THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 57, No. 6, July-August 1978
Printed in U. S. A.

UNIX Time-Sharing System:

Circuit Design Aids

By A. G. FRASER
(Manuscript received November 29, 1977)

The circuit design engineer who wants to obtain rapid prototype construction of one-of-a-kind designs can get help by using programs that run under the UNIX operating system. The programs include means for drawing circuit schematics on a graphics terminal and semi-automatic production of wire lists from those drawings. Included in the package are programs that check for simple errors and an interactive graphics program for performing the physical design of a circuit board.*

The design aids are oriented toward the production of circuits composed mainly of dual in-line packages mounted on wire-wrap boards. While the graphic editor, called **draw**, *can be used for many purposes, the physical design layout program, called* **place**, *is valid only for boards equipped with sockets for dual in-line packages. The programs know how to prepare numerical control information for automatic and semi-automatic wire-wrap machines.*

The programs use a PDP-11 computer with a Tektronix 4014 graphics terminal attached. Hard copy graphic output is obtainable on some systems from a device attached directly to the terminal; otherwise, the data are transmitted over communications lines to a computing service machine.

I. INTRODUCTION

In the research and development process, there is frequently a

* UNIX is a trademark of Bell Laboratories.

need to rapidly obtain small numbers of prototype circuits. Ideally, the designer would like to be able to draw and edit circuit schematics in machine-readable form so that checking, physical layout, and prototype production are available at the touch of a button. Drafting standards and (frequently) physical design efficiency are second in priority to a fast turnaround. Automatic documentation and facilities for making changes to earlier designs are also required.

To a significant degree, this ideal has been achieved for those engineers who have access to a PDP-11 that runs the UNIX operating system. The following four command programs are used:

draw	to edit circuit drawings
wcheck	to perform consistency checks on a circuit
place	to place components on a circuit board
wrap	to generate data for wire wrapping

All programs are written in the C language and the interactive graphics terminal is a Tektronix 4014. Hard-copy output on paper or microfilm is obtained through a Honeywell 6070. The Honeywell 6070 also produces the paper tape which drives a Gardner-Denver 14YN semi-automatic wire-wrap machine. Punched cards suitable for use with a Gardner-Denver 14F automatic wire-wrap machine are obtained by executing a program on an IBM 370. In each case, use of the service machine is automatic and is possible through the communications lines linking the PDP-11 to the Honeywell 6070 and linking the Honeywell 6070 to the IBM 370.

The circuit design aid programs are intended for use with circuits composed primarily of dual in-line packages assembled on wire-wrapped boards. The intent is to provide the nonspecialist circuit designer with a tool suitable for laboratory use. It is assumed that the user's prime objective is to obtain an assembled circuit board and maintain consistent documentation with a minimum of fuss. In exchange, the user must be prepared to adopt some predefined conventions and a rather restricted drawing style.

Logic diagrams are first prepared using **draw**. One file of graphic information is constructed for each page of the drawing. When the logic drawing is complete, the graphics files are processed to yield a wire-list file. The user must now prepare a definitions file in which the circuit board and circuit components are described. Library files of definitions exist to simplify the task. The definitions file is a text file prepared using the UNIX text editor. Next, the wire-list file and the definitions file are fed to **wcheck**. That program performs a number of consistency checks and generates a cross-reference

listing. Any errors detected at this stage must be corrected in the logic diagram, and the process is then repeated. When a correct circuit description has been obtained, the definition and wire-list files are fed to **place**. That is an interactive program for choosing a circuit board layout. A drawing of the board is generated automatically. In addition, placement information is automatically written back into the logic drawings for future reference. Finally, the definitions file and a wire-list containing the placement data are fed to **wrap**, which generates the instructions necessary for wiring the circuit board. For machine-wrapped boards, **wrap** produces the appropriate paper tape or punched cards. Hard-copy printouts of the logic diagrams, the circuit board, and the cross-reference listings provide permanent documentation of the final product.

II. TERMINOLOGY AND CONVENTIONS

A circuit description contains two parts. A schematic drawing shows the interconnection of components, while a text file describes the physical properties of the components. Drawings prepared by **draw** are held in *drawing files* with one file per page of a drawing. These files have names that end in the page number and **.g**. The definitions file is prepared as ASCII text and is in Circuit Description Language (CDL). It is CDL that the **draw** program generates when asked to produce a wire list from the circuit drawing.

