AUTOMATIC GENERATION OF LOGIC DIAGRAMS

by

James A. Smith

Research Report CS-76-26

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada
May 1976

ACKNOWLEDGEMENTS

This report is based on the Ph.D. thesis "Automated Generation of Logic Diagrams" by J.A. Smith, October 1975. The guidence and support of Dr. J.G. Linders through this research project is gratefully acknowledged. This work was supported in part by the National Research Council under grant NRC A7640.

ABSTRACT

This report studies the automated generation of logic diagrams. Logic diagrams play an important role in the design, development and maintenance of logic circuitry. The automated production of logic diagrams results in increased speed and accuracy.

This report discusses the logic diagram problem and gives a set of objectives and requirements that an automated logic diagram generation system (ALDGS) should meet. The structural information of a logic circuit is represented by graphs and new methods are developed for classifying and transforming this information for use in the generation of logic diagrams. The steps involved in automatically generating a logic diagram are: partitioning the logic circuit, placement of components on the sheet, and routing the interconnections between the components. Methods are given for partitioning and placing the logic circuit elements on a logic diagram sheet and objective criteria are established for evaluating the placement. A new routing algorithm is given which reduces interconnection crossings on the logic diagram.

The algorithms and techniques have been implemented in a system which takes a pseudo-boolean representation of a circuit and generates the appropriate logic diagram(s). Examples are given of several circuits for which the system has generated the logic diagrams automatically.

TABLE OF CONTENTS

Appendix D. Pseudo-boolean Input Language Descriptions for the Automatically Produced Logic Diagrams of Chapter 6

0.0 Introduction.

In the mid-fifties the subject of computers designing other computers became a popular subject. People began to recognize that the design and development of a complex digital computer involved many routine and repetitive operations that lent themselves quite readily to mechanization. These early attempts at design mechanization formed the basis for what has become known as design automation (DA). The term *digital design automation* is usually taken to mean the art of applying computers and computer based techniques in the design of digital systems. A *digital design automation system* (DDAS) refers to a collection of programs which are used to assist the digital systems designer. These programs are usually defined around a file system or data base to assist in the design, analysis, and documentation, and in the generation of manufacturing data.

This report concerns itself with one segment of a design automation system, namely that of producing digital design documentation and in particular the *automated* generation of logic diagrams. Logic diagrams are a very important part of the documentation process. They are the basic source of information about a digital design from its conception through to the final design stages. When the design is complete they are used in the manufacturing and maintenance of digital equipment. Because of their importance, logic diagrams must be

produced quickly and accurately. If they are produced quickly there is a very short time lag until changes are reflected in the drawings and this may result in reduced design time. The need for accuracy speaks for itself. Speed and accuracy should result if a computer is used and hopefully the further benefits of decreased manpower and reduced costs would also be present.

In any design automation system, some method is needed to input a description of the logic which is to be processed. The common methods are either to capture the data from a logic sketch or to provide the information in the form of logic equations. From the logic sketch, people specify actual locations on a sheet for each component. Since the logic equations are a low level description of the logic it is also possible to specify these sheet locations in the logic equation description. The use of automatic generation techniques would allow, if desired, omission of this location information. A third method of generating the logic description is through the use of a high level computer hardware description language (CHDL). A survey of such languages in contained in [SU 74]. Since these languages usually describe hardware at a level much higher than boolean equations, it is not easy to specify positioning information and thus the automatic generation of logic diagrams is a necessity here.

In order to produce usable logic diagrams automatically, one has to be able to quantify some of the criteria a

draftsman uses in laying out a diagram. The operations necessary are the breaking of the logic circuit into sections which fit on a logic diagram sheet (partitioning), locating the elements of the circuit on the logic diagram sheet (placement), and generation of the interconnecting lines (routing). Much research has been performed on these operations for the generation of PC (Printed Circuit) board designs, but because of the end objective, the requirements and criteria for an acceptable PC board design vary greatly from the objectives of logic diagram generation.

The techniques involved in the generation of logic diagrams have not received much attention in the open literature. Many papers have been published on design automation systems in which logic diagrams were produced but the papers give only very cursory descriptions. The majority of these systems fall into one of the two following categories:

1) component positions are encoded from hand drawn sketches or are determined manually, or

2) some form of automatic positioning of components is performed but no mention is ever made of the techniques used.

In most cases the routing of connections is performed automatically but the techniques used are carried over from PC board routing schemes with minor modifications.

Thus, there has been no concentrated attempt to examine the algorithms needed for automatic generation of logic diagrams. This report addresses itself to this problem. It discusses each aspect of the generation of logic diagrams. The suitability of existing PC board techniques is discussed and where they are inadequate new methods are given. The algorithms and techniques have been implemented and thus form an automated logic diagram generation system, called ALDGS, which takes a pseudo-boolean representation of a logic circuit and produces logic diagrams. Examples of automatically generated logic diagrams are given.

Chapter 1 discusses logic diagrams and the problems involved in generating them and gives the features that should exist in a logic diagram generation system.

Chapter 2 evaluates the logic diagram generation techniques used by several design automation systems. It also discusses current PC board techniques as they relate to logic diagram layout.

The generation of logic diagrams involves the maintenance and manipulation of much circuit information at different levels of complexity. Chapter 3 discusses methods of representing this information in the form of graphs and gives new methods of classifying the nodes of the graph and

relationships between nodes such that the required manipulations may be performed easily and consistently. Chapter 3 also gives the basic configuration of the logic diagrams that will be used in this report.

Chapter 4 discusses the selection of the components of a circuit that should be placed on a logic diagram sheet as well as the positioning of the components. The second half of the chapter discusses placement of the signal inputs and outputs from/to other logic sheets. It gives methods of placing these in conjunction with a method of pin assignment.

Chapter 5 deals with the routing of connections on the diagram. It gives a new algorithm for routing connections on logic diagrams which will reduce crossovers.

Chapter 6 discusses the logic diagrams which are produced by the system using the techniques described in the previous chapters and gives recommendations for future research.

1.0 Introduction.

This chapter discusses the logic diagram layout problem and the features that should appear in a logic diagram generation system. It gives the capabilities expected and the information that the diagrams should contain. Also discussed is the value of interaction in an automated logic diagram generation system.

1.1 The Logic Diagram Layout System.

An automated logic diagram system may exist as part of an integrated digital design system or it may exist as a system by itself. For the purposes of this report it is required that the logic diagram system accept a representation of a logic circuit from which it is required to automatically generate a set of logic diagrams for this circuit.

This representation would be generated from input data which could have many forms. Low level logic equations are the form assumed for this report since they are the common denominator of all other forms. Other forms of input, eg. truth tables, and state tables, may be processed to generate these low level logic equations as may a high level description of the logic in a digital design language.

The system should be able to handle large sections of logic and perform the necessary partitioning of the logic to

fit it on a logic diagram sheet. This partitioning of the logic circuit should cluster heavily connected logic together on a single sheet to avoid cyclic references between sheets.

Intersheet connections between logic diagram sheets should be handled automatically by the system. It should also be able to generate different sizes of diagrams and use different sizes of components depending on the user's requirements.

1.2 Information to be Presented.

As mentioned in chapter 0, a logic diagram is the basic source of information throughout the design and manufacture of a digital system. Among the minimum information that should appear on a logic diagram would be: the logic symbols that represent a given function (AND, NAND, OR, etc.), the interconnections between the logic symbols, and the input-output pads indicating logic signals from/to the external environment and from/to other logic sheets. The logic signals should be identified according to their names in the input form and the source and destination information should also be included for off-sheet signals.

The system should have the capability to handle all types of components currently in use for digital logic diagrams, i.e. gates, flip-flops, and other elements, and be flexible enough to allow new components to be defined by the user. Then, on one set of diagrams a user could represent a

functional unit, such as an adder, by a box with its input and output circuitry and with another set of diagrams give the internal circuitry for the adder itself. The system should have the ability to use logic symbols which conform to the MIL-specifications [DEFE62]. For elements with unique inputs and/or multiple outputs, the input leads and the output leads should be identified on the logic diagram.

The remaining basic information that must be included is the standard drafting data such as circuit name and identification data, designer, sheet numbering information, and update or revision data.

The next level of information to be included, if available and desired, would be the information that describes the implementation of the logic circuit. This would include the chip types which were used to physically realize a particular function and even the function number if the chip contained many similar functions.

Furthermore, if the PC boards had been previously designed for the circuit then the PC board number and position of the chips on the PC board as well as the pin numbers could be included on the logic diagram. For those logic signals which enter or leave a PC board, the board pin number information should be placed on the diagram.

1.3 Analysis of the Problem.

Generation of logic diagrams involves taking some description of a logic circuit which represents the components and their interconnections, and embedding the circuit on one or more logic diagram sheets. Solution of this problem requires the solution of the three problems of: 1) partitioning, 2) placement, and 3) routing. Because of their interrelation, these three problems would best be solved by a simultaneous approach. However, because of the difficulty in solving these problems simultaneously, they have historically been attacked separately.

Partitioning involves the selection of a suitable subset of the circuit to be placed on each logic diagram sheet. If heavily connected components are kept together on logic diagrams the intersheet connections will be reduced and hence their readability will be improved. There is however another consideration. Besides using connectivity criteria some thought must be given to the placement of components in an attempt to improve the layout of a sheet.

Placement refers to mapping the components of this logic subset into positions on the logic sheet. In LD layout all placement is performed to aid the clarity of the logic diagram. For this reason it is preferable that the diagram have its inputs on the left and its outputs on the right. Those components with all inputs external to the sheet should be to the left of all logic that they are input to, and those

components with outputs external to the sheet should be on the right of those components that input to them. The remaining components should be suitably placed between them. The logic diagram will be more understandable and easier to follow if the components are placed to reduce crossovers and the number of reversed connections†. Thus the 'flow' of the logic should be from left to right as much as possible. If unconnected sections of logic are to be placed on one logic diagram then these sections should be separated from each other. The input and output signals should be placed close to the components to which they are connected.

Routing is concerned with drawing the interconnection lines between the components. For logic diagrams, a connection between two components should be as direct as possible to allow easy tracing of connections. It is further desired that the routing scheme be always successful. The routing of interconnection lines should be such that the number of crossings is mimimal.

1.4 Cost.

1.4.1 General Considerations.

The cost of generating a solution to the partitioning, placement, and routing problems determines the feasibility of automatically generating logic diagrams. For any given circuit

---

† A reversed connection is a connection whose source component is to the right of its destination component. A reversed connection which forms part of a loop in a circuit is called a feedback connection.

that is to be represented on a logic diagram many solutions are possible and although each solution has a particular associated cost, it is often difficult to evaluate the quality of the solutions. The crossing of interconnection lines is allowed and in fact a crossing of lines is much better than a line wandering all over the sheet in order to avoid one crossing. The criterion of 'goodness' depends mostly upon ease of comprehension. The number of logic diagrams to be produced does not affect the cost of generating the layout. Logic diagrams should be as readable as possible and produced as cheaply as possible.

1.4.2 Interaction.

The value of interaction is determined by the quality of the solution desired and the associated cost. Given the need to generate large diagrams it is doubtful that interactive graphics intervention is feasible at the partitioning or even the initial-placement and routing stage because of the large amount of information involved. Adequate algorithms may be developed for partitioning the logic, performing a placement, and doing the routing.

Interaction does have a place in revision of logic diagrams. In this case, if logic is to be changed or if a small bit of logic is to be added to a logic section and it may be done on the same sheet, then interaction could be used to place the logic such that the original circuit layout on the sheet remained the same.

1.5 Summary of Desirable Features for LD Layout.

The following is a rearranged summary of the desired features as proposed in the earlier sections. An automated logic diagram generating system should:

1. produce legible diagrams at a reasonable cost

2. handle large sections of logic circuitry and different sizes of diagrams and components with heavily connected logic grouped together on a sheet

3. have sheet inputs on the left and sheet outputs on the right with the components suitably placed such that the 'flow' is from left to right and reversed connections are minimized. Unconnected sections of logic should be separated on the diagram and input and output connections should be placed close to the components to which they are connected.

4. have a routing scheme which routes 100% of the interconnections using paths which are as direct as possible, and which attempts to minimize crossings without violating item 3

5. be able to handle all types of components including user defined ones and be able to generate MIL-specification symbols

6. have provision to present information concerning: circuit identification, circuit implementation, signal names, and off-sheet signal references.

2.0 Introduction.

Many of the major computer manufacturers have had design automation systems in existence for many years and logic diagram capabilities have existed in these systems almost from the start. Generation of logic diagrams involves the three steps of partitioning, placement, and routing. These operations may be performed either manually or automatically.

The use of logic diagrams in an automated design system first appeared in [KLO058] from IBM. In this system, an engineer's logic sketch was encoded and reproduced on a line printer. Placement information was specified for the components but the routing was performed automatically [WARB61a], [WARB61b]. A more recent paper [HAYA70] from Fujitsu describes a similar system which also uses the line printer as an output device. The systems [DEHA66], [WISE67], and [SAND72] also take component coordinate information from hand drawn schematics but in these cases a plotter is used as the output device. The system described in [BALD67] uses a manual initial placement followed by an automatic placement refinement.

Early attempts at automatic placement of components were made by [ROCK61] and [KALI63]. The latter system also used a cathode ray tube to preview the logic diagram before drawing it on a plotter. The systems described in [ROTH65] and

[LINN71] performed automatic partitioning of the boolean equations to select components for each sheet and then automatically placed the components on each sheet.

Interactive graphics was used in the generation of logic diagrams by [ALAI67]. In this system coordinate locations were not required as input data but a light-pen and cathode ray tube were used to interactively place the modules. Others to take this approach are [UHLI68] and [HOWI71].

In terms of the input description, boolean equations, logic sketches, a computer hardware description language [FRIE66], and interactive graphics have all been used, although the most common form was boolean equations.

The above papers describe the logic diagram aspects of the systems of six different companies. The following section discusses each of these systems, gives examples of their output, and shows their drawbacks. The remaining section discusses PC board techniques which are relevant to LD layout.

2.1 Logic Diagram Generation Systems and Their Capabilities.

2.1.1 IBM and Fujitsu.

Both these companies use a printer as the output device. The use of a printer means rapid and cheap generation of documents. A printer does however impose a limit on the size of documents that may be produced. The limit on one dimension (length or width depending on the orientation of the diagram

on the printer) is governed by the printer width and is usually around 14 inches wide with 120 to 133 columns in this width. To generate the required drawing, both systems use a character memory image of the print page.

Because of a limited character set and the low resolution, the logic diagrams of the above two manufacturers are limited to representing logic elements as rectangles with a mnemonic to designate the type (ie. AND, OR, FLIPFLOP, etc.). Thus they are unable to achieve the MIL-STD-806A specifications.

The input to the Fujitsu system and the early IBM system was in the form of logic diagrams from which the placement and connection data were extracted. In both cases the routing of the connections was performed by computer. IBM progressed to accepting boolean equations as input but their placement procedures were not specified†. In the paper by Uhlik an interactive system for updating these logic diagrams was described but the same type of drawing was output.

Examples of diagrams generated automatically by IBM are shown in Figures 2.1a and 2.1b. In these figures, several unnecessary crossings of interconnections exist. These crossings are circled on the diagrams. In Figure 2.1b, the line separates 2 groups of components which are not connected to each other. These groups should have been separated to make the diagram clearer.

---

† A preliminary effort in this area was described in the paper by Rocket [ROCK61] but this effort seems to have been ignored.

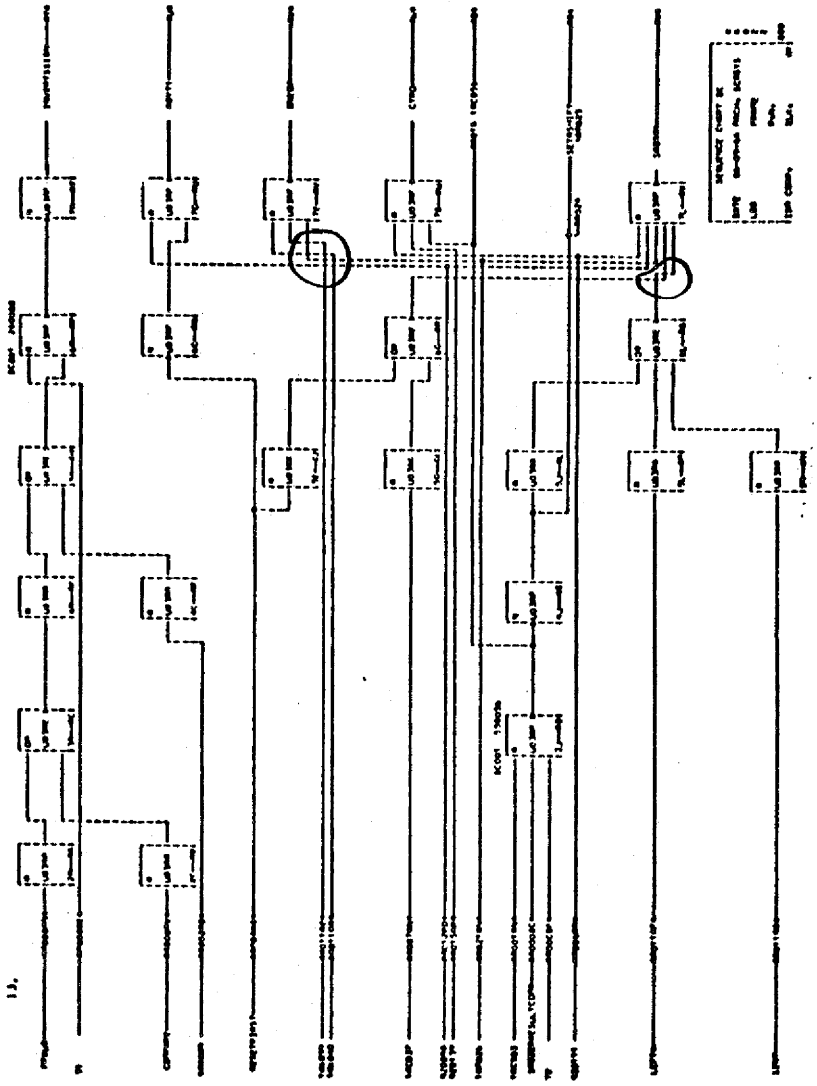


Figure 2.1a Example of a Logic Diagram Produced by IBM [ROTH65]

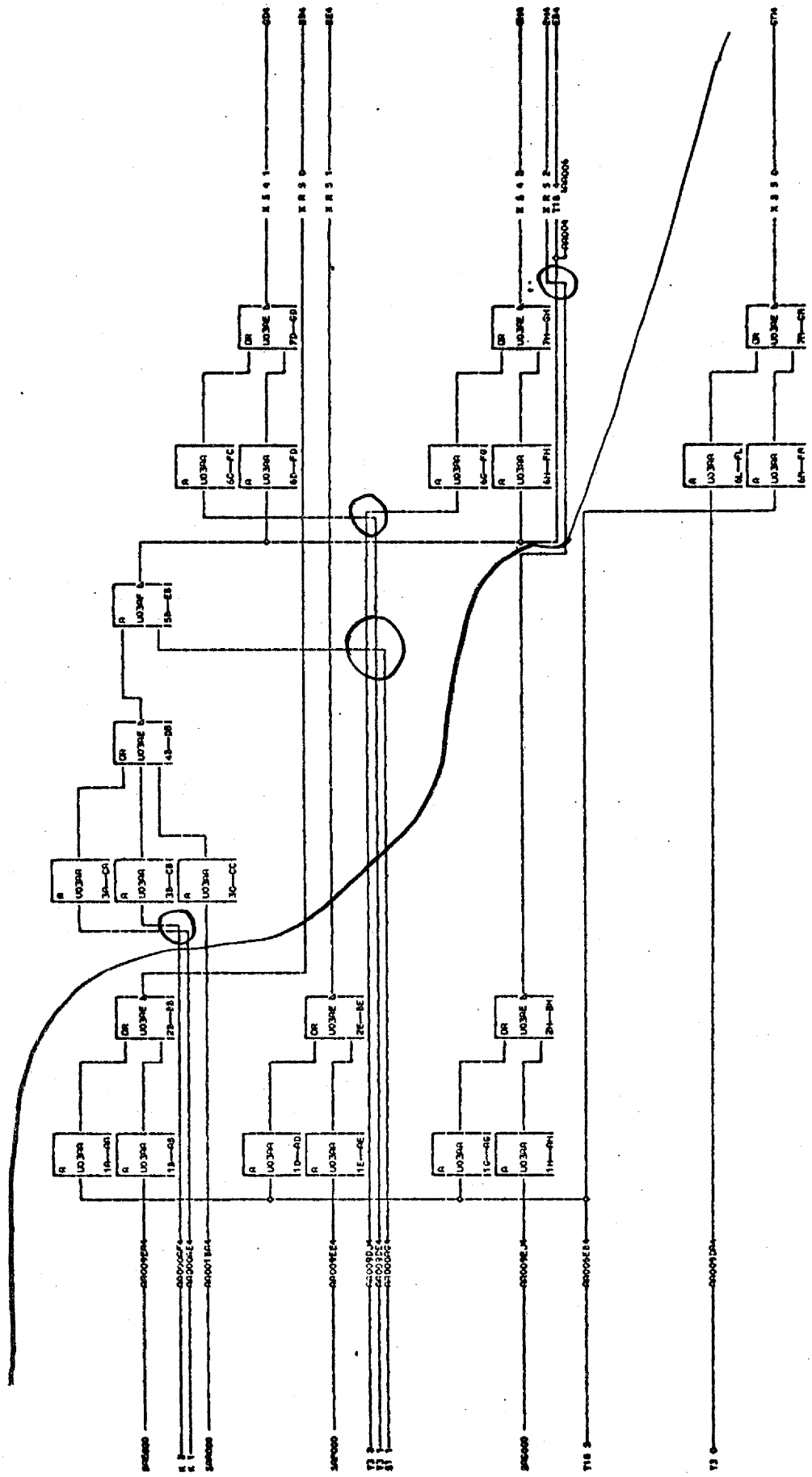


Figure 2.1b Example of a Logic Diagram Produced by IBM

An example of the diagrams produced by the Fujitsu system is shown in Figure 2.2. This diagram was produced with manual placement and automatic routing. The routing is acceptable except for the reversed connections. An example of this is the interconnection between the top two components in the second column. This connection would be improved by routing the connection between the components. In fact, if the indicated components were moved, there would not be any reversed connections on the diagram. The illustrated circuit is regular in structure and combinational, thus it does not give any indication of the ability of their system to handle routing on a complicated diagram.

2.1.2 Bell Telephone Laboratories.

The three papers mentioned from the Bell Telephone Laboratories are entirely concerned with the generation of schematic diagrams. The first paper [KALI64] is concerned with the methods for generating optimum layouts for electrical diagrams. Although this system was an excellent attempt at automatic placement and routing, it contains several drawbacks.

The problem of feedback connections was mentioned only in describing the type of inputs allowed and it is not clear how these would be handled. Furthermore, their technique of detecting crossings using a geometric solution of lines is not acceptable for large diagrams. Another problem with this system is that it was limited to 'L' shaped interconnection

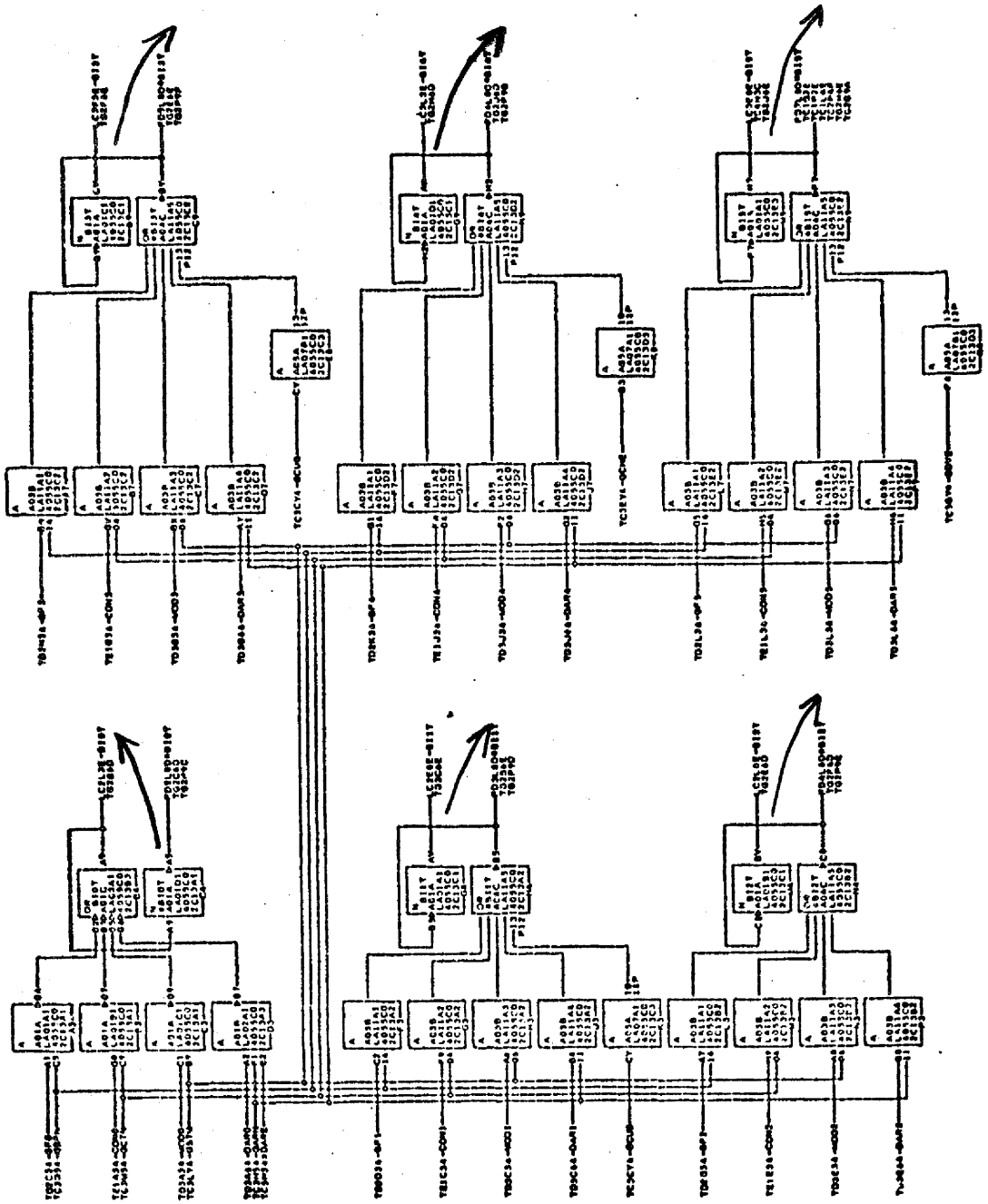


Figure 2.2 Example of a Logic Diagram Produced by Fujitsu [HAYA70]

runs. It was however, a good attempt at solving the component placement problem. This was an early attempt at performing what has become known as initial-placement and placement refinement operations.

Production systems for generation of logic diagrams were described in [DEHA66] and [ALAI67]. The first paper describes a system which accepts input which is coded from a logic sketch. The information coded consists of element type inputs and outputs, orientation and coordinate information. Thus the element positions are fixed leaving only routing to be done automatically. The routing algorithm works on a basis similar to maze threading using a grid which defines allowable paths for the interconnections†.

The other paper [ALAI67] describes their GRAD (Graphics Aided Drafting) program which allows interaction through the use of a light-pen and a CRT. The user uses the light-pen to place or rearrange the elements on the CRT. The automatic routing is a Lee-type algorithm.

The first system has the handicap of requiring the specification of coordinate information manually on cards

---

† There is a class of algorithms using maze-threading techniques and are based on the work of Moore [MOOR59] and Lee [LEE 61] with simplifications by Majorani [MAJO64] among others. This class of algorithms, which shall be referred to here-after as Lee-type algorithms, is characterized by the use of a matrix or grid to determine allowable paths for interconnections. With this class, first applied to printed circuit board routing problems, the resolution required for routing determines the matrix size which is usually maintained in memory. This type of algorithm will always find a path, if one exists, but the routing may meander all over the diagram on a complicated circuit if it is not restrained.

while the second system requires it either manually on cards or manually through interaction. The ideal approach is automated placement with minor manual modifications only if necessary. For both systems the routing method could require excessive amounts of memory for large diagrams or require much time if the data were compactly packed. With a Lee-type algorithm it is difficult to prevent unnecessary crossings of the interconnections.

An example of the diagrams produced by the GRAD system is given in Figure 2.3. The placement for this diagram was specified manually and routing was performed automatically. It contains groups of components which may be moved to the top or bottom of the diagram without affecting the other components since the only connections they have in common are the connections indicated by an asterisk†. The routing is reasonably good but has some jogs which are not necessary. Again this circuit is regular in structure and does not have any feedback connections.

2.1.3 General Electric Company.

In the Automatic Logic Implementation (ALI) system of the General Electric Company, [BALD68], logic sheets are coded in IPEL (Intermediate Programming Engineering Language), from which the system produces manufacturing and documentation output.

---

† For some reason these connections have no source.

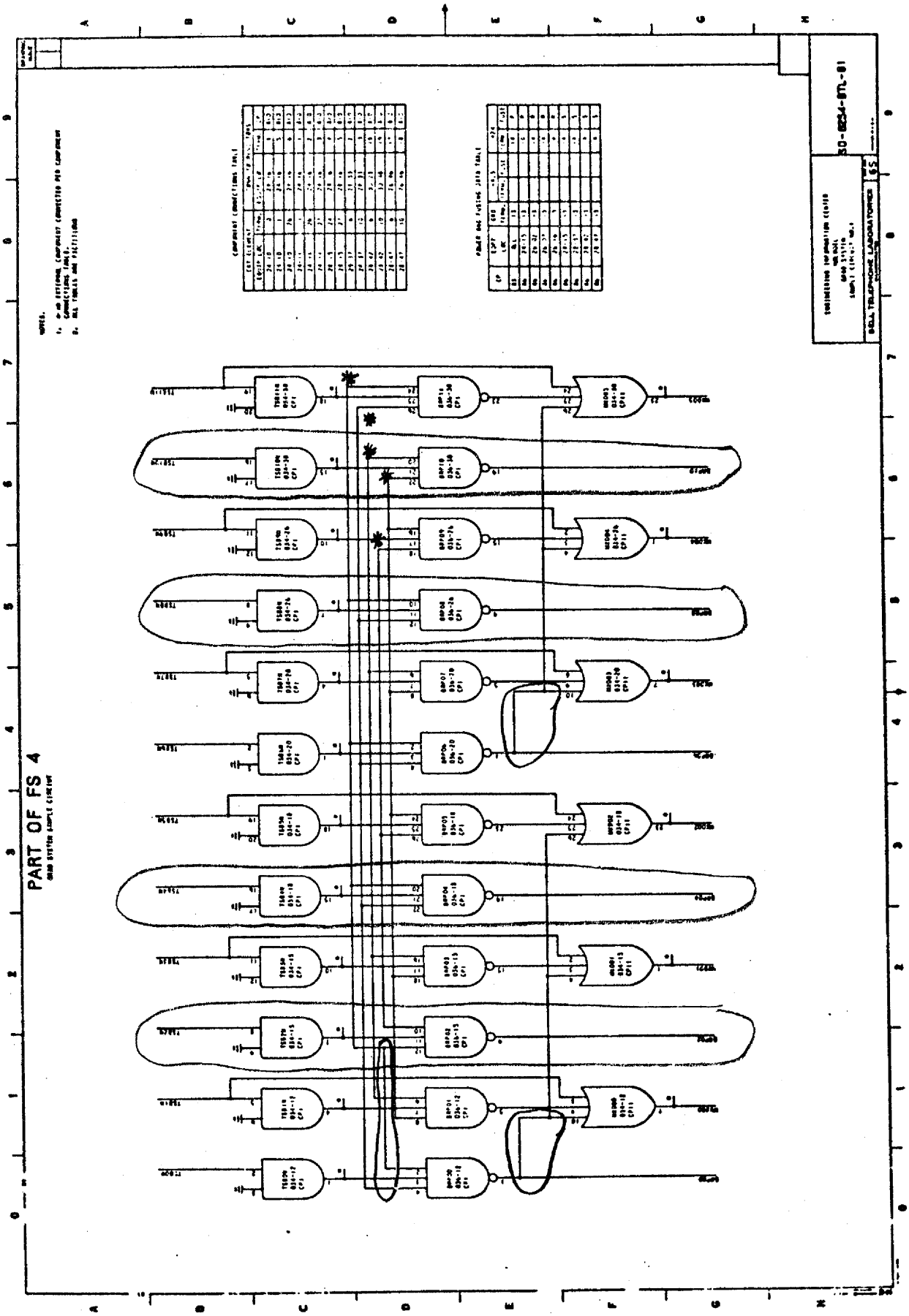


Figure 2.3 Example of a Logic Diagram Produced by the Bell Telephone Laboratories [ALAI67]

To achieve a reasonable positioning of elements from the manual input, the system is prepared to perform a five step process composed of the following steps: a) initial placement b) expand, c) squeeze, d) stomp, and e) align. The initial placement is coded on cards and defines the topology of the elements. The remaining steps are aimed at improving the geometry of the elementst. The alignment process aligns input and output signal names wherever possible to eliminate bends in the paths. The routing of connections is performed using a Lee-type algorithm.

This system goes through many gyrations to achieve good placement just to correct bad aspects of the manually produced logic sketch and may actually fail in this attempt.

An example of the diagrams produced by this system is contained in Figure 2.4. The components were placed manually from a logic diagram as were the positions of the input and output signal names. The routing was performed automatically and generates connections which are difficult to follow. Their router also generates unnecessary crossings, some of which are circled in the diagram.

2.1.4 Sperry-Rand.

For the logic flow diagrams of the Sperry Rand Corporation [SAND72], data is taken directly from original logic sketches. The information input is a) logic and circuit

---

† It is not always possible to correct a logic diagram. Upon encountering this case the program admits defeat.

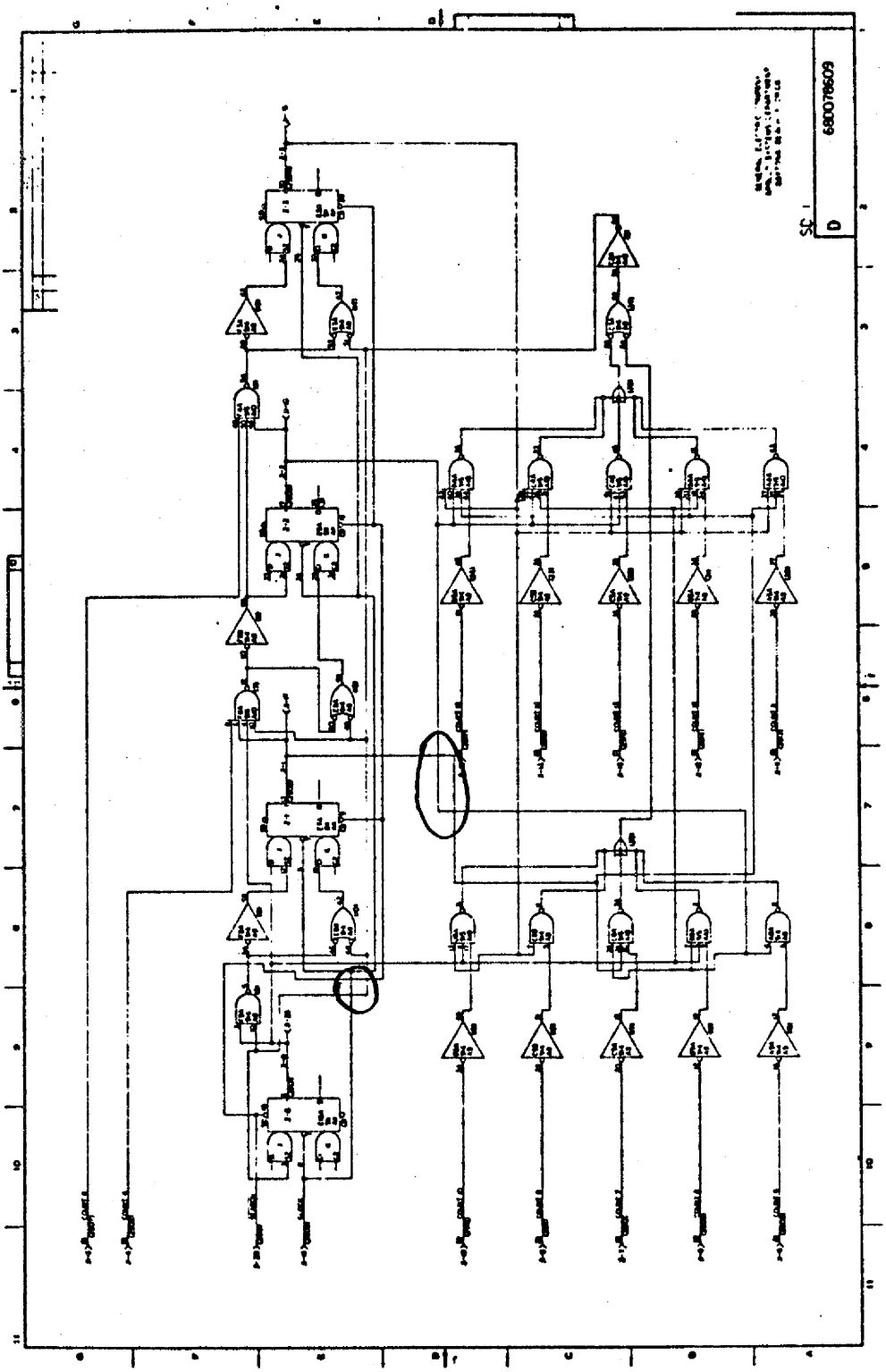


Figure 2.4 Example of a Logic Diagram Produced by the General Electric Company [BALD68]

symbol data, b) interconnection data, and c) signal flow data (i.e. information concerning the disposition of signals which enter or leave the diagram). It is required to specify the location and data for each symbol on the final diagram. Routing is performed using a modified Lee type algorithm.

An example of their diagrams is contained in Figure 2.5. It was generated using manual placement with automatic routing. The routing generates indirect connections. Unnecessary crossings are circled on the diagram. The diagram is cluttered by the fact that there are many output names for each output signal.

2.1.5 Sanders.

The Logic Design System (LDS) of Sanders Associates [LINN71] is a set of FORTRAN programs that converts boolean equations and component descriptions into logic diagrams and other manufacturing information. Data is supplied to indicate dimensions of the logic design, the logical elements to be used, the correlation between logical names and particular devices, and the input and output signals. The system has the capability to handle gates, flipflops, and modules. The Logic Diagram program uses MIL-STD-806A symbology and creates C,D,E, and F size drawings using 5/8, 3/4 or full scale 806A symbols. All modules are laid out one equation at a time until a page is filled, then another page is started. The system looks after relating input and output connectors with the pages where they originate or to where they are going.

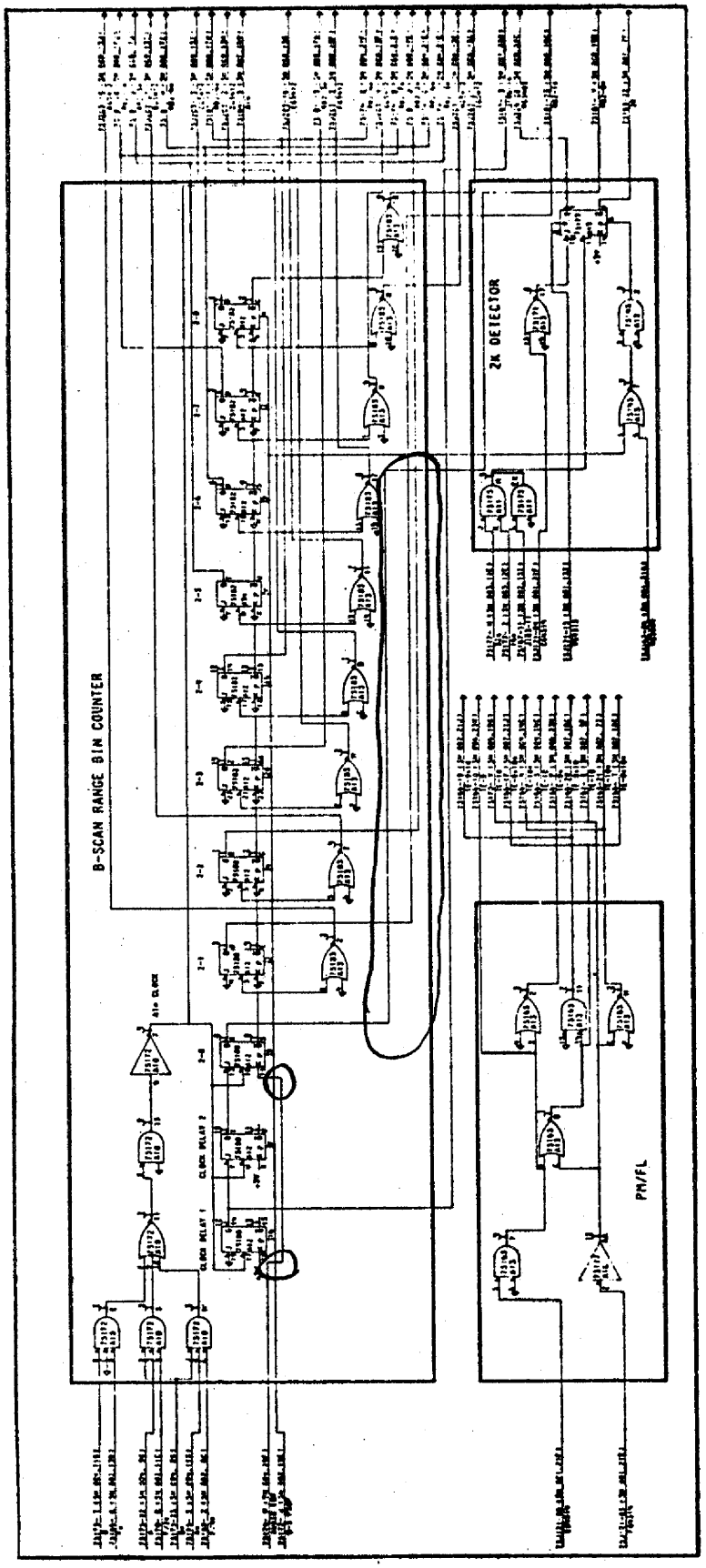


Figure 2.5 Example of a Logic Diagram Produced by Sperry-Rand [SAND72]

The logic diagrams generated by this system suffer greatly since they perform their layout one equation at a time. This means that heavily connected circuitry could be spread over several pages necessitating frequent page turning to trace a circuit. Such a serial assignment, without considering connectivity, cannot begin to optimize placement. Neither their placement technique nor their routing scheme were described in the paper.

