The design of an interactive computer system for microelectronic mask making.

J D Eades

Thesis presented for the Degree of Doctor of Philosophy of the University of Edinburgh in the Faculty of Science, December 1976

I declare that the work described in this thesis is entirely my own except where the appropriate acknowledgements are given in the text. I also declai that the work has not previously been submitted as par of a higher degree thesis.

J D Eades

Acknowledgements

The research work for this thesis was started in the Wolfson Microelectronic Liaison Unit in the Department of Electrical Engineering of the University of Edinburgh. It was continued in the Computer Aided Design Project in the Department of Computer Science.

The initial computing facilities were provided by Systemshare Limited, an Edinburgh based time-sharing company, and were financed by the Wolfson Unit. Subsequent computer facilities were financed by the Science Research Council with grant number B/SR-8874.

My thanks are due to my supervisors for their source of constant encouragement and for their useful advice and constructive criticism. My thanks also go to the other members of both the Wolfson Unit and the CAD project for useful discussions on the projects objectives and requirements.

CONTENTS

Contents

SUMMARY

This thesis describes the development of a suite of computer programs that assist in the design and production of integrated circuit layouts. The suite is called GAELIC which is an acronym for Graphic Aided Engineering Layout of Integrated Circuits.

The purpose of the suite is to provide an efficient interactive facility for designing integrated circuit layouts that can run on a variety of computers and requires the minimum of capital expenditure. GAELIC, consequently, is the first integrated circuit design facility to work on a time-shared computer and in order to do this efficiently, the data is stored on disc using several novel features.

The first chapter introduces the problems of mask design and manufacture and why computer aids are required. The second expands on the possible computer aids and describes the presently available programs. Chapter 3 describes how GAELIC can be used to design masks and Chapter 8 evaluates its performance. Chapters 9 describes the future work that is possible using GAELIC and Chapter 10 contains the conclusions. The internal details of the program are given in Chapters 4, 5 and 7 and the factors involving the choice of graphics hardware is discussed in Chapter 6.

Chapter 1: Introduction

1.1 The reason for GAELIC

In 1969 the Department of Electrical Engineering at the University of Edinburgh were awarded a grant by the Wolfson Foundation to set up the Wolfson Microelectronics Liaison Unit. The main aim of the Wolfson Unit, as it is usually called, was to encourage industry to use the new microelectronic technologies to build their equipment instead of descrete components. In order to do this efficiently, it was essential that the unit staff were experienced in these new technologies. The Unit consequently set up a small pilot production facility for the manufacture of thin film and hybrid circuits enabling the staff to get the required experience. However, obtaining the necessary experience of the various integrated technologies was more difficult for two reasons. Firstly, it was not economic to set up a pilot production facility: secondly integrated circuits are an order of magnitude more complex than film or hybrid circuits. It was therefore decided to concentrate on the circuit and layout design and use the production facilities of existing semiconductor manufacturers to fabricate any integrated circuits designed. As it was intended to design only one or two circuits each year, special staff could not be employed to produce the artwork etc. and so the layout design had to be done by the

existing staff in addition to their normal duties. It was therefore essential to have as many computer aids as possible to speed up the design cycle. The author was employed initially by the Wolfson Unit to provide these aids and this thesis is a description of the research work involved in the development of the resultant suite of programs. The programs are known collectively as GAELIC, the name being an acronym for Graphic Aided Engineering Layout of Integrated Circuits.

1.2 The Design and Manufacture of Integrated Circuits

The starting point for the design of an integrated circuit is the specification of the system to be implemented as one or more integrated circuits and proceeds through several stages until the tested cicuits are supplied to the customer. The flow diagram for a logic system is shown in fig. 1.1 and this will be discussed in detail. A similar flow diagram exists for linear systems.