The terminology used in the circuit design aid programs is described below. It will be seen that this terminology is oriented toward integrated circuits and wire-wrap boards. However, with a little imagination, the user will find that a wide range of "non-standard" situations can be handled quite effectively.

signal	The information that passes over a wire and is carried by a net of wires.
special signal	A signal that is distributed about the circuit board by printed circuitry. Ground and VCC often fall into this category.
chip	A circuit component such as a NAND gate.
package	The physical unit which houses one or more chips. A dual in-line package is an example.
pin	The point where a signal wire connects to a chip.
board	The physical unit on which packages are mounted.

socket	The device for mounting a package on a board.
connector	The interface between signal wires on the board and those external to the board.

Chips, packages, and sockets are grouped by type. All the items of one type exhibit the same properties. Types, like signals, chips, and connectors, are identified by name. Pins and sockets are identified by number.

III. DRAW: TO EDIT A LOGIC DRAWING

For simplicity, **draw** is a graphic editor that knows nothing about electricity. However, its features are designed to make the preparation of logic drawings easy. Using **draw** one could draw almost anything and there would be no immediate complaint. Errors would be reported only if the machine is asked to produce a wire list.

The user must accept certain conventions for circuit drawings. One convention relates to the size and scale of the drawing itself. The hard-copy version is intended to fit neatly on 11- by 17-inch drawing paper. (Fig. 1 is a reduced version of such an output.) The page is divided into two parts: a main drawing area and a narrow rectangular area known as the *border*. The latter is intended for annotations, such as the name of the component being designed. When displayed on the Tektronix 4014, the main drawing area occupies the entire screen (Fig. 2), and a special mode of **draw** is required to display the border. To avoid confusion about which lines touch which others, a 1/10-inch grid is used. All lines have their end points on grid points. (A single exception will be described later.) For reference purposes, the drawing area is divided into large squares 1 by 1 inch. These are the bases of a coordinate system by which the position of any component can be identified.

draw is used mainly to draw two things: chips and wires. So far as **draw** is concerned, no relationship exists between them. For example, it is unaware of any need for a wire to terminate on a chip. For the same reason, it does not object to a wire that runs right through a chip. Only when it comes time to generate a wire list and feed it to **wcheck** is it necessary for the drawing to meet such standards of consistency and reasonableness.

A wire is drawn as a connected series of straight lines. Drawing is achieved by first specifying the signal name and then by moving the cross-hairs (controlled on the 4014 by thumb wheels) to the position at which drawing should start. Striking an **e** starts a new line and an

