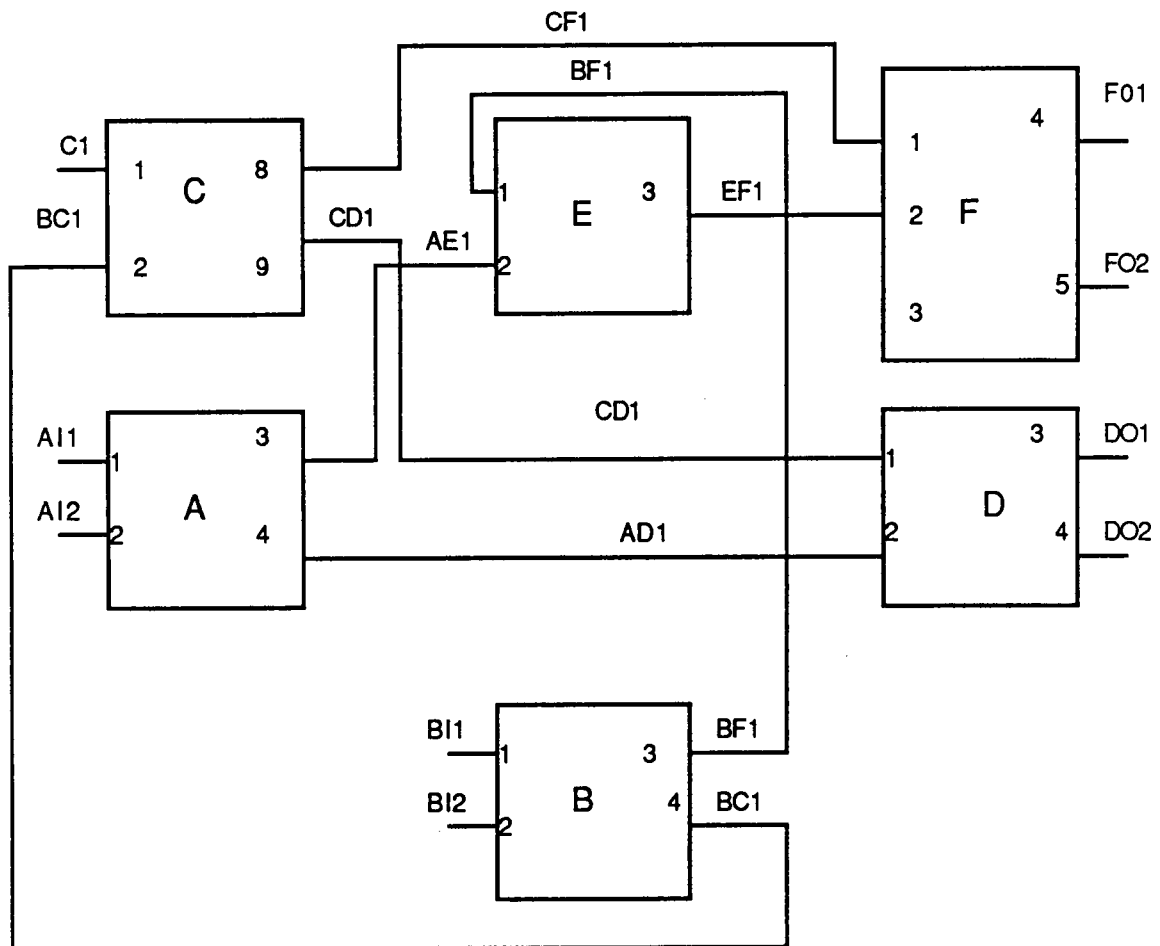
a) Original Sketch

Figure 18. Test Circuit 1



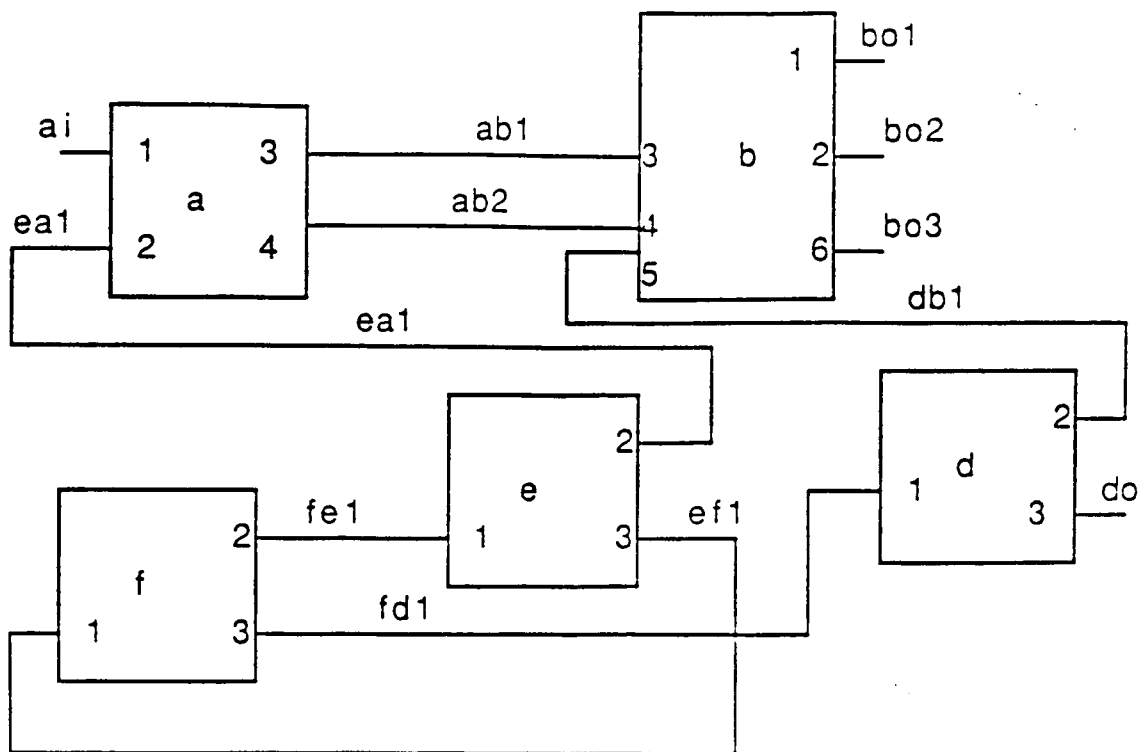
b) Intermediate Placement

Figure 18. Continued



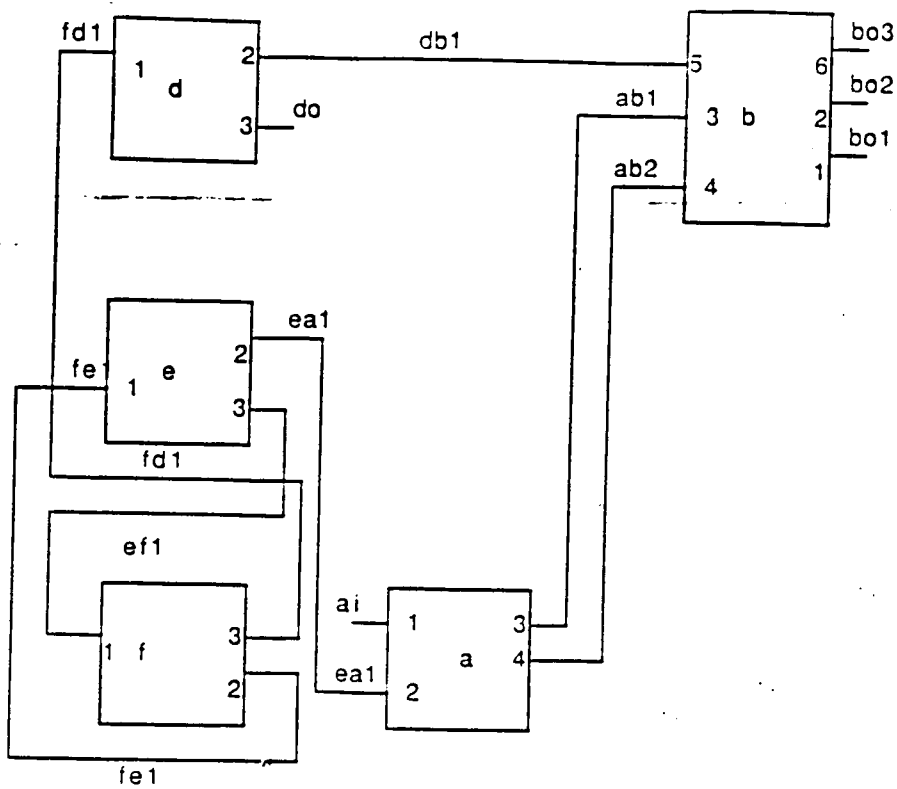
c) Final Placement

Figure 18. Continued



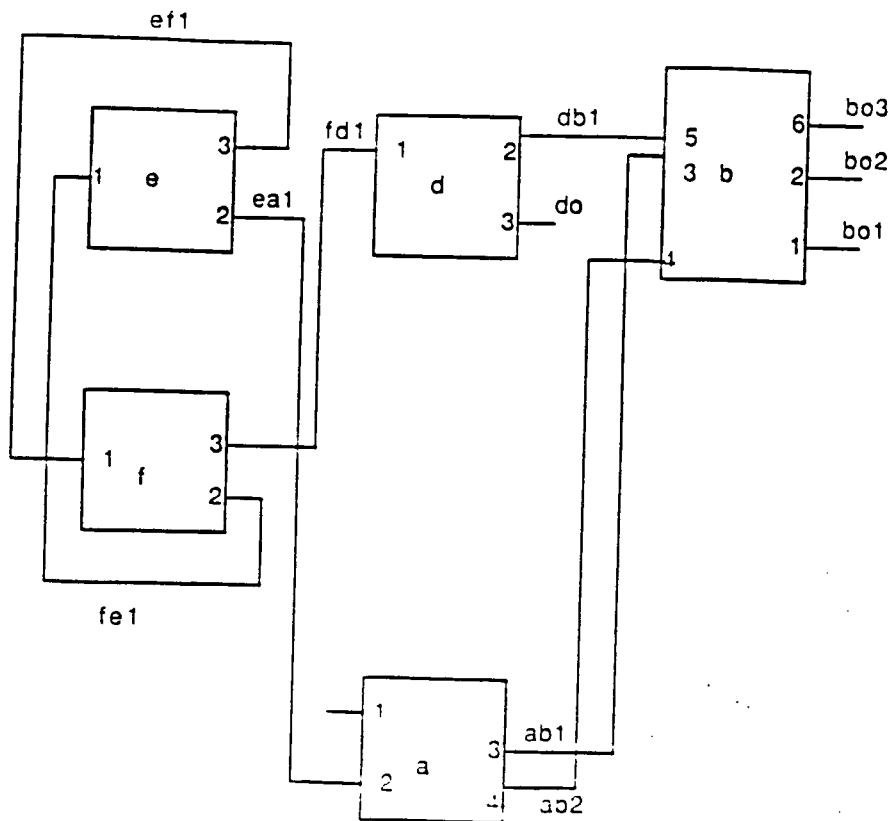
CIRCUIT NET-LIST

Figure 19. Test Circuit 2



INTERMEDIATE PLACEMENT

Figure 19. Continued



FINAL PLACEMENT

Figure 19. Continued

modules. For test two (Figure 19) the sequence of unplaced module selection is B, D, E and F. After each module is selected for placement, a slot location is assigned to each of them. Once this is done, each of the modules is drawn on the screen in its respective slot location.

Figures 20 through 23 are examples of additional schematics generated by LOGDESIGN. The placement for the "after" schematics were computed by the software package. These examples further illustrates the advantages of using LOGDESIGN to compute optimum placement of modules in a schematic diagram generation.

The logic diagram generator designed in this thesis is an excellent tool for a circuit designer. The designer can implement this package to have a pictorial representation of the optimum module placement for a logic diagram. The simplicity in using the package makes it's use worthwhile.