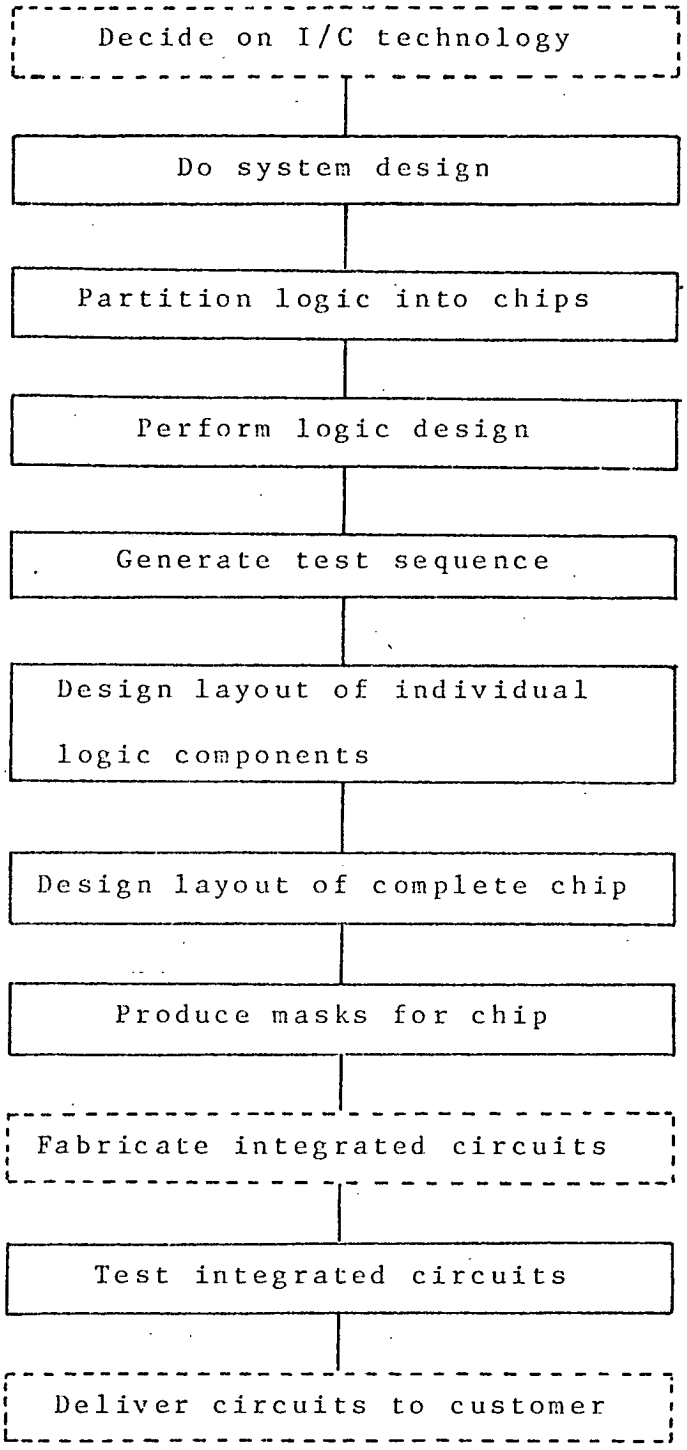


Fig. 1.1 Flow diagram for integrated circuit manufacture

The first problem is to decide on the technology to be used. Factors that have to be taken into consideration at this stage include the maximum power consumption and the speed of operation. Low power consumption will require an M.O.S. technology whereas ultra high operating speed requires a bipolar emitter coupled logic technology.

A system design is then done to decide on the number of counters and the sizes of the registers etc. before partitioning the system into subsystems. Each subsystem must be capable of being implemented on a single integrated circuit. This partitioning is a complex operation as it involves ensuring that there is a minimum number of interconnections between subsystems and that each subsystem can be tested when implemented as an integrated circuit.

Each individual subsystem is then designed in terms simpler logic elements such as gates and flip-flops and at the same time it is often convenient to design the sequence of tests to be applied to the subsystem that will ensure that it will function correctly.

The layout of each individual logic component is designed unless there are suitable designs already existing in a library. During this stage the designer must bear in mind how the components are to be interconnected and their operating speeds. These points are discussed in more detail in Chapter 2.

The layout of the complete integrated circuit is then designed by placing the components and routing all the interconnections. When a satisfactory layout has been obtained, the integrated circuit masks are produced.

The first samples of the integrated circuits are then produced and tested and if necessary modifications made to either the circuit or the layout design. These modification can mean a change of technology if, for instance, the power dissipation is too high.