An example of their diagrams is contained in Figure 2.6 Both the placement and routing are bad. Connected components are not placed close together and the routing scheme generates unnecessary crossings as indicated on the diagram.

2.2 A Survey of Techniques for LD Generation and Related Problems.

Partitioning, placement, and routing are usually performed for both logic diagram generation and PC board design. The objectives, however, are different in each case. PC boards are usually multi-layered and have many design objectives. It is impossible to directly incorporate all the circuit goals, and thus minimumizing the wirelength is a commonly used approximation. A logic diagram contains only a single layer and the objectives are concerned with making the logic diagrams easy to use and understand.

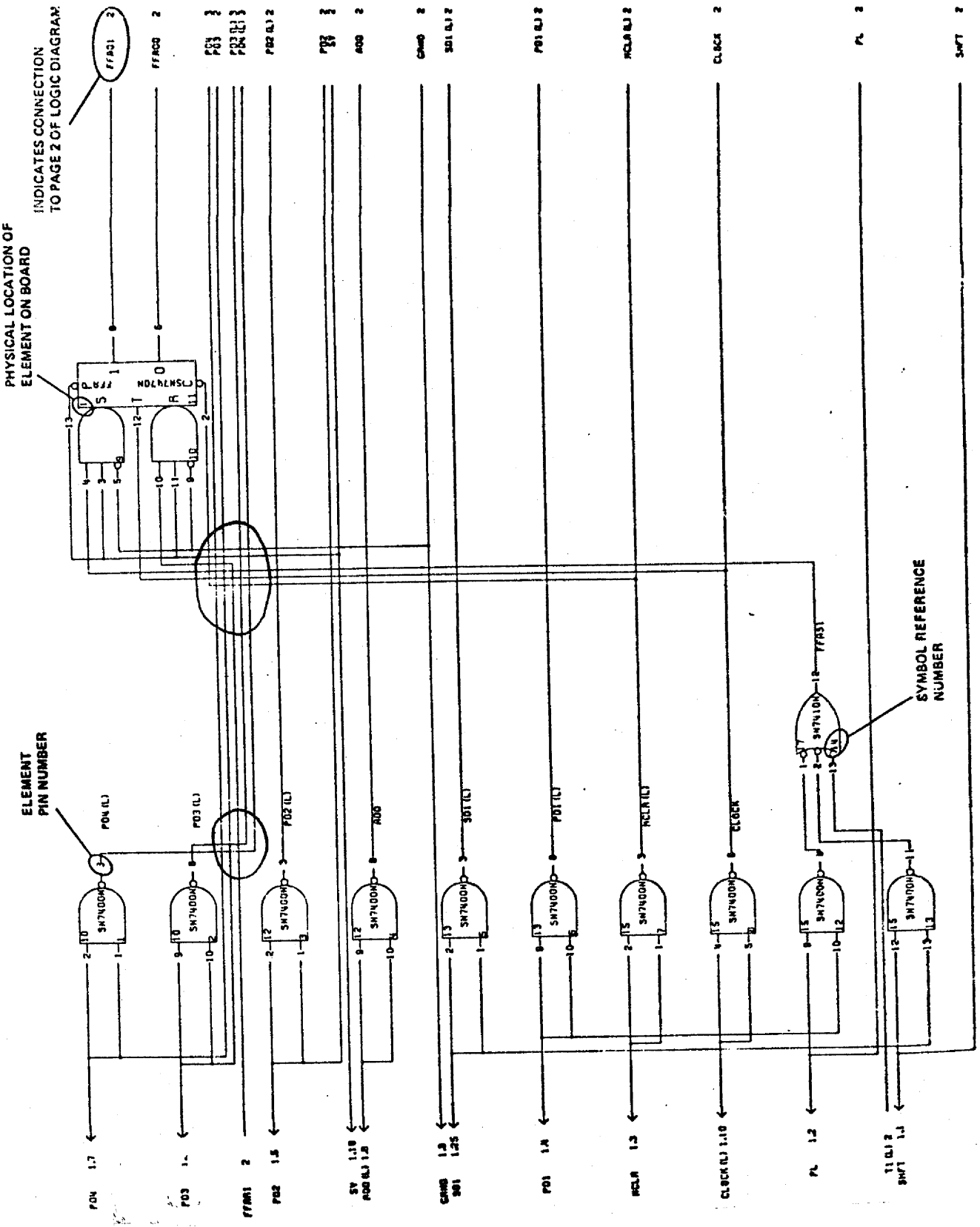


Figure 2.6 Example of a Logic Diagram Produced by Sanders [LINN71]

2.2.1 Partitioning.

A discussion of partitioning techniques for PC boards is given in [KODR72]. PC board partitioning techniques start out with one or more 'seed' components and then add components to the seed(s) according to connectivity. The upper bound on the number of components that may be added is usually a function of space and the number of external connections allowed. The space is determined by the number of positions available and the number of external connections is limited by the number of pins on the PC board.

Connectivity is not an adequate criterion by itself for LD generation, nor are the same bounds acceptable. For logic diagrams, the required size of the components and the logic diagram size determine the number of positions available but it is not desirable that all spaces be filled. The number of spaces that should be occupied is a function of the interconnections of the components that are to be placed on the sheet and thus varies from sheet to sheet. Thus the placement is the deciding factor in determining how many components and which components may be placed on a sheet.

2.2.2 Placement.

Placement is an important topic in PC board design and as a result much has been written on it [HANA72]. Of primary interest are the following two placement techniques:

1) constructive-initial placement

2) iterative placement-improvement

The first placement method is constructive in that at each stage a component is selected and then placed in a position from which it is not moved. The second placement method starts out with a placement configuration and then iteratively perturbs it in an attempt to lower some cost function (usually interconnection length). A common approach is to use a constructive-initial placement followed by an iterative method.

In constructive-initial placement methods, a component is positioned as a function of its previously placed neighbours. As a result an individual component may not occupy an optimum position when all of its neighbours have been placed. Placement refinement methods attempt to improve the placement by perturbing it and testing if the cost function has been reduced. A series of force directed perturbation techniques† are described in [HANA72], [GRAC73a], [GRAC73b], [GRAC73c] and a comparision of force directed techniques is given in [WILS74]. The difficulty with both these PC board methods for logic diagrams is that their objective is to minimize interconnection length, which by itself is not a good objective for logic diagrams.

---

† By calculating a force vector for the components, an indication is obtained of the preferred motion of each component. These vectors indicate the direction in which trial interchanges should be made or in some cases indicate exactly which two components should be interchanged.

2.2.3 Routing Schemes.

Maze-type routing schemes are sequential in nature, routing one connection at a time using a fixed size grid. The grid defines the allowable routing paths at each stage of the algorithm. The number of elements in the grid is determined by the overall dimensions of the routing surface and the required resolution. Since a fixed grid defines the routing paths, a connection routed in the wrong order on a logic diagram may cause unnecessary crossovers or cause the routing to meander in order to avoid crossings. Because of the sequential nature of the routing it is difficult to minimize crossovers.

For logic diagrams, these algorithms usually have modifications to prevent wandering. One method is to limit the number of bends in an interconnection route [DEHA66]. This unfortunately may cause some connections to be unroutable. Such a condition necessitates either rearrangement of the components on the diagram, removal of some logic components, or relaxation of the restrictions to allow the routing to be completed. Alternate methods to achieve complete routing are to use a finer grid (more routing paths, more storage) or to move some components apart sacrificing space elsewhere [UHLI68].

Because of the maze threading techniques, a larger grid will cause these algorithms to be slower. Thus modifications to speed them up involve windowing techniques and depth-first searches.

Although these algorithms are being used successfully for logic diagram routing there are newer algorithms which offer better speed, reduced memory requirements, and because of their nature prohibit wandering. An algorithm by Hightower [HIGH69] overcomes the first two objections. It is fast and requires less memory since it uses a linked list instead of a grid to store the routing obstructions. Unfortunately, it is still sequential in nature and thus encounters the same problem as the Lee-type algorithms.

The routing algorithm of Hashimoto and Stevens [HASH71] is called the channel routing technique and represents an attempt to combine the advantages of the line search technique of Hightower [HIGH69] or Mikami [MIKA68] and the stepping aperture of Lass [LASS69]. Their objective was to increase the speed of wiring and to increase the flexibility to achieve high routing success with a minimum of wire length.

Their method requires that the components be arranged on the board in straight rows and columns separated by horizontal and vertical spaces†. Many wires can be run in parallel within a space so a space is divided into channels. Each channel can contain many non-overlapping wire segments.

This channel routing technique is much faster than the maze-running algorithms and the space requirements are low. Furthermore, the connections are routed in paths which are

---

† A space is an area on the board which extends from one edge to the other.

more direct, since the paths are governed by the end points of the wire, thus preventing meandering.

The major drawback to this scheme for logic diagram routing is that by its nature it causes many crossings and makes no attempt to reduce them. Furthermore, this routing scheme introduces a class of conflict situations which are not adequately handled in their algorithm.

The paper by Mah and Steinberg [MAH 72] describes a method which, from the outset, tries to reduce crossings as much as possible. Their scheme which they called topological class routing, also requires the components to be arranged on the board in straight rows and columns separated by spaces.

The main attraction of this algorithm's approach is the attempt to minimize crossings from the start. However this minimization involves only one side of the board at a time and, as with the algorithm of Hashimoto and Stevens, when all paths are drawn on the same side as in logic diagrams, too many crossovers remain. Further, the above scheme would work well for PC board routing assuming that the connections were 'nice'. It does not make clear how they deal with unroutable wires (i.e. those that cause intersections).

Their routing classes do not account for diagonal connections which were not in adjacent rows. It is these connections which cause most of the problems due to the many different cases and end point conditions. One way these could

be handled would also be after the fact by a Lee-type algorithm.

2.3 Summary.

In the systems discussed, the partitioning techniques were either manual or automatic. The automatic methods of [ROTH65] and [LINN71] operate on a circuit at the logic equation level instead of the more basic level of component connectivity. The placement information is also specified manually or generated automatically. The criteria used in the automatic placement systems are not specified except that the components are also chosen at the equation level. The specified routing schemes are all automatic and are based on maze-threading techniques in which connections are routed sequentially. As a result it is difficult for these routers to avoid unnecessary crossings since previously routed connections may not have been placed optimally.

Standard PC board techniques for partitioning, placement and routing are not adequate for LD layout The partitioning methods are based on connectivity which by itself is not an adequate objective. The placement methods try to minimize interconnection length instead of trying to achieve a flow of the logic. The routing techniques are not oriented towards reducing crossings on just one surface and/or do not provide direct paths for the interconnections. The problem of signal name placement does not appear in PC board design.

3.0 Introduction.

This chapter gives methods of representing logic circuits for the purpose of generating logic diagrams and gives the configuration of the logic diagram itself. It gives a set of graph representations which are used to store the information about a circuit such that large sections of circuitry may be manipulated easily. There are several stages of representations. The end result is an intersheet connection list and graphs which represent the components which are to be placed on each sheet. This unique classification scheme allows the relevant circuit information to be maintained for and throughout the circuit layout operations. It allows easy generation of the intersheet connections and the labelling of them with the to/from information for simple signals as well as for those signals which go to/from the external environment and between different sheets at the same time. The tables giving the actual classifications and the transformations between the representations are contained in Appendix A.

The last section gives the configuration of the logic diagram that is be used so that the objectives of chapter 1 may be realized.

3.1 Definitions.

3.1.1 Graph. A graph G consists of an edge set EG and a vertex set VG. Each edge in EG is incident to a pair of vertices in VG, called its ends.

3.1.2 Directed Graph (Digraph). A graph in which for each edge @ one of its ends is designated the positive end p@, the other the negative end n@. Such an edge is called a directed edge.

3.1.3 Bipartite Graph (Bigraph). A bigraph G is a graph whose vertex set VG can be partitioned into two subsets V1 and V2 such that every edge of G joins a vertex in V1 with a vertex in V2.

3.1.4 Bipartite Directed Graph (Bi-Digraph). A bi-digraph B is a bipartite graph in which all edges of B are directed edges.

3.1.5 Valence. For X in VG, the valence of X in G, is the number of edges in G incident to X.

3.1.6 Invalence. For X in VG, the invalence of X in G, is the number of edges in G with negative end in X.

3.1.7 Outvalence. For X in VG, the outvalence of X in G, is the number of edges in G with positive end in X.

3.1.8 Source. A source is a vertex of invalence zero and outvalence > 0.

3.1.9 Sink. A sink is a vertex of outvalence zero and invalence > 0.

3.2 The Logic Circuit and Its Environment.

A logic circuit consists of components (eg. NAND, NOR, flipflops, etc.) and signal nets† which interconnect the components. For any piece of logic equipment that is defined there exists an environment surrounding it with which the equipment must interface. To indicate this interface to the environment a special type of component is defined which is called a terminal component. There are both input and output terminal components. A terminal is a component for most intents and purposes with an input terminal defining a source signal net of the same name (can have many outputs, no inputs) and an output terminal is a sink for one signal net (can not have any outputs). In fact an output terminal 'O' does define a signal net whose input is the input to the terminal 'O' but the output is never used. An output terminal may have only one signal node as its immediate input and a signal node may only be input to one output terminal node.

An example of a simple circuit is shown in Figure 3.1. In this figure the input and output terminals are A, B, C, D, E, F, G and I, J, K, L respectively.

---

† A signal net is a set of connections which have the same source signal.

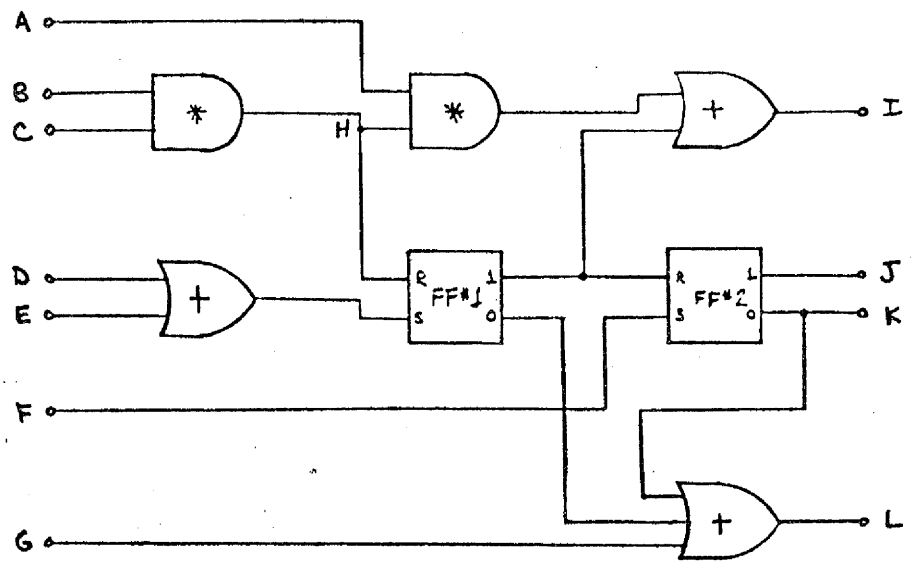


Figure 3.1 Logic Circuit 1

3.3 Representative Graphs.

One way to represent a logic circuit for performing data manipulations is in the form of a graph. To represent the structure of the logic circuits a bipartite directed graph is used. This type of graph was used by Kodres [KODR72]. By definition such a graph contains two types of nodes. For logic circuits the first node type represents the components of the logic circuit (AND, OR, flipflops, terminals, etc.) and is called a component node, while the second type of node represents the signal nets and is called a signal net node.

For the logic circuit of Figure 3.1, the corresponding directed bipartite graph is shown in Figure 3.2. This graph will be called the representative graph of the logic circuit.

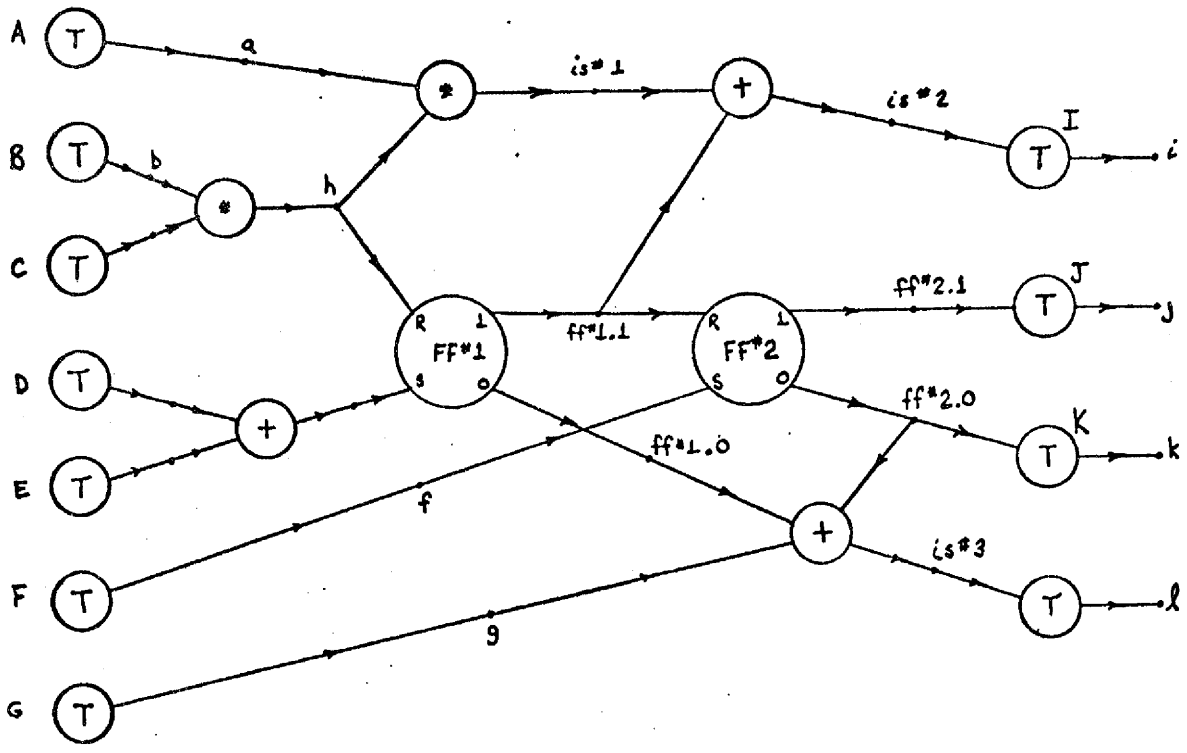


Figure 3.2 Representative Graph for Logic Circuit 1

The small nodes with lower case tags represent the signal nets and the larger nodes with upper case tags are the component nodes. In addition to the tags, the component nodes also have an identifier to further classify the component node as to its type (+ - OR, * - AND, T - terminal). The arrows on the edges of the graph indicate the direction of the edge from the source to sink. For gate type components, all input leads to the gates are identical, whereas for components such as flip-flops the input and output pins are unique and must be uniquely specified. The components with identifiers FF#1 and FF#2 represent flip-flops.

3.4 Reduced Graphs.

The representative graph is rather complicated since it is an exact representation of the entire logic circuit. For many operations on the circuits, much less information is actually needed and a simple bipartite directed graph is sufficient. Before extracting such a simple bipartite graph from the representative graph there is one preliminary simplification that can be made. The terminal components are not regular components as their only purpose is to signal a connection to the external world and they do not take up the space of a regular component on the logic diagram. As such they may be discarded when generating what will be called the reduced graph. The signal nets from input terminal nodes must be maintained and they will be called external input signal nets. The output signal nets of output terminal nodes serve no purpose and in fact are disconnected when the output terminals are deleted. Thus they may be discarded. Instead, those signal nets which are sources to the output terminals become important and are of two types:

1. those which are a source for an output terminal only, and

2. those which are a source for an output terminal and a source for one or more components other than output terminals.

The former become what will be called external output signal nets while the latter will be called dual external

signal nets since the node representing the signal net is both an external output and an external input node due to the new status we have imposed upon it.

This situation is best explained by considering the representative graph of Figure 3.2 and its corresponding reduced graph of Figure 3.3. In Figure 3.3 the signal nets a, b, c, d, e, f, g are external input signal nets and the signal nets IS# 2, IS# 3 are external output signal nets. Being an external output of one gate and an external input to another gates makes 'ff#2.o' a dual external signal net.

In addition to the external types of signal nets there will, of course, still be some internal signal nets left.

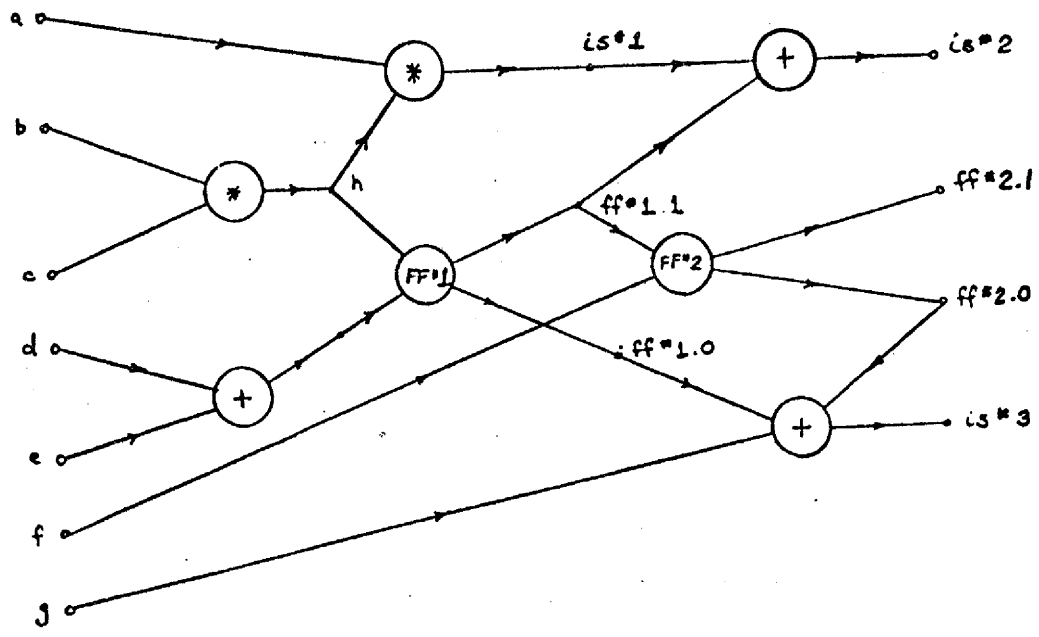


Figure 3.3 Reduced Graph of Figure 3.2

3.5 Partitioning the Reduced Graph.

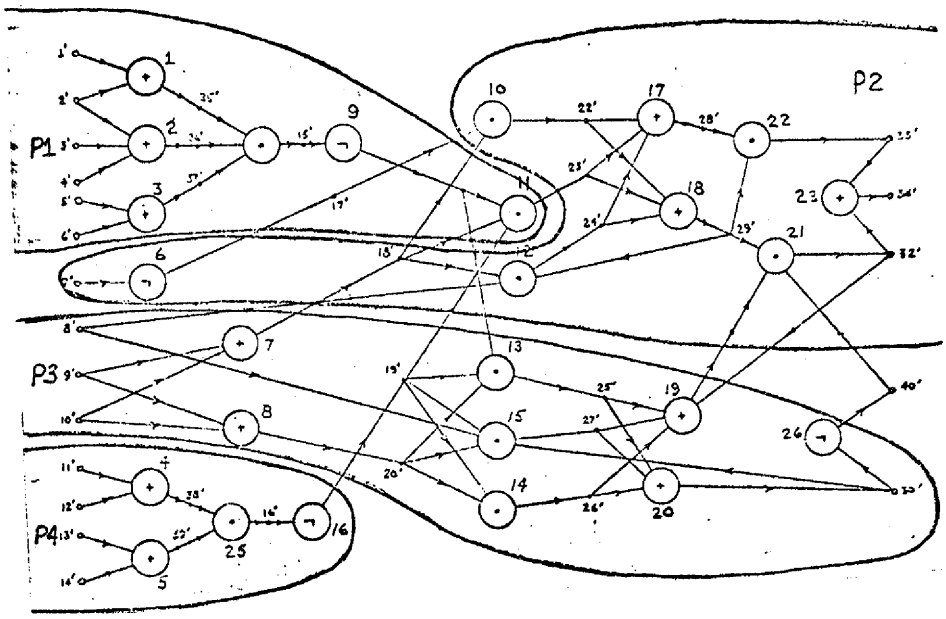
If the amount of logic to be processed is considerable, then in all probability one or more logic diagram sheets will be required to represent the entire circuit. In this case the reduced graph must be partitioned into manageable 'sheet size' groups. Figures 3.4a and 3.4b show two arbitrary partitions of a graph† for different size sheets. Each partition contains the components that fit on one logic sheet. Figure 3.4a shows a partitioning which would require 4 sheets while the partitioning of Figure 3.4b requires 2 larger sheets. For these graphs the external input signal net nodes are the signal nodes on the left of the figure while the external output and dual external signal net nodes are the signal nodes on the right.

3.6 Partition Node Graphs.

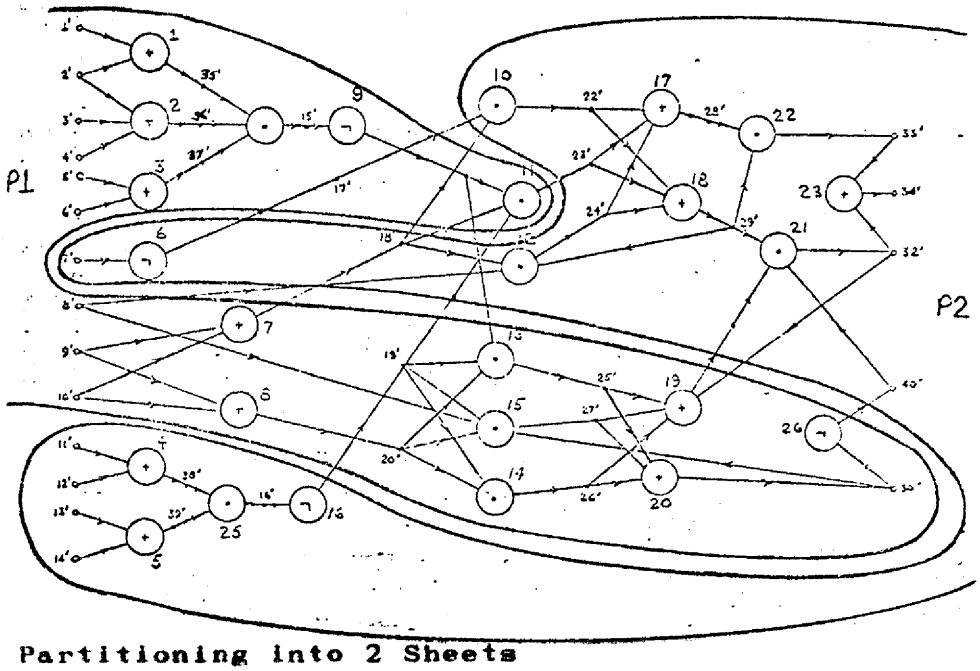
To obtain the intersheet connections, the reduced graph is transformed into a **partition node graph**. This graph is generated by collapsing all the component nodes of a partition into a single node (a partition node) and discarding those signal nets which are entirely internal to the partition. The remaining signal nets which are not entirely internal indicate the interpartition connections or the connections to the external environment. The partition node graph for the partitions of Figures 3.4a and 3.4b are shown in Figure 3.5a and 3.5b respectively. These graphs are bipartite directed

---

† This graph is a modified version of the one on page 191 of [KODR72]



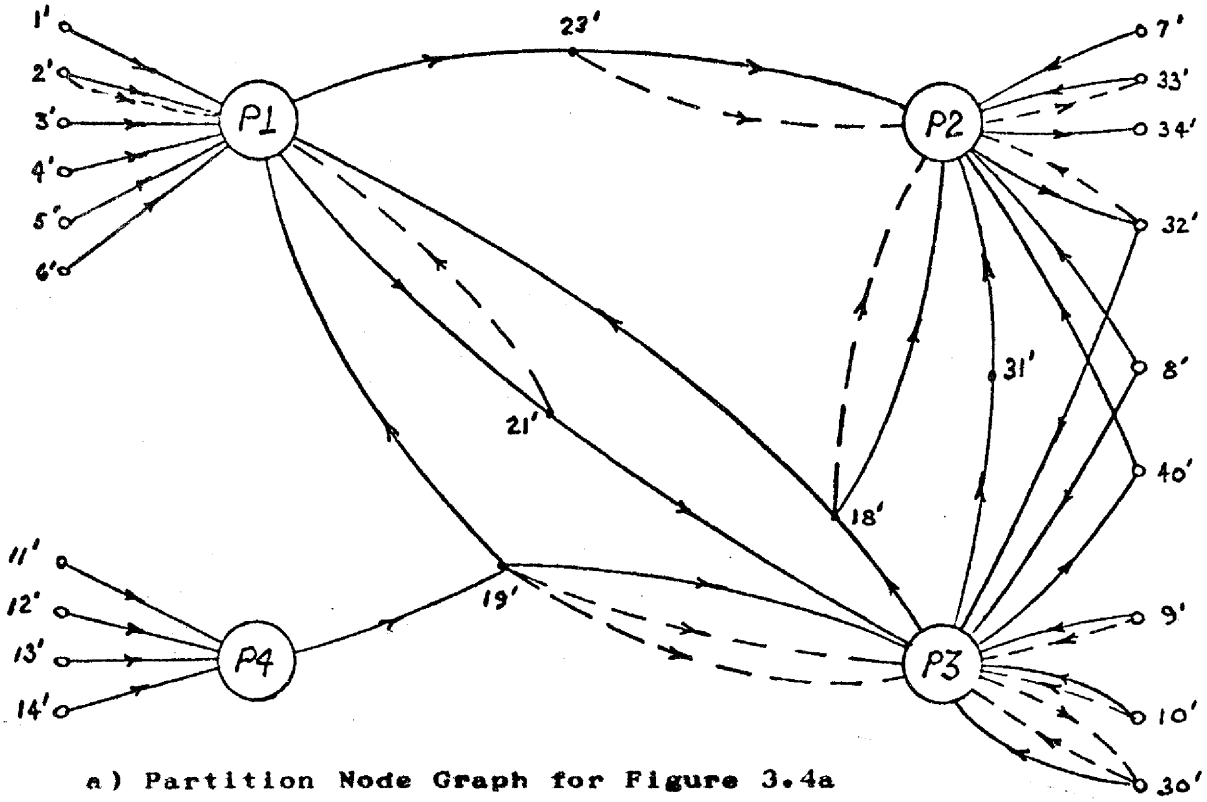
a) Partitioning into 4 Sheets



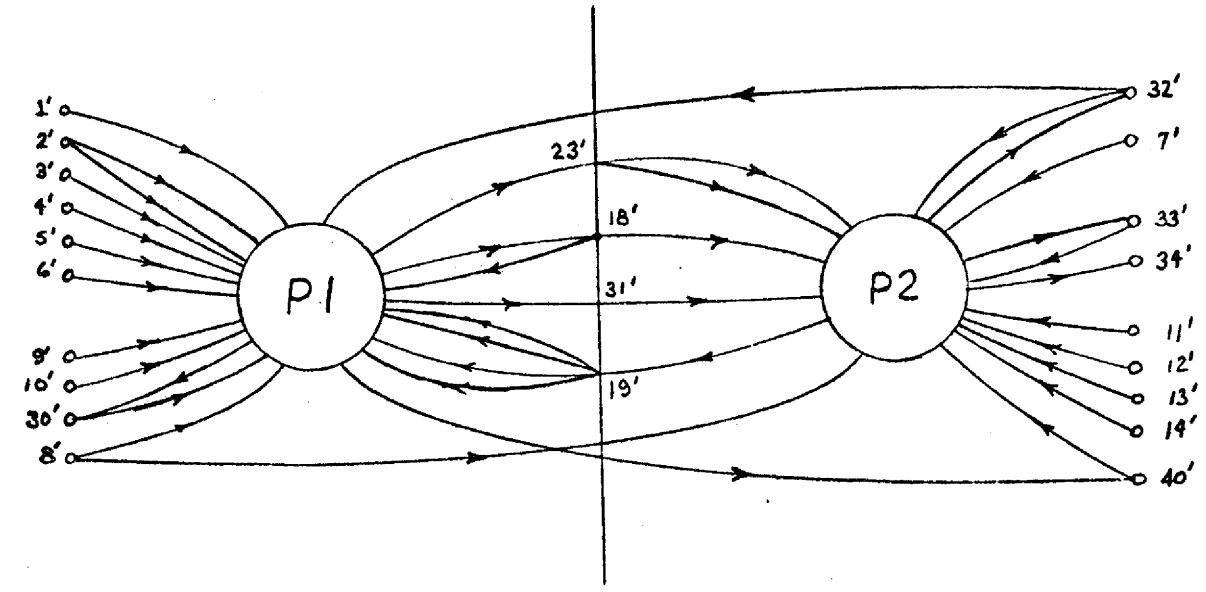
b) Partitioning into 2 Sheets

Figure 3.4 Two Possible Partitions of a Graph

graphs and maintain the original signal net numbering scheme. The signal nodes on the left and right sides of the figure are external signal nodes.



a) Partition Node Graph for Figure 3.4a



b) Partition Node Graph for Figure 3.4b

Figure 3.5 Partition Node Graphs of Figures 3.4a and 3.4b

All signal nodes of these graphs are important since they represent the intersheet and external connections. The

internal signal nodes of the partition node graph are actually nodes which will be used to indicate connections to other sheets. They are called intersheet signal nodes. On the partition node graph it is simple enough to find such nodes since all internal signal nodes are in this category. The other intersheet signal nodes are those which also connect to the environment as well as to different sheets. These intersheet signal nodes may be collected into an intersheet connection list which gives the sheet (or partitions) on which each intersheet signal node appears.

For the partition node graphs of Figures 3.5a and 3.5b the intersheet connection list would appear as given in Figures 3.6a and 3.6b.

3.7 Sheet Node Graphs and Sheet Graphs.

To represent the logic that goes on a single logic diagram sheet the individual nodes of the partition node graph may be separated to form the sheet node graphs. The signal nodes on the sheet node graph are either external signal nodes or intersheet signal nodes or combinations thereof. The information to distinguish between them may be derived from the partition node graph. Figure 3.7 contains the sheet node graph of Figure 3.5b.

In the sheet node graphs each node represents a partition and each partition contains many logic components. From this sequence it is possible to assemble together all the

Index	Signal Node	Absolute Type	Sheet List (sheet,relation)				
1	8	1	2,EI	3,EI			
2	18	6	1,I	2,I	2,I	3,O	
3	19	6	1,I	3,I	3,I	3,I	4,O
4	21	6	1,O	1,I	3,I		
5	23	6	1,O	2,I	2,I		
6	31	6	2,I	3,O			
7	32	3	2,EI	2,EO	3,EI		
8	40	3	2,EI	3,EO			

a) Intersheet Connection List for Figure 3.5a

Index	Signal Node	Absolute Type	Sheet List (sheet,relation)				
1	8	1	1,EI	2,EI			
2	18	6	1,O	1,I	2,I		
3	19	6	1,I	1,I	1,I	1,I	2,O
4	23	6	1,O	2,I	2,I		
5	31	6	1,O	2,I			
6	32	3	1,EI	2,EI	2,EO		
7	40	3	1,EO	2,EI			

b) Intersheet Connection List for Figure 3.5b

where: I - means the signal node is an intersheet input to the sheet
O - means the signal node is an intersheet output from the sheet
EI - means the signal node is an external input to the sheet
EO - means the signal node is an external output from the sheet

Figure 3.6 Intersheet Connection Lists for the Partitionings of Figure 3.5

component nodes which comprise a sheet of the logic diagram. This will give a new set of graphs called the sheet graphs which contain the logic components and interconnections for each sheet. This operation will add only component nodes and internal signal nodes to the graph.

On the partition node graph of Figure 3.5a, signal node 23 is an internal signal node of the graph and is an internal

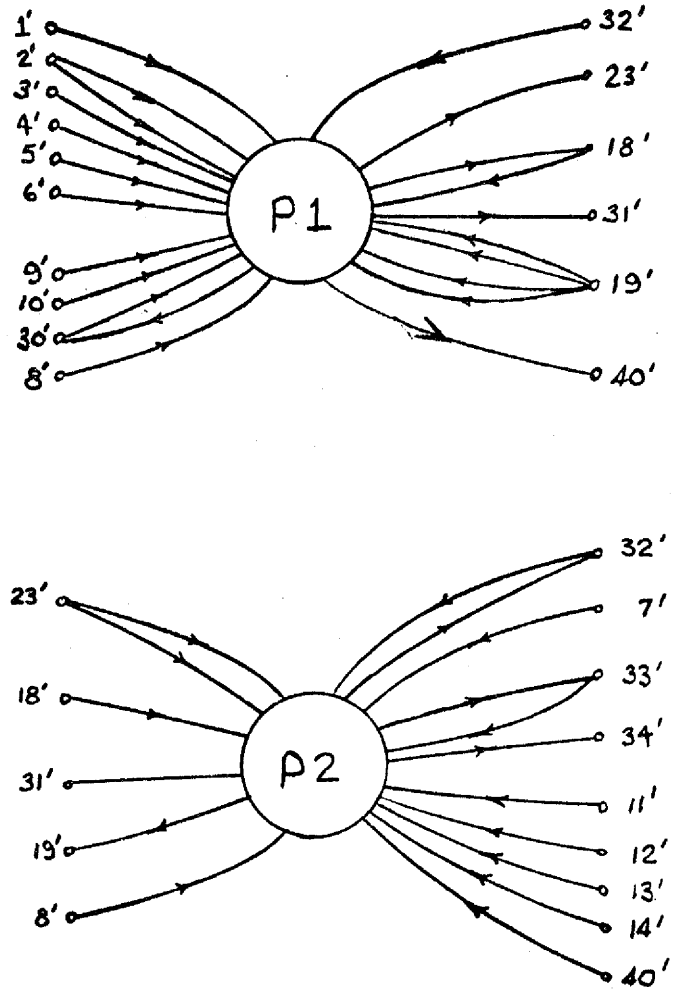


Figure 3.7 Sheet Node Graphs for Partitioning of Figure 3.5b

output of P1 and an internal input to P2. Thus it becomes an intersheet output of sheet 1 and an intersheet input to sheet 2. Signal node 21 is both an intersheet input and output of sheet 1 and thus is called a dual intersheet signal node.

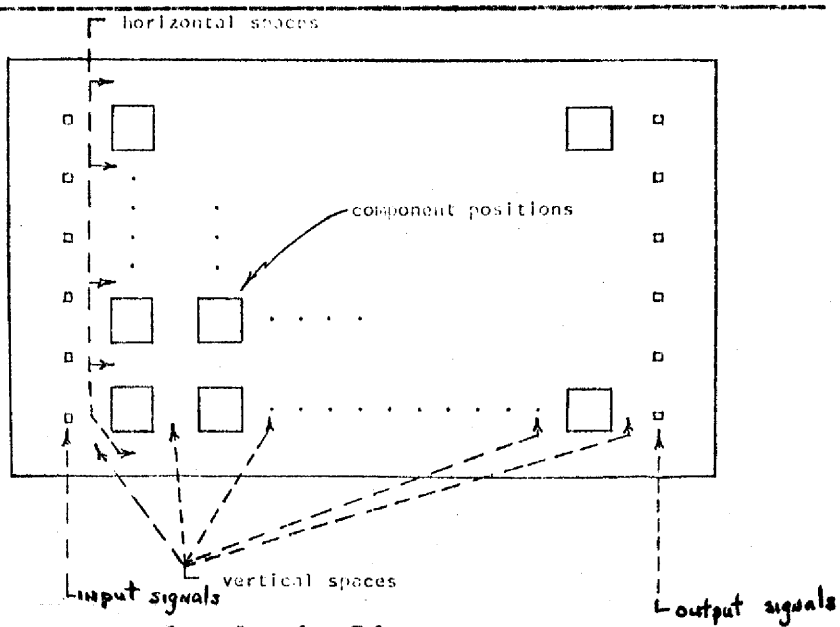
In the same figure there are signal nodes which provide signals to and receive signals from the external environment and also are intersheet signal nodes. Signal node 8 is an example of an external input signal node which is input to two different sheets, 2 and 3. Signal node 40 is an output to the environment and also input to another sheet and signal node 32

is an output to the environment from sheet 2 and is an input to sheets 2 and 3. These latter nodes are classified as external intersheet nodes.