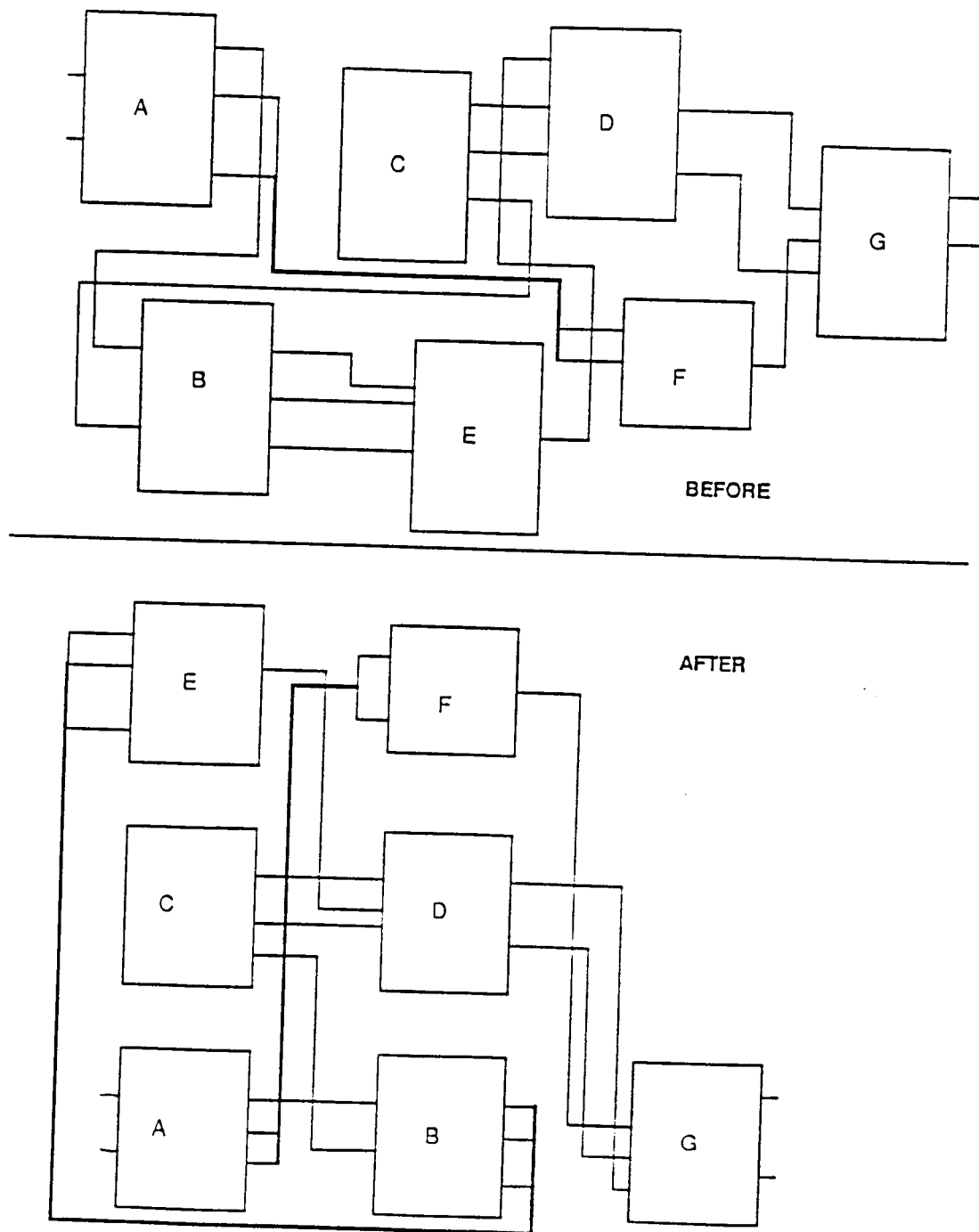


Figure 20. Example Circuit 1

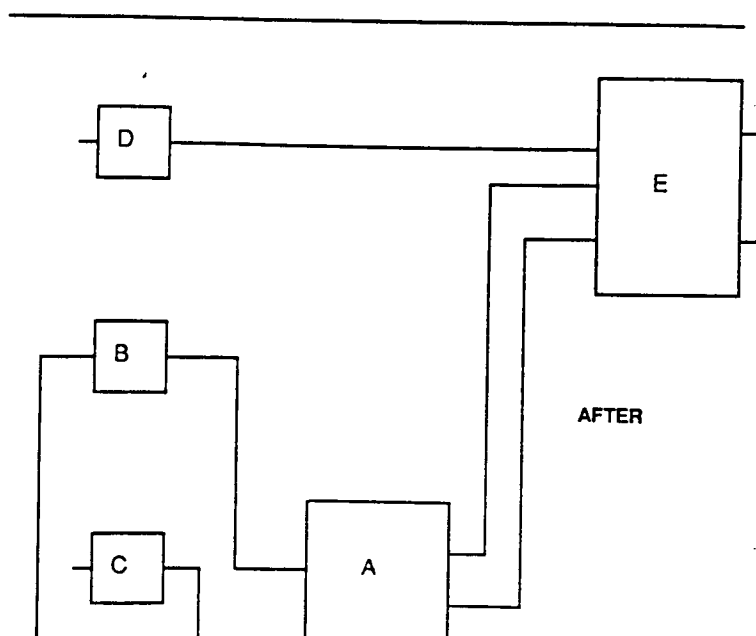
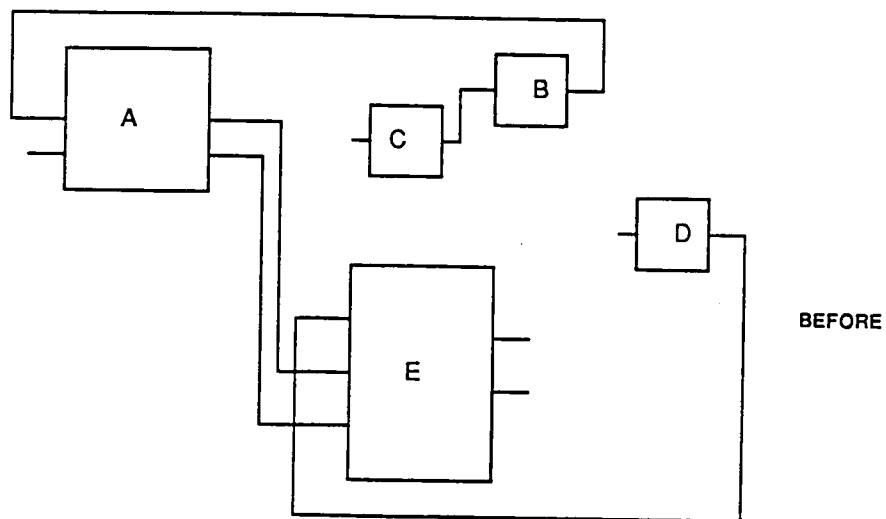
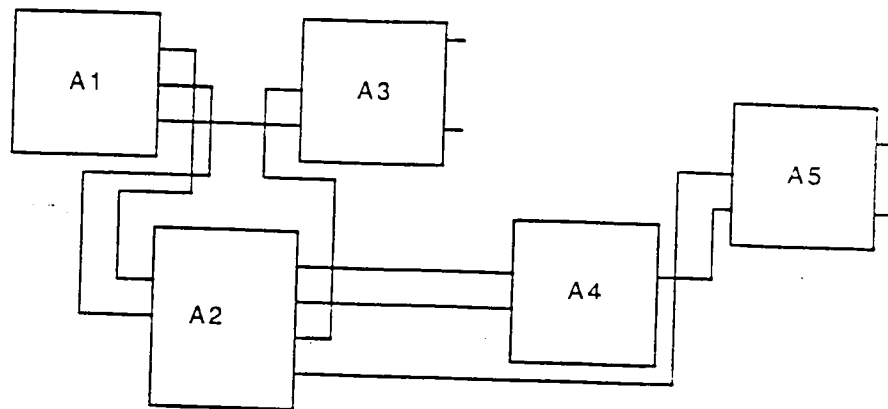
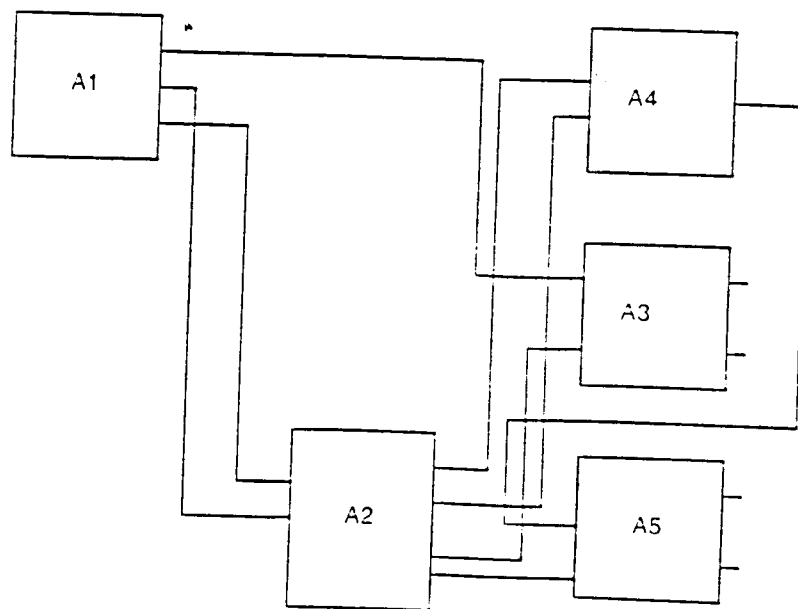