1.3 Possible computer aids

The processes shown with a solid border in fig 1.1 can be helped by the use of the computer. Most of the computer aids are concerned with checking and simulation rather than with the actual design process. There are two reasons for this: firstly it is difficult to write computer programs that can simulate the creative activity of the human brain and secondly the reluctance of the designer to use any program that threatens to make him redundant.

The system design can be speeded up by using a high level logic simulator to check that the design will perform the required functions. This type of simulator does not work with individual gates but rather works with counters and registers as its basic components. They often use a register transfer language [ref 1.1] or use a

similar technique which avoids the detailed specification of the system.

The partitioning into subsystems that can be implemented as individual integrated circuits is usually done by hand. However, the algorithms of Kerninghan and Ling [ref 1.2] used by Hope [ref 1.3] to partition components for printed circuit boards could well be useful at this stage.

The logic design for a single chip can be checked by means of a gate level logic simulator such as those of Stevenson [ref 1.4] and Kaposi [ref 1.5]. These simulators work with components of the complexity of gates and flip-flops: consequently any registers etc. must be constructed from these simpler components. The simulators, however, can usually predict the delays through the gates with sufficient accuracy for race hazards to be detected. The test sequence that is generated to test the finished circuit can also be checked by modifying the logic simulation program so that it simulates faults on each gate in turn and checks that the test sequence detects them. This technique can be taken a stage further by programming the computer to try all possible input sequences and noting those that shown up the faults i.e. automatically create the test sequence. This could take a long time for combinatorial circuits and even longer for sequential circuits.

The layout design of the individual components can be helped by transient analysis programs which predict the speed of the components for various geometries and can simulate the efects of capacitive loading and of fan-in and fan-out. Unfortunately this usually turns out to be expensive in computer time.

The computer driving an interactive graphics terminal can be used as a drawing board to design the layout of all the logic components. These components can be stored in the computer and then called up, moved to the correct position and the interconnection added as the complete layout is designed. The computer can also generate the drive tapes for a tape controlled coordinatograph or any other mask making machine.

Computer programs exist which attempt to automatically place and route the logic components to produce the final layout. These have not up to now been very successful mainly due to the reasons given above but will be discussed in more detail in chapter 2.

Once the desciption of the layout is stored in a computer, it is possible to use the computer to check the design. Programs have been written that check that the layout obeys the rules issued by the integrated circuit manufacturer. This is known as layout rule checking and typical of such programs is DIMCHK [ref 1.6]. A more difficult problem to solve is ensuring the layout will perform correctly, ie. whether the components have been

correctly interconnected etc.

The computer can be used to control the test equipment that checks the finished circuits using the test sequence generated earlier and can do the obvious commercial tasks of invoicing etc.

The part of the layout cycle that could benefit most from computer aided design was the layout of the circuit and at the time that the work started very little work had been done and so it was decided to concentrate the effort in this sector.

1.4 Guide to Thesis

The thesis will be of interest to two types of reader; one who wishes to use the programs to design integrated circuits, probably an electrical engineer, and one who wishes to know more about the programming techniques used i.e. a computer scientist. The reader just wishing to use the programs need only read chapters 1, 2, 3, 8, 9 and 10 whereas the reader wishing to write similar programs will also need to read chapters 4, 5, 6 and 7.

The next chapter (2) is devoted to integrated circuit mask design and manufacture. It starts with a brief description of the various methods that have been used to make masks and explains why the tape controlled coordinatographs and pattern generators have superceeded

other methods. The actual layout design process is discussed in some detail taking as an example the layout design for an integrated circuit correlator. The various possible ways in which the computer can help in the layout design are then discussed and this is followed by descriptions of the various programs that are available and which provide these aids. The chapter ends with the derivation of the requirements for an interactive design system.

Chapter 3 is devoted to a description of the ways in which the GAELIC programs can be used in the design of integrated circuit layouts. It describes how the data is prepared, how it is checked, how it is displayed on the Tektronix screen and how it is modified. It also describes how the tapes for the various mask making machines are produced.

Chapter 4 is a general review of the data structures used in interactive graphics. It starts with an explanation of why a data structure is required, then reviews the data structures and finishes with a detailed description of the structures used in the CAMP programs and on the Marconi Myriad computer.