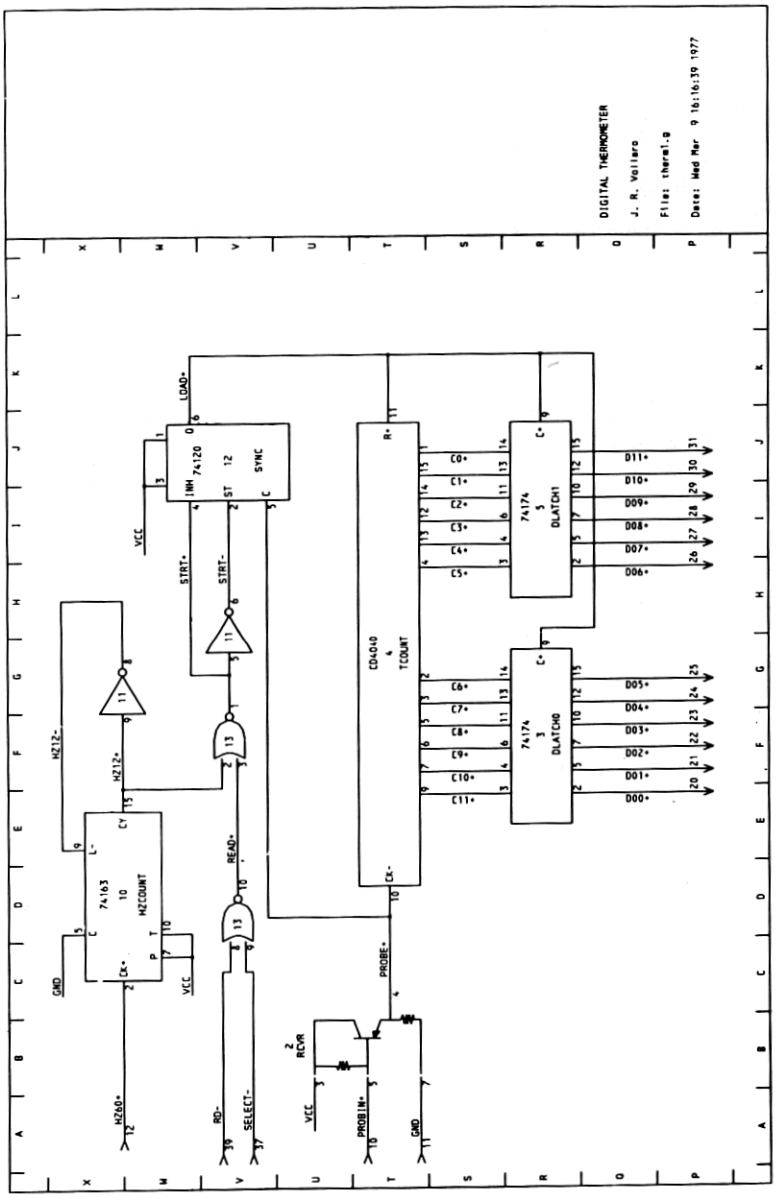

Fig. 1—A circuit schematic printed from microfilm output.

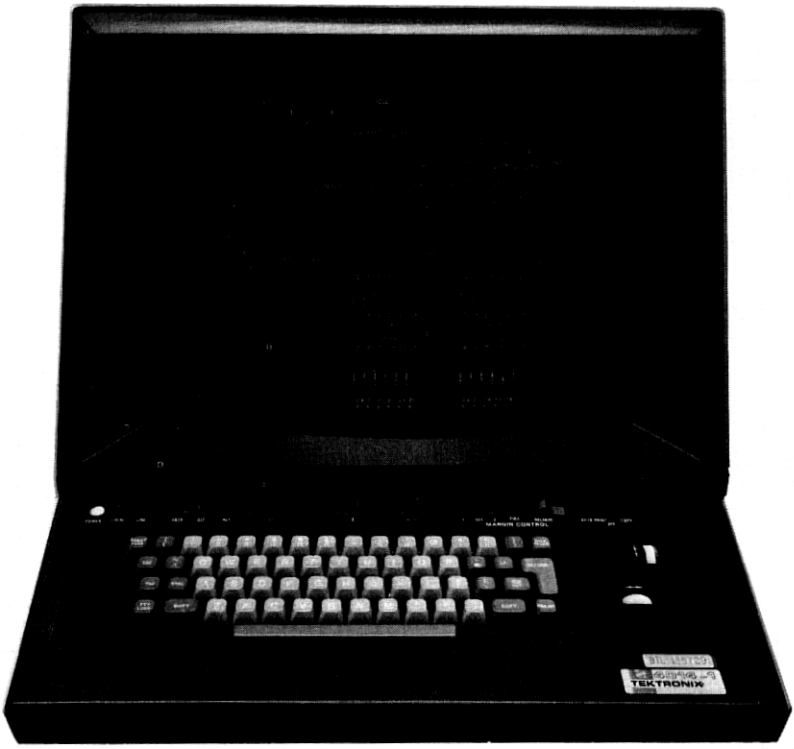

Fig. 2—The Tektronix 4014 terminal displaying a circuit.

a starts a new line segment which joins onto an existing line. Thereafter, striking the space bar extends the line to the current cross-hair position. Line segments drawn in this way are either vertical or horizontal. Diagonal lines are drawn by using **z** instead of **e**.

It may seem illogical, but **draw** treats pins as part of a wire and not part of a chip. A wire joins together a number of pins, and for each pin one can specify a name and a number. Chips are drawn separately and the association between chips and pins is made when the wire list is constructed. For that association to be made, the pin must lie within, or be very close to, a rectangle just large enough to contain the chip.

Chips are usually represented by closed geometric forms such as rectangles. To draw a chip it is only necessary to specify its shape and use the cross-hairs to give its position. A rectangle is drawn if no shape is specified. Two text strings can optionally be associated with a chip: a chip name and a type name. In some cases the shape

adequately identifies chip type and then no explicit type name need be given. Both names are treated by **draw** as text to be written, usually, within the drawing of the chip. An option, selected when defining a new shape, allows one to specify where such text should be displayed.