Figure 22. Example Circuit 2



BEFORE



AFTER

Figure 22. Example Circuit 3

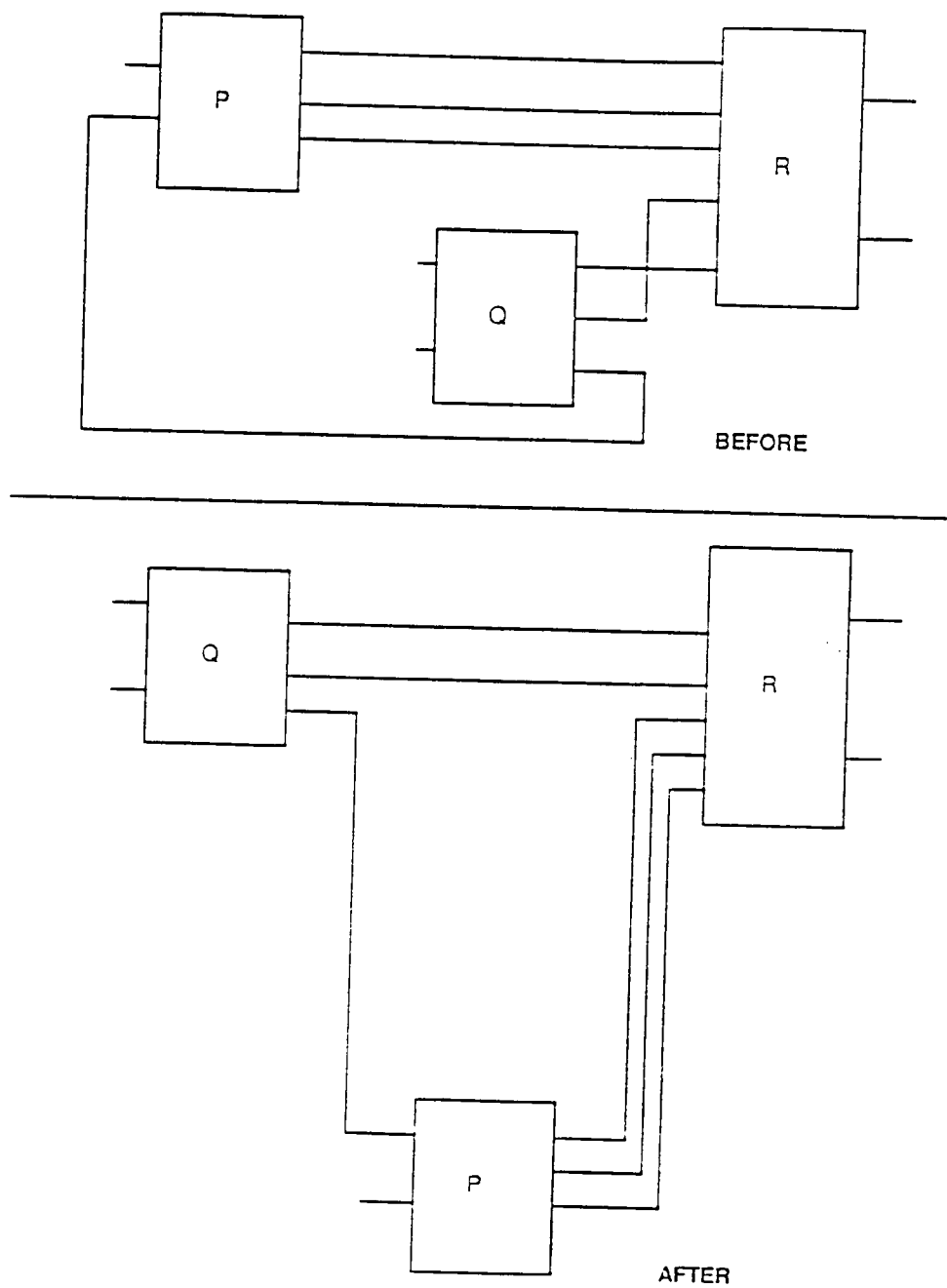


Figure 23. Example Circuit 4

CHAPTER 7

SUMMARY AND CONCLUSIONS

The software package LOGDESIGN presented in this thesis is an excellent schematic diagram generator. When this project began, it was difficult to set the design goals for the generated schematic diagram. Since an aesthetically pleasing diagram cannot be expressed in a quantitative manner, this made the task of choosing the design criteria very difficult. Finally, by studying the work of several authors in this area, a decision was reached concerning the design objectives for the generated schematic. During the development of the program, knowledge was gained about the use and features of the HP Advanced Graphics Package. This chapter contains a discussion about the limitations of LOGDESIGN as well as suggestions for future modifications of the package.

The graphics package LOGDESIGN is set to work with a maximum of nine modules. The schematic generated is a single page design and there is no option for handling a circuit with more than nine modules. If all the modules have the same number of connections to each other, using the package becomes redundant.

Another limitation may be the lack of having a router to draw the interconnections between modules. This is an excellent candidate for future modifications to the software package. The slot locations were designed such that adequate routing channels are already provided.

The package may also be implemented on a different machine. Since the program is coded in 'C' the only changes that have to be made are in the function `draw_box2.c()` and parts of the main program which calls commands specific to the HP 9000 computer.