The next chapter (5) is devoted to the data structures used in the development of the GAELIC programs. Three data structures are described: the first is a sequential data structure which was designed for speed of implementation rather than efficiency, the second was a

'ring' data structure which held the data on disc memory and the the third was another 'ring' data structure which held the data on disc memory in a more efficient manner. The final section in the chapter describes the way the data is transfered to and from disc memory.

Chapter 6 describes the various graphic input/output devices that can be used in an interactive program. The various methods of producing pictures and interacting with them are described with their advantages and disadvantages. The choice of a storage tube terminal is explained and the chapter ends with a discussion on the software requirements to draw pictures and interact with them on this type of terminal.

The programs in the GAELIC suite are described in Chapter 7. It concentrates on the problems that were encountered and how these were overcome rather than on a detailed description of the subroutines. Chapter 8 discusses the performance of the programs. It contains the results of the various measurements that were made on the programs during the development of the program.

Chapter 9 contains a discussion of the various ways in which GAELIC can be extended or used in future work. The possibilities range from manual interaction on the output from an automatic layout program to the design of timber framed houses. The final chapter discusses whether the requirements for the programs were sound and whether they were met.

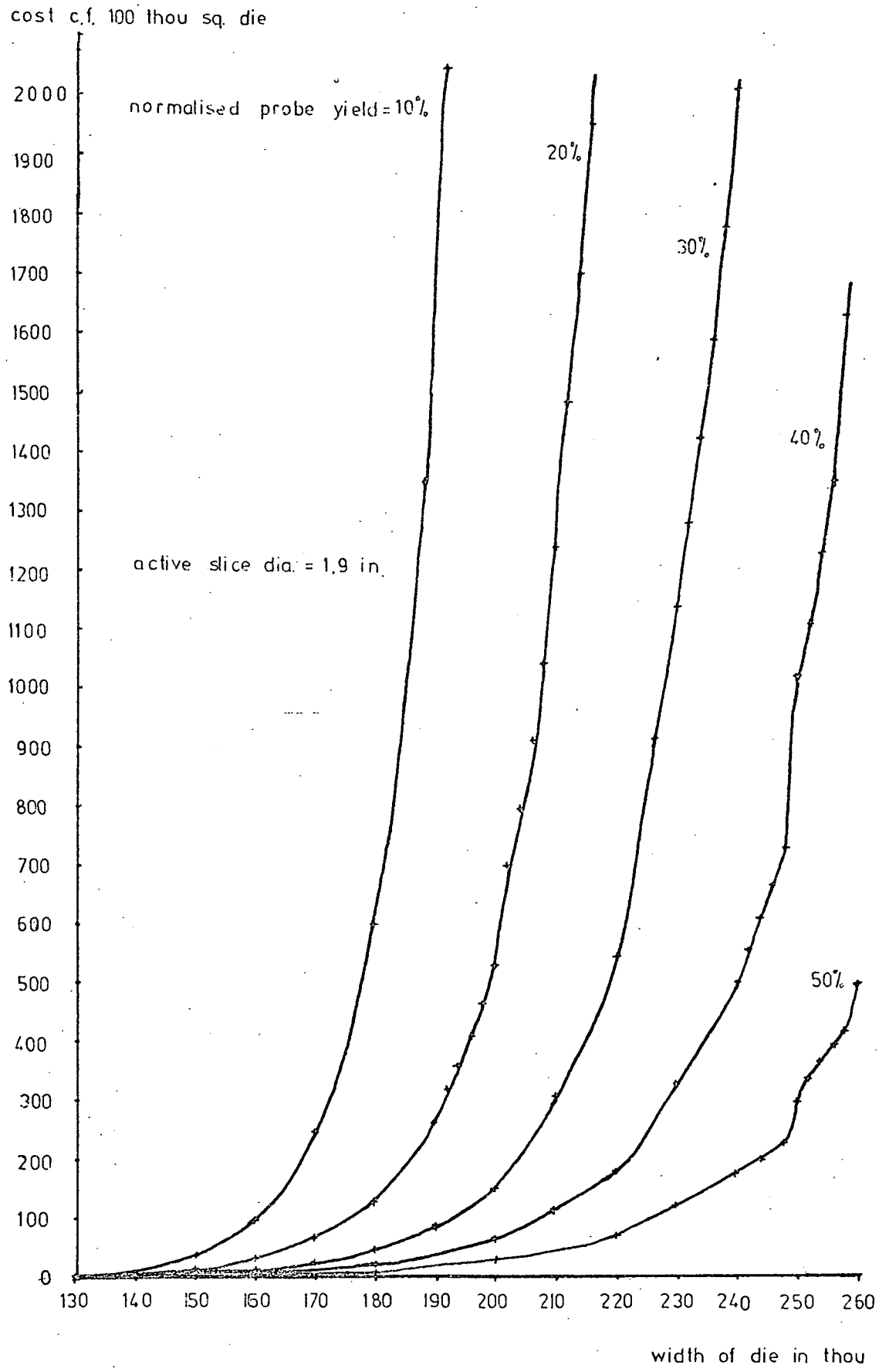


Fig 2.1. Graph of comparative cost vs die size

CHAPTER 2: Mask Making

This chapter discusses the process of designing an integrated circuit layout and then describes the development of the actual methods used to produce the mask masters and their respective advantages and disadvantages. The input data requirements to drive a tape controlled coordinatograph are then examined and ways in which the computer can assist in the reduction of the volume of data entered into the computer are discussed. Various computer systems that exploit this reduction and also assist in the actual design of the layout are described.

2.1 Manual Methods of Layout Design

The objective of the layout designer is to design a series of integrated circuit masks that will define all the components and interconnections of a circuit so that the finished device will meet its specification but nevertheless occupies the minimum area of silicon. The cost of producing an integrated circuit increases extremely rapidly with the size of the layout as can be seen in the graph shown in fig. 2.1. This shows the cost of producing a square integrated circuit die of a given size compared with the costs of producing a die of 100thou square assuming various yields levels. Further details about the graphs and the calculations on which they are based are given in Appendix 1: it will suffice here to note that the production costs double for a die that