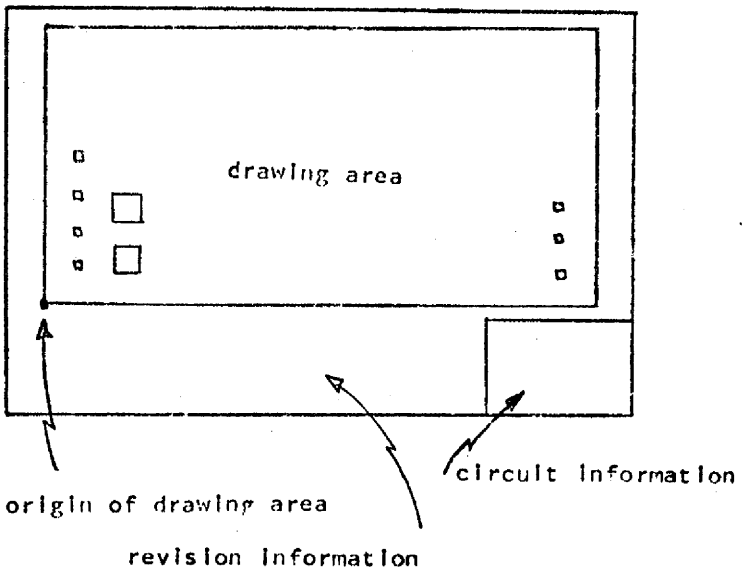
3.8 The Logic Diagram.

The drawing area of the diagram is divided as shown in Figure 3.8a. The components of a logic diagram are arranged on a grid. The inputs to a logic diagram sheet are on the left, the outputs on the right. Similarily, the components are restricted to have an orientation such that their inputs and outputs are on the left and right respectively. Figure 3.8b shows the relationship of the drawing area to the logic diagram sheet and shows the regular placement of the circuit and revision information.

In Figure 3.8 the component positions are indicated by boxes within which the MIL-spec symbols may be drawn. These boxes or zones define the maximum space needed by the particular symbol that is to be placed inside it. The set of basic symbols is defined and placed in a library. The library contains the geometric information, input and output pin specifications, alphanumeric symbols and tag field positions. The alphanumeric symbol fields are for static identification information while the tag fields are for insertion of variable identification information. By adhering to the prescribed format a user can define his own symbols and add them to the library. Figure 3.9 describes the format of the library entries and Figure 3.10 shows a MIL-spec symbol and a library



a) Drawing Area of a Logic Diagram



b) Drawing Area and the Logic Diagram

Figure 3.8 Layout of the Logic Diagram

description of that symbol. A similar definition scheme has been used in the Sperry Rand system [SAND72].

For different size diagrams, the drawing area differs in size as does the number of component positions (size of the grid). The size in which the components are to be drawn i.e.

```
TYPE name
PARM ID #S #T #L #C #IP #OP
ZONE xsize ysize
PINS 1 name1 I y1
     2 name2 I y2
     3 name3 O y3
     .   .   . .
     .   .   . .
SYM  xs ys size 'characters'
TAG  xt yt size
LINE x1 y1 x2 y2
CIRC xc yc radius s-angle e-angle
ENDS
```

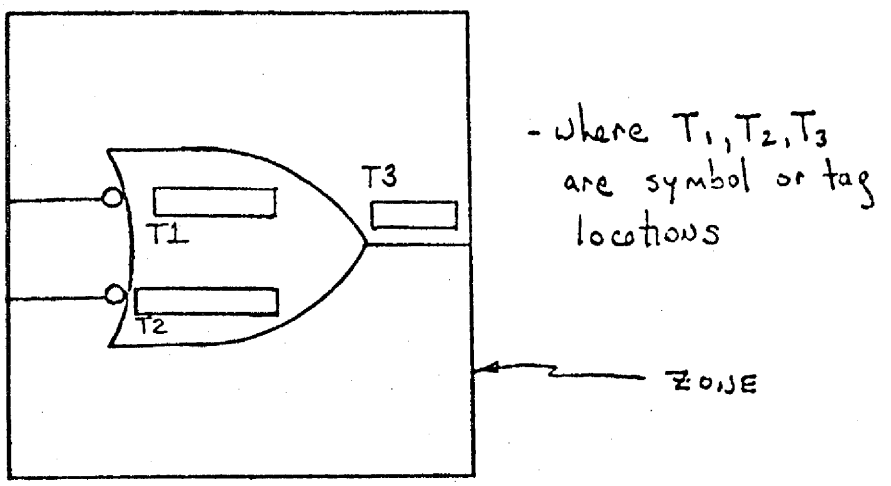
where

name	name of circuit
xsize,ysize	size of rectangular zone
ID	0 if input pins identical
	1 if input pins unique
#S	# of symbol entries
#T	# of tag entries
#L	# of line entries
#C	# of circle entries
#IP	# of input pins
#OP	# of output pins
name1,...,name3	names of pins
I	pin is input
O	pin is output
y1,y2,y3	y coordinates of pins
xs,ys,xt,yt	location of symbol or tag field
size	size of symbol or tag characters
x1,y2,x2,y2	coordinates of line end points
xc,yc	coordinates of centre of circle
radius	radius of circle
s-angle	start angle of circle
e-angle	end angle of circle

Figure 3.9 Symbol Library Entry Prototype

1/2, 3/4, or full size, will also affect the number of component positions.

Between each horizontal and vertical component row and column is space for component interconnection lines. These are called the horizontal and vertical spaces (Hashimoto and Stevens terminology for PC boards [HASH71]).



a) Geometric Drawing of Symbol

```
TYPE NAND2
PARM      0    0    3    5    5    2    1
ZONE    200  200
PINS      1I1       1  -15
          2I2       1   15
          30        2    0
TAG1    -25   15   15
TAG2    -40  -30   15
TAG3     50   10   10
LINE    -20  -40  -63  -40
LINE    -20   40  -63   40
LINE    -69  -15   -1  -15
LINE    -69   15   -1   15
LINE     50    0    1    0
CIRC   -130    0   80  330   30
CIRC    -20  -40   80   30   90
CIRC    -20   40   80  270  330
CIRC    -61  -15    8    0    0
CIRC    -61   15    8    0    0
ENDS
```

b) Library Description (units in 1/100th of an inch)

Figure 3.10 Sample Symbol and Its Library Description

The positions of the component rows and columns and horizontal and vertical spaces are not fixed but vary according to the number, type, and scale of components. Within the MIL-specs the relative size of components vary according

to their function. The height of any one horizontal component row or the width of any one component column will reflect the size of the largest component in that row or column. Since the drawing area is fixed for a given drawing size, any excess space may be absorbed by the horizontal or vertical spaces. If for a given circuit, the maximum number of rows or columns is not used then this space is also distributed equally among the appropriate' spaces. The input and output signal nodes, called boundary signal nodes indicate the connections to an external environment or to/from other logic sheets. Their horizontal positions are fixed but their vertical positions may vary to correspond with the components to which they are connected.

4.0 Introduction.

This chapter is concerned with the placement of components and signal names, and with the assignment of interconnections to pins on the component symbols. Component placement is performed by a constructive-initial placement step for which a new algorithm is given and by a placement refinement step for which new objective functions are presented. Automatic placement of boundary signal nodes has not been discussed before in the literature. Methods of placing boundary signal nodes, in conjunction with a new method of pin assignment, are given.

4.1 Placement.

The placement operation consists of two steps, a constructive-initial placement step followed by a placement refinement step. The constructive-initial placement step uses standard, single-seed partitioning techniques to select components on the basis of connectivity and the initial placement operation tries to place each component on the current sheet. If a component is unplaceable on the current sheet the placement operation rejects it and requests another component. The placement refinement step perturbs this initial placement in an attempt to reduce the objective functions. For each of these placement steps the objective functions vary

from those used in PC board design as described in the following sections.

4.1.1 Initial Placement.

When a new component is to be positioned, there is usually a set of empty grid positions into which it could be placed. The choice of the empty grid position is made according to some cost criterion. In PC board layout this cost is usually the interconnection wire length.

For logic diagrams, reduction of interconnection wire length is not a meaningful objective by itself. A placement is required such that the logic flows from left to right and reversed connections are minimized. Through the use of graph theory techniques, it is possible to eliminate all reversed connections except for a minimum number of feedback connections. To do this would require the use of a minimum feedback set algorithm. Such algorithms, however, are known to be very costly in terms of machine time. Furthermore, a minimum feedback algorithm requires the whole circuit (graph) for a diagram to be known at the time the algorithm is applied. This is not possible when using a combined partitioning and constructive placement approach.

A simple constructive method of placement to achieve the new logic diagram placement objectives is to position a component to the right of the components which are its

ancestors† and to the left of the components which are its descendants‡. Thus for any component with only ancestors or descendants, the rightmost ancestor or the leftmost descendant becomes what will be called a preferred neighbour. Ideally, the component is placed adjacent to its preferred neighbour. If this is not possible, then the component must be placed in a different row. In the case when a component has both ancestors and descendants, it is not always possible to eliminate reversed edges. In this case the component is be placed with respect to both its ancestors and descendants such that the number of reversed connections that it causes is minimal. Thus if the components are chosen in a suitable manner, reversed edges will be reduced and the flow of the logic will be maintained.

The diagram of Figure 4.1 illustrates the placement of components according to their ancestors and descendants. The numbers within the components represent the order in which they were selected and placed. The placement of components 1, 2 and 3 is straightforward. In each case the component is placed to the right of its ancestor.

Component 4 has two ancestors when it is placed thus a decision is made to place it to the right of the rightmost ancestor. Similarly, component 5 has two descendants but no ancestors yet, thus it is placed to the left of its leftmost descendant. Since the grid position in the same row to the left of component 2 is occupied, component 5 must be placed in

---

† An ancestor of a component 'C' is any component which is the source of a net that is input to 'C'.

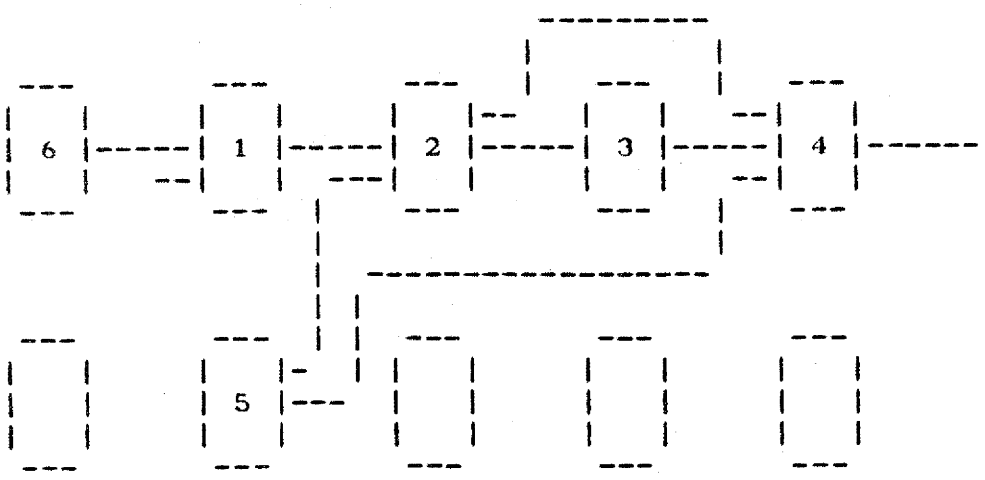


Figure 4.1 Horizontal Ordering of Components

another row. Component 6 has only one descendant thus it is placed to the left of that descendant. The use of such an ancestor-descendant technique gives an ordering to the components in the horizontal direction. In this simple example it was possible to place the components without any reversed edges.

When a component is to be placed in an adjacent row, the chance of introducing crossovers may be reduced by using a biasing scheme to influence the choice of rows. If the input pins of components 2 and 4 of Figure 4.1 are identical, then component 5 could be placed above or below the preferred row. If the input pins are not identical then the location of the pin may be used to bias the placement of component 4. The biasing of the components in such a manner gives a vertical ordering to the components.

The vertical placement is influenced to a greater degree, however, if the candidate component has other neighbours which are already placed on the sheet in different rows. Component 8 in Figure 4.2 illustrates this case. It is to be placed to the left of its preferred neighbour, component 2, but it is biased above the preferred row by the location of

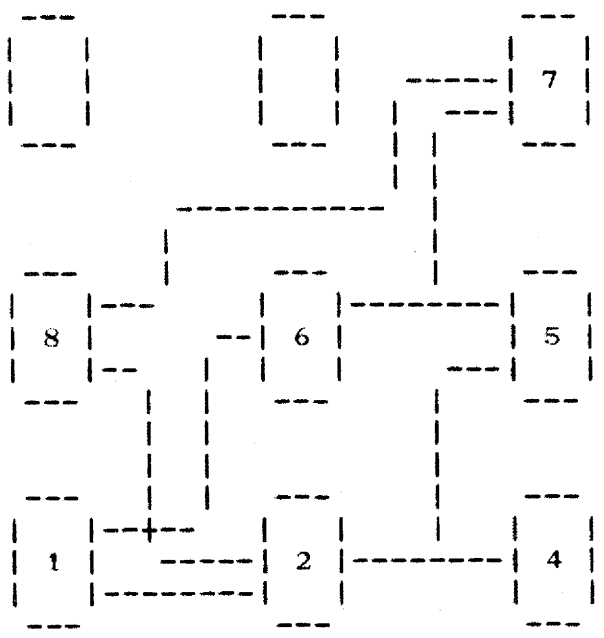


Figure 4.2 Vertical Biasing of Placement by Neighbours

component 7.

If a candidate component has many neighbours which are already placed, then determination of this bias becomes more difficult. The bias is obtained by using a centre of gravity or force placement techniques [HANA72].

When positioning a candidate component, it is often found that the desired grid position is already filled. In this case it is necessary to begin a search for an empty grid

position in that neighbourhood. If this search is unsuccessful there are two alternatives:

1) calculate a new desired grid position on the basis of the second best neighbour, and/or

2) choose another candidate component.

If it is impossible to find a grid position in which to place the chosen component then that component is abandoned and another component is chosen. The components which are abandoned will subsequently be placed on other sheets. Figures 4.3a and 4.3b give a flowchart for the initial placement of components. The switches SWX and SWY start the search in the appropriate quadrant depending upon whether the preferred neighbour is an ancestor or descendant and whether the candidate component is biased above or below the preferred row respectively.

4.1.2 Placement Refinement.

The new objective functions which are appropriate for placement refinement on logic diagrams are:

1. number of adjacent components,

2. length and number of reversed edges, and

3. interconnection length.

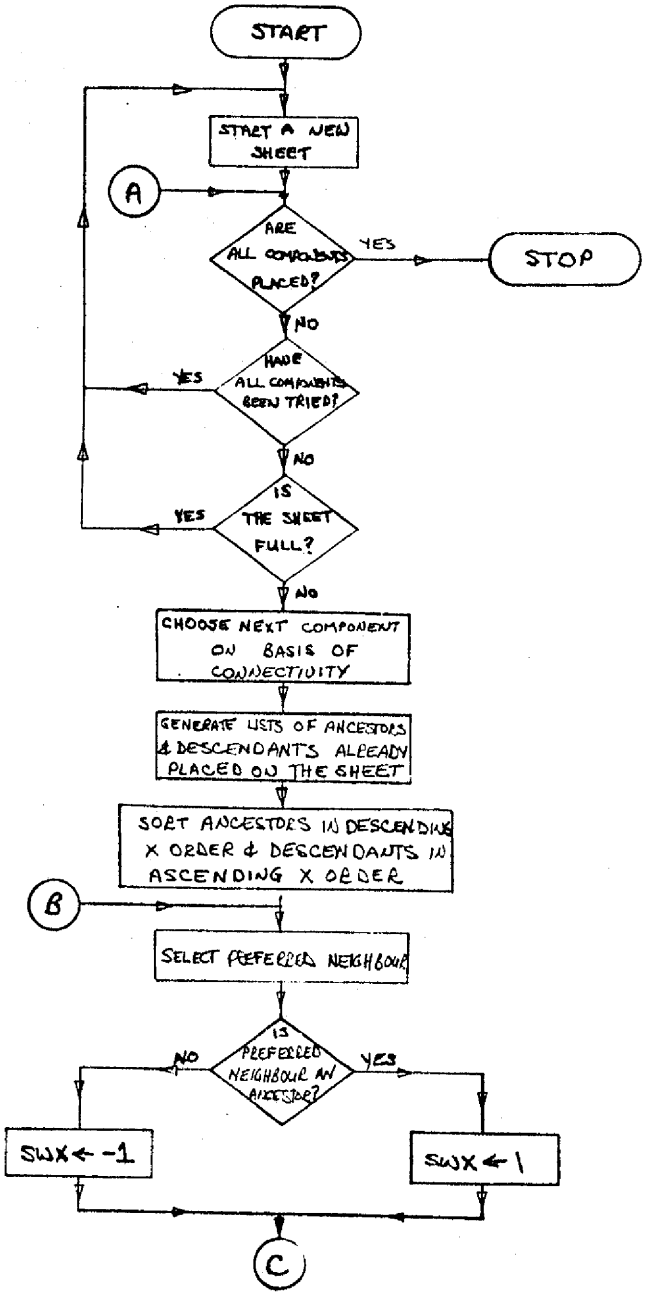


Figure 4.3a Flowchart for Initial-Placement of Components

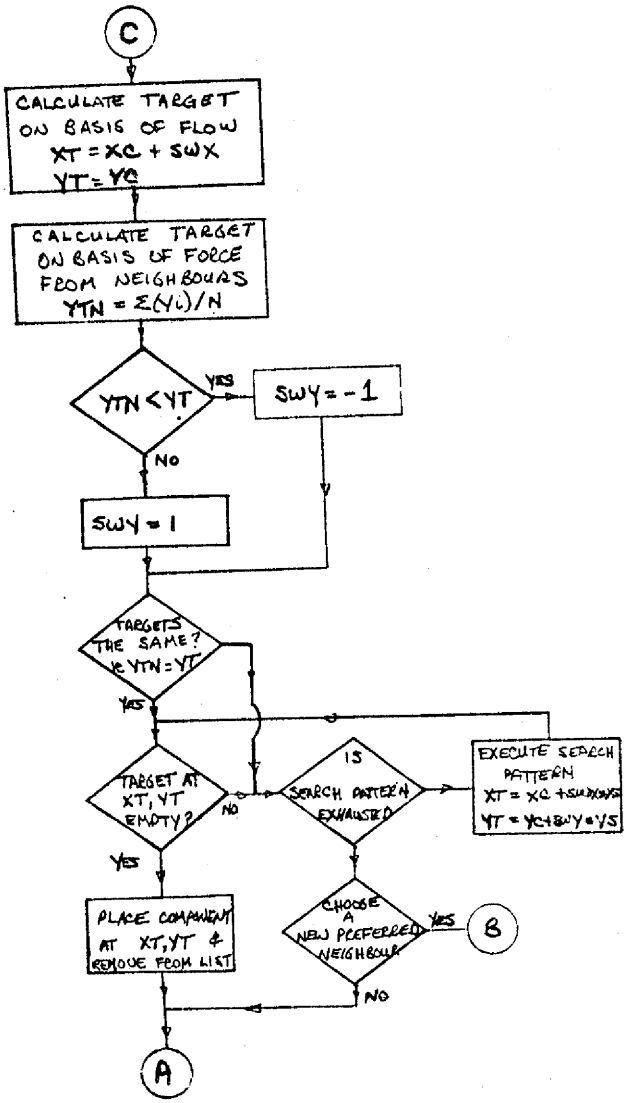


Figure 4.3b Flowchart for Initial-Placement of Components

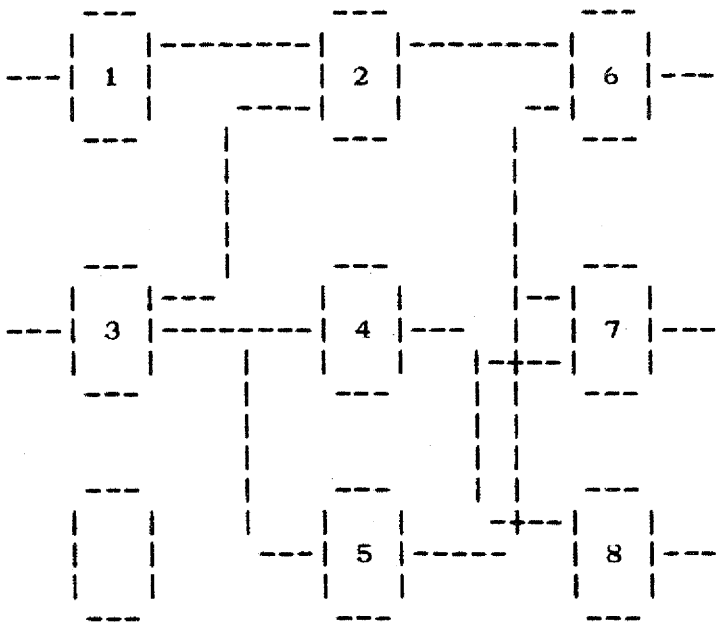
The adjacency and reversed objectives are a measure of the flow of the logic. The interconnection length is a measure of the "closeness" of the components. The number of adjacent components should be maximized and the number of reversed edges should be minimized. The interconnection length is minimized after the other two objectives are considered. The refinement process is broken into two steps, the vertical and horizontal refinements, which are performed separately.

In the vertical refinement step, components are moved (interchanged) in the vertical direction only. Since reversed edges may not be reduced by a vertical movement of components, the objective functions for vertical refinement are only adjacency and interconnection length. As with force placement techniques, the vertical forces exerted on each component by its neighbours are calculated and placed in a list. The component C with the highest force is chosen first for refinement. The sign of the force gives the direction of the refinement. Component C is trial interchanged with its adjacent component in the direction of the force and the changes in adjacency and interconnection length are computed. If the number of adjacencies is increased or if the number of adjacencies remains the same and the interconnection length is reduced, the trial interchange is confirmed. If component C has not reached its target location, then C is trial interchanged with its new adjacent component in the same direction. This process continues until the component reaches the target position. The vertical forces are now recomputed

and the component with the largest force is chosen again. If at any stage a component is not interchanged the next component on the list is tried. The vertical refinement ends when no trial interchange results in a move. A simple example is given in Figure 4.4a. The forces on the components of Figure 4.4a are listed in the table of Figure 4.4b along with the length and the number of adjacent components. Assume that the components have been placed as shown; the numbers give the order of placement.

The y force on each component is determined by performing a vector sum of the lengths of its connections to its ancestors and descendants only. For instance component 3 has three descendants 2, 4 and 5 which contribute +1, 0, and -1 respectively to the force sum. The lengths are calculated by summing the magnitude of the forces. The adjacency value for a component is determined by the number of components which are ancestors or descendants and are in adjacent columns and the same row or are in the same row and non-adjacent columns where the intermediate columns in the same row are empty. The maximum adjacency per component is 2. The list of Figure 4.4c results from sorting the components on the magnitude of their forces.

Since component 5 has the largest force it is selected for refinement first. The direction of the trial-interchange is in the positive y direction as determined by the sign of the force (+4). When component 5 and 4 are trial-interchanged and tested, the length of the connections is decreased by 4



a) Layout After Initial Placement

Component	Force (Y)	Length (Y)	Adjacents
1	0	0	1
2	-1	1	2
3	0	2	1
4	-1	1	2
5	+4	4	0
6	-2	2	1
7	-1	1	1
8	+1	1	0
TOTAL	12		8

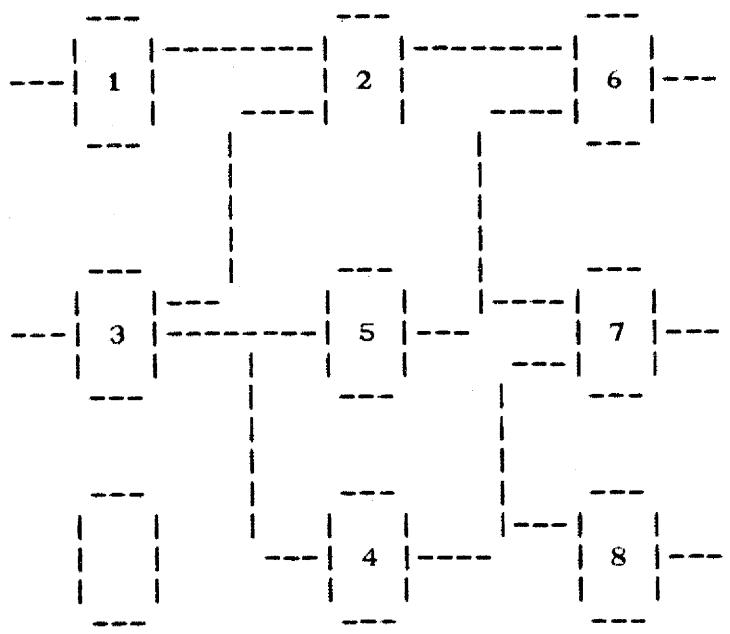
b) Forces and Number of Adjacent Components

Component	Force (Y)	Length (Y)	Adjacents
5	+4	4	0
6	-2	2	1
2	-1	1	2
4	-1	1	2
7	-1	1	1
8	+1	1	0
1	0	0	1
3	0	2	1

c) Sorted Forces and Number of Adjacent Components

Figure 4.4 Before Vertical Refinement of Components

and the number of adjacent components increases by 1. Thus the interchange is kept. The diagram of Figure 4.5 illustrates this change and a new list is calculated and sorted. For this new list an unsuccessful trial-interchange is tried on component 4 in the positive y-direction. Each of the other components is also tried in turn but no further interchanges



a) Layout After One Interchange

Component	Force (Y)	Length (Y)	Adjacents
1	0	0	1
2	-1	1	2
3	0	2	1
4	+2	2	1
5	+1	1	2
6	-1	1	1
7	-1	1	1
8	0	0	1
TOTAL		8	10

b) Forces and Number of Adjacent Components

Component	Force (Y)	Length (Y)	Adjacents
4	+2	2	1
2	-1	1	2
5	+1	1	2
6	-1	1	1
7	-1	1	1
1	0	0	1
3	0	2	1
8	0	0	1

c) Sorted Forces and Number of Adjacent Components

Figure 4.5 After One Interchange in Vertical Refinement

are made. Thus the placement of Figure 4.5a is the final placement in this step.

The next step to be performed is the horizontal refinement. Because of the initial-placement method described above the logic circuit will have a flow from left to right. Thus an interchange of component pairs in this x-direction

will seldom result in any improvement of the objective functions. An improvement in this direction could result however by moving components into empty adjacent grid positions. This situation is depicted in Figure 4.6. Again the numbers in the boxes give the order of the placement. No interchange of components in the x-direction will reduce the

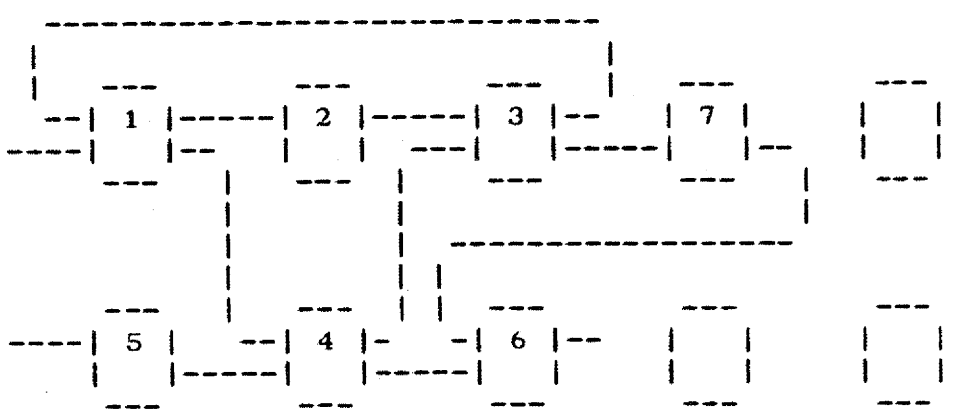


Figure 4.6 Horizontal Refinement of Components

number of reversed edges.

However component 6 has been placed to the right of 4 since it was its only neighbour at the time of placement. After component 7 is placed, an unnecessary reversed edge exists. It may be removed by moving component 6 two columns to the right. In this case the objective function for movement is the length of the reversed edges. A trial-movement of component 6 to the first empty space to the right reduces the length of reversed edges and a second trial-movement reduces the reversed length even further and removes the reversed edge.

4.2 Placement of Boundary Signal Nodes and Pin Assignment

The positioning of the boundary signal nodes may be accomplished in two different manners.

1) Arrange the boundary signal nodes such that the inputs on one sheet are aligned physically to match corresponding outputs on the other sheets.

2) Arrange the boundary signal nodes according to the position of their source and sink component(s) on each logic diagram.

The first technique is an attempt to improve intersheet references but does so at the expense of the current logic diagram. Often boundary signal nodes will be placed far from their source or sink resulting in increased intersections. Furthermore, if a source is input to more than one other logic sheet and the inputs are not at the same level on the other logic diagrams, then to maintain the alignment the output signal node must be duplicated. This also increases intersections as may be seen from the output in the paper by Sanderson [SAND72]. The majority of these intersections occur in the first and last vertical spaces and somewhat negate the improvement in intersheet reference.

The second method will result in better individual logic diagrams. There will be less congestion in the first and last vertical spaces since the boundary signal nodes will be near to their sources and sinks and there will be only one boundary

output signal node per source on the logic diagram. Since the boundary signal nodes will indicate the other logic sheets to which they connect, the intersheet referencing is not greatly hindered. For these reasons the second method was chosen.

The placement of the boundary signal nodes affects the assignment of connections to pins of a component†, and the routing of the connections. Conversely the pin assignment and routing may also affect the placement of the boundary signal nodes. Thus boundary signal node placement, pin assignment, and routing are performed in the following manner.

1) Preliminary boundary signal node placement

2) Pin assignment

3) Intermediate boundary signal node placement

4) Routing (chapter 6)

5) Final boundary signal node placement

The placement of boundary input signal nodes is more difficult than placement of boundary output signal nodes. Boundary input signal nodes may have one or more sinks (destinations) whereas boundary output signal nodes may have only one source‡. Signal nodes which are dual boundary signal nodes are handled as boundary output signal nodes.

---

† If a component has input pins which are logically equivalent then crossovers may be reduced if the connections may be assigned to these pins in any order.

‡ By convention, wired ORs must be indicated by a wired OR component (MIL-STD-806B).

4.2.1 Boundary Input Signal Nodes.

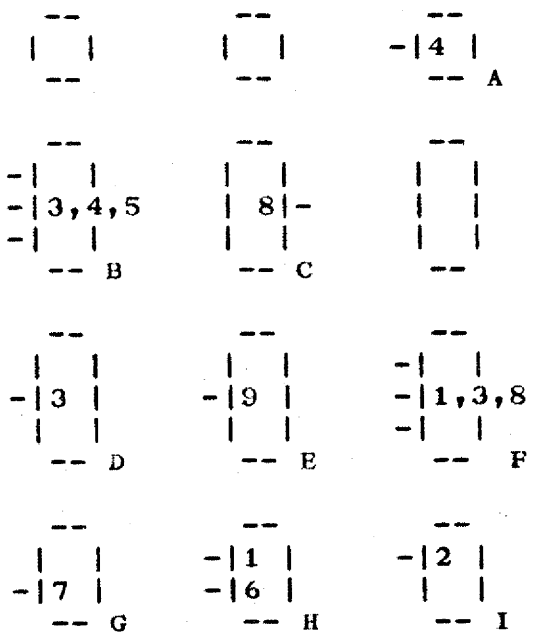
For preliminary placement, the various cases of a boundary input signal node with single and multiple sinks are listed below.

Condition	Preliminary Assignment
a) input to one sink	
1) sink in 1st column	- assign opposite centre of component
2) sink not in 1st column	- assign in centre of space above or below component row
b) input to more than one sink	
1) one sink in 1st column, others are not	- assign opposite component in 1st column, bias above or below centre according to other components
2) one or more components in 1st column	- assign opposite highest component in 1st column, bias as above
3) no components in 1st column	- assign centre of space above or below component nearest 1st column

In the above cases, the boundary input signal nodes were assigned their preliminary positions on the basis of a 'preferred sink'. If a boundary input signal node is input to only one component then that component becomes the preferred sink. For the cases where there is more than one such component the preferred sink must be chosen. A simple technique for determining the preferred sink involves a sort of the sinks of each boundary input signal node. If the grid positions of the sinks are sorted in ascending order on the column number and in descending order on the row number, the preferred sink will be at the top of the list for each signal

node. This technique is illustrated in Figure 4.7. Figure 4.7a shows a section of a logic circuit in which the connections to be made are indicated by number pairs. Components B and F have non-unique pins to which the connections 3,4,5 and 1,3,8 respectively must be assigned. The pins of component H are unique. The sources of all connections are boundary input signal nodes except for connection 8 which is an internal connection. Figure 4.7b gives tables containing the column number DA and the row number DB of the sinks (destinations) for all boundary input signal nodes. The left table contains the sink positions for each signal node in a random order. The right table shows the sink positions after the sort. For those connections with more than one sink the right table also contains the offset of all sinks from the first one and indicates the bias.

If a signal node has only one sink, it is placed opposite the centre of that sink or in the centre of the space above or below the component row. Signal nodes with more than one sink are placed opposite the centre of, or in the space above or below the sink in the leftmost column and are biased from their central position by the remaining sinks. This biasing of the signal nodes influences the pin assignment and thus helps to reduce intersections. Figure 4.8a shows the result of the preliminary assignment. In this case the choice was made to place boundary signal nodes 2 and 6 in the space above their destination row and to place boundary signal node



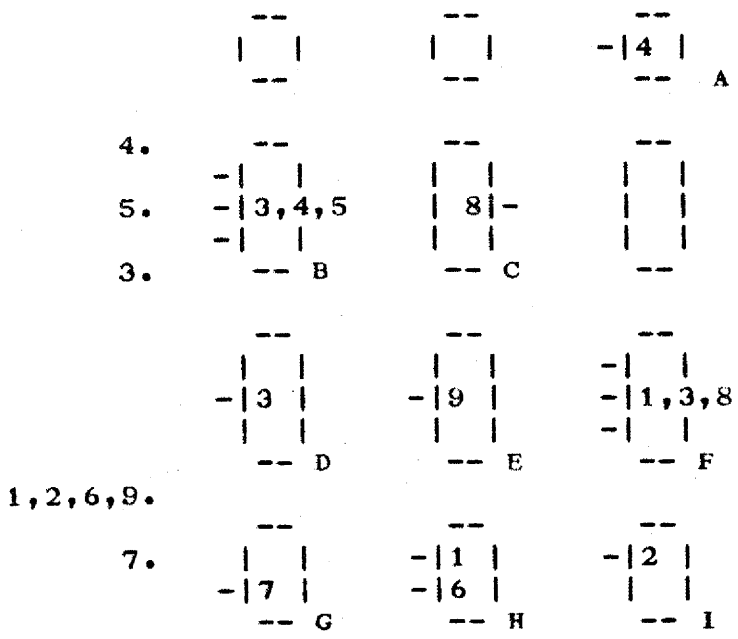
- sources for connections 1,2,3,4,5,6,7,9 are boundary input signal nodes, connection 8 is an internal connection

a) Before Preliminary Assignment

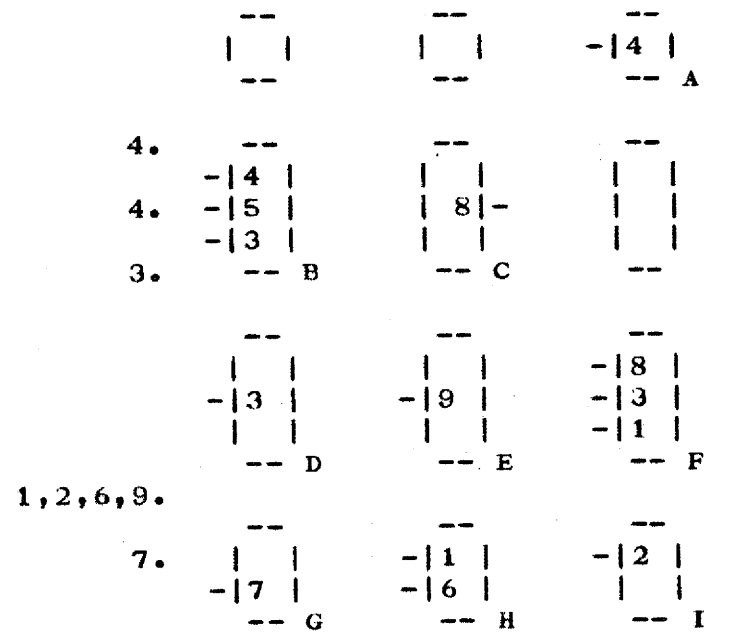
Signal Node	Sink #	col	row	Signal Node	Sink #	col	row	DELTA	
1	F	3	2	1	H	2	1		
1	H	2	1	1	F	3	2		
2	I	3	1	2	I	3	1		
3	F	3	2	3	B	1	3	0	
3	D	1	2	3	D	1	2	-1	bias below
3	B	1	3	3	F	3	2	-1	centre
4	A	3	4	4	B	1	3	0	
4	B	1	3	4	A	3	4	+1	bias above
5	B	1	3	5	B	1	3		
6	H	2	1	6	H	2	1		
7	G	1	1	7	G	1	1		
9	E	2	2	9	E	2	2		

b) Destination Information (before and after sorting)

Figure 4.7 Boundary Input Signal Node Placement Example



a) Preliminary Placement



b) After Pin Assignment

Figure 4.8 Preliminary Placement of Boundary Input Signal Nodes

9 in the space below its destination row.

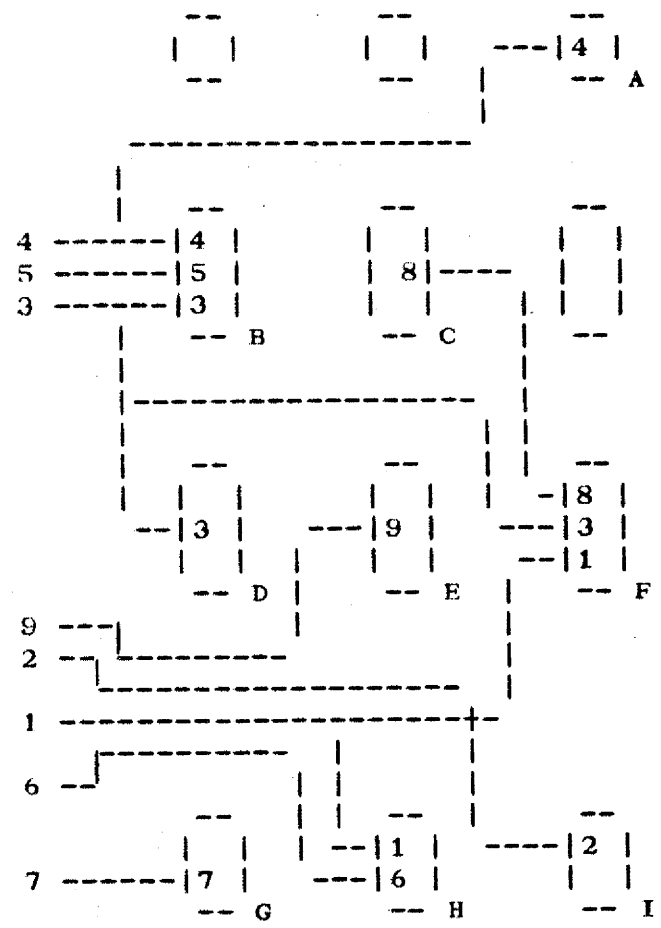
For the current example, pin assignment is only necessary for the component B which has inputs 3,4,5 and component F with inputs 1,3,8. The pin assignemnt is straightforward and is shown in Figure 4.8b. (Pin assignment is discussed in detail in section 4.2.3.)

The intermediate placement for boundary input signal nodes which are input to components in the first column is straightforward. They are placed directly opposite the pin. This in fact becomes their final placement. The remaining boundary input signal nodes at present are still assigned positions in the centre of the horizontal spaces. If a horizontal space contains more than one boundary input signal node then, the signal nodes must be given a vertical separation so routing may be performed. This may be accomplished using another sorting scheme considering each horizontal space separately. In this case the three columns, DB, DA, and DY are used. The left table of Figure 4.9a shows the boundary input signal nodes which are in the centre of a horizontal space.

When a boundary signal node is input to more than one component only the component which determined the preliminary position is used. Since boundary signal nodes may be placed in the centre of horizontal spaces above or below the component row, the sorting operation becomes more complicated. It is necessary for sort to be made in ascending order on DB, in ascending order on DA if the destination row is below the

Connection	DB	DA	DY	Connection	DB	DA	DY
1	1	2	y1	6	1	2	y6
2	1	3	y2	1	1	2	y1
6	1	2	y6	2	1	3	y2
9	2	-2	y9	9	2	-2	y9

a) Destination Information (before and after sorting)



b) After Intermediate Placement

Figure 4.9 Intermediate Placement of Boundary Input Signal Nodes

horizontal space and in descending order if the destination row is above the horizontal space, and then in ascending order on DY. This sort may be simplified if the destination DA of a boundary signal node is negated if the destination row is

above the horizontal space. Then the sort on DB, DA, and DY reduces to an ascending, ascending and ascending sort respectively. The right table of Figure 4.9a shows the result after this sort. This ordering of the boundary signal nodes gives their relative positions on the diagram. Figure 4.9b shows the intermediate placement for the boundary input signal nodes. Also given in the diagram is a possible routing of the connections.

There is no way to predict where the routing will place the horizontal connections to those boundary input signal nodes which were assigned positions in a horizontal space since there are internal connections which also use the same space. Thus placement of these signal nodes is not final.