The graphics package may also be used with a schematic capture software like P_CAD. Instead of having the net-list typed by the user, the input net-list may be a schematic

drawn by P_CAD. LOGDESIGN can then manipulate the schematic to generate a better placement. If the additional placement feature can be added to the schematic capture program, this will assist in generating superior schematics.

LIST OF REFERENCES

LIST OF REFERENCES

1. Smith, J.A. and Linders, J.G., "Automatic Generation of Logic Diagrams", *13th Design Automation Conference Proceedings*, San Francisco, pp. 377 - 391, 1976.
2. Peatman, John B., *Digital Hardware Design*. New York: McGraw-Hill, 1980.
3. Kumar, A., Arya, A. , Swaminathan, V. V. and Misra, A., "Automatic Generation of Digital Schematic Diagrams", *IEEE Design & Test*, 2, pp. 58 - 65 (1986).
4. Fakashi, Y. and Kuh, E.S., "Efficient Algorithms for Channel Routing", *IEEE Trans. CAD ICs and Systems*, 1, pp. 25 - 35 (1982).
5. Venkatraman, V.V. and Wilcox, C.D., "Gems: An Automatic Layout Tool for MIMOLA Schematics", *23rd Design Automation Conference Proceedings*, paper 8.1, pp. 131 - 137 (1986).
6. Hanan, M. and Kurtzberg, J.M., "Placement Techniques", *Design Automation of Digital Systems*, Prentice-Hall, Inc. vol. 1, pp. 213 - 282.
7. Kelley, A. and Pohl, I., *A Book on C*. Menlo Park, California: Benjamin/Cummings, 1984.
8. Hewlett Packard. *Graphics/9000 Advanced Graphics Package*. Part Number: 97085 - 90000, n.p.: June 1983.

APPENDICES

"local 2.h"

struct Module

```
{  
    char chipnam[10]; /* stores module name */  
    connection input_pin_numb;  
    connection output_pin_numb;  
  
}
```

struct connection

```
{  
  
    int pin_numb; /* stores pin number */  
    int chip_number; /* stores quantity of conneced chips */  
    char signal_name[10]; /* stores signal name */  
    char chip_name[10]; /* stores module name */  
  
}
```

struct slot1

```
{  
  
    int x_centr; /*stores center of x_slot */  
    int y_centr; /* stores center of y slot */  
  
}
```

Schematic Diagram Generator Program

Purpose	To generate a canonical schematic from a user specified text schematic
Process	* Prompt user for net-list * Select seed module and place on screen * Select remaining modules for placement * Assign slots for each module * Draw modules on screen
Variables	P stores P_list W stores pin quantity of modules R stores P_matrix entry chip stores module information C stores connection matrix O stores output matrix F stores F list sposi keeps track whether a slot is empty

***** /


```

int F[MAXNUMBER][1];

int loc; /* var gets the empty slot number for temp use */
int sposi[MAXSLOT]; /* sposi & b keeps track of empty slot locations

/* agp variables */

short id = 1, wspan, wspanm[2], iodlen, iodes[8], control, ierr;
char *wscnam = (char *)wspanm;
char *inces = (char *)iodes;

float XSI = 1.0, YSI = 0.763; /* used to determine aspect ratio */

float xxmin = 0.0, xxmax = 132.0, yymin = 0.0, ymax = 100.00;
short p4, p5;

k = mod_numb();
printf("\ntotal number of modules: %d\n", k);

for (i = 1; i < (k + 1); ++i)
    read_info (&chip[i]);

/* read input signal names */
getchar();
for (i = 1; i < (k + 1); i++) {
    printf("\nt*****input pin info module %s *****\n", chip[i].chipnam);
    for (j = 1; j < (chip[i].input_pin_numb.pin_numb + 1); j++)
        read_in_signal(&chip[i], j, i);
    } /* for loop */

/* reading connection details */

for (i = 1; i < (k + 1); ++i) {
    printf("\n*****output pin info module %s information *****", chip[i].chipnam);
    for (j = 0; j <= (chip[i].output_pin_numb.pin_numb - 1); ++j)
        read_connection (&chip[i], j, W, i);
    }

printf("\npress 'y' and then enter if want to use prefer neighbor algorithm\n");
scanf("%c", &y); /* gets response from user */

/* get slot location */
for (i = 1; i < MAXSLOT; i++)
    Get_slot(&slot[i], i);

for (i = 1; i < MAXSLOT; i++)
    printf("\n slot %d \tx = %f \ty = %f\n", i, slot[i].x_centr, slot[i].y_centr);

/* function develops the R (summation) matrix */
l = 1;

for (i = 1; i < (k + 1); ++i) {
    for (j = 0; j <= (chip[i].output_pin_numb.pin_numb - 1); ++j)
        sum (i, j, W, R, l);
    }

/* choosing the initial chip */

n = 4096 / (R, K);

```

```

R[nl[1]] = 0;
PIF[1][1] = 1; /* letting system know this chip has already been selected */
posi[6] = n; /* choose slot location */

for (i = 1; i < (k + 1); ++i) {
    if (placing (Pl, i) == 1)
        printf("\nThe chips that are unplaced are : %s\n", chip[i].chipnam);
}

/* constructing the connection matrix */
for (l = 1; l < (k + 1); l++)
    strcpy(Module[l], chip[l].chipnam);

for (i = 1; i < (k + 1); i++)
    for (j = 0; j <= (chip[i].output_pin_numb.pin_numb - 1); j++) {
        C[i][j] = 0;
        connection_matrix(C, &chip[i], k, Module, i, W, j, o);
    }
    printf("\n\n\t *** CONNECTION MATRIX *** \n\n");
    for (i = 1; i < (k + 1); i++) {
        printf("\n\n");
        for (l = 1; l < (k + 1); l++)
            printf("\t C[%d][%d] = %d\t", i, l, C[i][l]);
    }

/* F list computation */

for (j = 1; j < k; ++j) {
    for (i = 1; i < (k + 1); ++i) {
        if (placing (Pl, i) == 1)
            F_list (C, i, j, Pl, k, F);
    }

    hl = F_list_max (F, Pl, k, R);
    printf("Chip chosen is %s\n iteration number : %d\n", chip[nl].chipnam, j);

/* decides global input & output chips */
if (Global_input (nl, o, k) == 1) {
    printf ("Global input chip is %s\n", chip[nl].chipnam);
    loc = search_gi (sposi, nl);
    if (loc > 0) /* means slot available */
    {
        printf ("slot chosen is %d\n", loc);
    }
    else if (loc == 0) {
        loc = search_gi_2 (sposi, nl);
        if (loc > 0) /* means empty slot four