increase from 200thou to 220thou square assuming a normalised probe yield of 30%. The effect of this increase in production costs is not always realised and consequently the main objective of the layout design is often to design the series of masks in the shortest possible time. This objective is undoubtedly extremely desirable in view of the high cost of design effort, but can obviously give a higher overall cost.

Designing an integrated circuit can be compared to solving a jig-saw puzzle, in that the shapes are moved about until all have been inserted and the total occupies the minimum space. However, there is one important difference: in a jig-saw, when the last shape has been correctly entered, a complete recognisable picture is obtained. In contrast, in an integrated circuit layout the designer can never be certain that he has the optimum solution, as there are always changes that can be made which could possibly improve the layout. On the other hand changes may have the reverse effect, and in any case may take weeks of work to put to the test.

Manual layout design methods vary from semiconductor manufacturer to manufacturer and even from designer to designer and so any description of the method used can only hope to be a concensus of the various methods. In turn the concensus will inevitably be biased towards the author's ideas on how the layout should be designed.

The starting point for layout design is either a schematic diagram containing details of the components i.e. transistors, resistors etc. and their interconnections or a logic diagram containing details of the gates and other logical functions used, along with their interconnections. The designer first calculates the size of each component so that it will handle the required current and operate at the appropriate voltage at the required speed or frequency. If the circuit is to be made using a bipolar technology then at this stage it is advantageous to identify all components occupying the same isolation region. Most designers also identify all the shapes or components that are repeated in the circuit either individually or on a matrix. The geometric shapes that are required to form each component are then designed by drawing a rough sketch of the outlines of each mask in turn on squared paper. Usually the outline for each and every mask is superimposed onto the one sheet of paper and the drawing is then known as a 'composite'. An initial layout is then produced, sometimes on squared paper but more usually on plain paper. The reason for this step is to discover the positions and nature of the various crossovers to give the most compact layout. Various 'tricks of the trade' are used to avoid using specially designed crossovers. In a bipolar technology resistors are used as far as possible as this avoids use of the extra area of a specially designed crossover. It is also possible to move the collector region of a bipolar transistor away from its base region to allow an

interconnection to go between. However this latter method must be used with care, as the characteristics of the transistor are obviously modified by this change in the geometry. In the limit, this may adversly affect the performance of the circuit. With MOS technology crossovers can again make use of resistors, and also can occur over the 'P' diffusion which connects the source of one transistor to the drain of a second.