The final assignment is done after the routing is performed. It operates on the boundary input signal nodes which are not input to a component in the first column. The operation involves shifting the signal nodes until they are aligned with the horizontal part of the connection. The final placement is shown is Figure 4.10.

4.2.2 Boundary Output Signal Nodes.

For preliminary placement, the boundary output signal nodes are simpler to handle since they have only one source (source pins are unique). There are only two cases.

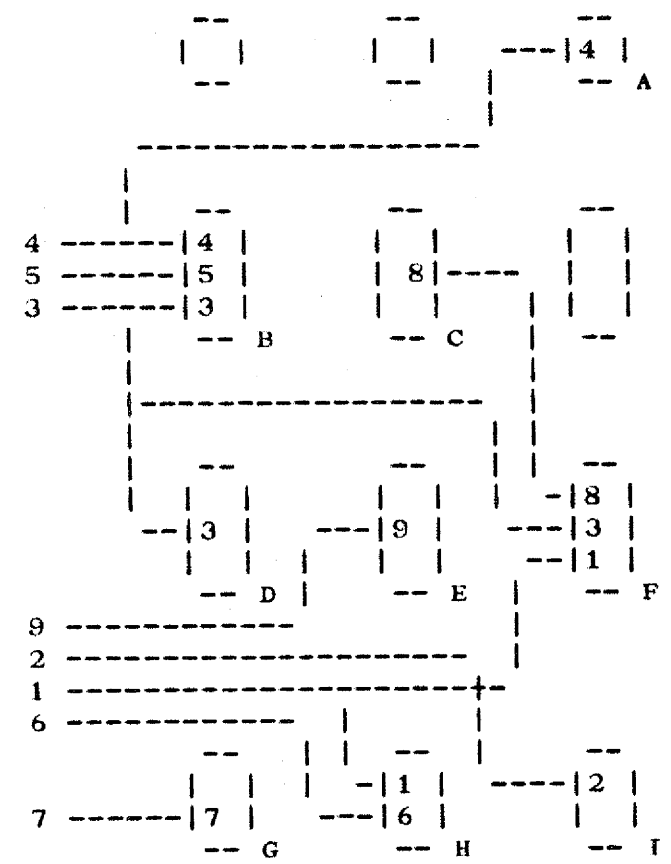


Figure 4.10 Final Placement of Boundary Input Signal Nodes

1) Source is in last column - assign the signal node opposite the source pin. This position is the final position of the signal node.

2) Source is not in last column - assign a preliminary position in the centre of the space above or below the source row.

The intermediate assignment is similar to that of the assignment for boundary input signal nodes. In this case the source positions SB, SA, and SY are collected. Using the intermediate placement technique introduced in section 4.2.1,

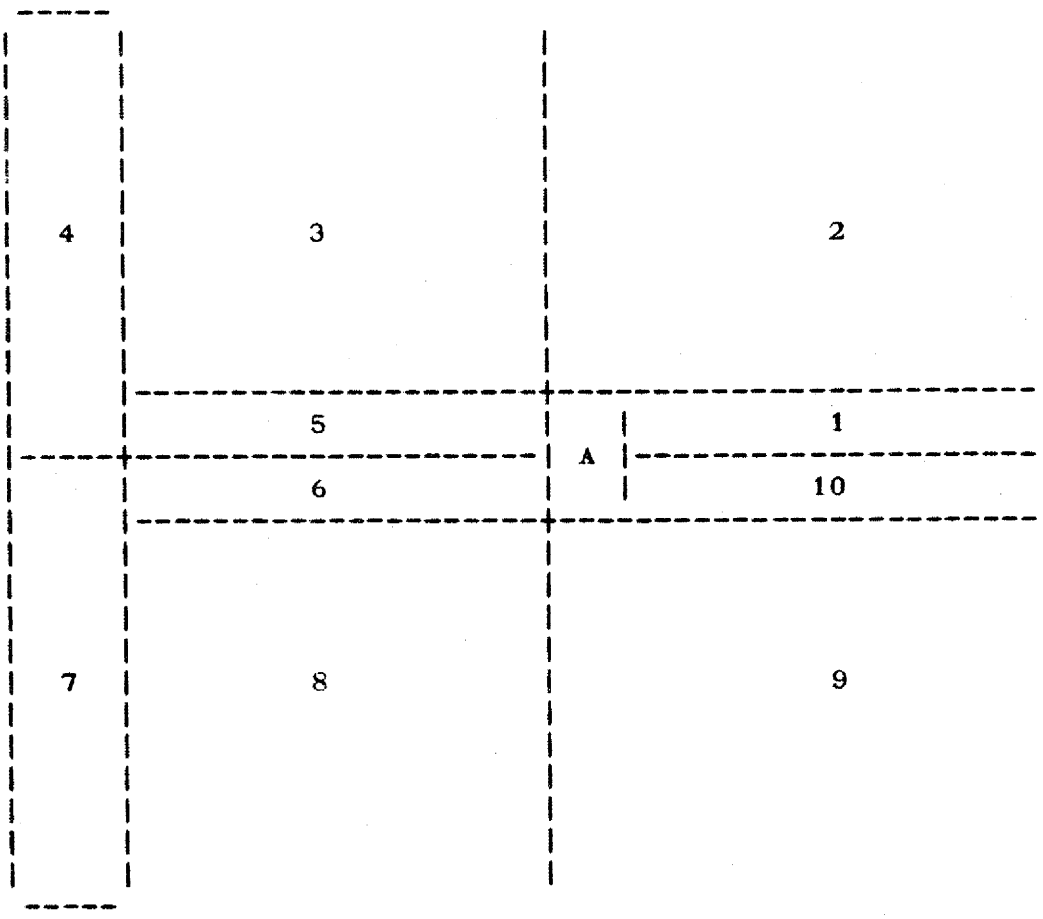
the source position SA is negated if the source row is above the horizontal space. Then the columns are sorted in ascending, descending and ascending order respectively. This will give their order on the diagram such that routing may be done.

Those boundary signal nodes whose sources are in the last column are already in their final postions. The remaining signal nodes must be shifted to align them up with the horizontal part of the connections as was done for the boundary input signal nodes.

4.2.3 Pin Assignment.

The assignment of connections to pins is necessary only for those components in which there is more than one input pin which is identical in function. A proper assignment of the connections to these pins will reduce crossings on the logic diagram. Achieving a proper assignment requires information concerning the source positions of the individual connections and the spaces in which they will be routed. Since the routing scheme of chapter 6 assigns connections to spaces on the basis of their sources and sinks, this information may be used in determining the pin assignment. The technique to be described classifies the sources of the connections to an individual component and then performs a multi-way sort.

For each component requiring pin assignment, the sources of the connections input to it may be classified into one of the regions shown in Figure 4.11. The tables of Figure 4.12 give the method of classifying the connection sources into the regions. Once all the sources for a component have been assigned to regions, the sources are sorted in ascending order on their region number and then a multi-way sort is performed in accordance with the table of Figure 4.13. This multi-sort is performed on the source row, source column and then on the Y position of the source pin. When sorting is complete the connections corresponding to the sources are assigned in reverse order, ie. top connection on the list is assigned to the highest numbered pin. Pins are numbered from bottom to top. Figure 4.14 and 4.15 give an example of this pin assignment. Figure 5 where 8a shows the example before pin assignment where the component labeled A has 7 identical inputs to which the sources AA, AB, AC, AD, AE, AF and AG are to be connected. The table of Figure 4.14b give the source information after it has been collected in random order and Figure 4.14c gives this information after it has been sorted into regions. Figure 4.15a gives the source information after sorting within the regions and Figure 4.15b gives the final assignment with a possible routing.



A - component for pin assignment
Regions 4 & 7 - boundary input signal nodes

Figure 4.11 Regions for Classification of Connection Sources for Pin Assignment

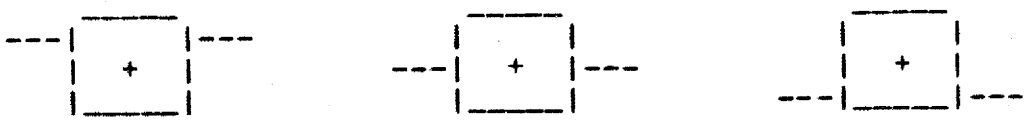
		DELB <0	DELB =0	DELB >0
DELA	<0	2	10	9
	=0	2	10	9
	>0	3	6	8

DISYP > 0

		DELB <0	DELB =0	DELB >0
DELA	<0	2	1	9
	=0	2	1	9
	>0	3	5	8

DISPY <= 0

where DELA = DA - DB (destination column - source column
DELB = DB - SB (destination row - source row)
DISPY = the Y displacement between the pin and the centre of the component (Y of centre - Y of pin)



DISPY < 0 DISPY = 0 DISPY > 0

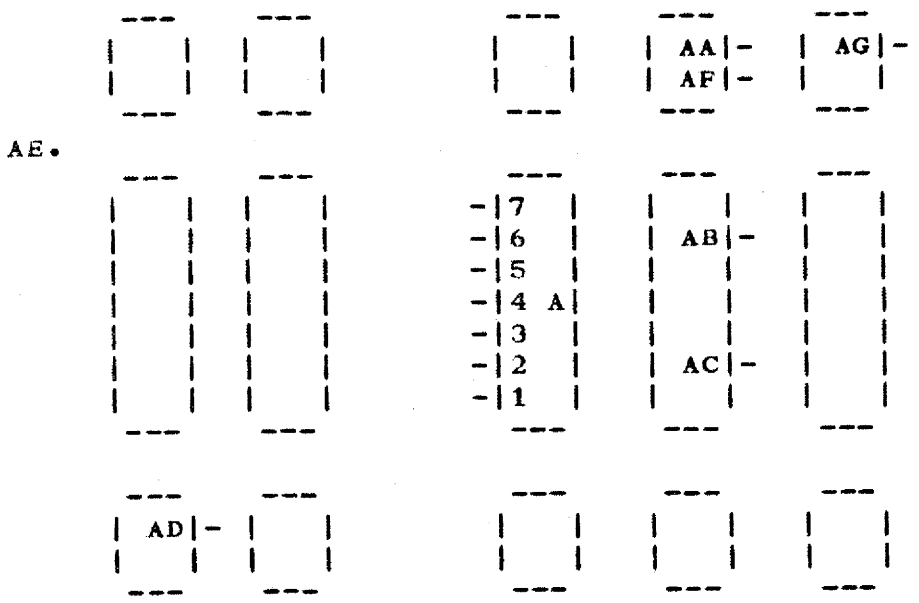
Figure 4.12 Classification of Connection Sources into Regions

Region	SB	SA	SY
1	N	A	D
2	A	A	D
3	D	D	D
4	N	N	D
5	N	A	D
6	N	D	D
7	N	N	D
8	D	A	D
9	A	A	D
10	N	D	D

where SA - source column
SB - source row
SY - absolute Y position of source pin

N - no sort
A - ascending sort
D - descending sort

Figure 4.13 Table to Control Sorting Within Regions for Pin Assignment



a) Connections for Pin Assignment

	Region	SB	SA	SY
AA	2	3	4	y1
AB	1	2	4	y2
AC	10	2	4	y3
AD	8	1	1	y4
AE	4	-	-	y5
AF	2	3	4	y6
AG	2	3	5	y7

b) Before Sorting

	Region	SB	SA	SY
AB	1	2	4	y2
AA	2	3	4	y1
AF	2	3	4	y6
AG	2	3	5	y7
AE	4	-	-	y5
AD	8	1	1	y4
AC	10	2	4	y3

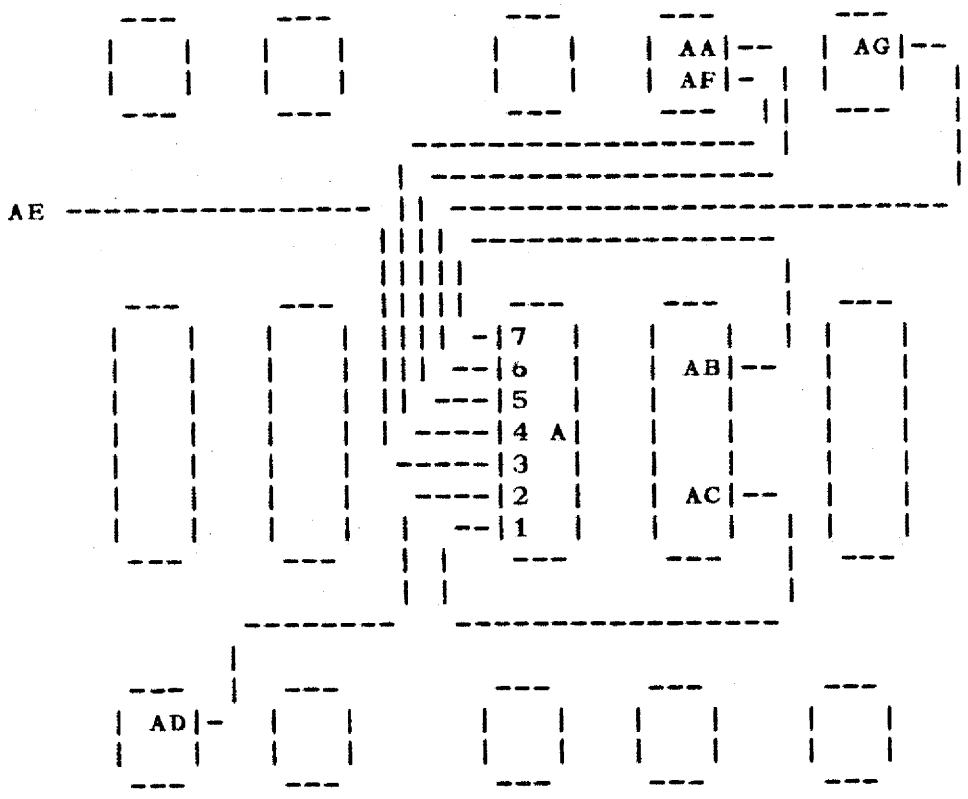
c) After Sorting into Regions

where y1 > y6

Figure 4.14 Pin Assignment Example and Table Sorts.

	Region	SB	SA	SY	Pin
AB	1	2	4	y2	7
AG	2	3	5	y7	6
AA	2	3	4	y1	5
AF	2	3	4	y6	4
AE	4	-	-	y5	3
AD	8	1	1	y4	2
AC	10	2	4	y3	1

a) After Multi-Sort



b) Final Pin Assignment

Figure 4.15 Multi-Sort and Final Pin Assignment

5.0 Introduction

This chapter deals with the routing of the connections between components which have already been placed. It gives a new technique for representing the connections as well as algorithms which allow the reduction of crossovers between the connections.

5.1 Definitions

5.1.1 Path. A sequence of vertices and edges (alternately) v1, e1, v2, e2,....., vk, ek, vk+1 where ei is incident with vi and vi+1 and vi =/= vj, ei =/= ej.

5.1.2 Directed Path. A path in which for every edge ei the positive end is incident to vi and the negative end is incident to vi+1.

5.1.3 Cycle. A path in which vk+1 = v1.

5.1.4 Directed Cycle. A directed path in which vk+1 = v1.

5.1.5 Acyclic Graph. A directed graph which contains no directed cycles.

5.1.6 Cover. A vertex and an edge are said to cover each other if they are incident. A set of edges which covers all the vertices of a graph G is called an edge cover. [HARA69]

5.1.7 Minimum Edge Cover. An edge cover of a graph G is called minimum if it contains the smallest number of edges in any edge cover of G. [HARA69]

5.1.8 Minimum Feedback Edge Set. A minimum set of edges in a directed graph which when deleted will make the graph acyclic.

5.1.9 Minimum Feedback Vertex Set. A mimimum set of vertices in a directed graph which when deleted will make the graph acyclic.

5.2 Paths

5.2.1 Classification of Paths

Since direct connections are desired, the paths for routing connections on logic diagrams have been restricted to those shown in Figure 5.1.
These paths reflect the orientation of components on a logic diagram, i.e. inputs on the left and outputs on the right.

The horizontal path class contains paths between components in the same row. The vertical path class contains the paths for interconnections between adjacent columns where the path source is in column i and the path sink is in column i+1. The diagonal path class contains paths for interconnections which are not in the same row and not in the same column.

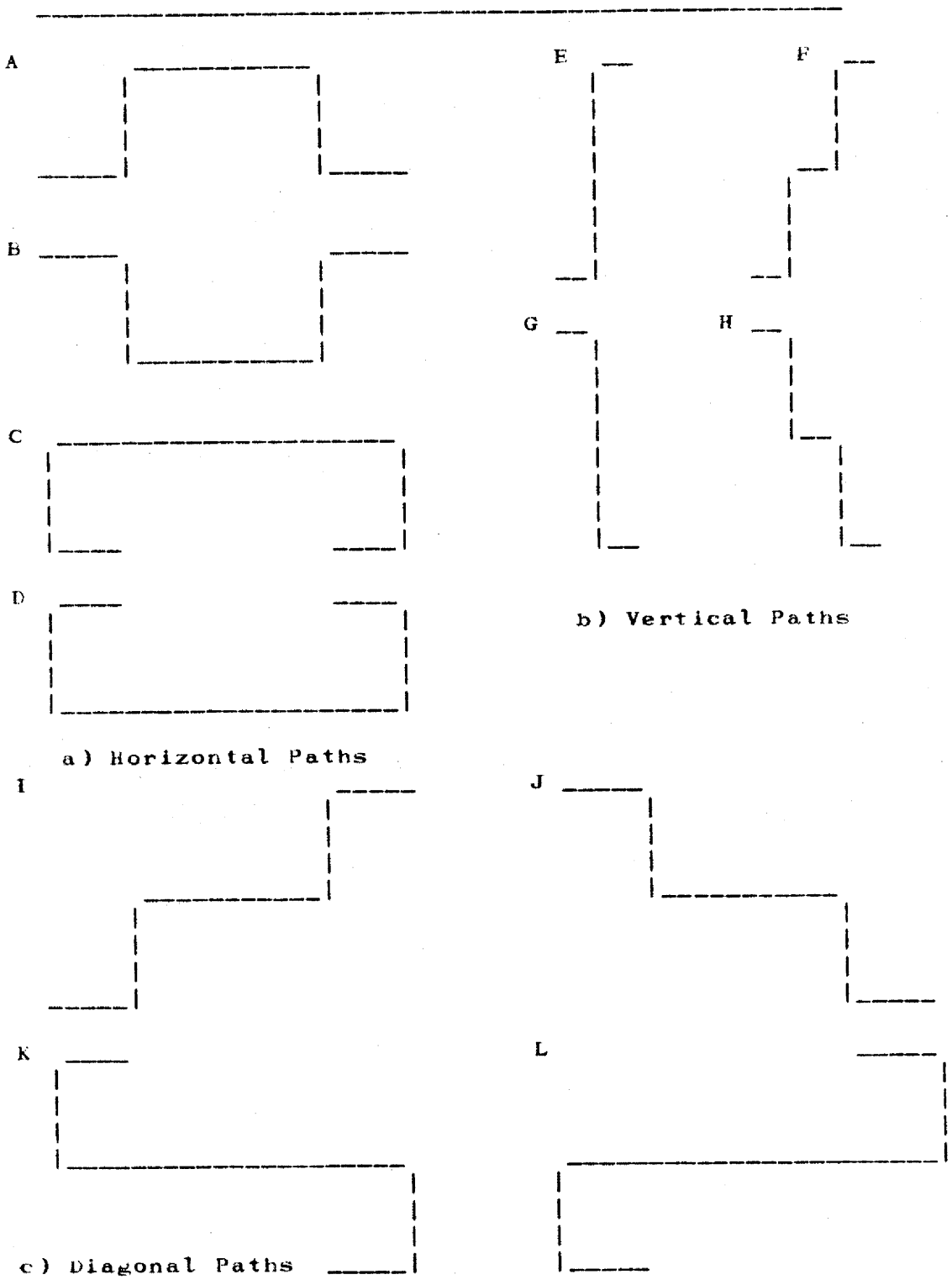


Figure 5.1 Classifications of Paths

5.2.2 Assignment of Connections to Paths.

The initial stage of this technique is to assign each connection to a path type. Depending upon the relative positions of the end points of a connection, this connection is assigned one of the path types A,B,C,D,E,G,I,J,K,L. Appendix B gives the conditions for assigning a connection to a path type.

There is one class of cases in which the path types A,B,C,D,E,G,I,J,K,L are not sufficient to allow the connections to be drawn without error. For instance if a space contains both an E and G type path and the ends of these paths have the same vertical positions, then the connections may not be routed correctly, i.e. they overlap. However, by changing one of these paths into a F or H type path respectively, the routing can be performed. It is for the correction of such cases that the path types F and H are necessary.

5.3 Segments.

5.3.1 Segments of Paths.

In the algorithm of Hashimoto and Stevens, each wire was broken up into horizontal and vertical segments which were collected according to the different horizontal and vertical spaces which they occupied. For subsequent processing each wire segment was represented by a straight line. Unfortunately

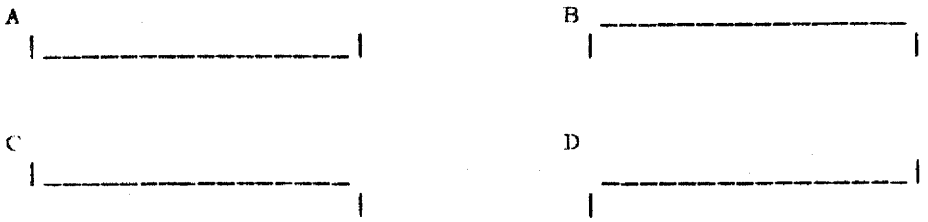
this representation loses much information which may be used to predict crossings and the conflicts.

Considering the path classes, as described in the previous sections, each path may be broken down into segments which fit into one of the segment classes shown in Figure 5.2. These segments maintain the important information concerning the ends of the segments which indicate the direction in which the path travels and whether the ends are free to move.

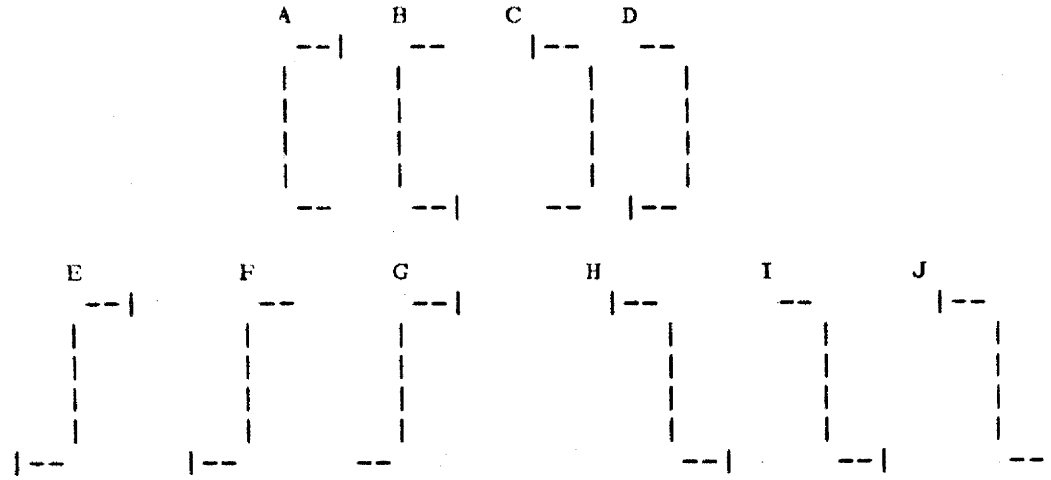
5.3.2 Intersection of Segments.

In any assignment of segments to channels, there are two types of segment intersections that can occur. The first type includes those intersections between segments that will occurs regardless of the channel assignment. These intersections will be called non-reducable (NR) intersections. The second type of intersections are those which will occurs only if a proper assignment of segments is not made. These are called reducable (R) intersections. An optimal assignment of segments to channels would be such that the number of R-type intersections is minimal.

Whether two segments intersect may be determined by using the intersection tables given in Appendix C. There are different tables for the vertical and horizontal segments. The relative positions of the end points of the two segments determines which table to use. These tables assume that the segments are sorted according to their maximum end points. The



a) Horizontal Segments



1) `--|` `|` - end is fixed to a component pin which may not move

2) `--` `|` - end occurs in a horizontal space and may move

3) both end types 1) and 2) may not have the same vertical position (would imply a component in the middle of a horizontal space)

b) Vertical Segments

Figure 5.2 Segment Classes

codes in the tables describe the relationships between segments. Using these relations it is possible to assign an order to the segments such that intersections are reduced.

5.3.3 Assignment of Segments to Channels. Example 5.1.

Consider Figure 5.3 which shows connections between components which are to be routed. Again, the connections to be made are indicated by number pairs. The connections 1-1, 3-3, 4-4, and 9-9 are connections between components while the connections 2-2, 5-5, 6-6, 7-7, 8-8, 10-10, and 11-11 are assumed to connect the components, as shown, to components on other parts of the diagram which are not shown. Figure 5.4 shows the vertical segments ordered according to their maximum end points. The intersection tables are used to generate the ordering relations shown in Figure 5.5. The code which caused the relationship is shown beside each one. The notation 'XL: s1 -> s2' implies that an intersection of segments s1 and s2 will occur and s1 should be assigned a channel to the left of s2. For those relations with a X code the order is not specified.

The information contained in these relations is best represented in the form of a directed graph. The corresponding graph of Figure 5.5, called the segment ordering graph is a directed acyclic graph as shown in Figure 5.6.
Each segment is represented by a vertex of the graph and two vertices of the graph are joined by an edge if an ordering relation exists between the two segments. If a relation s1 -> s2 exists then the edge between vertices 1 and 2 will have its positive end in vertex 1 and its negative end in vertex 2. By redrawing the segment ordering graph as a partial ordering, Figure 5.7, the information becomes more meaningful

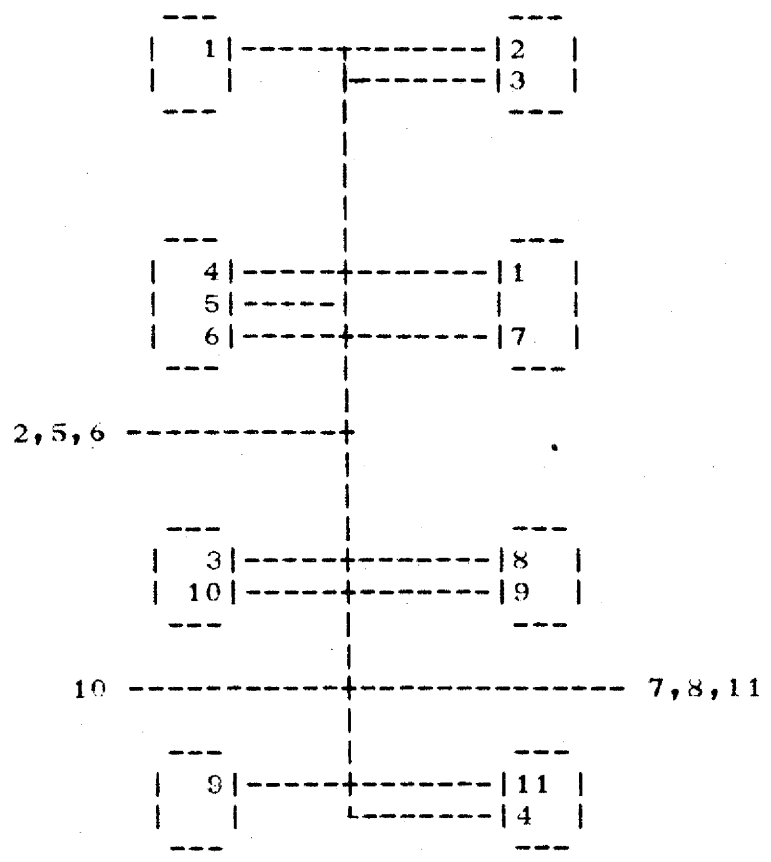


Figure 5.3 Example 5.1

and gives better insight into the procedure which will be described shortly.

The remaining information describes the compatibility of segments. This information is contained in the segment compatibility graph which also has a vertex representing each segment. As shown in Figure 5.8, it is an undirected graph with an edge between any two vertices if the corresponding segments are not compatible in the same channel. This data may be obtained from the simple check of end points necessary to determine into which Case the segments will fall. As indicated

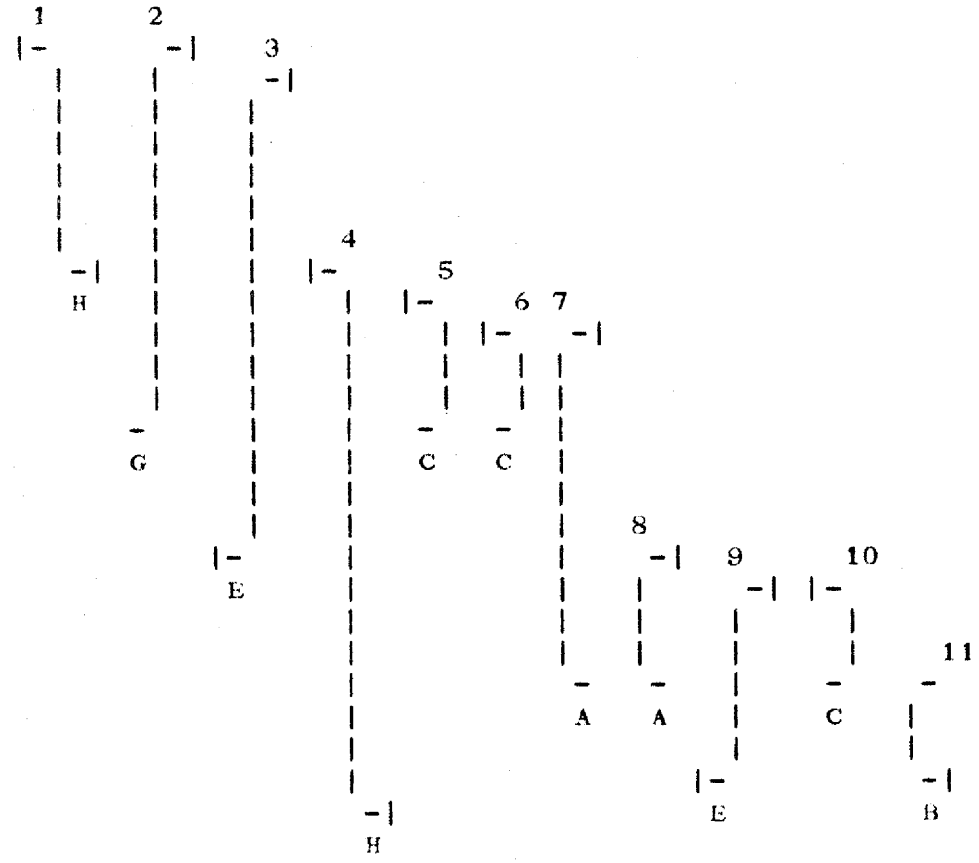


Figure 5.4 Ordered Segments from Figure 5.3

XL: s1 -> s2	R : s3 <- s6	R : s5 <- s6
X : s1 & s3	L : s3 -> s7	L : s5 -> s7
R : s1 <- s4	L : s3 -> s8	
		L : s6 -> s7
L : s2 -> s3	R : s4 <- s5	
X : s2 & s4	R : s4 <- s6	L : s7 -> s8
R : s2 <- s5	L : s4 -> s7	X : s7 & s9
R : s2 <- s6	L : s4 -> s8	R : s7 <- s10
L : s2 -> s7	X : s4 & s9	
	R : s4 <- s10	X : s8 & s9
X : s3 & s4	L : s4 -> s11	R : s8 <- s10
R : s3 <- s5		
		R : s9 <- s10
		L : s9 -> s11

Figure 5.5 Ordering Relations for Segments of Figure 5.4

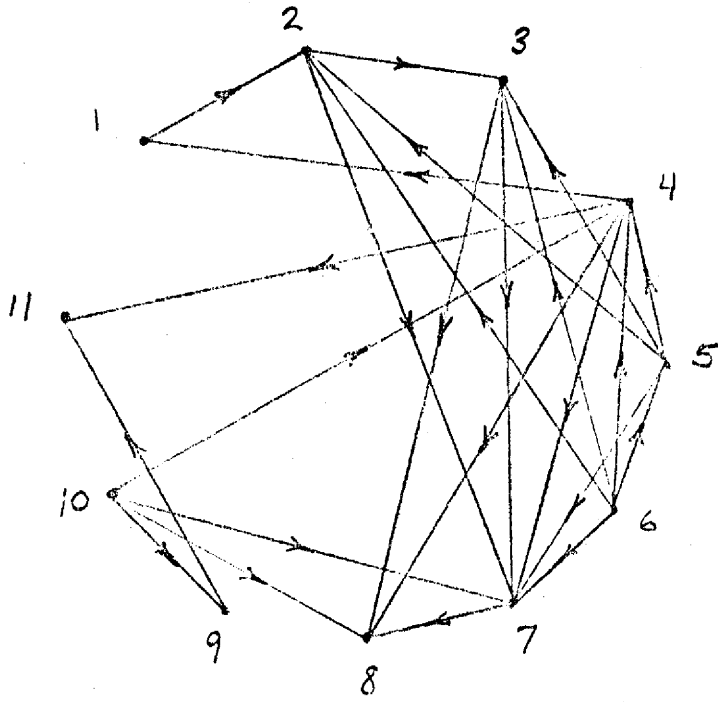


Figure 5.6 Segment Ordering Graph for Relations of Figure 5.5

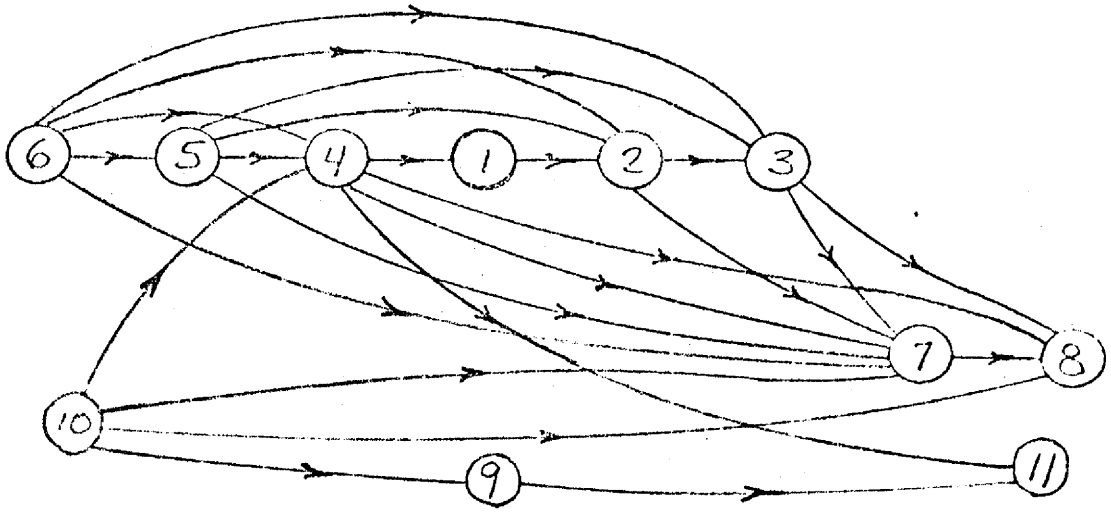


Figure 5.7 Segment Ordering Graph Drawn as a Partial Ordering

in Appendix C, for vertical segments only those which segments which fall into Case V1 are compatible.

The segments may be assigned to channels by looking at the invalencies of each vertex of the segment ordering graph.

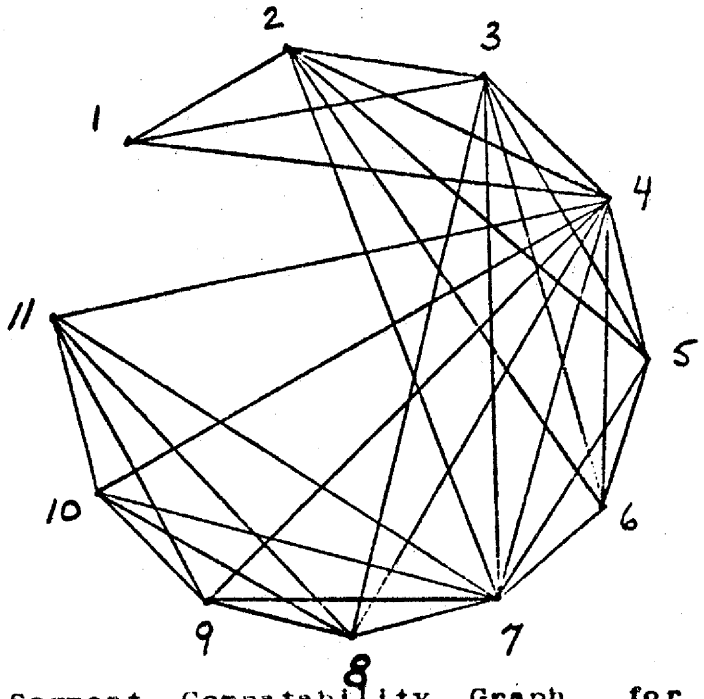


Figure 5.8 Segment Compatability Graph for Segments of Figure 5.4

These are listed in the leftmost column of the table in Figure 5.9. Since the directed graph is acyclic there will be one or more vertices with an invalence of zero.

vertex	invalency	c#1	c#2	c#3	c#4	c#5	c#6	c#7	c#8
1	1			0	#				
2	3	2	1		0	#			
3	3	2	1			0	#		
4	3	1	0	#					
5	1	0	#						
6	0	#							
7	6	4	3	2		1	0	#	
8	4	3		2			1	0	#
9	1	0	#						
10	0	#							
11	2		1	0	#				

Figure 5.9 Invalency Table and Its Processing

Select the first such one in the table and assign it to channel one. This is indicated in the table by a '#' in the

channel column of that vertex row. Search for the next vertex of invalence zero that is compatible with the previous segment (from compatibility graph) and assign it to channel one also. When no more segments may be assigned to this channel, remove the vertices (segments) in this channel from the graph. This is effected by decrementing by 1 the invalence of all vertices which have these vertices (segments) as sources. Repeat this procedure for the next channel. When the invalence of a vertex is decremented in the table, the result is given in the column of the channel being assigned. The result of the channel assignment is shown in Figure 5.10. The horizontal segments of the diagram are ordered as they would be after a similar processing of the horizontal segments. The assignment used 8 channels and 7 crossings. It may be noted that these crossings were predicted by the total number of X, XL, and XR codes in Figure 5.5.

5.3.4 Correction of Conflict Situations Using Segments.

The technique just described is effective providing that the segment ordering graph is acyclic. Unfortunately, this is not always the case. The cycles which appear in these graphs correspond to a conflict situation and make it impossible to choose an ordering of the segments. Thus each cycle must be found and removed by fixing the causes of the conflicts.

These conflicts occur when path types E and G have common end points and thus overlap on any order of channel assignment. The conflicts may be resolved by replacing one or

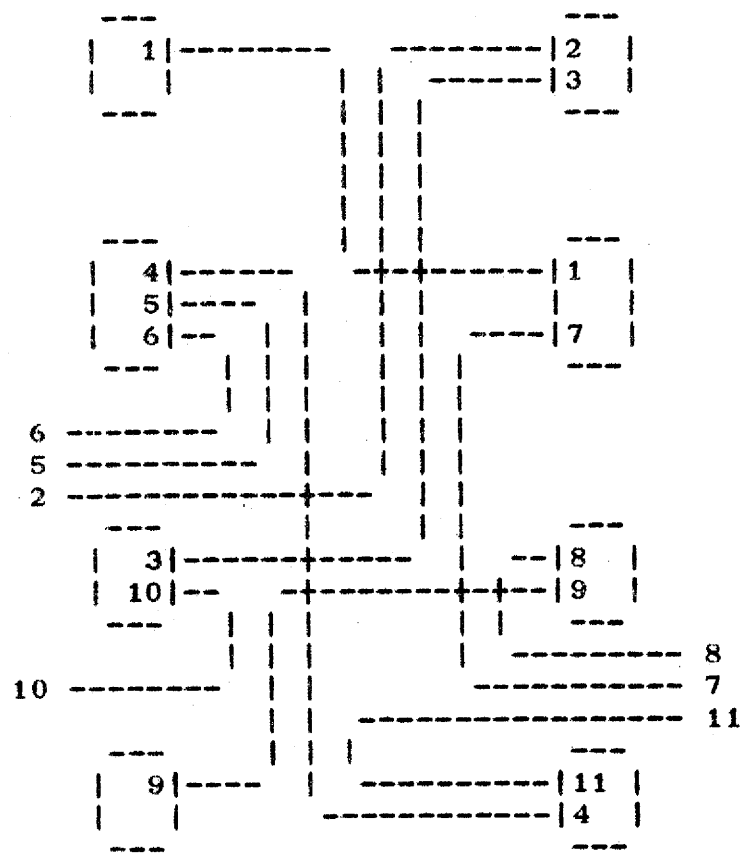


Figure 5.10 Final Routing for Example 5.1

more of the E and G path types with the path types F and H respectively. In terms of segments, this amounts to 'splitting' segments E and H (paths E and G) into segments F,G and I,J (paths F and H) respectively.

Consider the example of Figure 5.11. The two segments in Figure 5.11 have a code of E which implies they intersect but further implies that they in themselves constitute an conflict situation. For this code both the relations s1 -> s2 and s1 <- s2 are used as indicated in Figure 5.11b. The segment ordering graph for these segments, shown in Figure 5.11c, contains the cycle 1-2-1. In this case the arbitrary decision

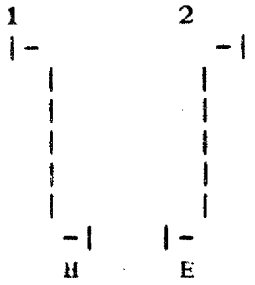
was made to split segment 2 into two segments as shown in Figure 5.11d. The split segment ordering graph of Figure 5.11e reflects the the splitting of segment (vertex) 2, but it may not be obtained from another processing of the new set of segments with the intersection tables. The relationship between segments 2a and 2b must be remembered (i.e. s2a <- s2b) since, in more complicated examples, new cycles will appear if the intersection tables are used.