To define a new shape, the user first enters its name. A blank screen is then presented on which the shape may be drawn at four times the final scale. In this mode line end points must lie on 1/40-inch grid points (not the usual 1/10-inch grid) so that reasonable approximations to curved lines can be made using straight line segments.

By now it should be apparent that **draw** has little built-in intelligence. For example, it does not have a repertoire of heuristics designed to automatically route wires around chips or to minimize crossovers. We judge that such heuristics are cumbersome to program and rarely satisfactory in practice. Layout of the drawing is entirely in the user's hands. However, a number of small conveniences are provided to make the task moderately easy. For example, chips are drawn by positioning the cross-hairs on the left-hand end of the required centerline. Most chip shapes are symmetric about the centerline, and the desired centerline position is frequently known more accurately than any other point. Another convenience is a command that draws a grid centered on the current cross-hair position with grid points spaced at intervals specified by the user. In this way, one can quickly and accurately place a series of equally spaced wires or chips.

Editing facilities are equally unsophisticated. One can delete specific wires, pins, chips, and other drawing features or one can draw a rectangle around a group of components and delete them all or move the rectangle with components inside. Rectangular sections of the drawing can also be copied either to other parts of the drawing or onto a file for inclusion in another drawing. The awkward edits can be most quickly handled by redrawing some of the wires. To facilitate this with minimum risk of error, the following technique is used. First, recall that each wire joins together a certain collection of pins. The pins are part of the wire so far as **draw** is concerned. Therefore, there is a mechanism by which the user can remove all the lines of a wire without removing the pins. The pins remain and are displayed as asterisks. A new series of lines can now be drawn joining the asterisks together. In this way the possibility of omitting a pin from the wire is minimized and, at the same time,

the pin display provides an effective visual aid towards rapid selection of a new wire route.

Manipulation of the Tektronix 4014 thumb wheels is an acquired skill. **draw** simplifies the task by not requiring high accuracy when positioning the cursor. For example, when extending a horizontal line only the vertical cross-hair need be accurately positioned. When pointing at a wire, pin, or chip one need only be close to it, not precisely over it.

It has been found that the speed with which one can use a graphics terminal diminishes as the need to move one's eyes between keyboard and screen increases. Ideally one should be able to watch the screen and not the keyboard. However, it takes two hands to operate a typewriter keyboard and one more to control the cross-hairs. Since **draw** requires that the user type in signal and shape names, some hand and eye movements are unavoidable. The compromise that we have adopted requires a hand movement when a name or number is typed but not when drawing takes place. In particular, while drawing a wire one can keep the left hand in the standard position for typing, while the right hand operates the thumb-wheels. The thumb of the right hand also is used to strike the carriage return key. Each drawing action is effected by striking a single key which is in the region controlled by the left hand. Striking the carriage return terminates a drawing sequence.

The command

```
draw    therm1.g
```

is used to initiate editing of the drawing held in the file **therm1.g**. Hard copy output of that drawing is produced on the Calcomp plotter by

```
draw   -c   therm1.g
```

A wire list is generated by

```
draw   -w   therm1.g
```

and it is placed in a file called **therm1.w**. The wire list is an ASCII file in Circuit Description Language.

IV. CIRCUIT DESCRIPTION LANGUAGE

The Circuit Description Language was first designed as a convenient notation for the manual preparation of wire lists. As the design aid programs were elaborated, their needs for information

increased and the Circuit Description language grew. It is apparent that further extensions to the language will be required as new capabilities are added to the software package. Consequently, the Circuit Description language has an open-ended structure.

A circuit description consists of command lines and signal description lines. The former have a period as the leftmost character, followed by a one- or two-letter command. For example, the following is a description of a 16-pin, dual in-line package.

```
.k   DIP16 16 80 30
.kp  1 05/00  -  8 75/00
.kp  9 75/30  -  16 05/30
```

The first line gives the name of the package type, the number of pins, and the dimensions (in hundredths of an inch) of the package. The remaining two lines give the positions of the pins relative to one corner of the package.

The description of a circuit is in four parts:

(*i*) Package Descriptions. A description of each package type.
(*ii*) Board Description. A description of the board, the sockets and connectors mounted on it, and any special signals for which there is printed circuitry on the wire-wrap board.
(*iii*) Chip Type Descriptions. A description of each chip type.
(*iv*) Wire List. A list of chips and the signals wired to them.