The main composite drawing is then produced on an accurately gridded paper or mylar, bearing in mind the position of the crossovers and the sizes of the components. This may sound an easy task but in actual fact it is extremely difficult, since when the shapes describing a component are drawn to scale, it may not be possible to place the crossover in the desired position. Sometimes not enough space is left between components to accommodate all metallisation tracks required.

There are certain criteria that need to be considered when designing multiphase or multiclocked MOS circuits. These and other more general criteria are best shown by the following description of the statagy adopted by Mr.R.Kelly of the Wolfson Microelectronics Liaison Unit when he designed the layout of an integrated circuit correlator.

When examining the system diagram of the correlator it was noticed that the greater part of the system was modular and involved a series of stages of the form shown

in fig. 2.2.

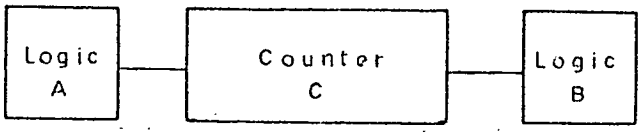


Fig. 2.2

It can be seen that each individual stage was made up of a counter 'C' preceded and followed by small sections of logic, 'A' and 'B'. The sizes of the blocks shown were roughly proportional to the estimated number of components and hence to the area of silicon required. It was therefore sensible to start the layout design by concentrating on minimising the area of silicon occupied by an individual counter.

With this aim in mind, various types of counter were considered, for example, the toggle and the feed back shift register counters, and more accurate estimates were made of the area that each type required in order to to meet the performance specification. The feed back shift register counter apparently required less silicon and so was the natural choice. The geometries of the various transistors, which would give the required power-speed tradeoff, were calculated giving an even better approximation of the area required. From this it was possible to tell that at least one stage i.e. shift register and logic, could easily be placed across the width of the chip and there was an extremely high probability of two stages being placed side by side

without the width becoming excessive. As the latter possibility was obviously extremely desirable, there was considerable incentive to design the shift register, and hence the individual bit, to have minimum width.

The layout of a previously designed shift register bit was modified to include the required preset and decode facilities and was compared with layouts designed from scratch, it was found to be superior because it obeyed certain features of good layout design. These were:

1) the various options for the bit layout were considered and the best one chosen under the circumstances. The factors affecting the choice can be understood by considering the layout of the double inverter, whose schematic diagram is shown in fig. 2.3. The principles affecting the choice of layout are the same as for the shift register bit but the schematic and layouts are easier to understand.

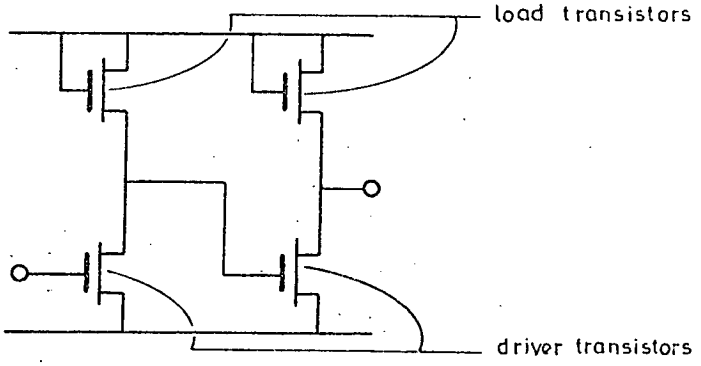


Fig. 2.3.

There are three possible layout options and these are shown in fig. 2.4.