The split segment ordering graph is obtained from the old segment ordering graph by splitting the desired vertex 'v' into two new vertices 'va' and 'vb'. The relationship between 'va' and 'vb' is determined by the type of segment 'v' as given in the table below.

	segment type		relation	split source	split sink
	va	vb			
segment type E	G	F	va <- vb	vb	va
segment type H	J	I	va -> vb	va	vb

All edges which had vertex 'v' as a source or sink are altered to have the split source vertex as source and the split sink vertex as sink respectively.

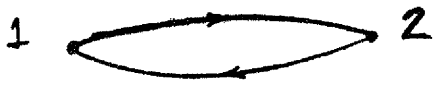
The split segment ordering graph must now be acyclic. Figure 5.11f shows the segments assigned to channels using the method described in the previous section. Note that when a segment is split a horizontal segment is generated which must be added to the other segments in the appropriate horizontal space for subsequent processing.



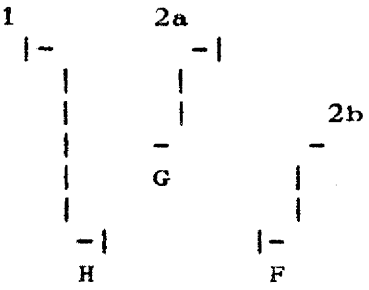
a) Conflicting Segments

E : s1 -> s2
s1 <- s2

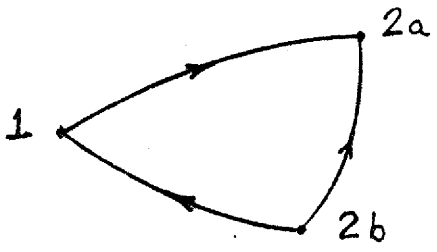
b) Relations for These Segments



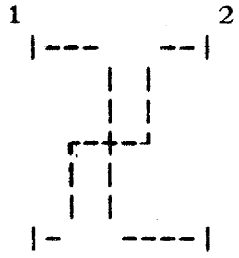
c) Segment Ordering Graph (cycle 1-2-1)



d) Segments after Splitting



e) Split Segment Ordering Graph (no cycles)



f) Segments Assigned to Channels

Figure 5.11 Simple Example of Splitting Segments

5.3.5 Assignment of Segments to Channels. Example 5.2.

The connections indicated in Figure 5.12 would generate many conflict situations if not handled properly. The segments that are initially assigned to the connections are given in Figure 5.13 and the segment ordering relations are shown in Figure 5.14. The segment ordering graph for these relations,

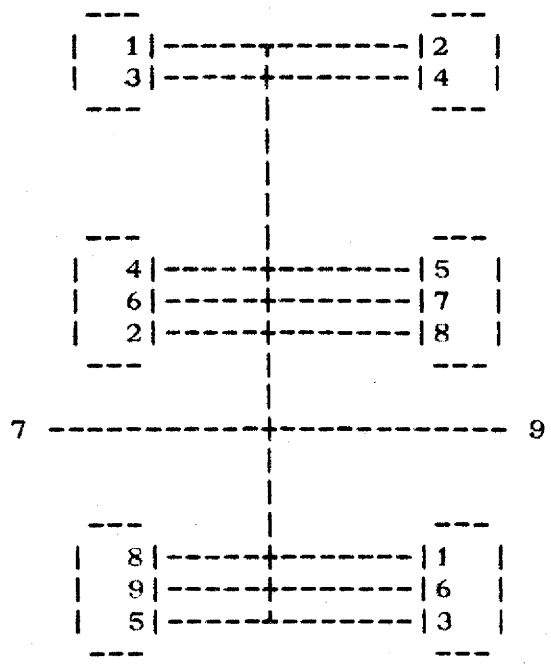


Figure 5.12 Example 5.2

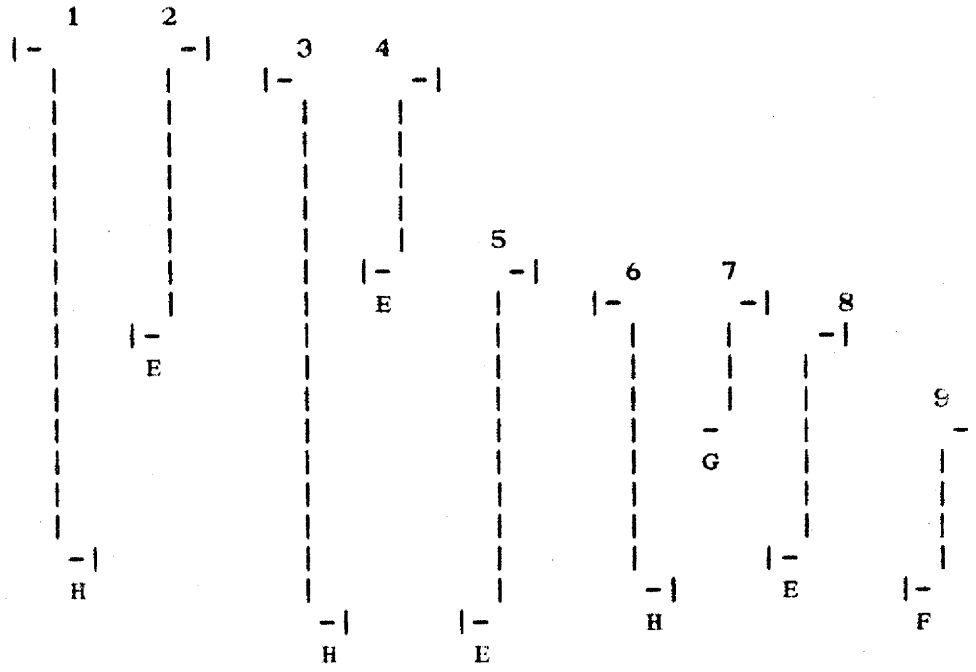


Figure 5.13 Ordered Segments of Example 5.2

XL: s1 -> s2	L : s2 -> s7	X : s5 & s7
R : s1 <- s3	L : s2 -> s8	X : s5 & s8
X : s1 & s4		X : s5 & s9
X : s1 & s5	XL: s3 -> s4	
R : s1 <- s6	XR: s3 <- s5	XL: s6 -> s7
X : s1 & s7	X : s3 & s6	X : s6 & s8
XR: s1 <- s8	X : s3 & s7	XR: s6 <- s9
X : s1 & s9	X : s3 & s8	
	X : s3 & s9	L : s7 -> s8
X : s2 & s3		
X : s2 & s4	L : s4 -> s5	L : s8 -> s9
L : s2 -> s5		
X : s2 & s6	X : s5 & s6	

Figure 5.14 Ordering Relations for Segments of Figure 5.13

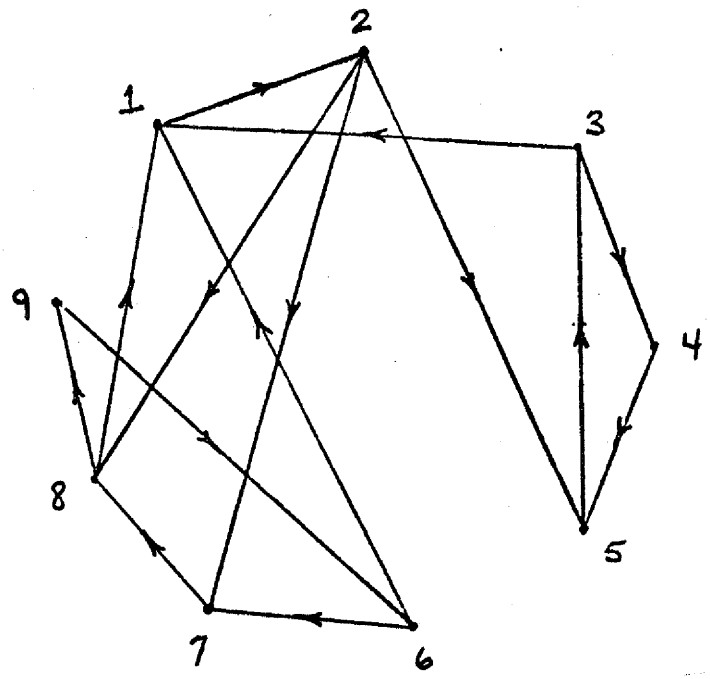


Figure 5.15 Segment Ordering Graph for Relations of Figure 5.14 (non-acyclic)

given in Figure 5.15, is not acyclic and the cycles which cause this are listed in Figure 5.16.

Cycle	Vertices
1	1, 2, 8, 1
2	1, 2, 7, 8, 1
3	1, 2, 5, 3, 1
4	1, 2, 7, 8, 9, 6, 1
5	1, 2, 8, 9, 6, 1
6	3, 4, 5, 3
7	6, 7, 8, 9, 6

Figure 5.16 Cycles of the Segment Ordering Graph of Figure 5.15

If the cycles of the segment ordering graph are independent†, then one vertex from each cycle must be chosen for splitting. If any cycles are not independent, then it is desired to choose a minimum set of vertices which, when split, will make the graph acyclic. In the graph of Figure 5.15, there is more than one such set. For instance, the sets {3,8} and {5,8} are both minimum.

Choosing the set {3,8} and splitting these vertices as outlined in the previous section will result in the split segment graph as shown in Figure 5.17.

When a segment is actually split, it may be necessary to decide in which horizontal segment this split should occur. For example, segment 3 may be split in either the top or bottom horizontal space. The choice of horizontal space will effect the number of channels and the number of crossings in the final routing. If segment 3 is split in the top horizontal space then the final routing which appears in Figure 5.18 uses 9 channels and has 24 crossings. By splitting segment 3 in the

† Two cycles will be said to be independent if they do not have any vertices in common

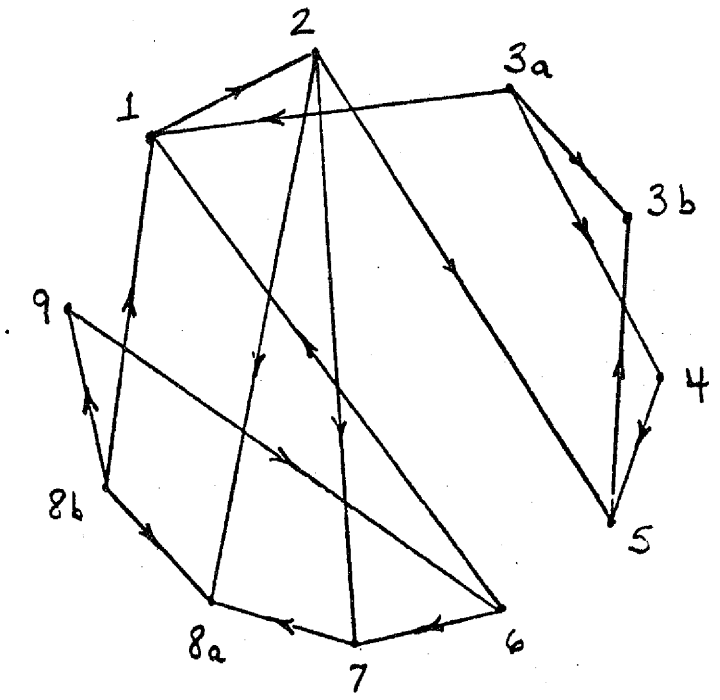


Figure 5.17 Split Segment Ordering Graph of Figure 5.15 (acyclic)

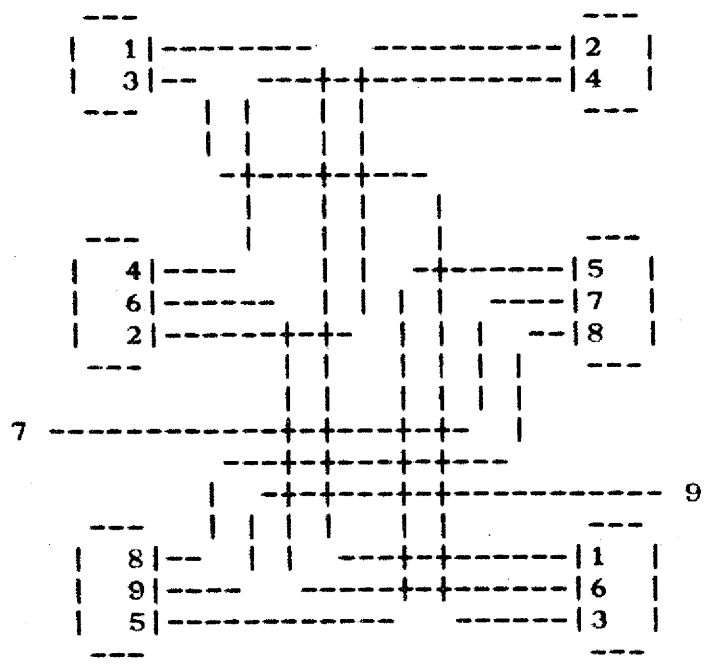


Figure 5.18 Final Routing for Example 5.2

bottom space the final routing would use 8 channels and have 26 crossings.

If the vertex set {5,8} had been used for splitting then the final routing would take 10 channels with 22 crossings.

5.3.6 Algorithms for Assignment of Segments to Channels.

The previous sections have outlined the difficulties in assigning segments to channels. The techniques described will now be formalized and algorithms will be given.

Let S = {s1, s2, s3,..., sn} represent the ordered segments of a space which are to be assigned to channels. Define a binary relation '->' such that si -> sj implies that sj should be assigned a channel to the right of si. Let G1 = (V1 , E1) be a graph, called the segment ordering graph, with vertex set and edge set E1 = {(s1,i , s1,j) | si -> sj for all i,j , i =/= j}.

Define a symmetric relation '%' such that si % sj implies that si and sj are channelwise incompatible. Let G2 = (V2 , E2) be a graph, called the channel incompatibility graph, with vertex set V2 = {s2,1, s2,2, s2,3,..., s2,n} and edge set E2 = {(s2,i , s2,j) | s2,i % s2,j for all i,j , i =/= j}.

If the segment ordering graph is acyclic then the segments may be assigned to channels immediately. The following algorithm will test if a graph is acyclic.

Algorithm to Test if a Graph is Acyclic.

Step 1: - let W = {invalence of S1,i , i=1,n}

Step 2: - let Y = { i | wi=-1}, and Z = { i | wi=0}
if Z is empty and |Y| ≠ n, go to step 5
if Z is empty and |Y| = n, go to step 6

Step 3: - let D be a set of vertices
D = { n | for every (p,n) ∈ E1, p ∈ Z}

Step 4:
$$wi = \begin{cases} -1 & \text{, if } wi = 0 \\ wi - 1 & \text{, for each } wi \neq 0 \text{ and } i \in D \\ wi & \text{, otherwise} \end{cases}$$

go to step 2.

Step 5: Stop, graph is not acyclic

Step 6: Stop, graph is acyclic

If the segment ordering graph is not acyclic, the problem of finding the minimum number of nodes to split to make it acyclic is known as the minimum feedback vertex set problem. Karp [KARP72] has shown that this problem is equivalent to a class of problems for which there is no known good algorithm.

In earlier sections, the acyclic problem was discussed and solved by finding the cycles of the graph and then choosing a minimum set of vertices which cover the cycles of the graph. Recent algorithms to find the cycles of a graph have been given by [TIER70], [WEIN72], and [TARJ73].

A simple scheme to find a covering set of vertices is to generate a list of the vertices and the cycles in which they are contained. Select the vertex with the maximum number of

cycles and delete this vertex and the cycles from the list. Select the vertex which now has the maximum number of cycles and repeat the operations until all cycles have been removed from the list. If the number of vertices and cycles is small then various combinations may be tried and the best result selected.

Once it has been decided which vertices are to be split to make the graph acyclic, the split segment ordering graph must be generated. The following algorithm describes how to split the vertices and generate the graph.

Algorithm to Split Vertices.

Step 1: - let D be the set of vertices to be split
D = {d1, d2,....., dm}

- let F be the set of split vertices
F = {d1a, d1b, d2a, d2b,.....,dma, dmb}

- then the vertex set of the split ordering graph is
SVG = VG - D + F

Step 2: - let G = {g1, g2,....., gm}

$$\text{where } gi = \begin{cases} 0, & \text{if di is of segment type E} \\ 1, & \text{if di is of segment type F} \end{cases}$$

- let Q be a set of edges with positive end in D
Q = {(p,n) | (p,n) ∈ E1, p ∉ D, n ∈ D}

- let R be a new set of edges
R = {r1, r2,.....,rk}

$$\text{where } ri = \begin{cases} (p,dja), & \text{if } (p,dj) \in Q,\ gj = 0 \\ (p,djb), & \text{if } (p,dj) \in Q,\ gj = 1 \end{cases}$$

- let T be the set of edges with negative end in D
T = {(p,n) | (p,n) ∈ E1, p ∈ D, n ∉ D}

- let U be a new set of edges
U = {u1,u2,.....,ul}

$$\text{where } ui = \begin{cases} (djb,n) , & \text{if } (dj,n) \in T,\ gj = 0 \\ (dja,n) , & \text{if } (dj,n) \in T,\ gj = 1 \end{cases}$$

- let W be a new set of edges joining the split vertices
  $W = \{w1,w2,.....,wn\}$

$$\text{where } wi = \begin{cases} (djb,dja) , & \text{if } gj = 0 \\ (dja,djb) , & \text{if } gj = 1 \end{cases}$$

- then the edge set of the split ordering graph is
  $SE1 = E1 - Q + R - T + U + W$

Step 3: stop

The algorithm for assigning segments to channels follows. It is based on the algorithm which test if a graph is acyclic.

Algorithm for Assigning Segments to Channels.

Step 1: - set m = 0, - let W = {invalence of s1,i , i=1,n}

Step 2: - let Z ={ i | wi = 0 }
if Z is empty go to Step 5

Step 3: - set m = m + 1
- set Cm = ϕ
- set Cm = Cm + { Zi | (cmj,zi) $\notin$ E2, j = |Cm|, i=1,|Z|}
(by definition (cm0,zi) $\notin$ E2)

Step 4: - let D = { n | for every (p,n) $\in$ E1, p $\in$ Z}

Step 5:
$$wi = \begin{cases} -m & , \text{if } wi = 0 \\ wi - 1 & , \text{for each } i \in D \\ wi & , \text{otherwise} \end{cases}$$

go to Step 2

Step 6: the sets Ci, i=1,m contain the numbers of the assigned to channel i.

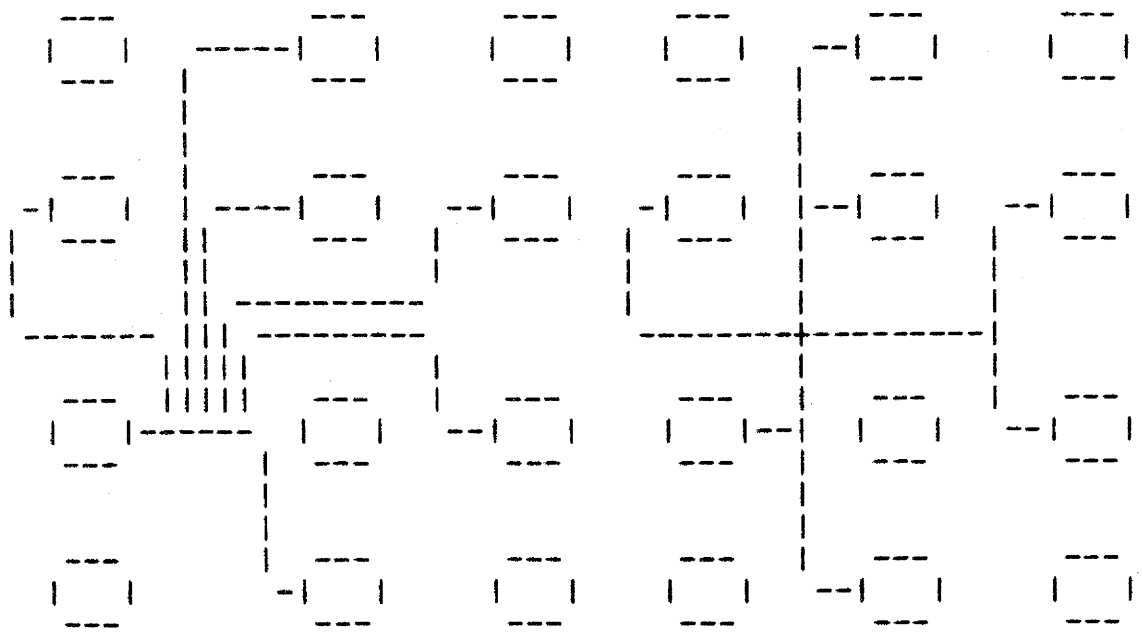
Step 7: Stop.

5.4 Nets.

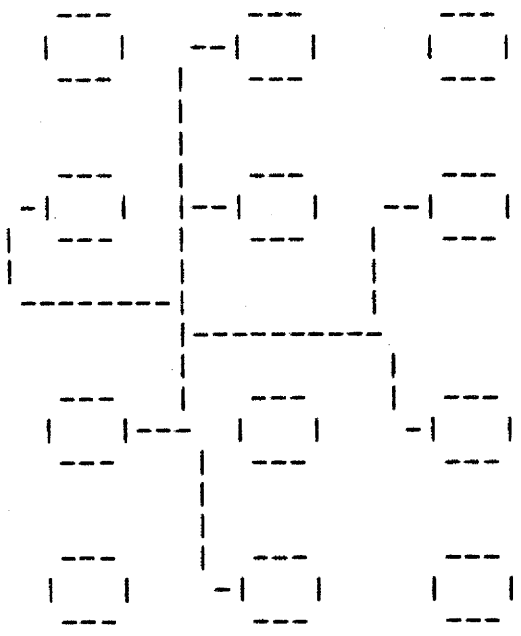
In all routing examples given previously, each connection was a net which had only one source and one sink. It is possible in a circuit that a net will have one source and many sinks. The segments as defined, along with the intersection tables, do route these nets correctly, but each connection of the net is considered by itself and handled separately. Each connection is assigned to spaces in such a way as to minimize its length. Note that this approach rarely results in a minimum net length. The length of the nets may be reduced within the constraints of the path types by changing the horizontal spaces to which the diagonal paths of the net have been assigned.

After the segments of the connections are assigned to spaces, the segments themselves are also considered separately. In Figure 5.19a the connections of a net have all been routed separately. Figure 5.19b shows one result possible when the segments are collapsed. In this case all segments which run in the same space are assigned to the same channel. Figure 5.19c shows a different method of collapsing the segments which, although not as neat, may allow more flexibility. Its segments are tied to the same channels only if they overlap, but not if their ends coincide. In all subsequent examples the first method will be used.

The segments which have been used up to this section will now be referred to as **simple segments**. When simple



a) Connections Routed Separately b) Collapsed Segments - Option 1



c) Collapsed Segments - Option 2

Figure 5.19 Collapsing of Segments to Form a Net

segments are collapsed, they form what will be called compound segments.

5.4.1 Collapsing of Simple Segments into Compound Segments.

There are two possible ways to collapse the segments.

1) collapse the simple segments and then assign them to channels or

2) assign the simple segments to channels and then collapse them

For the second method, segments which overlap would usually have to be collapsed into the larger segment. Segments, whose ends coincide, could not necessarily be assigned to the same channel. Thus if all overlapping segments were collapsed, the routing of the net would resemble that of Figure 5.19c. In many instances, collapsing segments will introduce new conflicts. Thus more checks must be performed. Figure 5.20 gives an example in which segments 3 and 4 have been collapsed but segment 5 may not be collapsed because of connection 2. The checking necessary to detect these cases almost duplicates the checking required in the initial assignment of segments to channels.

If the first approach is used, then the collapsing of the segments generate compound segments which are not accounted for directly in the segment type or in the intersection tables. This difficulty may be overcome by considering a compound segment as a collection of segments

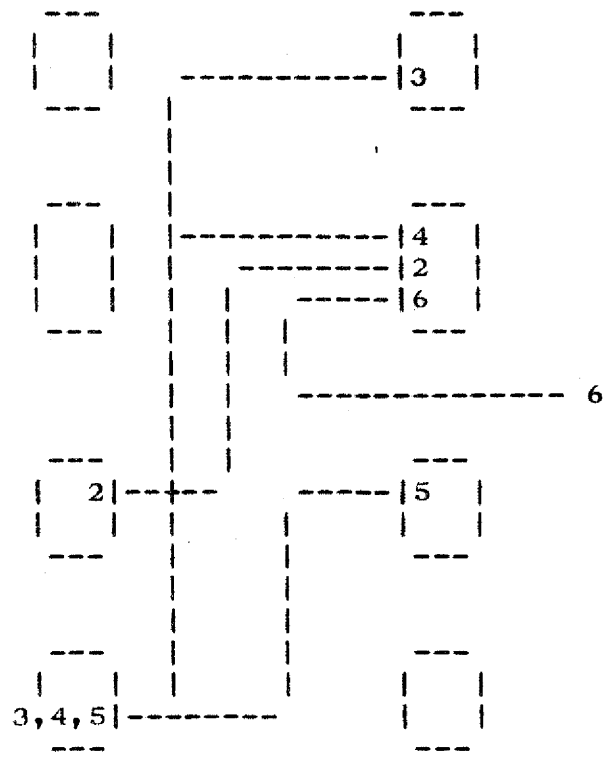


Figure 5.20 Collapsing of Segments Assigned to Channels

occupying the same channel and its assignment to a channel would be determined by the relations of its simple segments.

5.4.2 Assignment of Simple and Compound Segments to Channels.

Once the segments have been collapsed into compound segments, these segments are assigned to channels in the usual way. The compound segment maintains all the properties of the segments which compose it. Consider Figure 5.21. The connections to be made in Figure 5.21a contain three nets, N1 = {1,2,3}, and N2 = {4,5,6} and N3 = {7}. Figure 5.21b shows the segments of the second vertical space in terms of 2 compound segments and 1 simple segment. Figure 5.21c gives the segment ordering graph for these segments The compound

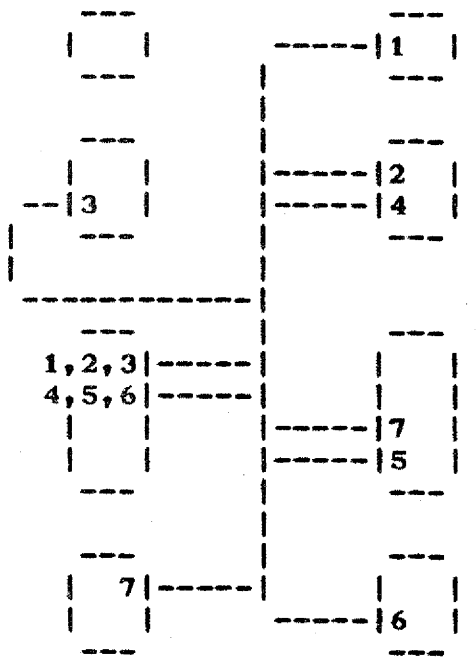
segments, corresponding to nets 1 and 2, are designated C1 and C2. It is not necessary to calculate the relationships within a compound segment. Thus compound segments may be considered as simple segments. Figure 5.21d shows the final assignment.

In Figure 5.21d, segment 7 crosses the compound segment 2 once. If the segments were not collapsed then segment 7 would cause two crossings. It would intersect simple segments 5 and 5. Because of the difference between simple segment crossings and net crossings, the assignment process may not give the minimum number of net crossings. For example if segment 7 would have been assigned to the right of the compount segment N2, then it would cross net N2 twice.

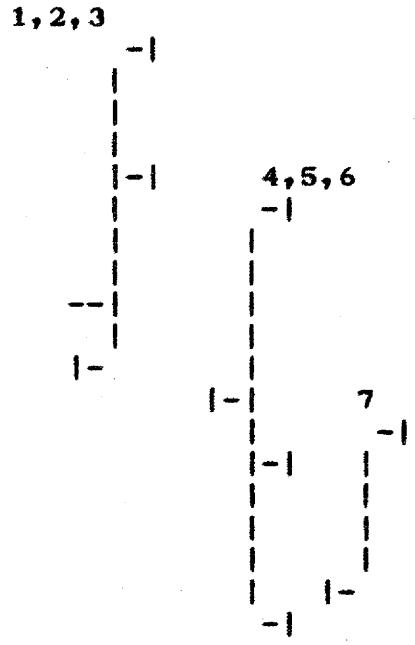
5.4.3 Nets and Non-acyclic Segment Ordering Graphs.

When a segment ordering graph is constructed for a compound segment, a cycle may occur which is not caused by a conflict situation. An example of this is given in Figure 5.22.

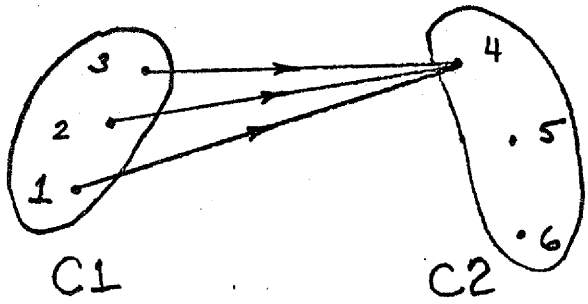
The cycle in the segment ordering graph is caused by the collapsing of the simple segments. This situation may be detected as the segment ordering graph is being constructed and may be resolved by deleting one of the offending edges of this graph. Confusion between this case and the conflict situation does not occur since the latter are caused by XL and XR relations and the former by just L and R relations. Depending on which of the relations is deleted, either of the diagrams in Figure 5.22e is possible. It shows the segments



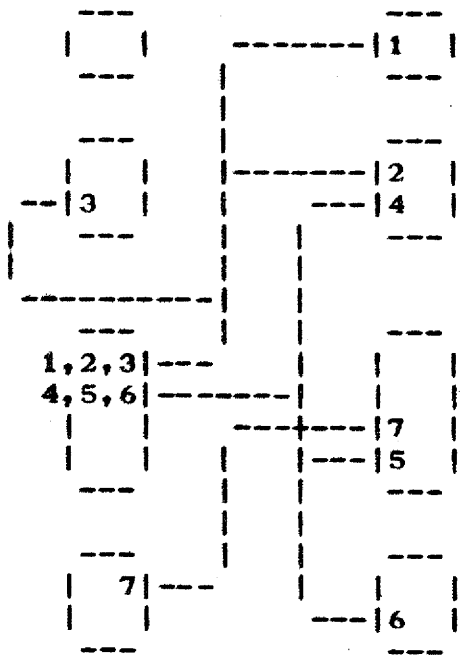
a) Segments Assigned to Centres of Spaces



b) Collapsed Segments

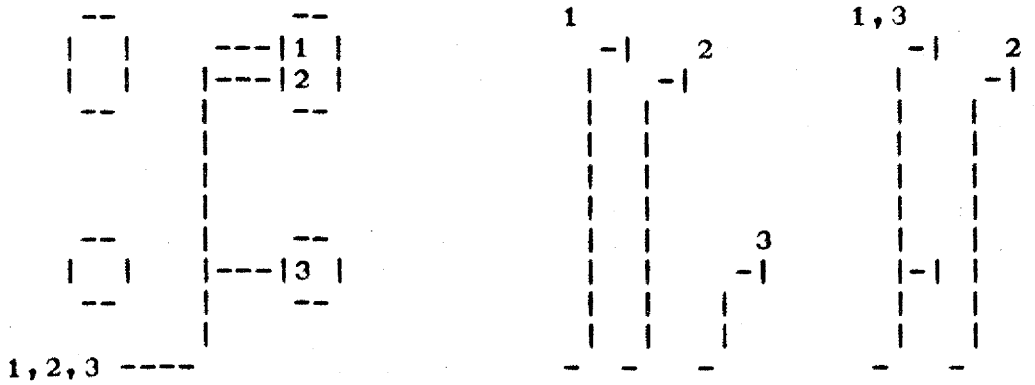


c) Segment Ordering Graph



d) Final Assignment

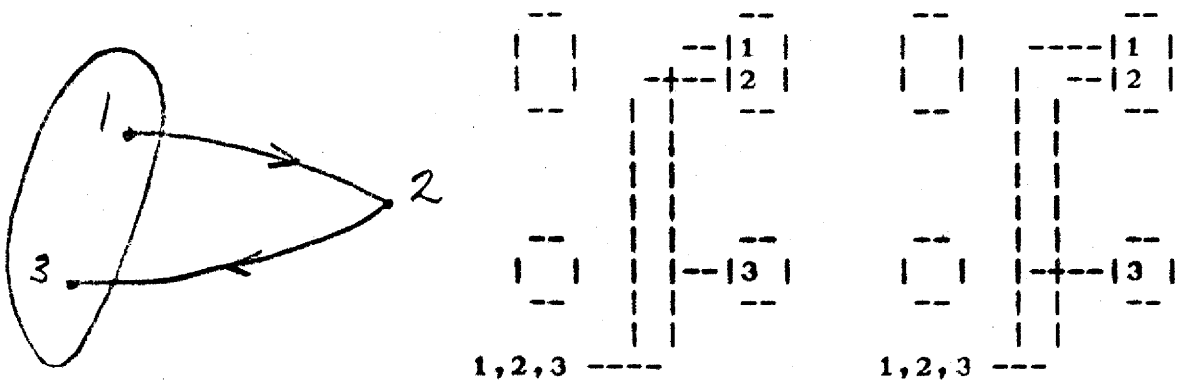
Figure 5.21 Routing of Simple and Compound Segments



a) Segments Assigned to Centres of Spaces (1 & 3 have same source)

b) Simple Segments

c) Collapsed Segments



d) Segment Ordering Graph

e) After Vertical Assignment, (two possibilities)

Figure 5.22 Cycles Caused by Collapsing Simple Segments

after they have been assigned to vertical channels. In this case both alternatives have one intersection and it is not possible to decide which alternative is best.

The conflict situations discussed earlier still exist but have not gotten much worse. In earlier sections, several examples of these situations, were given. These situations may involve two or more simple segments and for each segment, a potential conflict existed for each of its end points, i.e.

its source and its sink. Resolution of these conflicts by assigning an L,R order to the segments resulted in a cycle in the segment ordering graph.

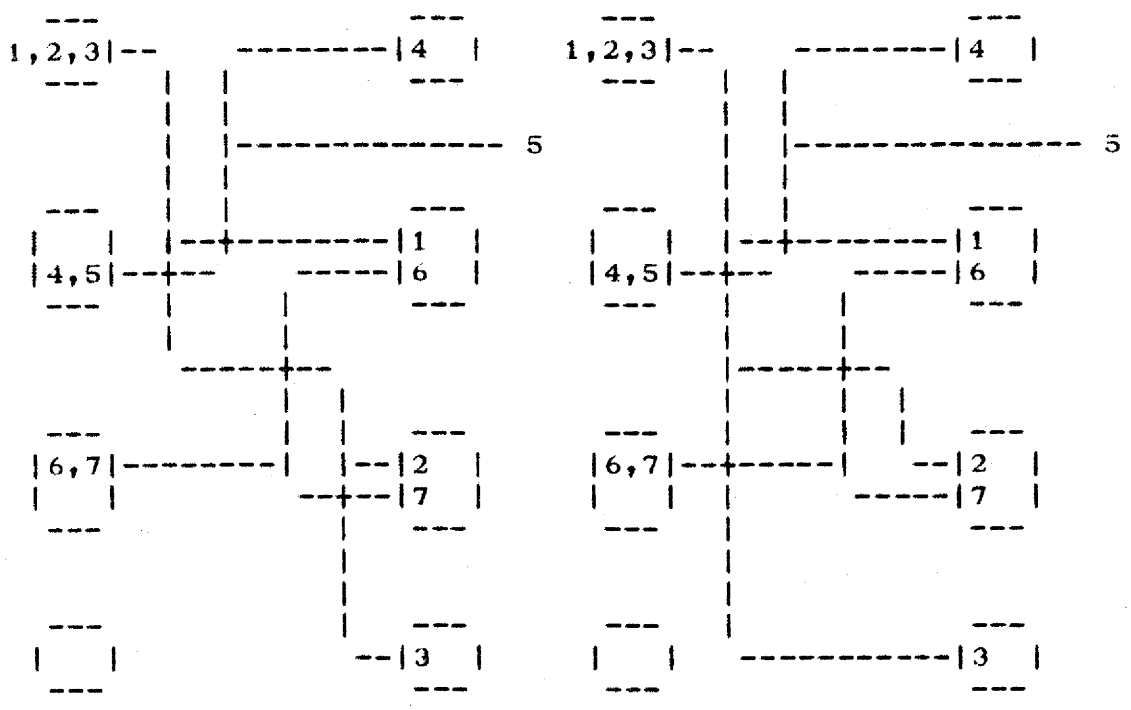
The conflict situations that exist now are still caused by simple segments but the complication is that the simple segments may be a member of a compound segment. Thus in the segment ordering graph a cycle may contain

1) all simple segments

2) simple and compound segments

3) all compound segments.

The first case has been discussed already. In the second case, the simplest solution is to split a simple segment. In the third case the compound statement must be split in some manner. Two possible ways of splitting a compound segment are shown in Figure 5.23. In the example it was decided that compound segment (1,2,3) was to be split. Figure 5.23a shows the split of the two simple segments 2, and 3. Figure 5.23b shows a simpler split of only segment 2 which results in an equally acceptable drawing.



a) Resolution - Option 1 b) Resolution - Option 2

Figure 5.23 Resolution of Conflict Situation Among Compound Segments

6.0 Introduction

This chapter discusses some of the results obtained by using the techniques and algorithms of the previous chapters. These results are reviewed in light of the objectives set in Chapter 1. Also included in this chapter are some suggestions for further research.

6.1 Samples of Automatically Produced Logic Diagrams.

The techniques of the previous chapters have been implemented into a system, called ALDGS, which takes a pseudo-boolean input description of a logic circuit and generates logic diagram(s) of the circuit. ALDGS was written in FORTRAN and comprises about 5000 statements. The system consists of 4 programs which are run sequentially to produce the diagrams. These programs perform the operations of 1) parsing, 2) partitioning and placement, 3) placement refinement, and 4) routing.

In order to generate a logic diagram for a logic circuit it is sufficient to supply the circuit description, the desired logic diagram size and the component size. The system then parses the circuit description to generate the graph representation, partition and place the circuit components on the logic diagram sheet, and perform the routing. Each type of component accepted by the system has an entry in a library

which contains a physical as well as a geometric description of the component with symbol and tag locations as described in chapter 3.

To demonstrate the capabilities of the system, several logic diagrams have been generated and are contained in Figures 6.1 to 6.8. The Figures 6.1 to 6.6 contain ALDGS generated logic diagrams for the circuits of Figures 2.1 to 2.6. The input descriptions for these circuits are given in Appendix D.

In Figures 6.1a and 6.1b, ALDGS detected several sections of logic which are ancestor-descendant disjoint†. It separated the logic of these sections and placed the sections on the same logic diagram.

When processing the circuit of Figure 2.2, ALDGS required two logic diagram sheets as shown in Figures 6.2a and 6.2b. It cured the reversed connection problem that existed in the original diagram by placing the offending components to the right of their ancestors.

The circuit of Figure 2.3 also contains some disjoint sections of logic which were placed at the top of the diagram. In the original diagram, this circuit contained several nets without sources. For Figure 6.3, external sources were supplied for these nets.

---

† If two sections of logic are ancestor-descendant disjoint, this means that there are no components which lie in one of the sections and have an ancestor or descendant in the other section.