```

```

        printf ("slot chosen in second try is %d\n", loc);
    }
    else
        printf ("\n no empty space this page go to next ");
        exit(0); /* exit main program */
}
/* end global input loop */
else if (Global_output (n1, o, k) == 1) {
    printf ("Global output chip is %s\n", chip[n1].chipnam);
    loc = search_go (sposi, n1);
    if (loc > 0) /* means slot available */
    {
        printf ("slot chosen is %d\n", loc);
    }
    else if (loc == 0) {
        loc = search_go_2 (sposi, n1);
        if (loc > 0) /* means empty slot found */
        {
            printf ("slot chosen in second try is %d\n", loc);
        }
        else
            printf ("\n no empty space this page go to next ");
    } /* loc == 0 loop */
} /* end global output */
/* process general modules */
else if ( y == 'y' || y == 'Y') {
    loc = prefer_neigh(n1, C, Pl, sposi, o, k);
    printf("\n slot location for chip %t% is %t%d\n", chip[n1].chipnam, loc) ;
} /* end of else if */
else {
    for (l = 1; l < MAXSLOT; l++) {
        if (sposi[l] == 0) {
            sposi[l] = n1; /* chooses slot for arbitrary case */
            break;
        } /* if */
    }
} /* end of else */

} /* end of iteration loop */

/* App subroutines */
jheqn();
--
n1len = 4;

```

```

wacnam[0] = 'D';
wacnam[1] = 'B';
wacnam[2] = 'O';
wacnam[3] = 'Y';
control = 0;
iodlen = 8;
jdint(&id, &wspn, wspnam, &iodlen, "/dev/tty", &control);
jwon(&id);
jaspk(&XS1, &YS1);
jwind(&xxmin, &xxmax, &yymin, &yymin);
for(j = 1; j < MAXSLOT; j++)
{
    if (sposi[j] > 0) { /* cks if slot is occupied */
        p4 = sposi[j];
        p5 = j;
        printf("\nchip chosen \t%d\t slot numb %d\n", sposi[j], j);
        draw_box(&chip[p4], p4, p5);
    }
} /* end of if loop */
} /* for end */
jwoff(&id);
jwend(&id);
jend();

} /* end of main */

```

```

int getline(s)
char s(80);
{
    int c, i;
    for(i = 0; i < 80 && (c = getchar()) != EOF && c != '\n'; i++)
        s[i] = c;
    if (c == '\n')
        s[i] = '\0';
    return(i);
}

```

```

sum(i, j, P, R, l)
int i, j, P[MAXLENGTH][MAXLENGTH], R[MAXLENGTH][MAXLENGTH], l;
{
    int k, sum;
    sum = 0;
    for (k = 0; k <= j; ++k)
        sum += P[i][k];
    P[i][j+1] = sum;
}

```

```

int mod_numb()
{
    int i;
    printf("\nEnter total number of modules in system:\t");
    scanf("%d", &i);
    getchar();
    return(i);
}

```

```

read_info (m) /* function reads names of chips and quantity of pins for */
module *m;

{
    printf ("\nenter chip name(string) & hit return\n");
    if (scanf("%s", m -> chipnam) == 1)
        printf ("\nenter number of input pins(int) hit return\n");
    if (scanf("%ld", &m -> input_pin_numb.pin_numb) == 1)
        printf ("\nenter output pin number & hit return\n");
    if (scanf("%ld", &m -> output_pin_numb.pin_numb) == 1)
        ;;
    printf("\n%s\n%s\n%s",
        " ***** ",
        " ***** ",
        " ***** ");
}

sort (P, n) /* function stores the maximum entry in P matrix returns the val */

int P[MAXLENGTH][MAXLENGTH], n;

{
    int i, max, j;
    max = 0;
    for (i = 1; i < (n + 1); ++i) { /* beginning of loop */
        if (P[i][1] > max) {
            max = P[i][1];
            j = i;
        }
        else if (P[i][1] == max)
            ;;
    } /* end of loop */
    return (j);
}

placing (P, n) /* checks if a module have been selected or not */
int P[MAXLENGTH][1], n;
{
    if (P[n][1] == 1)
        return (0);
    else
        return (1);
}

```

```

read_conn (m, j, M, i) /* function gets names and connection details */
module "m;
int j; /* pin number */
int M[MAXLENGTH][MAXLENGTH];
int i; /* module number */
{
    char temp[80], tmp2[80];
    int x, l, n, o = 1, x1, p = 1, x2, n1, i1;

    printf("\n\nfor chip %s number of output pins are %d\n", m -> chipname, m -> output_pin_num.pin_num);
    switch (j) {
        case 0:
            printf("\nenter quantity of modules pin 1 is connected to:\n");
            break;
        case 1:
            printf("\nenter quantity of modules pin 2 is connected to:\n");
            break;
        case 2:
            printf("\nenter quantity of modules pin 3 is connected to:\n");
            break;
        case 3:
            printf("\nenter quantity of modules pin 4 is connected to:\n");
            break;
    } /* end of switch */

    if (scanf("%d", &m -> output_pin_num.chip_number) == 1)
        printf("\nconnected module quantity %d", m -> output_pin_num.chip_number);

    getchar();
    x = m -> output_pin_num.chip_number;
    M[i][j] = x;
    if (x == 0) {
        printf("\nif no connected modules, enter signal name of pin, hit enter\n");
        getline(temp);
        strcpy(m -> output_pin_num.signal_name(n), temp);
    }
    else {
        printf("\n\nnote: DO NOT leave any space between chip name and signal name, the comma is the separator");
        for (l = 1; l < (x + 1); ++l) {
            if (o == 1)
                x1 = x;
            n = ((x1 * j) + l);
            printf("\nenter connected chipname(string) %s comma", n);
            if (l == x + 1)
                printf("\nenter\n");
            getline(temp);
            for (i1 = 0; i1 < 80; i1++) {
                if (temp[i1] == ',')
                    break;
            }
            else
                temp2[i1] = temp[i1];

            if (i1 == 0)
                temp2[i1] = '\0';
            strcpy(m -> output_pin_num.chip_name(n1, tmp2));
            strcpy(m -> output_pin_num.signal_name(n1, &temp[i1]));
        }

    } /* end of else */

    --o;
    x1 = x;
    for (l = 1; l < (x + 1); ++l) {
        if (p == 1)
            x2 = x;
        n1 = ((x2 * j) + l);
        if (M[i1][l] == 0)
            printf("\n\nsignal name of pin is:\n", m -> output_pin_num.signal_name(n1));
        else
            printf("\n\nconnected module name is %s\t signal name is: %s\n", m -> output_pin_num.chip_name(n1), m -> output_pin_num.signal_name(n1));
    }

    --p;
    x2 = x;
    return (x);
}