The commands used in each of these sections of a circuit description are described briefly below.

(*i*) **package type** For each package type one must give its physical size and the number of pins attached to it. For each pin one must give the pin number and its coordinates. A shorthand notation for describing the positions of equally spaced pins is shown in the example above.
(*ii*) **board** The board name, number of sockets, and number of I/O connectors must always be provided. Additionally, one can specify the relationship between the board coordinate system as used in the circuit description and the coordinate system used by a wiring machine.
(*iii*) **socket** Sockets differ either in size or in the range of package types that can be plugged into them. For each socket type, one must list the package types that are acceptable and board coordinates at which these socket types occur. For example, the following two lines specify that there are 15

sockets numbered 22,31,...,148 equally spaced on the board between coordinates 295/660 and 295/100. Each can hold either 14-pin or 16-pin packages.

```
.s   SOCKET16 DIP16 DIP14
.sb  22 295/660  −9  148 295/100
```

(*iv*) **I/O connector** For each I/O connector one must give its physical size, its board position, and the number of pins it contains. One must also give the coordinates of each pin relative to the position of the connector.

(*v*) **special signals** A special signal is distributed by printed circuitry and made available for wire-wrapping by pins mounted at various places on the board. One must give the name of the signal and the locations of the pins. For example,

```
.v   0  GND
.vb  1 110/740  −  18 110/50
```

defines the positions of 18 equally spaced ground pins.

(*vi*) **chip type** For each chip type, it is required that one specify the package type and list the numbers of the pins which carry output signals from the chip. All other pins are assumed to be chip inputs. Optionally, one can specify that certain pins are to be wired to special signals. For example, the following describes a chip in which there are four outputs.

```
.t   2509 DIP16 8 16
.to  2 7 10 15
```

Special signals (GND and VCC) are connected to pins 8 and 16.

The wire list is divided into pages corresponding to pages of the original circuit drawing. The chips on each page are described separately, and for each chip the signals connected to it are listed. For example,

```
.c    OVR 74S74
.cs   12
clock+   3
aluv+    2
ovr+     6
```

describes a chip called OVR which has signals connected to pins 3, 2, and 6. The chip is to be placed in socket number 12.

V. WCHECK: TO CHECK A CIRCUIT DESCRIPTION

Suppose that the file **therm.chp** contains definitions of packages, sockets, and chip types. The file **therm1.w** contains a wire list. Together these files completely describe a circuit and the following command line will check that the description is self-consistent.

```
wcheck  therm.chp  therm1.w
```

The consistency checks performed by **wcheck** are relatively simple. For example, it checks that each pin is used for only one signal, that all chips assigned to one package are of the same type, and that at most one package is assigned to a socket. Loading rules are not checked but one can obtain a list of signals that do not connect to exactly one output pin, or are not connected to at least one input pin, or connect to more than 10 input pins. Unused inputs to a chip can also be listed.

wcheck will generate a cross reference listing. For each signal there is a list telling what pins are visited, where those pins appear in the circuit drawings, and whether the pins are chip outputs. For each occupied socket, **wcheck** gives the types and names of chips assigned to it.

VI. PLACE: TO OBTAIN A BOARD LAYOUT

The purpose of **place** is to assign socket positions to the packages and to assign routes for the wires. Unfortunately, the common criterion for an optimum placement, minimum wire length, is not always adequate and, even if it were, there is no known algorithm (other than enumeration) for obtaining it. A fully automatic placement strategy cannot, therefore, be totally satisfactory. Yet the task is sufficiently complex that some degree of automation is a necessity. The course we have chosen combines automatic placement procedures with human interaction.

Automatic placement is aimed at minimizing wire length. To do that, one must be able to estimate the wire length for any given configuration of the packages on a board. That requires a solution to the wire routing problem.