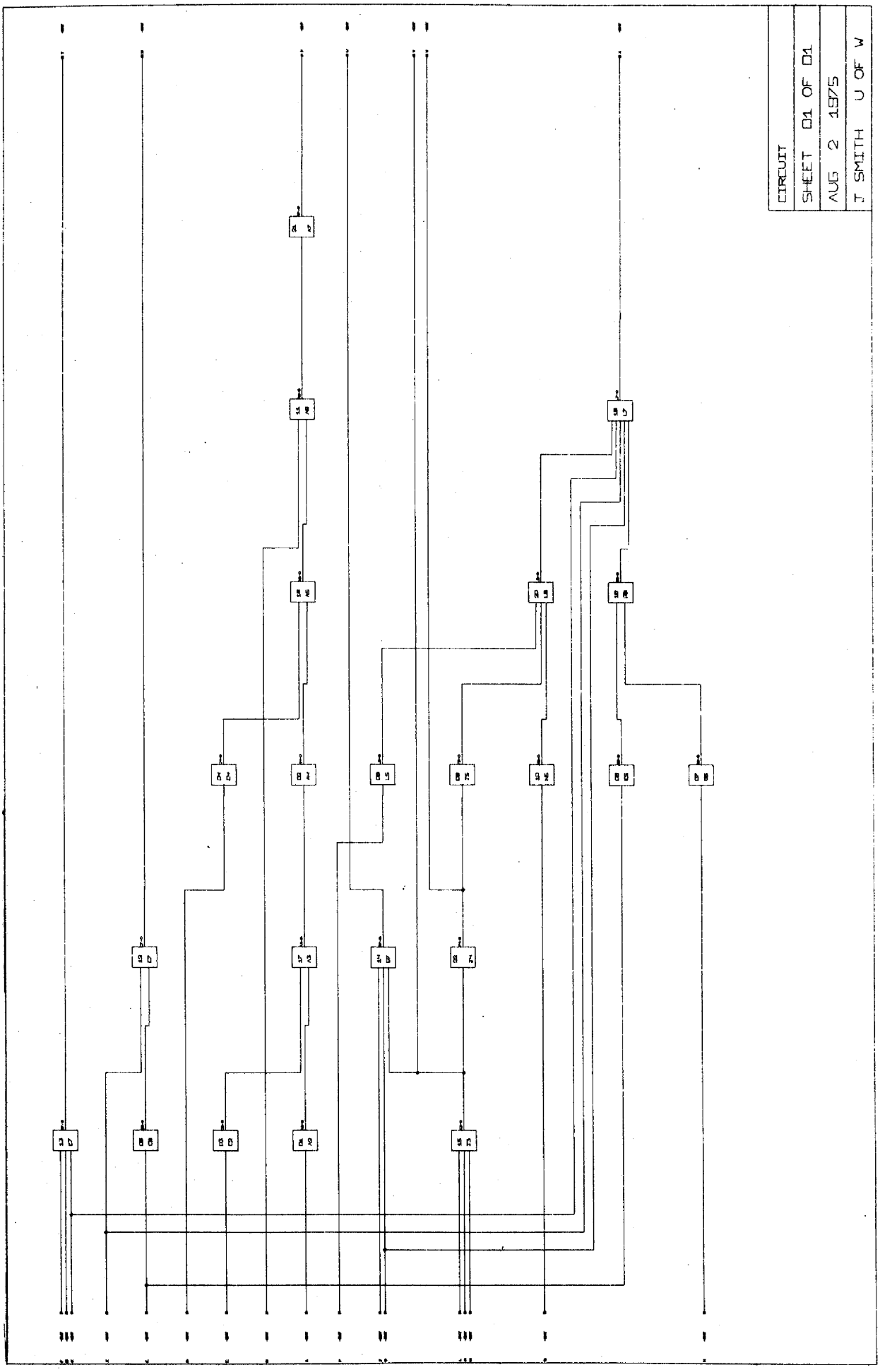


Figure 6.1a ALDGS Generated Logic Diagram for the Circuit of [ACM65]

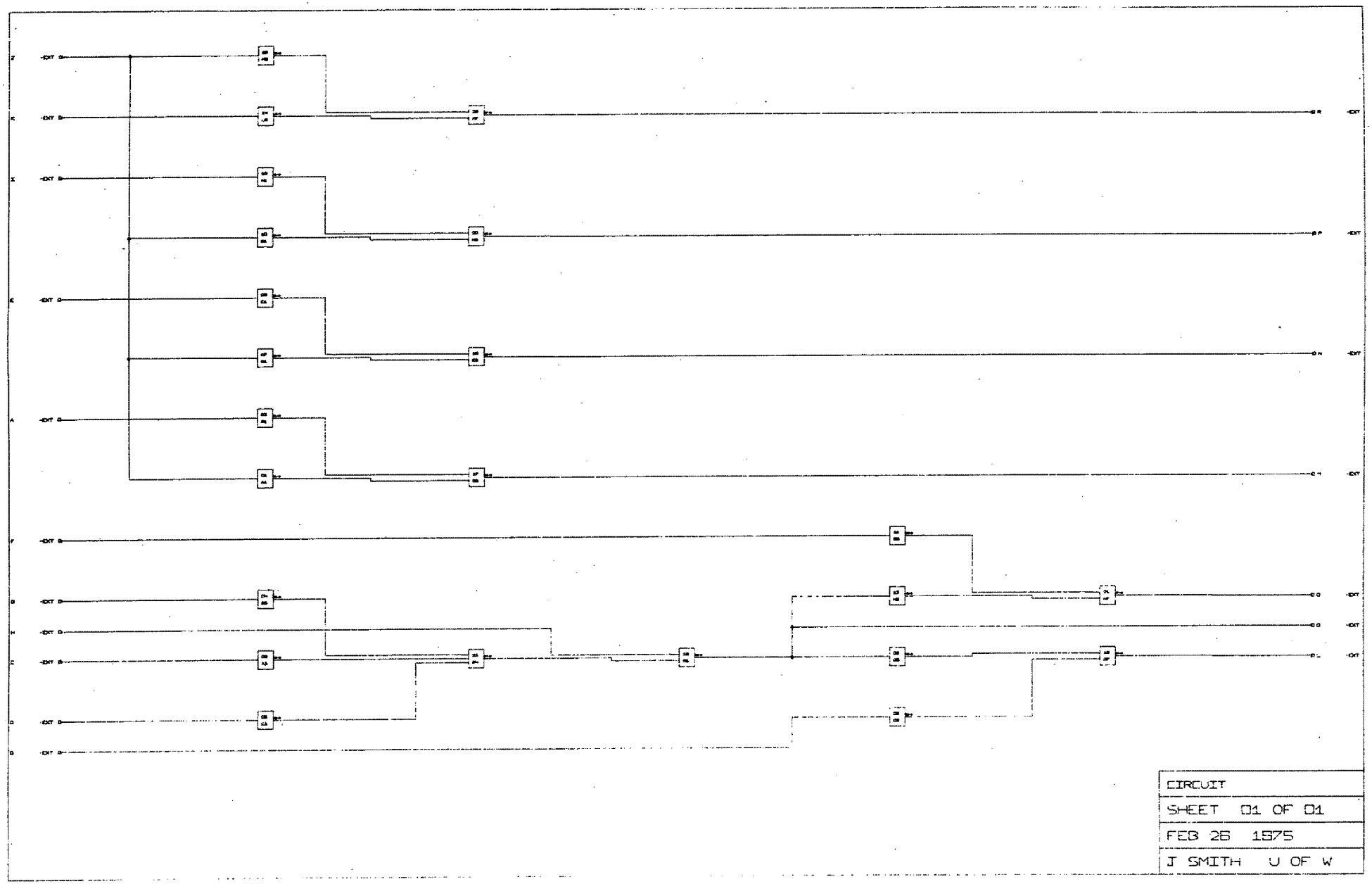


Figure 6.1b ALDGS Generated Logic Diagram for the Circuit 01 [FRIE69]

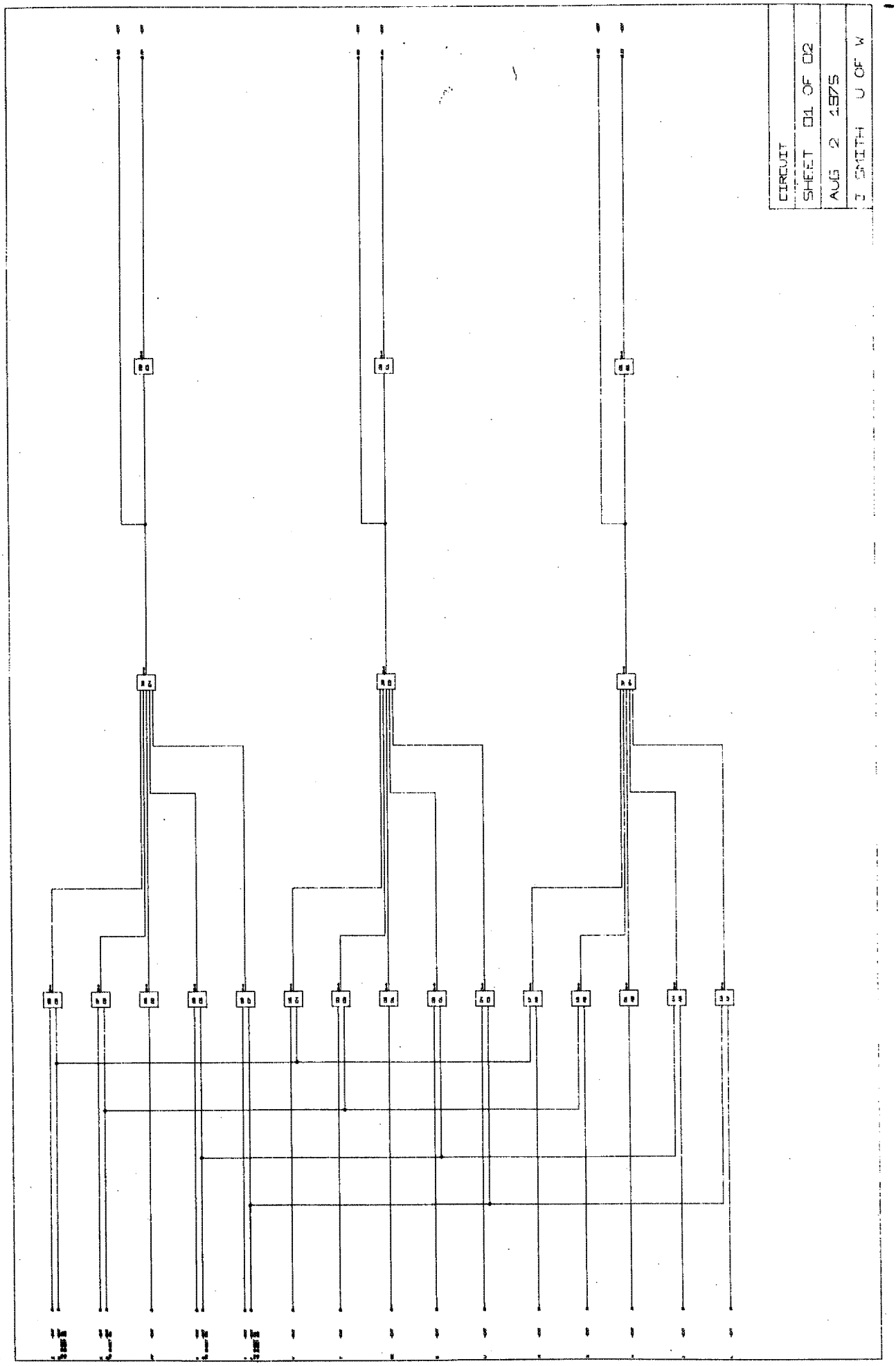


Figure 6.28 ALDGS Generated Logic Diagram for the Circuit of [HAYA7J] - Sheet 1

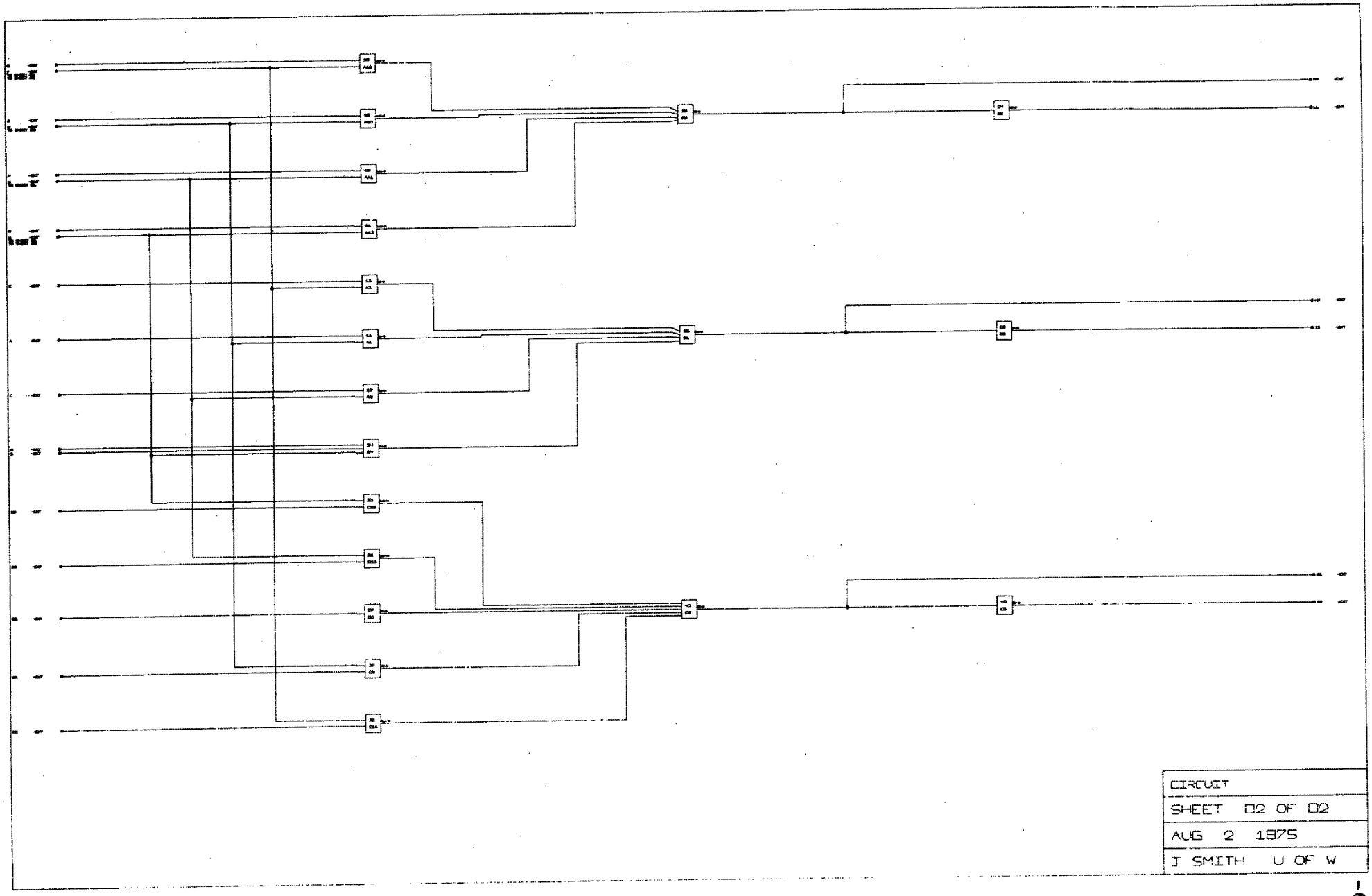


Figure 6.2b ALDGS Generated Logic Diagram for the Circuit of [BAYA70] - Sheet 2

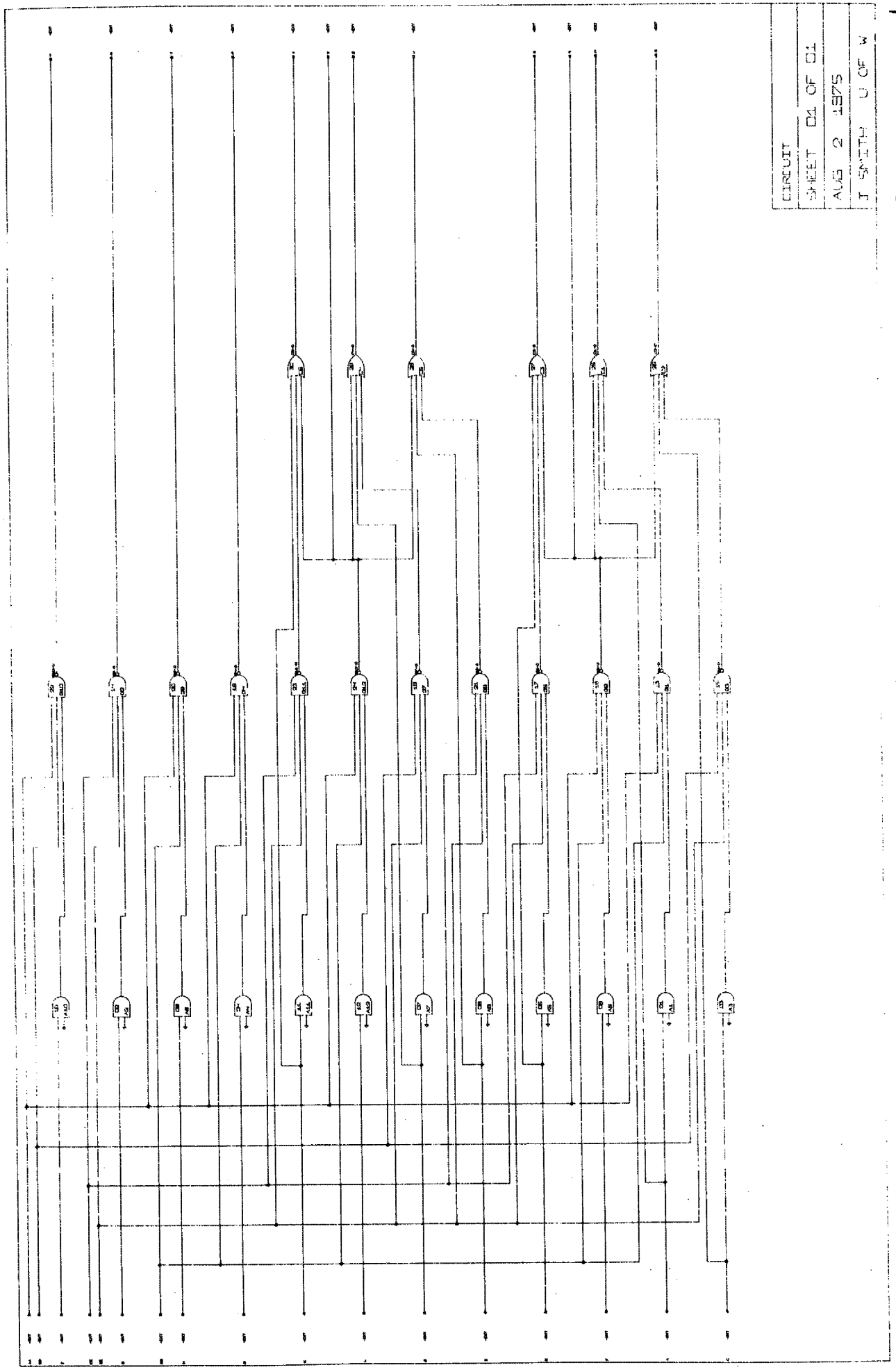


Figure (6.3) ALDGS Generated Logic Diagram for the Circuit of [ALA167]

Both Figures 6.4a and 6.4b are the same circuit as Figure 2.4. In Figure 6.4a, the layout was performed entirely automatically by ALDGS. This circuit is a difficult one for ALDGS to lay out as the symmetry is hard to detect. In this case the manual placement is better than the automatic placement. Figure 6.4b uses the same manual placement for the components as in the original diagram with the routing performed automatically by ALDGS. The routing on the new diagram is easier to follow than on the original diagram since the new paths make more direct connections. In one comparision the new diagram does suffer in that ALDGS (by intent) does not allow signal names to be placed in the middle of the diagram. All signal names must appear at the edges and this causes some long connections on this diagram.

When Figure 6.5 is compared to Figure 2.5, it may be seen that ALDGS separated out the sections of logic and placed them above each other. The automatic placement maintains the functional characteristics of the circuit and the routing is much cleaner than on the original. ALDGS does not allow (by philosophy) multiple output signal names for a net. Thus the last vertical space is much cleaner with fewer connections. (It does allow multiple destinations, however, and these would be listed below the signal name with the signal names separated appropriately to make room.)

The diagram of Figure 6.6 corresponds to the circuit of Figure 2.6 and generates a much better layout. It separates the sections of logic and places connected components close

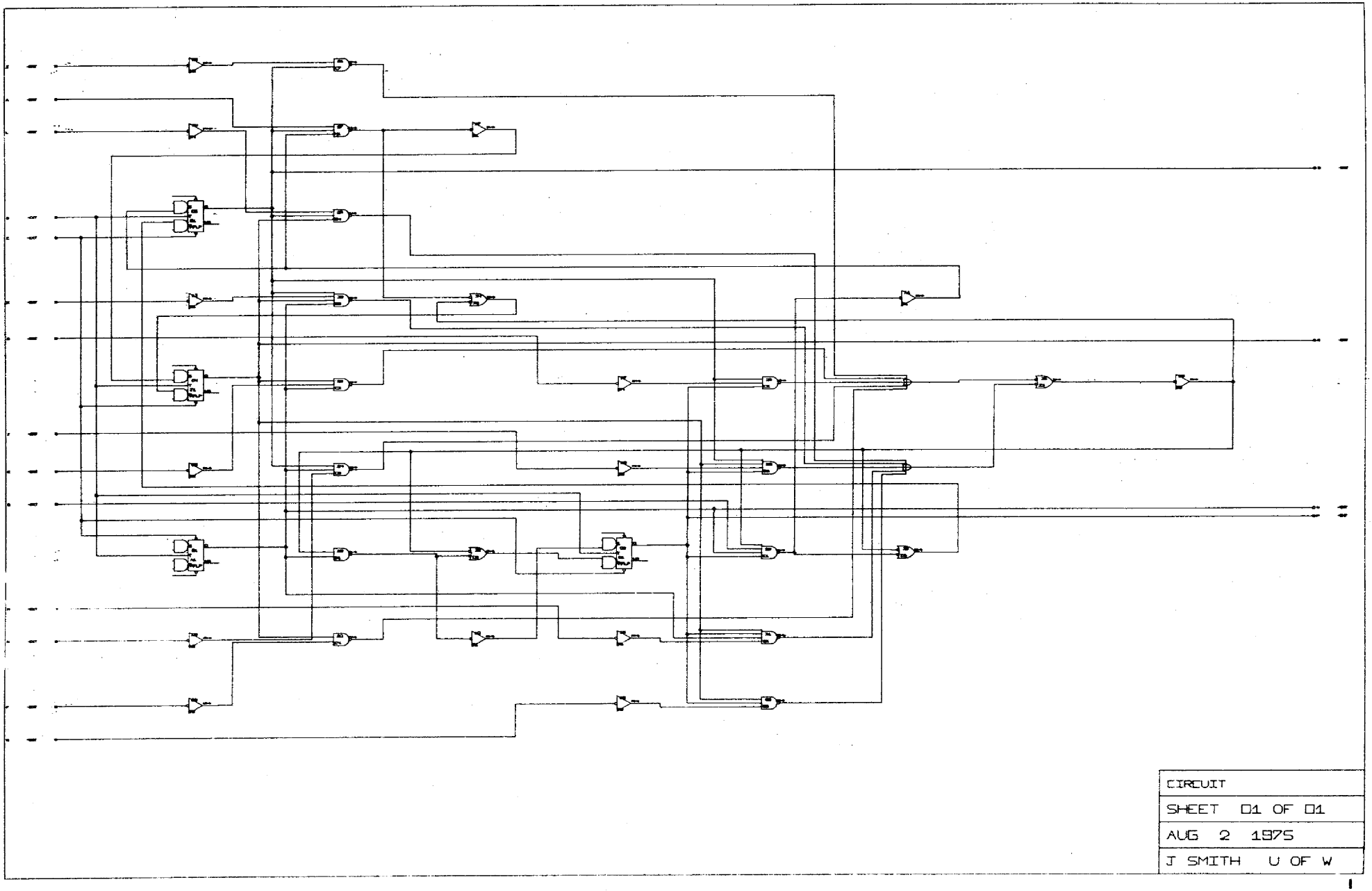


Figure 6.4a ALDGS Generated Logic Diagram for the Circuit of [BALD68] - Version 1

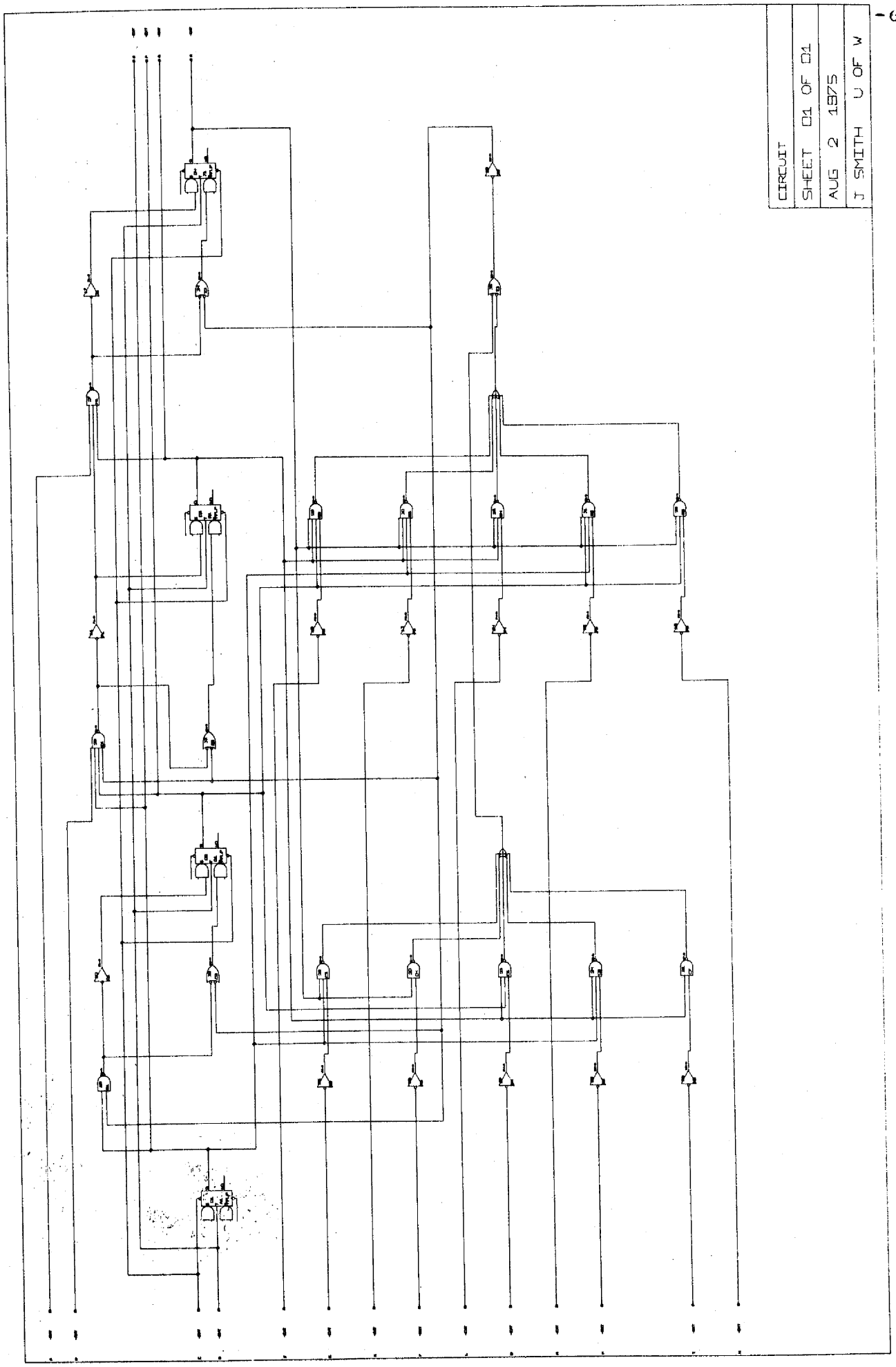


Figure 6.4b ALDGS Generated Logic Diagram for the Circuit of [BALD68] - Version 2

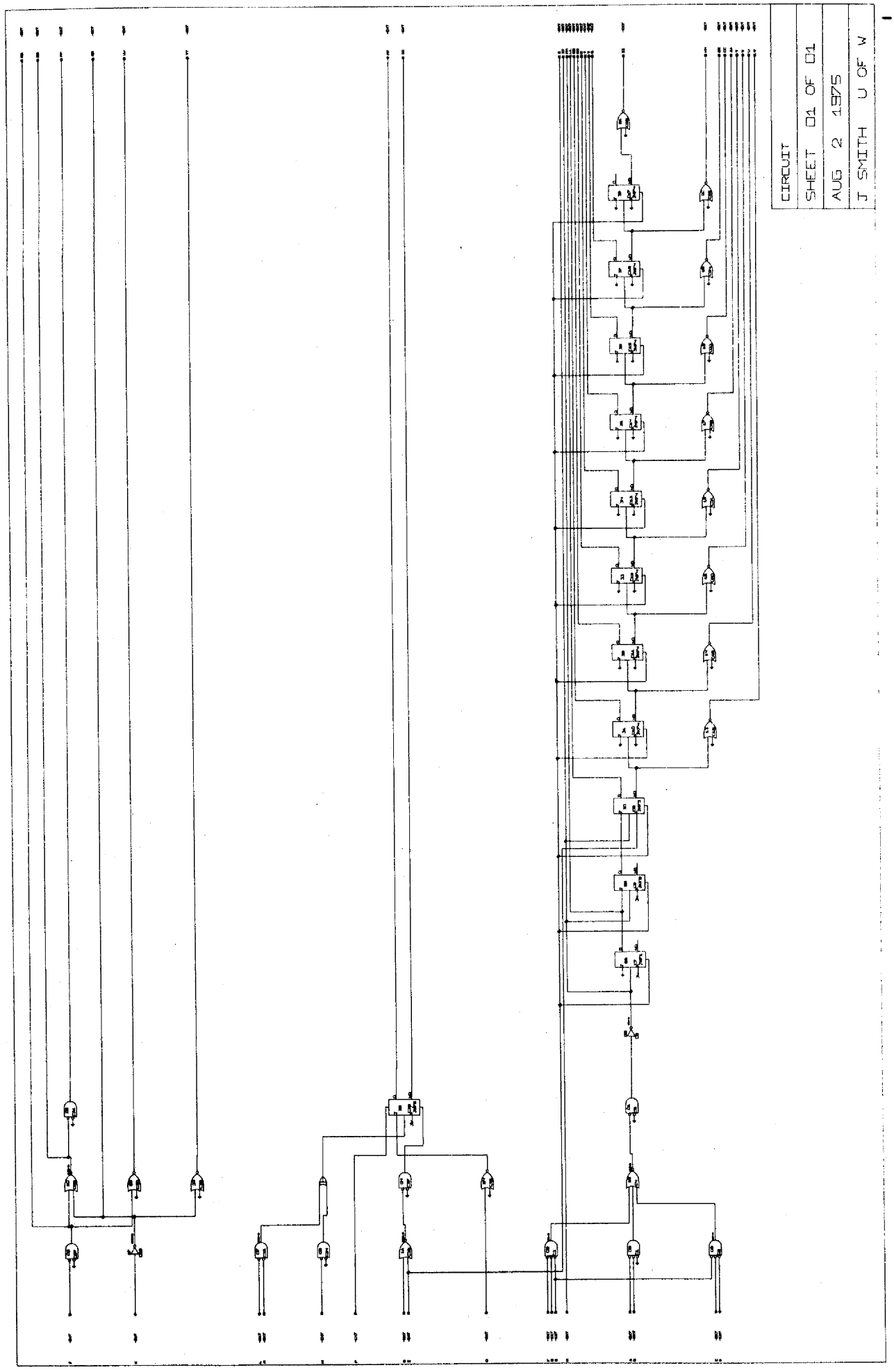


Figure 6.5 ALDGS Generated Logic Diagram for the Circuit of [SAN072]

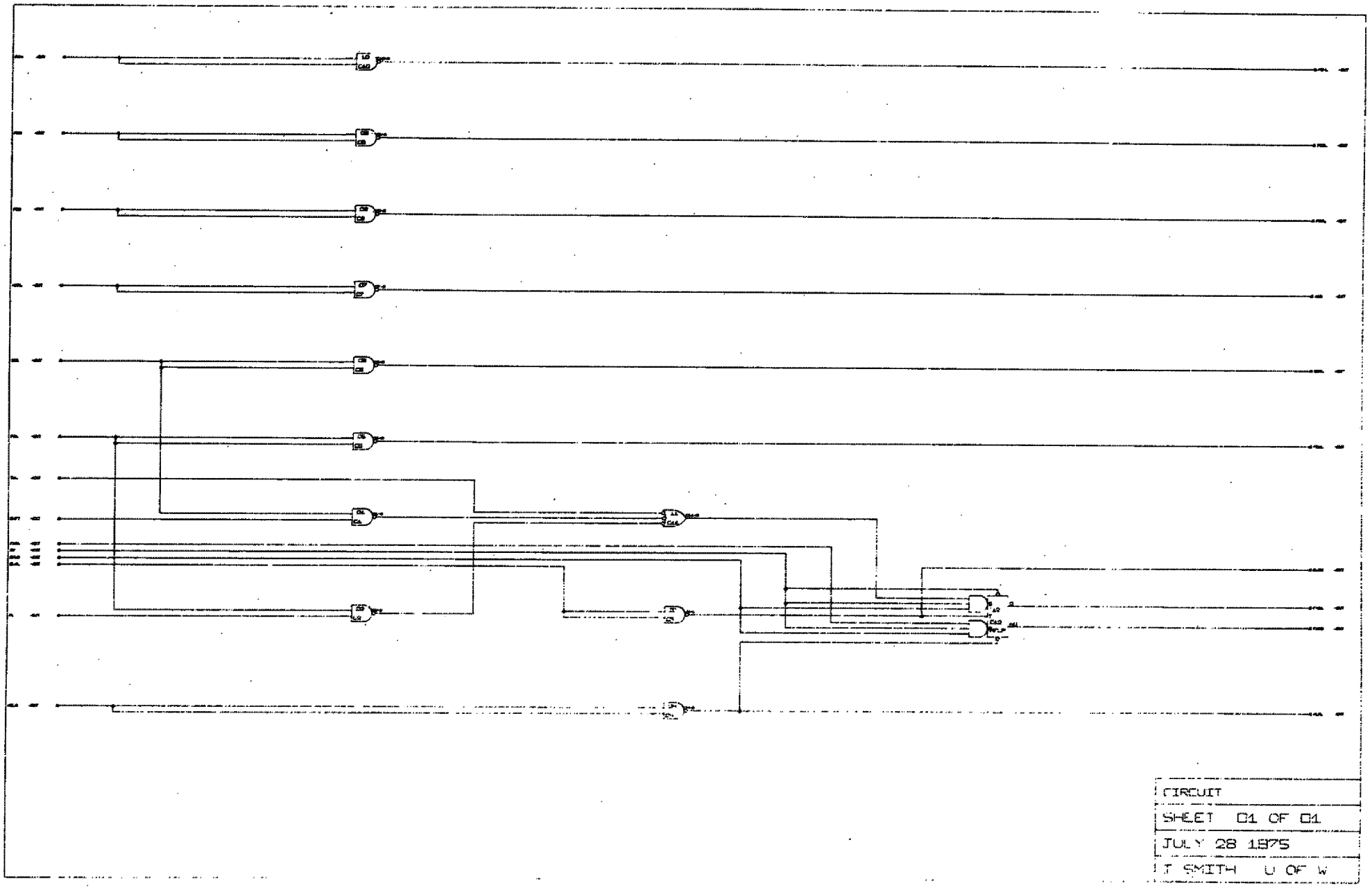


Figure 6.6 ALBUS Generated Logic Diagram for the Circuit of [LINN71]

together. The original diagram has some signals which both enter and leave the diagram. ALDGS does not allow this but would convey this information at the source of the signal.

The diagrams of Figures 6.7 and 6.8 are of the circuit represented by the graph shown in Figure 3.4. In Figure 6.7, the circuit is drawn on one large sheet and Figures 6.8a, 6.8b, and 6.8c show the circuit when it is drawn on three smaller sheets.

6.2 Performance of ALDGS - Complexity of Algorithms.

Statistics concerning each logic diagram example have been collected in Table 6.1. It gives the time required to generate each logic diagram. Times are included for partitioning and initial placement (P&P column), for placement refinement (Ref column), for pin assignment and boundary signal name placement (P&A column), for routing (R column), and for generation of the lines to draw the diagrams (GL column). These programs were run on a IBM 360/75 under the IBM FORTRAN-H compiler. Each program runs in less than 180K bytes, for these examples.

Of all the above operations, only the partitioning and placement operation works with the entire circuit. The other operations work with one diagram at a time, and thus their memory requirements and speed are bounded by those of the largest size of diagram. This means that their memory requirements are still limited to 180k bytes and the maximum

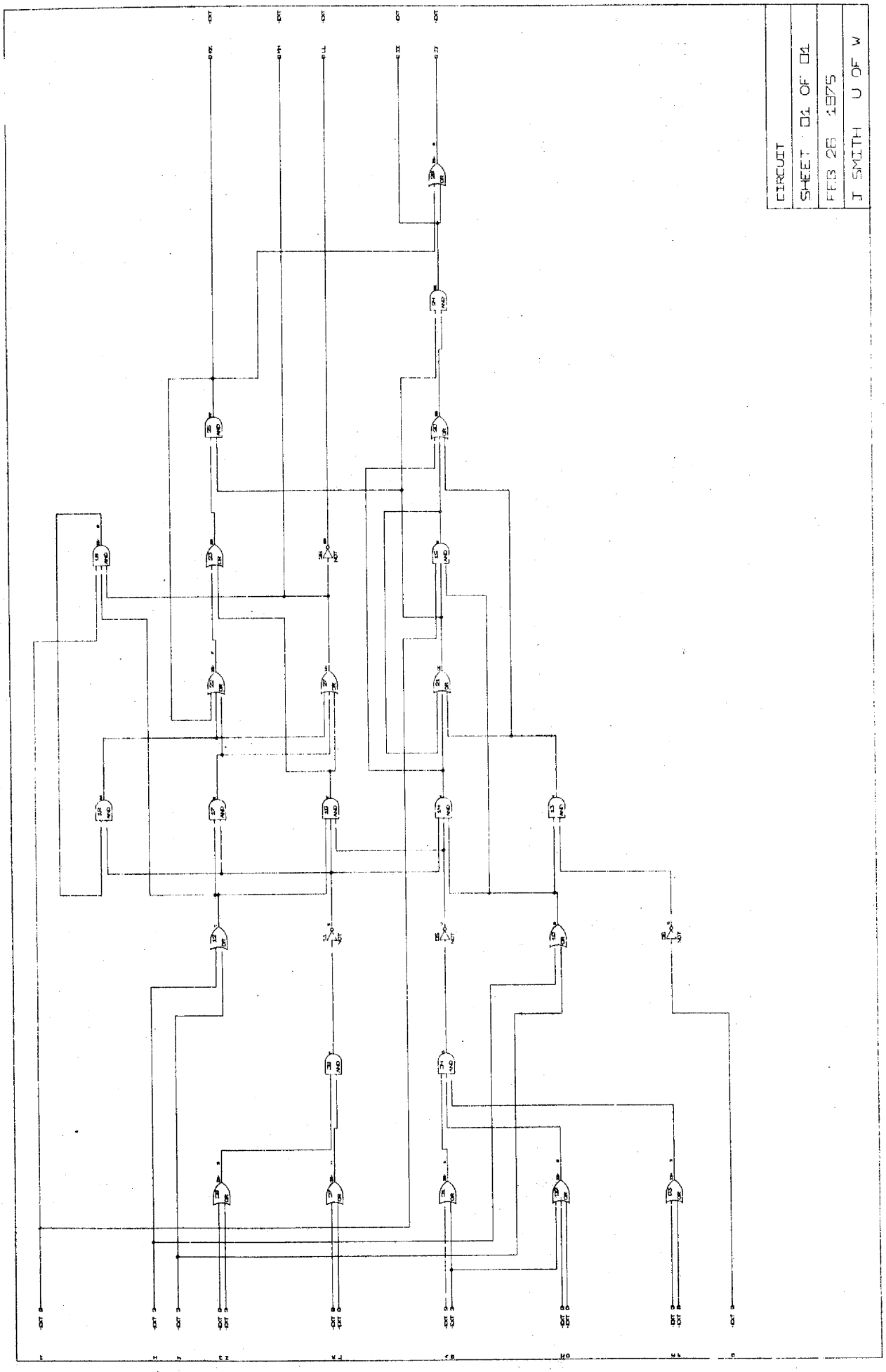


Figure 6.7 ALDGS Generated Logic Diagram for the Circuit of Figure 3.4 - Sheet 1 of 1

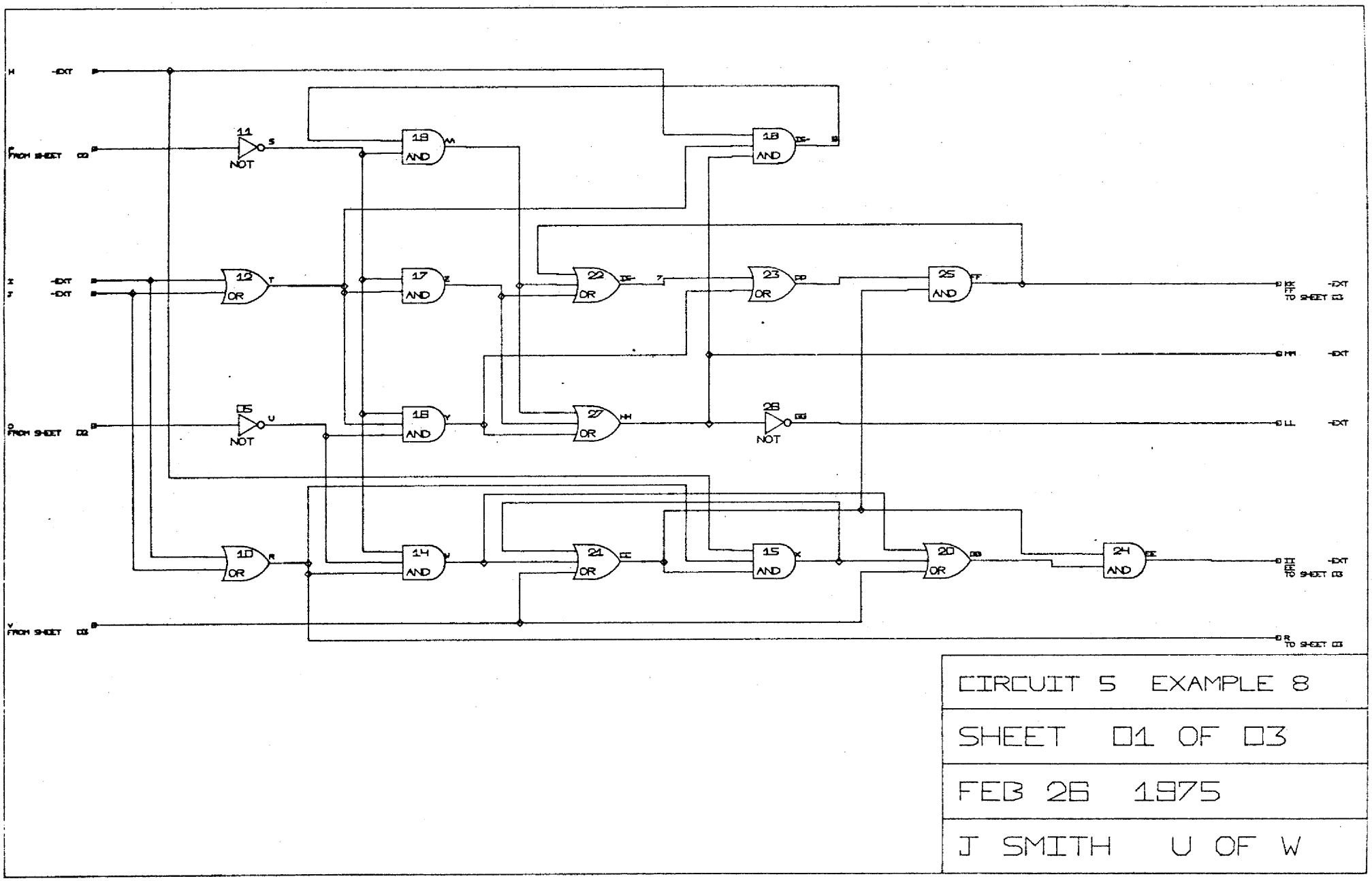


Figure 6.8a ALDGS Generated Logic Diagram for the Circuit of Figure 3.4 - Sheet 1 of 3