```

```

connection_matrix (c, m, k, n, i, W, j, o) /* function gets the connection matrix C */
int c[MAXLENGTH][MAXLENGTH], k, i; /* c is the connection matrix */
char n[MAXLENGTH][MAXLENGTH];
int W[MAXLENGTH][MAXLENGTH];
int i;
module *m;
int o[MAXLENGTH][MAXLENGTH];
{
    int l, b, q, q1;
    int l1 = 1, p, b1, x;
    char O[MAXLENGTH]; /* var gets the name for the chip connected to */
    l = 0;
    c[l][l] == 0;

    x = W[i][j];
    for (b = 1; b < (x + 1); ++b) {
        if (l1 == 1)
            b1 = x;
        p = ((b1 * j) + b);
        --l1;
        b1 = x;
        strcpy (O, m -> output_pin_numb.chip_name[p]);

        l = 0;
        while (strcmp (O, n[l1]) != 0 && l < (k + 1))
            ++l;

        ++c[l][l];
        ++c[l][i];
        o[i][l]++;
        printf ("\n c[%d][%d] = %12d c [%d][%d] = %5d \n", i, l, c[i][l], l, i, c[l][i]);
        printf ("\n output matrix is : o[%d][%d] = %5d\n", i, l, o[i][l]);
        l = 0;
    }
}

```

```

#include "local2.h"

F_list (c, i, j, pl, k, f) /* F_list.c gets the f_list, */
int i, j, k; /* i is chip id, j is numb. of iterations, k is numb. of
int c[MAXLENGTH][MAXLENGTH]; /* connection matrix */
int pl[MAXLENGTH][1]; /* placement matrix */
int f[MAXNUMBER][1]; /* F - list matrix */

{
    int l, m, n, s; /* temporary var */
    int save[MAXNUMBER];

    for (l = 1; l < (k + 1); l++) {

        if (l != i) {
            if (pl[l][1] == 1) { /* ck. if module is placed */
                save[l] = 1;

                ++m;
            }
        }
        /* for loop */
        save[m] = '\0';
        s = m;
        m = 0;
        for (l = 1; l < (k + 1); ++l)
            for (m = 0; m <= (s - 1); ++m)
                if (l == save[m]) {

                    n += c[l][1];
                }

    }

    f[i][1] = n;
    printf ("\nF_list is F[%d] = %d\n", i, f[i][1]);
}

```

```

#include "local2.h"

Get_slot (s, i) /* function gets the slot locations to systems */
slot1 *s; /* var is slot identifying structure */
int i; /* incr var */
{
    int of = 4; /* var holds offset value */

    if (i >= 1 && i <= 3) {
        s -> x_centr = 20.0;
        s -> y_centr = 84.0 - ((i - 1) * 8 * of);
    } else if (i == 4 || i == 5 || i == 6) {
        s -> y_centr = 84.0 - ((i - 4) * 8 * of);
        s -> x_centr = 56.0;
    }

    else if (i == 7 || i == 9 || i == 8) {
        s -> y_centr = 84.0 - ((i - 7) * 8 * of);
        s -> x_centr = 92.0;
    }
}

```

```

}
F_list_max(f, p1, k, r) /* F_list_max.c function finds the maximum of F list
int f[MAXNUMBER][1]; /* F_list matrix */
int k, p1[MAXLENGTH][1]; /* k is numb of modules, P1 is selection matrix, p1[0] = 0
int r[MAXLENGTH][MAXLENGTH]; /* r matrix is the p_list matrix */
{
    int i, j, y, max; /* temporary variables */
    int save[MAXLENGTH]; /* VAR STORES modules where a tie exists */

    for (j = 1; j < (k + 1); ++j) {
        if (j == 1) { /* 1st iteration */
            max = f[j][1];

            printf ("\n j is %d\n ", j);
        }
        else if (j > 1) {
            if (max < f[j][1]) {
                max = f[j][1];
            }
        }
    } /* for loop */

    y = 1;
    i = 1;

    for (j = 1; j < (k + 1); ++j) { /* save to resolve ties */
        if (max == f[j][1]) { /* save value to resolve ties in future */
            save [y++] = j;
        }

        for (j = 1; j < y; ++j) {
            if (j == 1) {
                max = save[j];
            }
        }

        else if (i != j) {
            if (r[max][1] < r[j][1]) {
                max = save[j];
            }
        }

        else if (r[max][1] == r[j][1]) {
            max = save[j];
        }
    } /*else if loop */

    p1[max][1] = 1; /* chip is selected */
    f[max][1] = 0;
    return (max); /* return selected chip number */
}

```

```

#include "local2.h"
/* function determines whether chip is global input module revised 8/8/87*/
Global_input (m, o, k)

int m, o[MAXLENGTH][MAXLENGTH]; /* m is module number, o is output matrix */
int k; /* k is number of modules */
{
    int i; /* i is the increment operator */
    int sum = 0;
    for (i = 1; i < (k + 1); i++)
        sum += o[i][m];
    if (sum > 0)
        return (0);
    else
        return (1);
}

Global_output (m, o, k) /* function finds global output */
int m, o[MAXLENGTH][MAXLENGTH]; /* m is module, o is output matrix */
int k;
{
    int i, sum; /* i is increment oper, sum gets the sum */
    sum = 0;
    for (i = 1; i < (k + 1); i++)
        sum += o[m][i];
    if (sum > 0)
        return (0);
    else
        return (1); /* chip global output */
}

```

```

#include "local2.h"

search_gi (s, i) /* se_arch2.c _ has the latest functions to facilitate the
                    program . Aug 8, 1987*/
{
    int s[MAXSLOT]; /* string keeps the slot available info */
    int i; /* module numb */

    {
        int j;
        for (j = 1; j <= 3; j++) {
            if (s[j] == 0) {
                s[j] = i; /* module is located in slotnumber j */
            }
        }
        return (j); /* slot number returned */
    }
    for (j = 4; j <= 6; j++) {
        if (s[j] == 0) {
            s[j] = i;
        }
    }
    return (j);
} /* for loop ends */

search_go (s, i) /* function finds the slot location for a global output chip
{
    int s[MAXSLOT];
    int i;
    {
        int j;
        for (j = 7; j <= 9; j++) {
            if (s[j] == 0) { /* chks if slot is empty */
                s[j] = i;
            }
        }
        return (j);
    } /* if */

    for (j = 4; j <= 6; j++) {
        if (s[j] == 0) {
            s[j] = i;
        }
    }
    return (j);
} /*for */

/* function looks for empty slot space for global output chip