Wire routing operates under the following constraints: A net of wires connects together a certain set of pins. There can be at most two wraps on each pin so that the net is simply a linear ordering of the pins. The route followed by a wire between one pin and the

next must always be parallel to one or the other of the board edges. Diagonal wiring is not permitted. No constraints are imposed on the number of wires that can be funneled through any section of the board. The minimum wire route problem is similar to the traveling salesman problem and the algorithms used by **place** are based upon the work of Lin.[1]

The package placement routines are based upon the work of D. G. Schweikert.[2] Two routines are involved: One makes an initial placement of packages not already placed anywhere on the circuit board. The other takes those packages already assigned positions on the board and attempts to improve matters by moving them around. Both operate within the following constraints: Packages cover a rectangular area of the board, and no two packages can be so placed that the area covered by one overlaps the area covered by another. Sockets can secure to the board only those packages listed in the socket definition. No socket can hold more than one package, but it may take several sockets to handle a large package. When a package is placed in a socket, the origin of the package coordinate system is made to coincide with the origin of the socket coordinate system (i.e., the lower left-hand corners of the two rectangles are aligned). If the package is larger than the socket, all other sockets falling partly or entirely inside the package rectangle are "occupied" by that package and cannot be used for another package. A connector whose position on the board is always fixed also occupies a rectangular area of the board. A package cannot be placed so that it overlaps the connector. These constraints are considerably more elaborate than those used in other placement programs of which we are aware. They are made necessary by the growing diversity in package sizes and socket configurations.

place starts with the package placement indicated on the logic drawings. It uses the Tektronix 4014 to display the current board layout. Sockets and packages are represented as rectangles with socket numbers and chip names written within them (Fig. 3). Using the cross-hairs, one can point to a chip or socket and, by typing a command letter, cause some change either in the display or in the current placement. For example, one can move a package from one socket to another, place an unplaced package, or remove a package from the board. Other commands invoke the automatic placement routines.

Each package can be in one of three states: It can be unplaced or its placement may be either soft or hard. A soft placement is one that can be altered by an automatic placement routine while a hard

Fig. 3—Display of circuit board layout.

placement can be changed only by the user. An unoccupied socket can be in two states: It can be available for placement or it can be blocked. Packages cannot be placed in blocked sockets. Sockets are blocked and unblocked by user command.

One quite successful method of obtaining a placement is to use the automatic placement and improvement routines on selected groups of packages or sections of the board. Between such optimizations the user inspects the current placement, makes key moves, and changes the soft/hard and blocked/unblocked states of sockets and packages in preparation for the next use of the automatic routines. To assist with this process one can display an airline map of the optimized wire route for wires connected to selected packages. One can also obtain summary information such as the length of wire required by the current placement. Another helpful display is the original logic drawing with the current placement displayed on the drawing for each chip. When a satisfactory placement has been obtained, the drawing files can be permanently updated in this way.

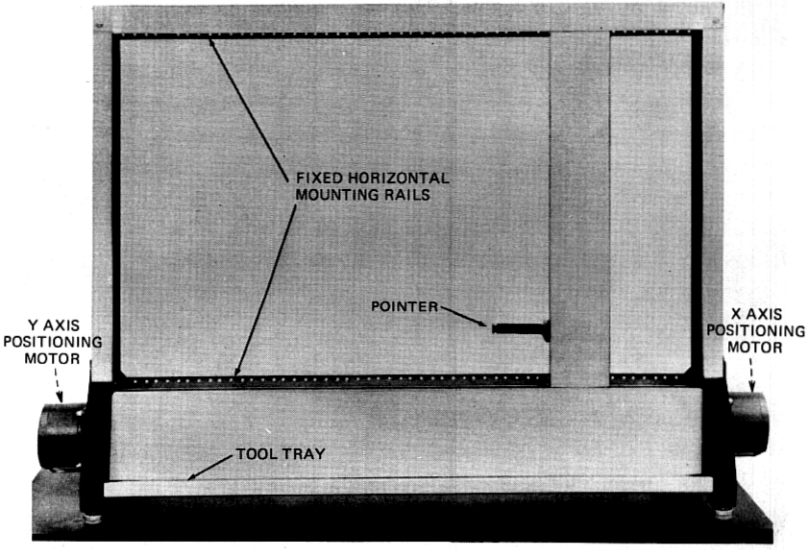

Fig. 4—Standard Logic semi-automatic wire-wrap machine.

A hard copy printout of the board layout can also be obtained from **place**.

VII. WRAP: TO WIRE-WRAP A BOARD

Three different methods of wiring a circuit board are currently supported: use of hand wiring, use of a Gardner-Denver 14YN semi-automatic wiring machine, and use of a fully automatic Gardner-Denver 14F wiring machine. Interfaces to two on-line machines are under development. One, a Standard Logic WT-600 wiring machine, is an inexpensive semi-automatic device for making wire-wrapped boards (Fig. 4). The other, a custom-made device, is used to assemble circuits using a more recently developed wiring technique called "Quick-Connect."[3] Each requires a different style of output from **wrap**. The format and information content of a hand-wiring list was designed to suit a professional wireperson. The semi-automatic machine reads paper tape and the more complex automatic machine is driven by punched cards. The last of these has been programmed by C. Wild[4] and **wrap** simply transmits the wire net data as input to C. Wild's program which runs on an IBM 370.