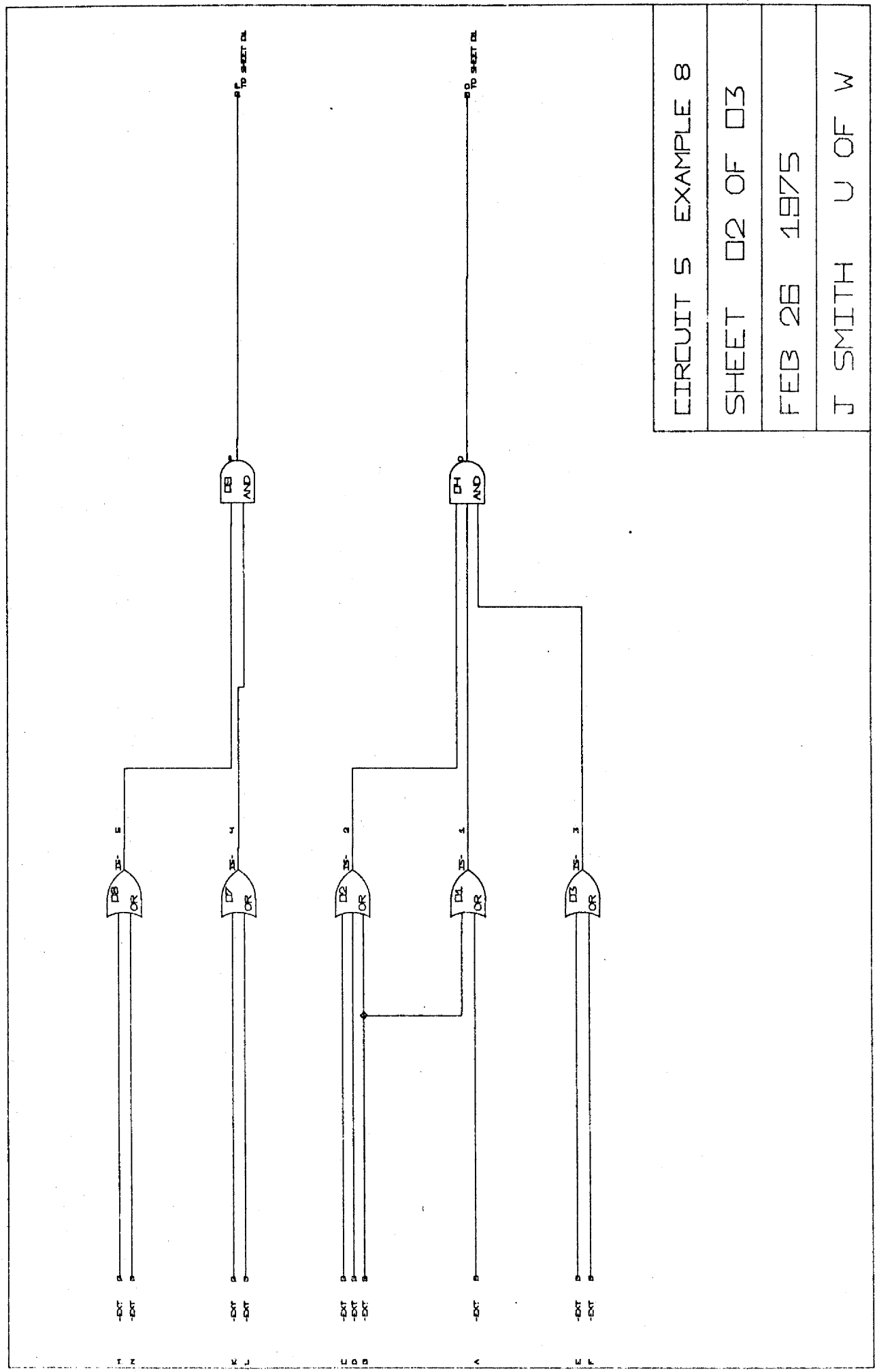


Figure 6.8b ALDGS Generated Logic Diagram for the Circuit of Figure 3.4 - Sheet 2 of 3

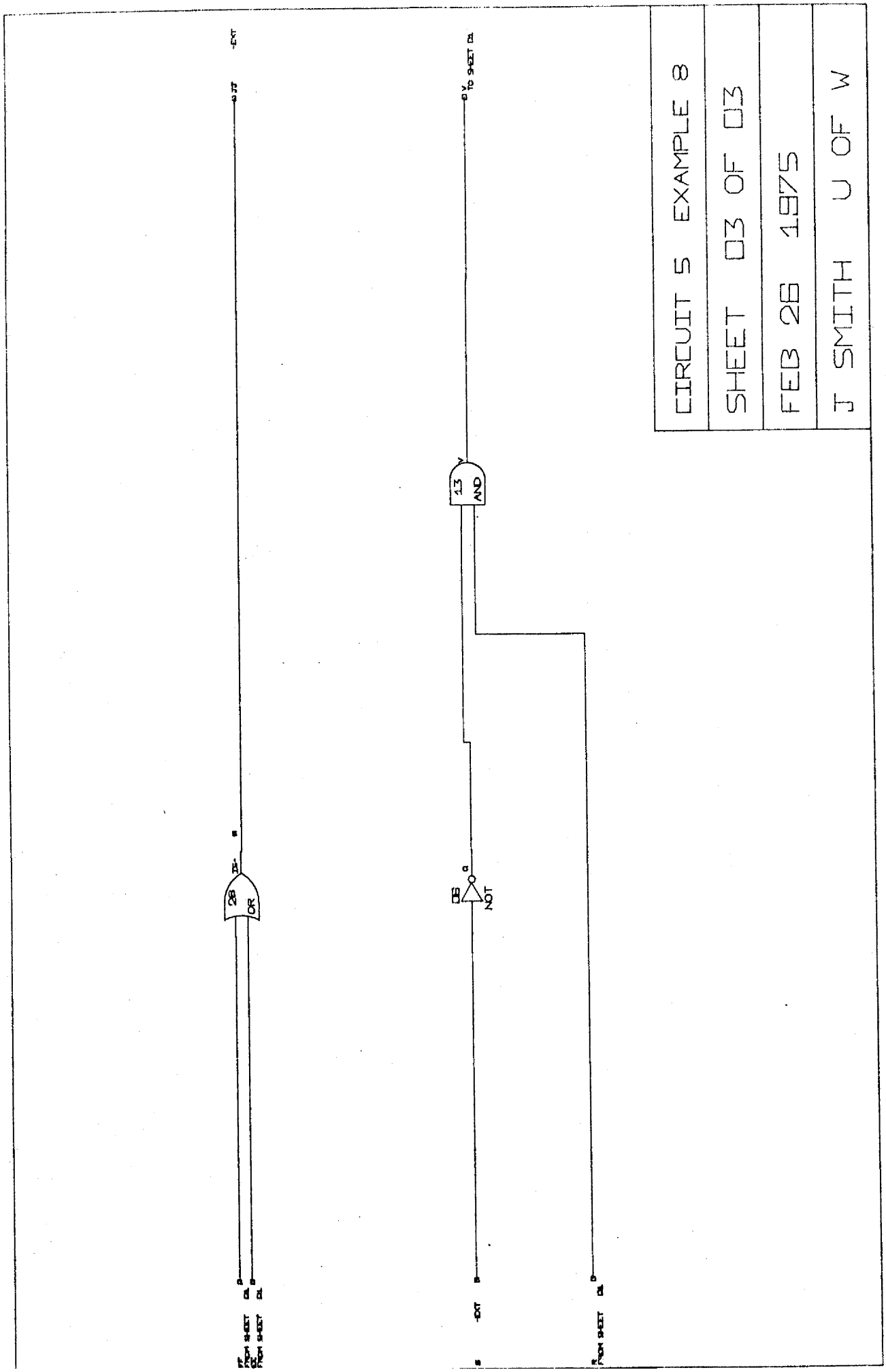


Figure 6.8c ALDGS Generated Logic Diagram for the Circuit of Figure 3.4 - Sheet 3 of 3

Figure	# of Elements Nets and Conns.	# of sheets sheet size, Comp. Size	P&P Time (sec)	Ref Time (sec)	P&A Time (sec)	R Time (sec)	GL Time (sec)	Total Time (sec)
6.1a	22,38,46	1-D,3/8"	2.4	1.2	6.4	0.4	7.8	18.2
6.1b	23,34,39	1-D,1/2"	3.0	1.2	4.9	0.3	9.0	18.4
6.2	40,73,99	2-D,3/8"	6.6	3.0	11.9	0.8	21.0	43.3
6.3	30,47,78	1-D,1/2"	4.8	1.8	6.4	1.1	15.6	29.7
6.4a	37,55,90	1-D,3/8"	4.5	2.4	17.2	0.9	20.4	45.7
6.4b	37,55,90	1-D,3/8"	manual		17.2	0.9	20.4	38.5
6.5	39,68,94	1-D,3/8"	4.8	2.4	17.4	1.0	24.0	49.6
6.6	12,27,42	1-D,1/2"	2.4	1.2	2.2	0.4	10.2	16.4
6.7	27,47,67	1-D,1/2"	3.0	1.8	4.9	0.5	17.4	27.6
6.8	28,47,71	3-B,1/2"	3.6	2.4	6.6	0.7	19.8	33.1

where: P&P Time - includes partitioning, and initial placement
Ref Time - placement refinement
P&A Time - includes accessing library to get pin information, pin assignment, boundary signal node assignment
R Time - route connections and assign segments to channels
GL Time - includes accessing the library to get geometric information and generating all the lines on disk

Table 6.1 Generation Times for the Logic Diagrams

exection time for these operations combined is approximately 1 minute per diagram.

The partitioning and placement operation requires 200 bytes per component and thus for a 1000 component circuit would need 200k bytes storage for the circuit itself and 65k bytes for the program for a total of 265k bytes. The speed of

the partitioning operation, by itself, varies according to its sophisication. The speed of the partitioning algorithm used in this thesis is, at worst, of order n**3 where n is the number of components. The placement operation itself is linear with respect to the number of components that the partitioning operation gives it.

6.3 Conclusions.

This thesis has presented a thorough investigation of the generation of logic diagrams and provides new techniques which have been combined to form an automated logic diagram generation system. The levels of representation and classifications of the logic circuit are original as are the techniques for assignment of boundary signal nodes and pins. The method of initial-placement is new while the placement-refinement concept, originally conceived for PC layout has been given new objective criteria which allow it to be used for logic diagram generation. The routing scheme with its classification and ordering of segments is also original.

The AIDGS system meets the objectives established for it in chapter 1. It is able to handle large sections of logic circuitry and different sizes of diagrams. It partitions the logic circuit according to connectivity and thus heavily connected logic will be grouped together. The placement techniques result in the logic having a flow from left to right, and feedback edges are minimized. The routing scheme is always able to route the connections since the

interconnections are not routed through the use of a fixed routing grid. The routing scheme eliminates most of the unnecessary crossings which seem to plague other systems. It can handle all types of components and can supply the required circuit information on the diagram.

Objective 1 is the most important test of the system. It concerns generation of legible diagrams at reasonable cost. The automatically produced logic diagrams, when compared with the source diagrams, are as legible if not more so. Thus if the source diagrams were usable then so must be the new diagrams. The time required to produce the diagrams was given in Table 6.1. The cost to generate the data to draw all diagrams shown was less than $21 at commerical rates. As may be seen from Table 6.1, a large percentage of this cost was in computer time to generate the lines of the diagram as opposed to the layout times. If simple blocks were used instead of the MIL-specification symbols, the line generation time would be cut tremendously and it is felt that the layout time itself could also be improved by careful tuning of the system. The plotting time for all diagrams was less than 19 minutes. In view of the time required for a draftsman to produce a similar quality diagram, these costs are reasonable.

6.4 Recommendations for Future Research.

1) The techniques described in this thesis are oriented towards the placement of components on a grid with routing spaces between each row and column. By removing

this restriction, and allowing placement of components anywhere on the sheet, it may be possible to generate even better placements. Of course, if this were done, the routing scheme would have to be revised but a similar approach should be possible.

2) Topological techniques for circuit layout use planarity techniques to obtain their layoutt. They combine placement and routing together. With modifications, the topological circuit layout methods could also be used to generate LD layouts. Since they combine partitioning and placement, they offer potential for generating better LD layouts.

6.5 Enhancements.

6.5.1 Partitioning, Placement and Refinement

1) In the current placement scheme, all components are placed in the centre of their grid position. This often results in a jog in the interconnection between two adjacent components since the input of a component is not in line with the output of its ancestor. In certain cases either the ancestor or descendant component may be shifted, still within the zone which does not move, to eliminate the jog (e.g. logic diagram 1, components 08 and 13).

---

† Vancleemput,W.M - An Improved Theoretic Model for the Circuit Layout Problem, 11th Design Automation Workshop 1974, pp.82-90

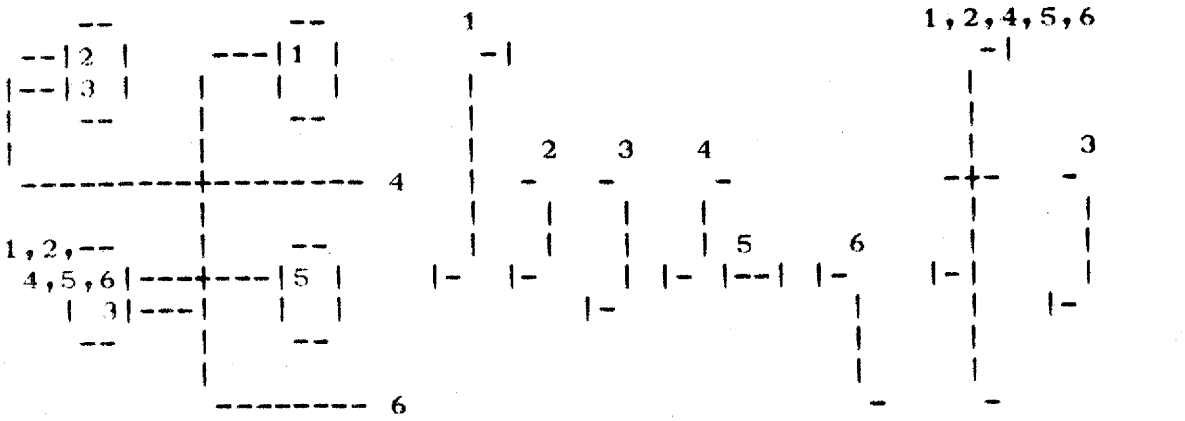
2) In some logic diagrams a component row becomes occupied by row disjoint sections of logic. If in these cases the maximum number of component rows is not used, then these row-disjoint components could be separated onto adjacent rows.

6.5.2 Routing.

1) In some cases the routing spaces become congested due to the number of connections using that space. This congestion could be reduced if the width of the routing spaces was allowed to vary according to the maximum number of channels in use. Thus, in logic diagram 6 - sheet 2, the third horizontal routing space would be allocated a larger width at the expense of the other sparsely used horizontal spaces.

2) As implemented, a path type was chosen for each connection of a net on the basis of the positions of the connections source and sinks. If the path types would have been assigned considering the nets as a whole, then some parallel runs could be reduced. This can be accomplished by using an algorithm to find an almost-optimal Steiner tree [HAKI71]. No change in the segments would be required.

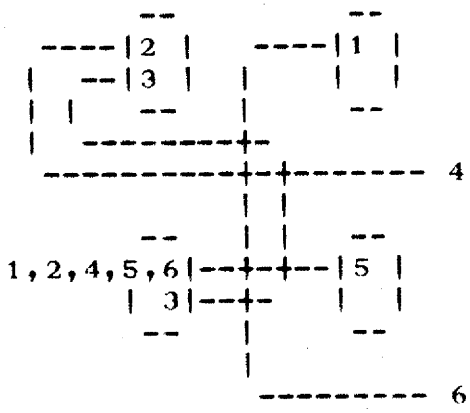
3) In certain cases, when assigning segments to channels, it is possible to reduce crossings further. This situation arises when there are many X relations



a) Segments Assigned to Centres of Spaces

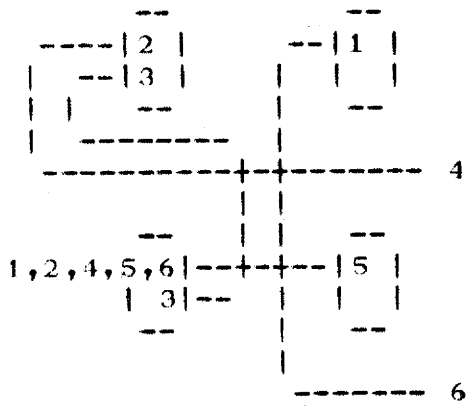
b) Simple Segments from 2nd Space

c) Collapsed Segments



d) Segment Ordering Graph

e) Assignment According to Segment Ordering Graph (4 crossings)



f) Better Assignment (2 crossings)

Figure 6.9 Further Reduction of Crossings With Compound Segments

between two compound segments. The example in Figure 6.9 is an extreme case demonstrating this situation. The segment ordering graph indicates that the compound segment 1,2,4,5,6 should be placed to the left of segment 3. This assignment, shown in Figure 6.9e generates 4 intersections. If the segments were assigned in the reverse order only 2 intersections are caused as shown in Figure 6.9f. Such cases may prove difficult to detect.

A simpler version of the above situation is easily solved. This is the case when simple segment 2 does not exist. Then there are no ordering relations between the compound segments and the number of intersections between them could be used to determine which was assigned first.

Appendix A

- Classifications and Mappings of the Signal Nodes of the Logic Circuit

- Summary of Sheet Graph Signal Node Types

A.1 Classification of the Signal Nodes of the Representative Graph.

In order to be able to perform operations on the representative graph, it is possible to classify the signal nodes as shown below.

Type	Classification	Mode
1	terminal input	A,R
2	terminal output	A,R
3	internal input	R
4	internal output	R
5	internal	A

In the table, the signal nodes are given a type number from 1 to 5 and a corresponding description. Also included is a mode for each type which is either A-for absolute or R-for relative. The mode of a signal node may be different depending upon how it is viewed, i.e. relative to a given component or absolutely with respect to the given circuit. This distinction is necessary for the transformations.

A.2 Classification of Signal Nodes of the Reduced Graph.

The following table summarizes the types of signal nodes of the reduced graph.

Type	Classification	Mode
1	external input	A,R
2	external output	A,R
3	dual external	A

4	internal input	R
5	internal output	R
6	internal	A

A.3 Mapping of Signal Nodes into Reduced Graph Classes.

The classification of the signal nodes for the reduced graph may be achieved in a two step process as outlined below.

a) The following table will give the absolute signal node classification for signal nodes on the reduced graph. The type 2 signal nodes are terminal output signal nodes which are deleted.

Absolute Signal Node Status on Representative Graph	A	B	C
1	1	1	1
2	-	-	-
5	2	3	6

where the column codes are:

A - if signal node is input to output terminal component only,
B - if signal node is input to output terminal component node and input to a non-terminal component node,
C - if signal node is input and output to/of non-terminal component nodes only.

b) Once each signal node has been classified absolutely for the reduced graph this table may be used to get

the relative classifications for the signal nodes.

Absolute Signal Node Status on Reduced Graph

	A	B
1	1	-
2	-	2
3	1	2
6	4	5

if signal node is: A - type 3 on representative graph (internal input)
B - type 4 on representative graph (internal output).

A.4. Classification of Signal Nodes of the Sheet Graphs.

Type	Description	Mode
1	external input	A,R
2	external output	A,R
3	dual external	A
4	internal input	R
5	internal output	R
6	internal	A
7	intersheet input	A,R
8	intersheet output	A,R
9	dual intersheet	A
10	external input to many sheets	A
11	external intersheet input	A,R
12	external intersheet output	A,R
13	dual external intersheet	A

A.5 Mapping of Signal Nodes into Sheet Graph Classes.

The following table gives the method of classifying the signal nodes of the sheet graph. An entry of -1 in the table indicates that the condition should not occur.

Signal Node Status on Reduced Graph

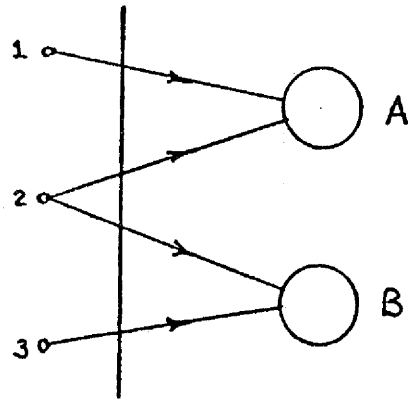
	A	B	C
1	1	10	11
2	2	-1	-1
3	3	-1	12
4	4	7	-1
5	5	8	-1
6	6	9	-1

if signal node is: A - not in intersheet connection list
B - in intersheet connection list but not dual external on reduced graph
C - in intersheet connection list and is dual external on reduced graph.

A.6 Summary of Sheet Graph Signal Node Types

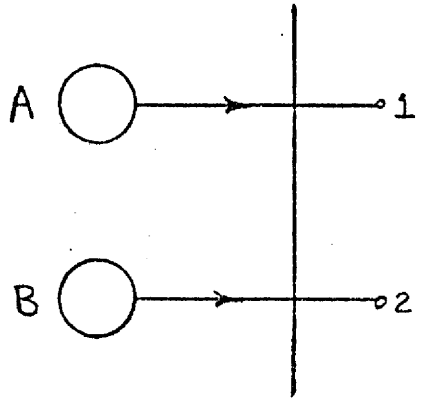
Type 1. External Input.

Signal nodes 1,2 and 3 are of absolute type 1 and are type 1 relative to component nodes A and B.



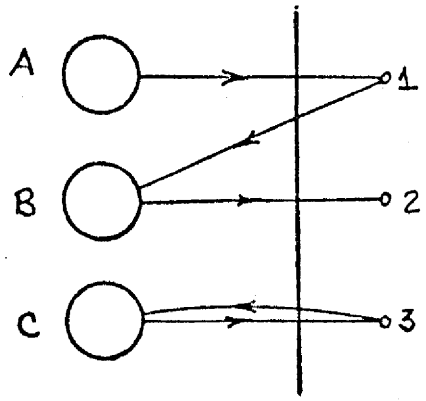
Type 2. External Output.

Signal nodes 1 and 2 are of absolute type 2 and are type 2 relative to component nodes A and B.



Type 3. Dual External.

Signal nodes 1 and 3 are of absolute type 3. Signal node 1 is type 2 relative to component node A and type 2 relative to component node B. Signal node 3 is both type 1 and 2 relatively to component node C. (Valid for component nodes which are blackboxes).

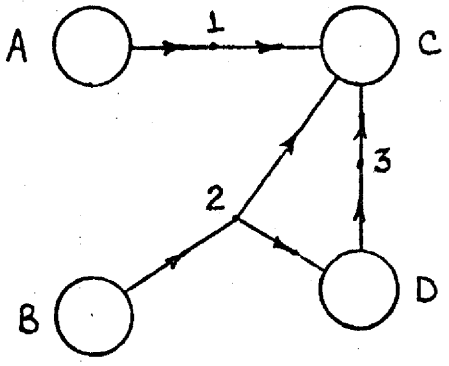


Type 4. Internal Input.
Type 5. Internal Output.

These signal node types define a signal nodes status relative to a given component node. See type 6 for clarification.

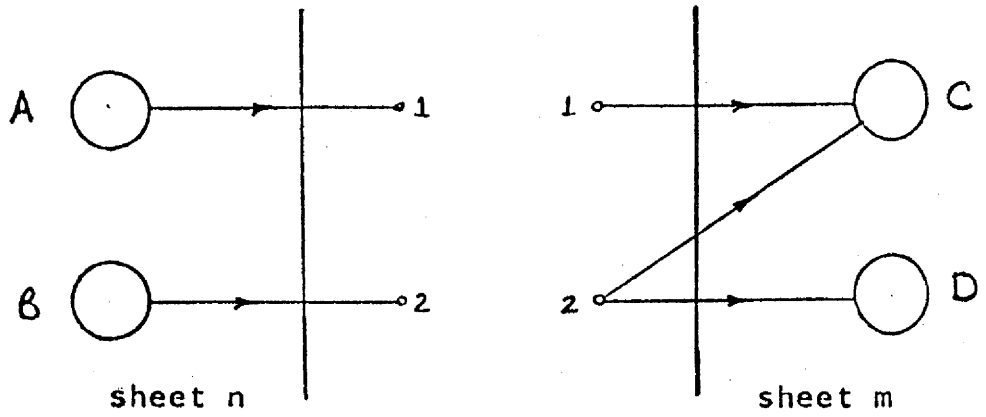
Type 6. Internal Signal Node.

Signal nodes 1,2 and 3 are of absolute type 6. Relatively signal node 2 is internal output of component node B and internal input to C and D. Signal nodes 1 and 3 are internal outputs of component nodes A and D respectively and internal inputs to component node C.



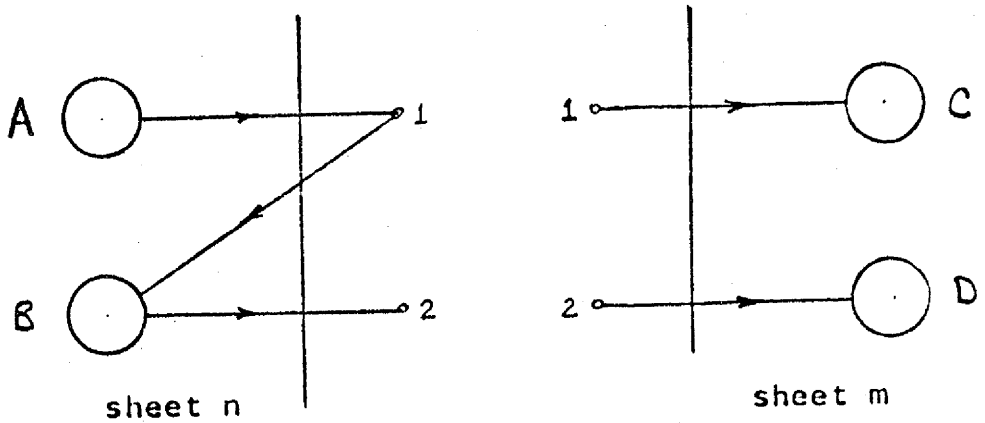
Type 7. Intersheet Input Node.
Type 8. Intersheet Output Node.

Signal nodes 1 and 2 have an absolute type of intersheet output (type 8) on sheet n and intersheet input (type 7) on sheet m. Relatively their types are the same. On the partition node graph these signal nodes would be internal.



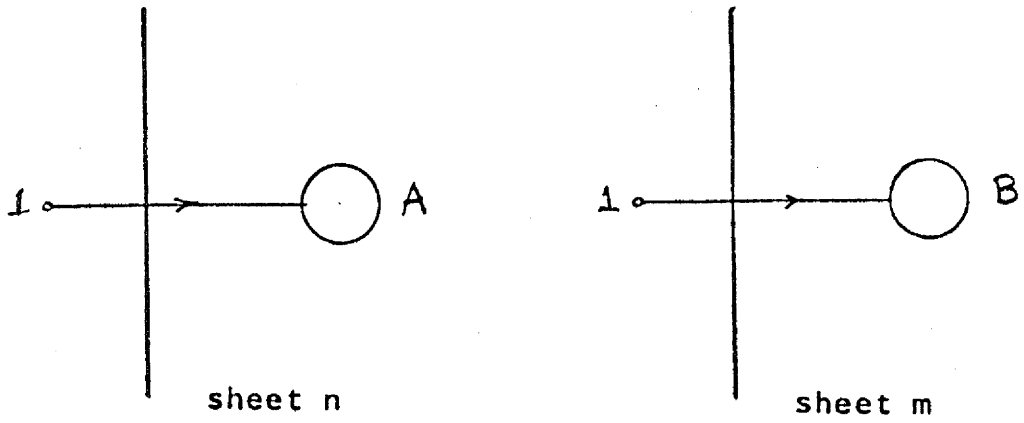
Type 9. Dual Intersheet Node.

Signal node 1 has an absolute type of dual intersheet (type 9) on sheet n and relatively is an intersheet input to component node B. On the partition node graph it would be an internal node.



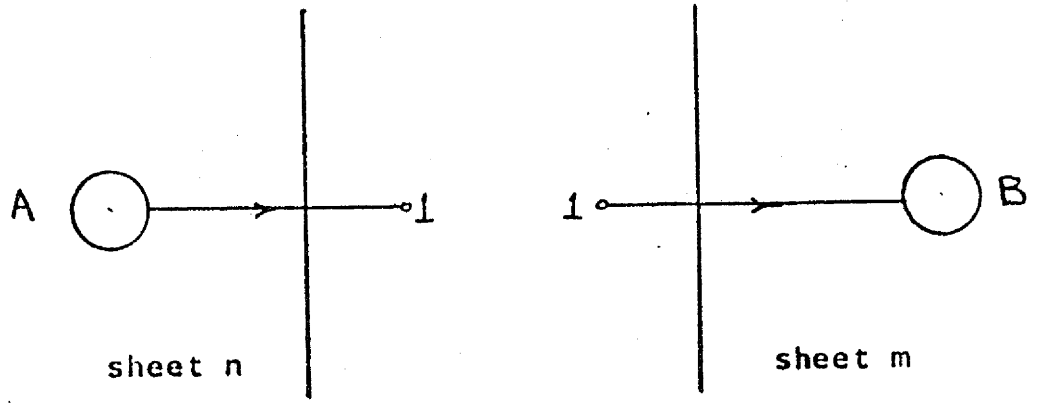
Type 10. External Input to More Than 1 Sheet.

Signal node 1 is an external input signal node to both sheets n and m. On the partition node graph it was external input to two group nodes.



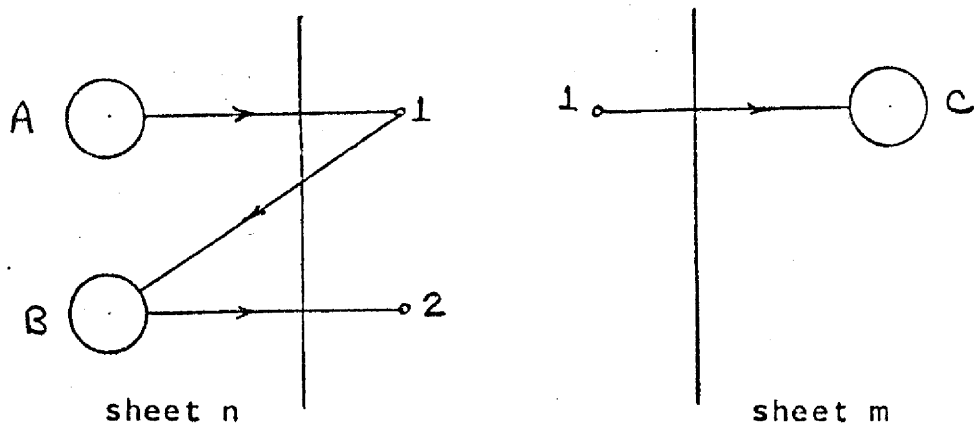
Type 11. External Intersheet Input.
Type 12. External Intersheet Output.

Signal node 1 has an absolute type of external intersheet output (type 12) on sheet n and external intersheet input (type 11) on sheet m. It also has the same types relative to component nodes A and B respectively. It would have been a dual external signal node on the partition node graph.



Type 13. Dual External Intersheet.

Signal node 1 has an absolute type of dual external intersheet on sheet n. Relatively to component nodes A and B it is an external intersheet output and an external intersheet input respectively. With respect to sheet m it is an external intersheet input. On the partition node graph this signal node was a dual external signal node which input to more than one partition node.



Appendix B

- Path and Segment Assignment Conditions

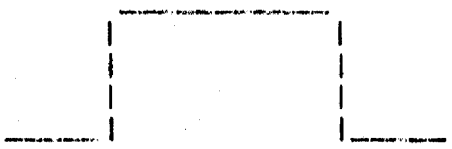
B.1 Assignment of Connections to Paths

DELX = column # of destination - column # of source

DELY = row # of destination - row # of source

Horizontal Paths

A

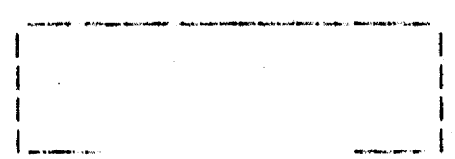


DELX > 1, DELY = 0

B

DELX > 1, DELY = 0

C



DELX <= 0, DELY = 0

D

DELX <= 0, DELY = 0

Vertical Paths

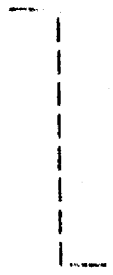
E

DELX = 1, DELY => 0

F

not assigned by end points

G



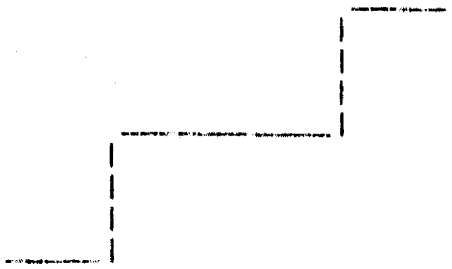
DELX = 1, DELY <= 0

H

not assigned by end points

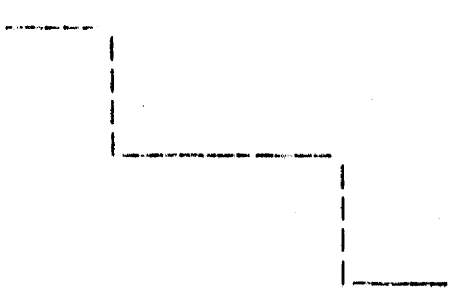
Diagonal Paths

I



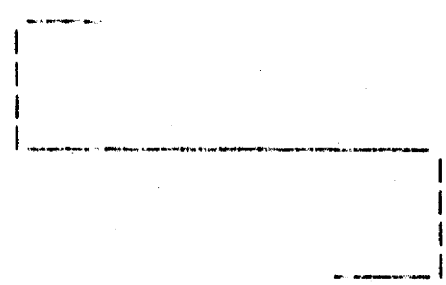
DELX > 1, DELY > 0

J



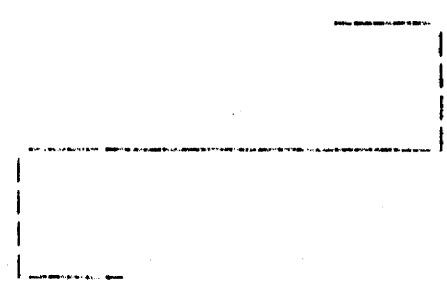
DELX > 1, DELY < 0

K



DELX <= 0, DELY > 0

L



DELX <= 0, DELY < 0

B.2 Summary of Path Assignment Conditions

The following table summarizes the conditions for path assignment which were given in section A.1.

		DELY		
		<0	=0	>0
DELX	<1	L	C,D	K
	=1	G	G	E
	>1	J	A,B	I

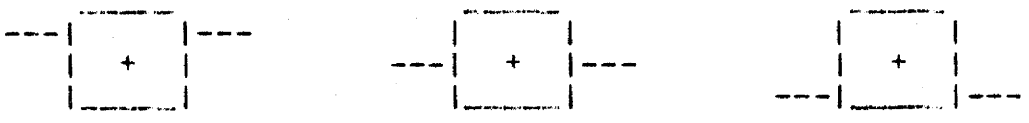
path types

The tables below provide the further distinction necessary for assignment of paths A,B and C,D. When they are in the same row (DELY = 0).

DISPY = the Y displacement between the pin and the centre of the component (Y of pin - Y of centre)

SDISPY = the Y displacement of source pin

DDISPY = the Y displacement of destination pin



DISPY > 0 DISPY = 0 DISPY < 0

Type A,B

		DDISPY		
		<0	=0	>0
SDISPY	<0	B	B	B
	=0	B	A	A
	>0	A	A	A

path types

Type C,D

		DDISPY		
		<0	=0	>0
SDISPY	<0	D	D	D
	=0	D	C	C
	>0	C	C	C

path types

B.3 Segments Composing Paths.

The path are divided into either 1 or 3 segments. The following table summarizes the segments which compose each path.

		segment number in path		
		1	2	3
	A	F	B	I
	B	J	A	G
	C	D	B	B
	D	C	A	A
	E	E		
	F	F	D	G
path type	G	H		
	H	J	C	I
	I	F	D	G
	J	J	C	I
	K	D	C	A
	L	C	D	B

segment type

Segment numbers 1 and 3 are vertical segments and segment number 2 is a horizontal segment.

Appendix C

- Segment Intersection Tables

C.0 Introduction.

There are two sets of intersection tables, one for vertical segments and the other for horizontal segments. The vertical tables are concerned with ordering vertical segments from left to right. The horizontal tables are for ordering horizontal segments from bottom to top. In the tables, the assumption is made that the segments are ordered according to their maximum end points and for each segment that Sn > En. The elements of the vertical tables contain codes which have the interpretations listed below.

1. L - segment 1 will not intersect with segment 2 if segment 1 is assigned a channel to the left of segment 2

2. R - segment 1 will not intersect with segment 2 if segment 1 is assigned a channel to the right of segment 2

3. - - the segments have the same non-source, non-sink end points and are not compatible; they can be assigned in either order

4. P - segments have same source and branch in opposite directions; they can be assigned in either order

5. O - segments have same source and overlap; they can be assigned in either order

6. OL - segments have same source and overlap, segment 1 is assigned a channel to the left of segment 2

7. OR - segments have same source and overlap, segment 1 is assigned a channel to the right of segment 2

8. E - error in drawing will occur irregardless of channel assignment

9. X - intersection will occur irregardless of channel assignment, no assignment order is specified

10. XL - intersection will occur irregardless of channel assignment, segment 1 must be assigned to the left of segment 2 to avoid a conflict

11. XR - intersection will occur irregardless of channel assignment, segment 1 must be assigned to the right of segment 2 to avoid a conflict

12. / - if this condition occurs the segments are in error

The elements of the horizontal tables constitute a subset of the above codes except that 'B' and 'T' refer to Bottom and Top instead of right and left.

In the case of the -, P, and X codes, the order of assignment is not really important and segment 1 is

arbitrarily assigned to the left of segment 2. The E code signifies that the segments will cause a conflict situation if either left or right assignment is performed. With the XL and XR codes the order of assignment is imparative since a conflict may result. They are used when the two segments have a common value for their fixed end points.

C.1 Vertical Segments.