```

```
search_g1_2 (s, i) {
```

```
int s[MAXSLOT];
```

```
int i;
```

```
{
```

```
int j;
```

```
for (j = 7; j <= 9; j++) {
```

```
if (s[j] == 0) {
```

```
s[j] = i;
```

```
return (j);
```

```
}
```

```
} /*for */
```

```
}
```

```
search_g0_2 (s, i) /* function looks to find empty slots for global  
output chips _ 2nd try */
```

```
at s[MAXSLOT];
```

```
i;
```

```
{
```

```
for (j = 1; j <= 3; j++) {
```

```
if (s[j] == 0) {
```

```
j);
```

```
s[j] = i;
```

```

#include "local2.h"

prefer_neigh (i, c, pl, a, o, k) /* prefer_neigh developed 0.12.07 */
{
    int i, k; /* i is module name, k is quan */
    int s[MAXSLOT]; /* s is slot quan */
    int c[MAXLENGTH][MAXLENGTH], cmax[MAXLENGTH][MAXLENGTH]; /* c is connection matrix,
    int pl[MAXLENGTH][1]; /* placement matrix */
    int temp, temp1, j, temp2, l; /* temporary var */
    int save[MAXSLOT];
    int m;

    for (j = 1; j < (k + 1); j++) c[j][i] /* gets the maximum connection to module */
        if (j == 1) {
            temp = c[i][j];
            temp1 = j; /* gets module number */
            save[0] = j;
        }

        if (c[i][j] > temp) { /* checks for higher connections */
            temp = c[i][j];
            temp1 = j; /* gets the identification of placed chip */
            save[0] = j;
        }
    }
    else if (temp == c[i][1] && j > 1) {
        m++;
        save[m] = j;
    }
    /* else if loop */
    /* for loop ends */
    if (m == 0)
        printf("n only one possible neighbor ");
    else
        printf("number of possible neighbors are %d n", m + 1);

    for (l = 0; m-- > 0) {
        temp1 = save[l];
        printf("n chip chosen is %dn", temp1);

        if (pl[temp1][1] == 1) {
            printf("placed neighbor chip is %dn", temp1);

            for (j = 1; j < MAXSLOT; j++) {
                if (s[j] == temp1)
                    temp2 = j;
            }
            printf("slot number for placed module is: %dn", temp2);
            /* for loop */
        }

        /* determines whether selected module output module */
        if (c[i][temp1] > 0) {
            printf("nselected chip is source ");

            if (temp2 > 0 && temp2 < 4) {
                for (l = 1; l < 4; l++) {
                    if (s[l] == temp2) {
                        l11 = l;
                        printf("n slot chosen for selected module %dn", l11);
                        return l11;
                    }
                }
            }
        }
    }
}

```

```

} /* for loop */

} /* if loop */

else if (temp2 > 3 && temp2 < 7) {
    for (i = 1; i < 4; i++) {
        if (s[i] == 0) {
            s[i] = 1;
            printf("No slot chosen for selected module %d\n", i);
            return (1);
        }
        if (s[i] == 0) {
            s[i] = 1;
            printf("No slot chosen for selected module %d\n", i);
            return (1);
        }
    }
} /* for loop */

} /* else if */

else if (temp2 > 6 && temp2 < 10) {
    for (i = 4; i < 7; i++) {
        if (s[i] == 0) {
            s[i] = 1;
            printf("No slot chosen for selected module %d\n", i);
            return (1);
        }
    }
} /* for loop */

for (i = 1; i < 4; i++) {
    if (s[i] == 0) {
        s[i] = 1;
        printf("No slot chosen for selected module %d\n", i);
        return (1);
    }
} /* for loop */

} /* else if */

} /* end of if loop */

} /* main if loop */

if (m == 0 && p1[temp1] == 0)
    printf("No none of the connected chips is placed\n");

} /* for loop */

for (i = 1; i < MAXSIZE; i++) {
    if (s[i] == 0) {
        s[i] = 1;
        return (1);
    }
}

} /* for loop */

} /* end of main */

```

```

#include "local2.h"
#include <stdio.h>

draw_box(m, i, q) /* draw_box2.c - draws boxes */
module *m; /* module number */
short i; /* module number */
short q; /* slot number */
{
    extern slot1 slot(MAXSLOT); /* slot locations */
    float x_loc, y_loc, yof = 4.0, xof = 0.0; /* x_loc and y_loc are var, yof and xof are ntr
    short j, op, ip, ml = 3, nj /* op, ip, n local var, m is for marking */

    /* following var are for text */
    short nl, jl, tmp1[8]; /* nl, jl local var, tmp1 stores string */
    char *p1 = (char *)tmp1;
    char temp[15]; /* stores string */

    strcpy(temp, m->chipnam);
    nl = strlen(temp);
    for (jl = 0; jl < nl; jl++) /* stores string as short */
        pl[jl] = temp[jl];

    printf("\nmodule number %d slot numb %d module name %s\n", i, q, m->chipnam);
    x_loc = slot[q].x_center;
    y_loc = slot[q].y_center;

    j2mov(&x_loc, &y_loc);
    jtxl(&nl, tmp1);
    j2mov(&x_loc, &y_loc);
    printf("location: x=%f y=%f\n", x_loc, y_loc);

    op = m->output_pin_numb.pin_numb;
    ip = m->input_pin_numb.pin_numb; /* gets pin num */
    printf("\n pin numbers are : \n input pins: %d output pins: %d\n", ip, op);

    if (ip > op)
        n = ip;
    else if (op > ip)
        n = op;
    else
        n = ip;

    n += 1; /* increm n */
    y_loc = (n * (yof)) / 2.0;
    printf("\n y location after move is %f\n", y_loc);

    j2mov(&yof, &y_loc);
    printf("\nyof value is %f\n", yof);

    for (j = 1; j < n; j++) {
        ynf = -4.0;
        j2dr(&xof, &yof);
        printf("\nj value %d\n", j);
        j2mk(&xof, &yof, &ml);
        printf("\nj value %d\n", j);
    }

    j2dr(&xof, &yof);
    ynf = 2 * yof;
    j2dr(&yof, &yof); /* draws bottom of box */
    yof = ynf / 2;
    printf("\n after bottom of box drawn xoff = %f yoff = %f\n", xof, yof);
    printf("\n n is %d ip is %d\n", n, ip);
    ynf = -(n * yof) / (ip + 1); /* gets offset val for right side */
    printf("\n offset value of left side is: %f\n", ynf);

    for (n = 1; n <= ip; n++) {
        j2dr(&xof, &yof);
        j2mk(&xof, &yof, &ml);
    }

    j2dr(&xof, &yof);
    ynf = 4.0;
    ynf = 2 * (ynf);

    j2dr(&yof, &xof); /* draws top of box */

    printf("\n relative draw is : \n xoffset: %f yoffset: %f\n", xof, yof);
}

```

VITA

Syed Kamruddin Hussain was born in Mymensingh, Bangladesh on January 13, 1960. He attended Residential Model School and passed the Secondary School Certificate Examination with distinction in 1976. The following fall he began coursework at Tri-State University in Angola, Indiana. In March, 1981 he was awarded the degree of Bachelor of Science in Electrical Engineering.

Upon graduation, he joined Square D Company as an applications engineer. In September, 1984 he joined the Electrical Engineering Department at the University of Tennessee, Knoxville to pursue a masters degree in electrical engineering. He was awarded a teaching assistantship. In December 1987 he was conferred the degree of Master of Science in Electrical Engineering .