The wire list is produced by building a model of the circuit board in which there is a data structure for each socket, chip, and pin. Special signal nets are then broken up into many small nets by routing wires to the nearest special signal pin. On some circuit boards the special signal pins are part of a socket in the position most commonly required by integrated circuit chips. Wires are not required if the socket is used to hold a chip with standard format. However, the special signal pin must be unsoldered if another type of chip occupies the socket. This is one of the special cases that **wrap** handles by modifying the circuit board model before generating the final wire list.

VIII. IMPLEMENTATION

The design aid programs are written in the C language and, for speed and portability, avoid use of floating-point arithmetic. The only assembly code program is a subroutine that uses double-length, fixed-point arithmetic to scale graphic vectors.

Standard operating system features are used. In particular, the graphic terminal is connected as a normal character device that is operated in RAW mode (no operating system interpretation of the character stream).

Twenty separate programs communicate with one another using temporary files. Standard programs such as **sort** and the shell are also used in this way. The four command programs **draw**, **wcheck**, **place** and **wrap** are steering programs that do the necessary "plumbing" to connect together a number of subprocesses. The main work load is performed by the subprocesses. In this way the relatively complex tasks required for circuit design and analysis are broken into manageable pieces.

Memory space requirements are further reduced by using as simple a data structure as possible. For example, the graphic editor represents each chip and wire as a named list of points and shapes. No attempt is made to store cross-reference information in the data base if it can be computed directly. Thus the programs use memory space efficiently and CPU cycles less so.

The largest programs are the drawing editor and the placement program. The drawing editor requires 13,000 words of main memory plus enough space to hold one page of a logic drawing. Typical drawings occupy less than 3,000 words. The placement program requires about 11,000 words of main memory plus space to store a description of the physical characteristics of the circuit. The

latter is approximately 3 words × the number of dual in-line package pins plus 25 words × the number of packages (about 7,300 words for 100 16-pin packages).

Communication with the computing center machines is controlled by a single program whose arguments include the job control language required for each particular purpose. Shell sequences contain the job control instructions for standard tasks using the Honeywell 6070. Different installations of the UNIX operating system require different job control instructions and use different communications facilities. By concentrating the communications functions in one program, the task of installing the design aid programs on different installations is simplified.

IX. CONCLUSION

The objective in preparing these design aid programs included the hope that one day it would be possible to go automatically from schematic circuit drawing to finished circuit board. We have come close to achieving that objective for wire-wrapped boards. However, there is a need to be able to make printed circuit boards from the same circuit drawings. Some steps in this direction were taken in the form of an experimental translator which converts between Circuit Description Language and Logic Simulation Language as used by the LTX system.[5]

X. ACKNOWLEDGMENTS

The design aid programs were a joint project by G. G. Riddle and the author. We were assisted by J. Condon and J. Vollaro, and received valuable help in chosing algorithms from S. Lin and D. G. Schweikert. C. Wild assisted with the interface to the Gardner-Denver machines, and D. Russ contributed to the development of an efficient hand-wiring list. The on-line wire-wrap machine interface was developed by E. W. Stark under the supervision of J. Condon.

REFERENCES

1. S. Lin, "Computer Solutions of the Traveling Salesman Problem," B.S.T.J., *44* (December 1965), pp. 2245-2269.
2. D. G. Schweikert, "A 2-Dimensional Placement Algorithm for the Layout of Electrical Circuits," *Proc. Design Automation Conf.* (1976), pp. 408-416. IEEE Cat. No. 76-CH1098-3C.

3. "New Breadboard Design Cuts Wiring and Testing Time," Bell Labs Record, *53* (April 1975), p. 213.
4. C. Wild, "WIREWRAP—A system of programs for placement, routing and wire-wrapping," unpublished work.
5. G. Persky, D. N. Deutsch, and D. G. Schweikert, "LTX—A Minicomputer Based System for Automated LSI Layout," J. Design Automation and Fault Tolerant Computing, *1* (May 1977), pp. 217-255.