Case V1: S1 > S2
E1 > S2 no intersections (segments are compatable)

Case V2: S1 > S2, E1 = S2

Edge 1 \ Edge 2	A	B	C	D	E	F	G	H	I	J
A	/	-	/	-	/	-	/	/	-	/
B	/	/	/	/	/	/	/	R	/	R
C	/	-	/	-	/	-	/	/	-	/
D	L	/	P	/	L	/	L	P	/	P
E	L	/	P	/	L	/	L	P	/	P
F	L	/	P	/	L	/	L	P	/	P
G	/	-	/	-	/	-	/	/	-	/
H	/	/	R	/	/	/	/	R	/	R
I	/	/	R	/	/	/	/	R	/	R
J	/	-	/	-	/	-	/	/	-	/

Case V3: S1 > S2, E1 > E2

Edge 2

Edge 1	A	B	C	D	E	F	G	H	I	J
A	X	X	R	R	X	X	X	R	R	R
B	X	X	R	R	X	X	X	R	R	R
C	L	L	X	X	L	L	L	X	X	X
D	L	L	X	X	L	L	L	X	X	X
E	L	L	X	X	L	L	L	X	X	X
F	L	L	X	X	L	L	L	X	X	X
G	L	L	X	X	L	L	L	X	X	X
H	X	X	R	R	X	X	X	R	R	R
I	X	X	R	R	X	X	X	R	R	R
J	X	X	R	R	X	X	X	R	R	R

Case V4: S1 > S2, E1 = E2

Edge 2

Edge 1	A	B	C	D	E	F	G	H	I	J
A	L	/	R	/	/	/	X	/	/	R
B	/	/	/	R	XR	XR	/	/	/	/
C	L	/	R	/	/	/	L	/	/	X
D	/	L	/	CR	OL	OL	/	XL	XL	/
E	/	L	/	CR	OL	OL	/	XL	XL	/
F	/	L	/	CR	OL	OL	/	XL	XL	/
G	L	/	R	/	/	/	L	/	/	X
H	/	/	/	XR	XR	XR	/	/	/	/
I	/	/	/	XR	XR	XR	/	/	/	/
J	L	/	R	/	/	/	X	/	/	R

Case V5: S1 > S2, E1 < E2

Edge 2

Edge 1	A	B	C	D	E	F	G	H	I	J
A	L	L	R	R	X	X	X	X	X	X
B	L	L	R	R	X	X	X	X	X	X
C	L	L	R	R	X	X	X	X	X	X
D	L	L	R	R	X	X	X	X	X	X
E	L	L	R	R	X	X	X	X	X	X
F	L	L	R	R	X	X	X	X	X	X
G	L	L	R	R	X	X	X	X	X	X
H	L	L	R	R	X	X	X	X	X	X
I	L	L	R	R	X	X	X	X	X	X
J	L	L	R	R	X	X	X	X	X	X

Case V6: S1 = S2, E1 > E2

Edge 2

Edge 1	A	B	C	D	E	F	G	H	I	J
A	/	/	R	/	/	/	/	R	/	R
B	/	R	/	R	/	R	/	/	R	/
C	XL	/	OL	/	XL	/	XL	OL	/	OL
D	/	L	/	L	/	L	/	/	L	/
E	/	/	XR	/	/	/	/	XR	/	XR
F	/	L	/	X	/	L	/	/	X	/
G	/	/	XR	/	/	/	/	XR	/	XR
H	XL	/	OR	/	XL	/	XL	OR	/	OR
I	/	X	/	R	/	X	/	/	R	/
J	XL	/	OR	/	XL	/	XL	OR	/	OR

Case V7: S1 = S2, E1 = E2

Edge 2

Edge 1	A	B	C	D	E	F	G	H	I	J
A	/	/	R	/	/	/	/	/	/	R
B	/	/	/	R	/	R	/	/	/	/
C	/	/	O	/	/	/	L	/	/	OL
D	/	/	/	O	/	OL	/	/	L	/
E	/	/	/	/	/	/	/	E	/	/
F	/	L	/	OR	/	O	/	/	XL	/
G	/	/	R	/	/	/	/	/	/	XR
H	/	/	/	/	E	/	/	/	/	/
I	/	/	/	R	/	XR	/	/	/	/
J	L	/	OR	/	/	/	XL	/	/	O

Case V8: S1 = S2, E1 < E2

Edge 2

Edge 1	A	B	C	D	E	F	G	H	I	J
A	/	/	R	/	/	/	/	XR	/	XR
B	/	L	/	R	/	R	/	/	X	/
C	L	/	OR	/	XL	/	XL	OL	/	OL
D	/	L	/	R	/	X	/	/	L	/
E	/	/	R	/	/	/	/	XR	/	XR
F	/	L	/	R	/	R	/	/	X	/
G	/	/	R	/	/	/	/	XR	/	XR
H	L	/	OR	/	XL	/	XL	OL	/	OL
I	/	L	/	R	/	X	/	/	L	/
J	L	/	OR	/	XL	/	XL	OL	/	OL

C.2 Horizontal Segments.

Case H1: S1 > S2
E1 > S2 no intersections (segments are compatable)

Case H2: S1 > S2
E1 = S2 should not occur

Case H3: S1 > S2, E1 > E2

Edge 1 \ Edge 2	A	B	C	D
A	X	T	T	X
B	B	X	X	B
C	X	T	T	X
D	B	X	X	B

Case H4: S1 > S2, E1 = E2

Edge 1 \ Edge 2	A	B	C	D
A	/	T	/	XT
B	B	/	XB	/
C	/	T	/	XT
D	B	/	XB	/

Case H5: S1 > S2, E1 < E2

Edge 1 \ Edge 2	A	B	C	D
A	B	T	X	X
B	B	T	X	X
C	B	T	X	X
D	B	T	X	X

Case H6: S1 = S2, E1 > E2

Edge 1 \ Edge 2	A	B	C	D
A	/	T	T	/
B	B	/	/	B
C	XB	/	/	XB
D	/	XT	XT	/

Case H7: S1 = S2, E1 = E2

Edge 1 \ Edge 2	A	B	C	D
A	/	T	/	/
B	B	/	/	/
C	/	/	/	/
D	/	/	/	/

Case H8: S1 = S2, E1 < E2

Edge 1 \ Edge 2	A	B	C	D
A	/	T	XT	/
B	B	/	/	XB
C	B	/	/	XB
D	/	T	XT	/

Appendix D

- Pseudo-boolean Input Language Descriptions for the Automatically Generated Logic Diagrams of Chapter 6

D.1a Input Description for Circuit of Figure 6.1a.

```
NEWSEG-FIGURE 6.1A
ITERM-A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q
OTERM-R,S,T,U,V,W,X
ABOX1-A2,A4,C2,C4,C6,E5,G5,J5,L5,N5
ABOX2-A6,C7
ABOX3-E7,G7,J3
ABOX5-L7
ORB2-A3,A5,G6
ORB3-L6
NORB-A7,J4
A2.I1 = A
A3.I1 = C2.O
A3.I2 = A2.O
A4.I1 = A3.O
A5.I1 = C4.O
A5.I2 = A4.O
A6.I1 = B
A6.I2 = A5.O
A7.I = A6.O
C2.I1 = C
C4.I1 = D
C6.I1 = E
C7.I1 = C6.O
C7.I2 = I
E5.I1 = E
E7.I1 = K
E7.I2 = F
E7.I3 = J
G5.I1 = H
G6.I1 = G5.O
G6.I2 = E5.O
G7.I1 = J3.O
G7.I2 = J
G7.I3 = O
J3.I1 = N
J3.I2 = M
J3.I3 = L
J4.I = J3.O
J5.I1 = J4.O
L5.I1 = P
L6.I1 = N5.O
L6.I2 = L5.O
L6.I3 = J5.O
L7.I1 = L6.O
L7.I2 = G6.O
L7.I3 = K
L7.I4 = I
L7.I5 = O
N5.I1 = Q
R = A7.O
S = C7.O
T = E7.O
```

```
U = G7.O
V = J3.O
W = J4.O
X = L7.O
```

D.1b Input Description for Circuit of Figure 6.1b.

```
NEWSEG-FIGURE 6.1B
ITERM-A,B,C,D,E,F,G,H,I,J,K
OTERM-L,M,N,O,P,Q,R
ABOX1-A1,A3,B1,B3,C3,C6,D1,D6,E1,G1,G6,H1,H6,L6,M6
ABOX2-B5
ORB2-B2,D7,E2,H2,H7,M7
ORB3-B4
A1.I1 = J
A3.I1 = C
B1.I1 = A
B2.I1 = B1.O
B2.I2 = A1.O
B3.I1 = B
B4.I1 = C3.O
B4.I2 = B3.O
B4.I3 = A3.O
B5.I1 = H
B5.I1 = B4.O
C3.I1 = D
C6.I1 = G
D1.I1 = J
D6.I1 = B5.O
D7.I1 = D6.O
D7.I1 = C6.O
E1.I1 = E
E2.I1 = E1.O
E2.I2 = D1.O
G1.I1 = J
G6.I1 = F
H1.I1 = I
H2.I1 = H1.O
H2.I2 = G1.O
H6.I1 = B5.O
H7.I1 = H6.O
H7.I2 = G6.O
L6.I1 = K
M6.I1 = J
M7.I1 = M6.O
M7.I2 = L6.O
L = D7.O
M = B2.O
N = E2.O
```

```
O = H7.O
P = H2.O
Q = B5.O
R = M7.O
```

D.2 Input Description for Circuit of Figure 6.2.

```
NEWSEG-FIGURE 6.2
ITERM-A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
ITERM-AA,BB,CC,DD,EE,FF,GG
OTERM-HH,II,JJ,KK,LL,MM,NN,OO,PP,QQ,RR,SS
ABOX1-A9,B2,B3,B5,D1,D2,D3,E1,E3,E5
ABOX2-A1,A2,A3,A5,A6,A7,A8,A10,A11,A12,A13,C1,C2,C3,C4,C5,C6,C7,C8
ABOX2-C9,C10,C11,C12
ABOX3-A4
ABOX4-B1,B6
ABOX5-B4,E2,E4,E6
A1.I1=B
A1.I2=A
A2.I1=D
A2.I2=C
A3.I1=F
A3.I2=E
A4.I1=I
A4.I2=H
A4.I3=G
A5.I1=B
A5.I2=J
A6.I1=D
A6.I2=K
A7.I1=F
A7.I2=L
A8.I1=H
A8.I2=M
A9.I1=N
A10.I1=B
A10.I2=O
A11.I1=D
A11.I2=P
A12.I1=F
A12.I2=Q
A13.I1=H
A13.I2=R
B1.I1=A4.O
B1.I2=A3.O
B1.I3=A2.O
B1.I4=A1.O
B2.I1=B1.O
B3.I1=B4.O
B4.I1=A9.O
B4.I2=A8.O
B4.I3=A7.O
```

```
B4.I4=A6.O
B4.I5=A5.O
B5.I1=B6.O
B6.I1=A13.O
B6.I2=A12.O
B6.I3=A11.O
B6.I4=A10.O
C1.I1=B
C1.I2=S
C2.I1=D
C2.I2=T
C3.I1=F
C3.I2=U
C4.I1=H
C4.I2=V
C5.I1=B
C5.I2=W
C6.I1=D
C6.I2=X
C7.I1=F
C7.I2=Y
C8.I1=H
C8.I2=Z
C9.I1=B
C9.I2=AA
C10.I1=D
C10.I2=BB
C11.I1=F
C11.I2=CC
C12.I1=H
C12.I2=DD
D1.I1=EE
D2.I1=FF
D3.I1=GG
F1.I1=E2.O
E2.I1=D1.O
E2.I2=C4.O
F2.I3=C3.O
E2.I4=C2.O
F2.I5=C1.O
E3.I1=E4.O
E4.I1=D2.O
E4.I2=C8.O
F4.I3=C7.O
E4.I4=C6.O
E4.I5=C5.O
E5.I1=E6.O
E6.I1=D3.O
E6.I2=C12.O
F6.I3=C11.O
E6.I4=C10.O
F6.I5=C9.O
HH=B1.O
I1=B2.O
JJ=B3.O
```

```
KK=B4.O
LL=B5.O
MM=B6.O
NN=E1.O
OO=E2.O
PP=E3.O
QQ=E4.O
RR=E5.O
SS=E6.O
```

D.3 Input Description for Circuit of Figure 6.3.

```
NEWSEG-FIGURE 6.3
ITERM-A,B,C,D,E,F,G,H,I,J,K,L,AA,BB,CC,DD,EE
OTERM-M,N,O,P,Q,R,S,T,U,V,W,X
AND2G-A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12
NANDN3-B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12
OR3-C1,C2,C3,C4,C5,C6
A1.I2=A
A2.I2=B
A3.I2=C
A4.I2=D
A5.I2=E
A6.I2=F
A7.I2=G
A8.I2=H
A9.I2=I
A10.I2=J
A11.I2=K
A12.I2=L
B1.I1=BB
B1.I2=A1.O
B1.I3=AA
B2.I1=EE
B2.I2=A2.O
B2.I3=CC
B3.I1=DD
B3.I2=A3.O
B3.I3=EE
B4.I1=BB
B4.I2=A4.O
B4.I3=AA
B5.I1=EE
B5.I2=A5.O
B5.I3=CC
B6.I1=BB
B6.I2=A6.O
B6.I3=AA
B7.I1=DD
B7.I2=A7.O
B7.I3=EE
B8.I1=BB
```

```
B8.I2=A8.O
B8.I3=AA
B9.I1=EE
B9.I2=A9.O
B9.I3=CC
B10.I1=AA
B10.I2=A10.O
B10.I3=DD
B11.I1=EE
B11.I2=A11.O
B11.I3=CC
B12.I1=BB
B12.I2=A12.O
B12.I3=AA
C1.I1=B6.O
C1.I2=B1.O
C1.I3=A
C2.I1=B6.O
C2.I2=B3.O
C2.I3=C
C3.I1=B6.O
C3.I2=B5.O
C3.I3=E
C4.I1=B12.O
C4.I2=B7.O
C4.I3=G
C5.I1=B12.O
C5.I2=B9.O
C5.I3=I
C6.I1=B12.O
C6.I2=B11.O
C6.I3=K
M=C1.O
N=B2.O
O=C2.O
P=B4.O
Q=C3.O
R=B6.O
S=C4.O
T=B8.O
U=C5.O
V=B10.O
W=C6.O
X=B12.O
```

D.4 Input Description for Circuit of Figure 6.4.

```
NEWSEG-FIGURE 6.4
ITERM-A,B,C,D,E,F,G,H,I,J,K,L,M,N
OTERM-O,P,Q,R
MFLOP-A1,D1,G1,J1
AMP-B2,B3,B4,B5,B6,C1,F1,F2,F3,F4,F5,F6,I1,J2
NANDN2-B1,C4,C7
NANDN3-C3,C5,C6,G4,G6,H1
NANDN4-E1,G2,G3,G5
NANDP2-C2,E2,I2,I3
WIREOR5-D2,H2
A1.I4=D
A1.I7=C
B1.I1=J2.O
B1.I2=A1.Q
B2.I1=E
B3.I1=F
B4.I1=G
B5.I1=H
B6.I1=I
C1.I1=B1.O
C2.I1=J2.O
C2.I2=B1.O
C3.I1=B2.O
C3.I2=A1.Q
C3.I3=J1.Q
C4.I1=B3.O
C4.I2=J1.Q
C5.I1=B4.O
C5.I2=D1.Q
C5.I3=G1.Q
C6.I1=B5.O
C6.I2=A1.Q
C6.I3=G1.Q
C7.I1=B6.O
C7.I2=G1.Q
D1.I1=C
D1.I3=C2.O
D1.I4=D
D1.I5=C1.O
D2.I1=C7.O
D2.I2=C6.O
D2.I3=C5.O
D2.I3=C4.O
D2.I5=C3.O
E1.I1=J2.O
E1.I2=A1.Q
E1.I3=D1.Q
E1.I4=B
E2.I1=J2.O
E2.I2=E1.O
F1.I1=E1.O
F2.I1=J
F3.I1=K
```

```
F4.I1=L
F5.I1=M
F6.I1=N
G1.I1=C
G1.I3=E2.O
G1.I4=D
G1.I5=F1.O
G2.I1=F2.O
G2.I2=D1.Q
G2.I3=G1.Q
G2.I4=J1.Q
G3.I1=F3.O
G3.I2=A1.Q
G3.I3=G1.Q
G3.I4=J1.Q
G4.I1=F4.O
G4.I2=G1.Q
G4.I3=J1.Q
G5.I1=F5.O
G5.I2=A1.Q
G5.I3=D1.Q
G5.I4=J1.Q
G6.I1=F6.O
G6.I2=D1.Q
G6.I3=J1.Q
H1.I1=G1.Q
H1.I2=F1.O
H1.I3=A
H2.I1=G6.O
H2.I2=G5.O
H2.I3=G4.O
H2.I4=G3.O
H2.I5=G2.O
I1.I1=H1.O
I2.I1=J2.O
I2.I2=H1.O
I3.I1=D2.O
I3.I2=H2.O
J1.I1=C
J1.I3=I2.O
J1.I4=D
J1.I5=I1.O
J2.I1=I3.O
O=A1.Q
P=D1.Q
Q=G1.Q
R=J1.Q
```

D.5 Input Description for Circuit of Figure 6.5.

```
NEWSEG-FIGURE 6.5
ITERM-A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q
OTERM-R,S,T,U,V,W,X,Y,Z,AA,BB,CC,EE,FF,GG,HH,II,JJ,KK,LL,MM,OO
OTERM-PP,QQ,RR,SS,TT,UU
NORP2G-C5,C27,C34,C37,C31
NORP3G-C1
NORP2-C33
NORP3-C2,C3
NORN2-C29,C36
NORN3-C4
NORN2G-C18,C19,C20,C21,C22,C23,C24,C25,C26,C30,C32,C35
WIREOR2-C39
AMP-C6,C28
JKFLOP1-C7
JKFLOP2-C8
JKFLOP3-C9
JKFLOP4-C10,C11,C12,C13,C14,C15,C16,C17
JkFLOP5-C38
C1.I2=B
C1.I3=A
C2.I1=D
C2.I1=C
C2.I3=E
C3.I1=G
C3.I2=F
C3.I3=E
C4.I1=C3.O
C4.I2=C2.O
C4.I3=C1.O
C5.I2=C4.O
C6.I=C5.O
C7.I1=I
C7.I3=C6.O
C8.I1=I
C8.I3=C6.O
C8.I4=C7.Q
C9.I1=I
C9.I2=H
C9.I3=C6.O
C9.I4=C8.O
C10.I1=I
C10.I3=C9.NQ
C11.I1=I
C11.I3=C10.NQ
C12.I1=I
C12.I3=C11.NQ
C13.I1=I
C13.I3=C12.NQ
C14.I1=I
C14.I3=C13.NQ
C15.I1=I
C15.I3=C14.NQ
C16.I1=I
```

```
C16.I3=C15.NQ
C17.I1=I
C17.I3=C16.NQ
C18.I2=C9.NQ
C19.I2=C10.NQ
C20.I2=C11.NQ
C21.I2=C12.NQ
C22.I2=C13.NQ
C23.I2=C14.NQ
C24.I2=C15.NQ
C25.I2=C16.NQ
C26.I2=C17.NQ
C27.I2=J
C28.I=K
C29.I1=C28.O
C29.I2=C27.O
C30.I2=C27.O
C31.I2=C29.O
C32.I2=C28.O
C33.I1=M
C33.I2=L
C34.I2=N
C35.I2=O
C36.I1=O
C36.I2=I
C37.I2=C36.O
C38.I1=C37.O
C38.I3=C39.O
C38.I4=C35.O
C38.I5=P
C39.I1=C34.O
C39.I2=C33.O
R=C18.O
S=C16.Q
T=C14.Q
U=C19.O
V=C15.Q
W=C20.O
X=C13.Q
Y=C21.O
Z=C6.O
AA=C22.O
BB=C12.Q
CC=C23.O
EE=C24.O
GG=C11.Q
HH=C25.O
II=C26.O
JJ=C7.Q
KK=C38.Q
LL=C10.Q
MM=C9.Q
OO=C38.NQ
PP=C31.O
QQ=C27.O
```

```
RR=C28.O
SS=C29.O
TT=C32.O
UU=C30.O
```

D.6 Input Description for Circuit of Figure 6.6.

```
NEWSEG-Figure 6.6
ITERM-PO1,PO2,PO3,PO4,FAR1,ADDL,GRND,SO1,MCLR,CLKL,PL,T1L,SHFT,SV
OTERM-FAD1,FADD,ADD,CLCK,MLRL,PO1L,PO2L,PO3L,PO4L,SO1L
NANDN2-C1,C2,C3,C4,C5,C6,C7,C8,C9,C10
NANDP3-C11
MFLOP-C12
C1.I1=SO1
C1.I2=SHFT
C2.I1=PO1
C2.I2=PL
C3.I1=CLKL
C3.I2=CLKL
C4.I1=MCLR
C4.I2=MCLR
C5.I1=PO1
C5.I2=PO1
C6.I1=SO1
C6.I2=SO1
C7.I1=ADDL
C7.I2=ADDL
C8.I1=PO2
C8.I2=PO2
C9.I1=PO3
C9.I2=PO3
C10.I1=PO4
C10.I2=PO4
C11.I1=T1L
C11.I2=C1.O
C11.I3=C2.O
C12.I1=C4.O
C12.I2=GRND
C12.I3=SV
C12.I4=FAR1
C12.I5=C3.O
C12.I6=GRND
C12.I7=SV
C12.I8=C11.O
C12.I9=SV
FAD1=C12.O2
FADD=C12.O1
PO3L=C9.O
PO4L=C10.O
PO2L=C8.O
ADD=C7.O
```

```
SO1L=C6.O
PO1L=C5.O
MLRL=C4.O
CLCK=C3.O
```

D.7 Input Description for Circuit of Figures 6.7 and 6.8.

```
NEWSEG-FIGURE 6.7 AND 6.8
ITERM-A,B,C,D,E,F,G,H,I,J,K,L,M,N
OTERM-II,JJ,KK,LL,MM
O  = (A+B) * (B+C+D) * (E + F)
U  = ¬O
Q  = ¬G
P  = (K+L) * (M+N)
R  = I+J
S  = ¬P
T  = I+J
V  = Q*R
W  = U*R*S
X  = H*R*CC
Y  = S*T*U
Z  = S*T
AA = S*T*H*HH
BB = V+W+X
CC = V+W+X
DD = Y+Z+AA+FF
EE = BB*CC
FF = CC*DD
GG = ¬HH
HH = Y+Z+AA
II = EE
JJ = EE+FF
KK = FF
LL = GG
MM = HH
```

Bibliography

[AARO61] Aaronson,D.A. and Kinnaman,C.J.
Production of Large and Variable Size Logic Block Diagrams on a High Speed Digital Computer
AIEE Fall General Meeting 1961, Paper No. CP61-116

[AKER72] Akers,S.B.
Routing
Design Automation of Digital Systems, Vol.1, Chapt 6, pp.283-333, Edited by M. A. Breuer, Publisher Prentice-Hall, 1972

[ALAI67] Alaimo,C.
A Graphics Aided Drafting System (GRAD)
4th Ann. Design Automation Workshop, June 1967, pp.4.0-4.23

[ALBF72] Albert,D.J.
Concept Communications with Design Automation Systems
9th Ann. Design Automation Workshop 1972, pp.11-14

[BALDU68] Balducci,E.G. et al
Automated Logic Implementation
Proc. of 23rd National Conference of the ACM 1968, pp.223-240

[BALDW63] Baldwin,G.L. et al
Design Automation - A Look at the Future
Communications and Electronics, AIEE Paper No.62-1205, January 1963, pp.510-512

[BERG62] Berge, C.
Theory of Graphs and Its Applications, Publisher Wiley, 1962

[BREU66] Breuer,M.A.
General Survey of Design Automation of Digital Computers
Proc. of the IEEE, Vol.54, No.12, December 1966, pp.1708-1721

[BREU69] Breuer,M.A.
Logic Synthesis
Joint Conference on Mathematical and Computer Aids to Design, 1969, pp.146-164

[BREU72a] Breuer,M.A.
Recent Developments in the Automated Design and Analysis of Digital Systems
Proc. of IEEE, Vol.60, No.1, January 1972, pp.12-27

[BREU72b] Breuer,M.A.
Recent Developments in Design Automation
Computer, May/June 1972, pp.23-35

[BREU72c] Breuer,M.A.
Design Automation of Digital Systems
Edited by M.A. Breuer, Publisher Prentice-Hall, 1972

[BROW68] Brown,J.A. et al
Design Automation and the Wrap System
5th Ann. Design Automation Workshop 1968, pp.19.0-19.29

[BROW69] Brown,R.R.
A Decade of Design Automation at Honeywell EDP
Honeywell Computer Journal, Vol.1, Winter 1969, pp.32-45

[CASE64] Case,P.W. et al
Solid Logic Design Automation
IBM Journal, Vol.8, No.2, April 1964, pp.127-140

[CASE71] Case,P.W.
An Overview of Design Automation
5th Ann. IEEE Int. Computer Group Conference, 1971, pp.61-62

[CASE72] Case,P.W.
Evolution of Design Automation
Computer, May/June 1972, pp.21-22

[CORR68] Correia,M. and Daubenmire,R.L.
The Use of Engineering Documentation in Support of a High Density Logic Test System
5th Ann. Design Automation Workshop 1968, pp.1.0-1.11

[CRAY56] Cray,S.R. and Kish,R.N.
A Progress Report on Computer Applications in Computer Design
Proc. of West Joint Computer Conference, Vol.10, 1956, pp.82-85

[CROC70] Crockett,E.D. et al
Computer-Aided System Design
Proc. of Fall Joint Computer Conference, Vol.37, 1970, pp.287-296

[DEFE62] Dept. of Defense
Graphic Symbols for Logic Diagrams
Military Standard, MIL-STD-806b, February 1962

[DEHA66] Dehaan,W.R.
The Bell Telephone Laboratories Automatic Graphic Schematic Drawing Program
3rd Ann. Design Automation Workshop 1966, pp.1-25

[DONA68] Donath,W.E.
Hardware Implementation
Proc. of Fall Joint Computer Conference, Vol.13, 1968, pp.1502

[EVEN73] Even,S.
Algorithmic Combinatorics, 1973
Publisher Macmillan

[FARL70] Farlow,C.W.
Machine Aids to the Design of Ceramic Substrates Containing Integrated Circuit Chips
7th Ann. Design Automation Workshop 1970, pp.274-285

[FORD71] Fordiani,W.O.
New Vistas for CAM
WESCON Convention Record 1971, Session 22, Paper 4, 3pp.

[FREE68] Freeman,M.F. and Resnick,M.
GENDA - A Generalized Design Automation System for Modular Hardware
5th Ann. Design Automation Workshop 1968, pp.2.0-2.21

[FRIE66] Freidman,T.D.
Alert: A Program to Produce Logic Designs from Preliminary Machine Descriptions
IBM Research Report, RC-1578, March 1966

[FRIE67] Friedman,T.D.
Alert: A Program to Compile Logic Designs of New Computers
Digest 1st Annual IEEE Computer Conference 1967, pp.128-130

[FRIE68] Friedman,T.D. and Yang,S.C.
Quality of Designs from an Automatic Logic Generator
IBM Research Report, RC-2068, April 1968

[FRIE69] Friedman,T.D. and Yang,S.C.
Methods Used in an Automatic Logic Design Generator (ALERT)
IEEE Transactions on Computers, Vol.C-18, No.7, July 1969, pp.593-614

[FRIE70a] Friedman,T.D. and Yang,S.C.
Quality of Designs from an Automatic Logic Generator (ALERT)
7th Ann. Design Automation Workshop 1970, pp.71-89

[FRIE70b] Friedman,T.D. and Liu,C.H.
The Design of a Design Language
Proc. of the National Electronics Conference, Vol.26, 1970, pp.89-93

[GAMB62] Gamblin,R.L. et al
Automatic Packaging of Minaturized Circuits
Advances in Electronic Circuit Packaging, Vol.2,
1962, pp.219-232

[GLAS69] Glaser,E.L.
Computer Aided Design for Computing Systems
Conference on Computer Science and Technology
1969, pp.13-19

[GORD60] Gordon,W.L.
Data Processing Techniques in Design Automation
Proc. of Eastern Joint Computer Conference,
Vol.18, 1960, pp.205-209

[GORM62] Gorman,D.F. and Anderson,J.P.
A Logic Design Translator
Proc. of Fall Joint Computer Conference, Vol.22,
1962, pp.251-261

[GORM67] Gorman,D.F.
Systems Level Design Automation: A Progress Report
on the System Descriptive Language (SDL II)
Digest 1st Annual IEEE Computer Conference 1967,
pp.131-134

[GORM68] Gorman,D.F.
Functional Design and Evaluation
Proc. of Fall Joint Computer Conference, Vol.33,
1968, pp.1500-1501

[GRAC73a] Gracer,F. et al
Force-Directed Pairwise Interchange (FDPI) for the Module Placement Problem and the Quadratic Assignment Problem
IBM Technical Disclosure Bulletin, Vol.16, No.3, August 1973, pp.860-861

[GRAC73b] Gracer,F. et al
Force-Directed Interchange (FDI) for the Module Placement Problem and the Quadratic Assignment Problem
IBM Technical Disclosure Bulletin, Vol.16, No.3, August 1973, pp.862-863

[GRAC73c] Gracer,F. et al
Force-Directed Relaxation (FDR) for the Module Placement Problem and the Quadratic Assignment Problem
IBM Technical Disclosure Bulletin, Vol.16, No.3, August 1973, pp.864-865

[HAKI71] Hakimi,S.L.
Steiner's Problem in Graphs and its Implications
Networks, Vol.1, No.2, pp.113-133, 1971

[HANA72] Hanan,M. and Kurtzberg,J.M.
Placement Techniques
Design Automation of Digital Systems, Vol.1, Chapt.4, pp.173-212, Edited by M.A. Breuer, Publisher Prentice-Hall

[HANN60] Hannig,W.A. and Mayes,T.L.
Impact of Automation on Digital Computer Design
Proc. of Eastern Joint Computer Conference, Vol.18, 1960, pp.211-232

[HANA72] Hanan,M. and Kurtzberg,J.M.
Placement Techniques
Design Automation of Digital Systems, Vol.1, Chapt.4, pp.213-282, Edited by M.A. Breuer, Publisher Prentice-Hall

[HARA69] Harary,F.
Graph Theory
Addison-Wesley Publishing Company, Inc

[HARV72] Harvey,J.G.
Automated Board Layout
9th Ann. Design Automation Workshop 1972, pp.264-271

[HASH71] Hashimoto,A. and Stevens,J.
Wire Routing by Optimizing Channel Assignment within Large Apertures
8th Ann. Design Automation Workshop 1971, pp.155-169

[HASP65] Haspel,C.H.
Automatic Packaging of Computer Circuitry
IEEE International Convention Record, Vol.13, Pt.3, 1965, pp.4-20

[HAYA70] Hayashi,T.
Facom 230-Series Computer Design Automation System
7th Ann. Design Automation Workshop 1970, pp.230-242

[HAYS74] Hays,G.G.
Computer Aided Schematics
11th Design Automation Workshop 1974, pp.143-148

[HERB61] Herbst,R.T. et al
Mechanization of Manufacturing Information for Digital Equipment
AIEE paper No. CP61-180

[HIGH69] Hightower,D.W.
A Solution to Line-Routing Problems on the Continuous Plane
6th Ann. Design Automation Workshop 1969, pp.1-24

[HORN69] Hornbuckle,G.D. et al
Computer-Aided Logic Design on the TX-2 Computer
6th Ann. Design Automation Workshop 1969, pp.357-369

[HOUG69] Houghton,J.
A System for the Placement of Circuit Modules
Proc. International Conference on Computer Aided Design 1969, pp.82-88

[HOUL73] Houle,J.L. and Linders,J.G.
A Formal Language for Description Modelling and Implementation of Digital Systems
Presented at the IEEE TCCA and ACM SIGARCH Workshop on Computer Description Languages, September 1973, Rutgers University, New Brunswick, New Jersey.

[HOUL74] Houle,J.L.
A Formal Language for Description Modelling and Implementation of Digital Systems
PhD. Thesis, 1974, Dept. of Applied Analysis, 1974 and Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[HOWI71] Howie,H.R. and Tavan,R.M.
OLLS: The On-Line Logical Simulation System
8th Ann. Design Automation Workshop 1971, pp.314-323

[IBM 67] IBM
Solid Logic Design Automation, Introduction to the Automated Logic Diagram Used at ESC
IBM Report No.67-579-001, September 1967

[IVER62] Iverson,K.E.
A Common Language for Hardware, Software and Applications
Proc. of Fall Joint Computer Conference,Vol.22, 1962, pp.121-129

[JACO65] Jacoby,K.
Ensuring Use of Design Automation
IEEE International Convention Record, Vol.13, Pt.3, 1965, pp.1-3

[KALI63] Kalish,H.M.
Machine Aided Preparation of Electrical Diagrams
Bell Laboratories Record, Vol.41, No.9, October 1963, pp.338-345

[KALL64] Kallas,J.L.
Computer-Aided Wiring Design
Bell Laboratories Record, Vol.42, No.11, 1964, pp.342-349

[KARP72] Karp,J.L.
Reducibility Among Combinational Problems
Proc. of the Symposium om Complexity of Computations, pp.85-103, Marhc 1972, Yorktown Heights, Edited by R.E. Millar and J.W. Thatcher, Publisher Planum Press

[KIRB61] Kirby,D.B.
Computer Program for Preparing Wiring Diagrams
Communications and Electronics, No.57, AIEE Paper No.60-1007, November 1961, pp.509-513

[KLOO58] Kloomok,M. et al
The Recording, Checking, and Printing of Logic Diagrams
Proc. of Eastern Joint Computer Conference, Vol.14, 1958, pp.108-118

[KODR61] Kodres,U.R.
Formulation and Solution of Circuit Card Design Problems Through Use of Graph Methods
Advances in Electronic Circuit Packaging, Vol.2, 1961, pp.121-152

[KODR64] Kodres,U.R. and Lippman,H.E.
SLT Board Layout
IBM Technical Report No. TR00.1010, March 1964

[KODR69] Kodres,U.R.
Logic Circuit Layout
Joint Conference on Mathematical and Computer Aids to Design, 1969, pp.165-191

[KODR72] Kodres,U.R.
Partitioning and Card Selection
Design Automation of Digital Systems, Vol.1, Chapt.4, pp.173-212, Edited by M.A. Breuer, Publisher Prentice-Hall

[KORE72] Koren,N,L.
Pin Assignment in Automated Printed Circuit Board Design
9th Ann. Design Automation Workshop 1972, pp.72-79

[KRIE71] Kriewell,T.J. and Millar,N.R.
CIBOL - An Interactive Graphics Program Used in the Design of Printed Wiring Boards and Generation of Associated Artmasters
8th Ann. Design Automation Workshop 1971, pp.304-307

[KURT68] Kurtzberg,J.M.
Computer Design Automation: What Now and What Next
Proc. of Fall Joint Computer Conference, Vol.33, 1968, pp.1499-1500

[LABA71] LaBahn,P.R.
CAM Software: Inputs Required--and Outputs Produced
WESCON Convention Record 1971, Session 22, Paper 2, 3pp.

[LASS69] Lass,S.F.
Automated Printed Circuit Routing with a Stepping Aperture
Comm. of the ACM, Vol.12, May 1969, pp.262-266

[LEE 61] Lee,C.Y.
An Algorithm for Path Connections and Its Applications
IRE Transactions on Electronic Computers, Vol.EC-10, September 1961, pp.346-365

[LEIN61] Leiner,A.L. et al
Using Computers in the Design and Maintenance of New Computers
IRE Transactions on Electronic Computers, Vol.10, December 1961, pp.680-690

[LERD64] Lerda,F. and Majorani,E.
An Algorithm for Connecting N Points with a Minimum Number of Crossngs
CALCOLO, Vol.1, 1964, pp.257-265

[LEWI69] Lewin,D.W. and Waters,M.C.
Computer Aids to Logic System Design
The Computer Bulletin, November 1969, pp.382-388

[LEWI72] Lewin,D.W. et al
Computer Assisted Logic Design - The CALD System
Proc. International Conference on Computer Aided Design 1972, pp.343-351

[LINN71] Linnemann,F.G. and Minich,C.E.
Logic Design System
8th Ann. Design Automation Workshop 1971, pp.324-346

[LYNN72] Lynn,D.K.
Computer Aided Design for Large-Scale Integrated Circuits
Computer, May/June 1972, pp.36-45

[LYON66] Lyons,D.
Concepts of Building a Design Automation System
3rd Ann. Design Automation Workshop 1966

[MAH 72] Mah,L. and Steinberg,L.
Topological Class Routing for Printed Circuit Boards
9th Ann. Design Automation Workshop 1972, pp.80-93

[MAJO64] Majorani,E.
Simplification of Lee's Algorithm for Special Problems
CALCOLO, Vol.1, 1964, pp.247-256

[MAND65] Mandell,R. and Estrin,G.
Specifications for a Design Automation System
2nd Ann. Design Automation Workshop 1965

[MAND65] Mandell,R. and Estrin,G
A Meta-Compiler as a Design Automation Tool
3rd Ann. Design Automation Workshop 1966

[MAND68] Mandell,R.L.
Tools for the Construction of Design Automation Systems
PhD. thesis, 1968, University of California, Los Angeles, Report 69-6, January 1969, University Microfilms, No.69-5335

[MARS70] Marsh,C.D.
Automation of the Design and Manufacture of a Large Digital Computer
Electronics and Power, October 1970, pp.375-379

[MILL71] Miller,D.M.
Economic Advantages of Integrating Standard DIP Hardware with Flexible Software
WESCON Convention Record 1971, Session 22, Paper 3, 3pp.

[MIKA68] Mikami,K. and Tabuchi,K
A Computer Program for Optimal Routing of Printed Circuit Conductors
Proceedings of IFIP Congress, Edinburgh 1968, pp.1495-1478

[MOOR59] Moore,E.F.
Shortest Path Through a Maze
Annals of the Computation Laboratory of Harvard University, 1959, Vol.30, pp.285-292

[MORE61] Morenti,O.J.
Implications of Machine Aids to Design
February 1961,AIEE paper No. 61-104

[PAUL64] Paul,F.G.
Design Automation Effects on the Organization
1st Ann. Design Automation Workshop 1964, pp.1-14

[PICK65] Pickrell,W.E.
An Automated Interconnect Design System
Proc. of Fall Joint Computer Conference, Vol.27, Pt. , 1965, pp.1087-1092

[PREI60] Preiss,R.J.
Design Automation Survey
IBM Technical Note, TN 00.480, 1960

[PREI61] Preiss,R.J.
Design Automation Survey: A Report to the AIEE Membership
AIEE Conference 1961, Paper No. CP61-178

[PREI62] Preiss,R.J.
An Experimental System for Logic Design Data Accumulation and Retrieval
Proc. of IFIP Congress, Vol.78, 1962, pp.678-683

[PREI72] Preiss,R.J.
Introduction to Design Automation of Digital Systems
Design Automation of Digital Systems, Chapter 1, Vol.1, pp.1-19, Edited by M.A. Breuer, Publisher Prentice-Hall

[REED72] Reed,I.S.
Symbolic Design Techniques Applied to a Generalized Computer
Computer, May/June 1972, pp.47-52

[ROCK61] Rocket,F.A.
A Systematic Method for Computer Simplification of Logic Diagrams
IRE International Convention Record 1961, Part 2, pp.217-223

[ROSE61] Rosenthal,C.W.
Computing Machine Aids to a Development Project
IRE Transactions on Electonic Computers, Vol.10 , No. , September 1961, pp.400-406

[ROSE62] Rosenthal,C.W.
Machine Aids to Design
Bell Laboratories Record, Vol.40, No. , 1962, pp.51-54

[ROTH65] Roth,J.P.
Systematic Design of Automata
Proc. of Fall Joint Computer Conference, Vol.27, Pt.1, 1965, pp.1093-1100

[RUSS68] Russo.R.L.
The Logic-to-Hardware Interface Area of Design Automation
Proc. of Fall Joint Computer Conference, Vol.33, 1968, pp.1501-1502

[RUSS72] Russo,R.L.
Design Automation
Computer, May/June 1972, pp.18-20

[SAND72] Sanderson,G. and Milici,A.
A Logic and Signal Flow Diagram Subsystem
9th Ann. Design Automation Workshop 1972, pp.250-257

[SASS70] Sass,W.H. and Krosner,S.P.
An 1130-Based Logic, Layout and Evaluation System
7th Ann. Design Automation Workshop 1970, pp.64-70

[SCHE69] Scheff,B.H. et al
The Role of a Computer Machine Aids System in the Digital Design Process
Joint Conference on Mathematical and Computer Aids to Design, 1969, pp.414

[SCHO64] Schorr,H.
Computer-Aided Design and Analysis Using a Register Transfer Language
IEEE Transactions on Electronic Computers, Vol.EC-13, 1962, pp.730-738

[SMIT64] Smith,C.F.
Graphic Data Processing
1st Ann. Design Automation Workshop, 1964

[SU 74] Su,S.Y.H.
A Survey of Computer Hardware Description Languages in the USA
Computer, Vol.7, No.12, pp.45-51, December 1974

[TARJ73] Tarjan,R.
Enumeration of the Elementary Circuits of a Directed Graph
SIAM J. Comp., Vol.2, No.3, September 1973, pp.211-216

[TIER70] Tiernan,J.C.
An Efficient Search Algorithm to Find the Elementary Cirucits of A Graph
Communications of the ACM, Vol.13, 1970, pp.722-726

[UHLI68] Uhlik,R.J.
Graphic Update of Automated Logic Diagrams
5th Ann. Design Automation Workshop 1968, pp.12.0-12.24

[VANC73] VanCleemput,W.M.
An Interactive System for Computer-Aided Design of Printed Circuit Boards
Master's Thesis, Dept. of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ontario, Canada, December 1971

[VANC73] VanCleemput,W.M.
Automated Design of Digital Systems - A Bibliography
Dept. of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Rept 74-08, May 1974

[WARB61a] Warburton,C.R.
Automation of Logic Page Printing
IBM Data Systems Division Technical Report No. 00.720, January 1961

[WARB61b] Warburton,C.R.
Automation of Logic Page Printing
AIEE Summer Meeting 1961, Paper No. CP61-726

[WARS64] Warshawsky,E.H.
Design Automation
Datamation, Vol.10, No.6, June 1964, pp.25-28

[WATE64] Waters,R.H.
A Drafting Document Generation Language
IBM Technical Report No. TR02.328, October 1964

[WAXM66] Waxman,R. et al
Automated Logic Design Techniques Applicable to Integrated Circuit Technology
Proc. of Fall Joint Computer Conference, Vol.29, 1966, pp.247-265

[WEIN72] Weinblatt,H.
A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph
Journal of the ACM, Vol.19, 1972, pp.43-56

[WISE69] Wise,D.J.K.
LIDO - An Integrated System for Computer Layout and Documentation of Digital Electronics
Proc. International Conference on Computer Aided Design, 1969, pp.72-81

[WILS74] Wilson,D.C. and Smith,R.J.
An experimental Comparision of Force Directed Placement Techniques
11th Design Automation Workshop 1974, pp.194-199

[ZELI71] Zelinger,S.H.
Documentation by Automation
Proceedings of Technical Program - National Electronic Packaging and Production Conference, 1971, pp.221

[ZELI72] Zelinger, S.H.
On-Line Interactive Graphics, The Designer's Dream or the Production Manager's Nightmare
9th Ann. Design Automation Workshop 1972, pp.35-49