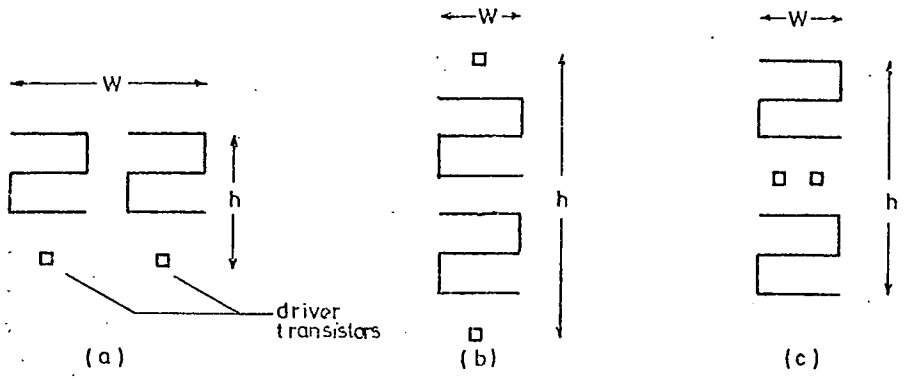


Fig 2.4

The first layout (a) basically follows the schematic diagram and as the area required for the transistor loads is the dominant parameter, this gives an extremely wide short layout for the circuit. The second layout (b) uses an extra supply rail so that the two halves of of the circuit can be stacked on top of each other. This gives a layout that is much narrower but is tall. The third layout (c) also uses an extra supply rail and makes use of the fact that the driver transistors are small and can be placed side by side to give the same minimum width as (b) but a saving in height, and so (c) is obviously the option to choose.

2) The metallisation tracks were kept as straight as possible and did not go round obstacles. The advantage of this can be seen by comparing the two diagrams in fig. 2.5. The first layout (a) shows that a dominant obstacle governs the width of an area of layout. However by taking two metallisation tracks round the obstacle then the total width is increased considerably.

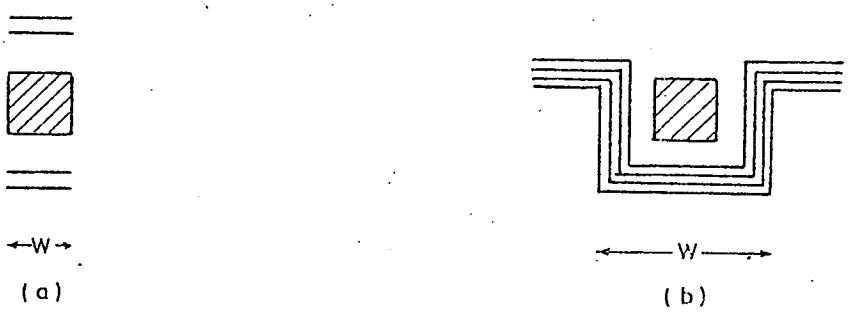


Fig. 2.5.

3) The amount of metallisation was kept to a minimum. An example of how this can be done is shown in fig. 2.6 which shows two areas of layout with the clock and logic lines running horizontally or vertically.

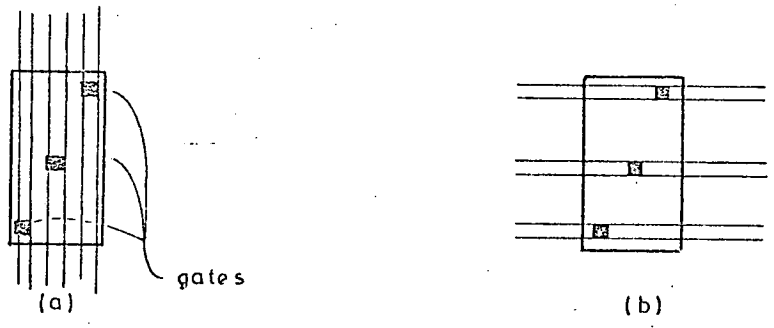


Fig. 2.6.

The metal over a gate is obviously serving the essential purpose of feeding signals to the gate. However the metal either side of the gate just occupies silicon and adds capacitance. The metallisation tracks should therefore be as short as possible between gates and, in general, this can be obtained by having metal running parallel to the shortest side as shown in fig.2.6.

The actual shift bit was clocked and so the layout required extra clock lines as well as the supply lines and the design of the metal interconnections were particularly important.

The layout of the actual shift register bit turned out to be 18.3 thou by 2.4 thou. Two stages could, therefore, be set side by side across a reasonable sized chip providing care was exercised over the design of the logic blocks A and B. The logic blocks at the end of the counter did not use the same clock lines as the counter itself and so some of the constraints were removed. However, the height of the logic became an important parameter. This was because the complete system consisted of several stages one above the other and the wasted area was dependent on the maximum height in the stage see fig. 2.7.

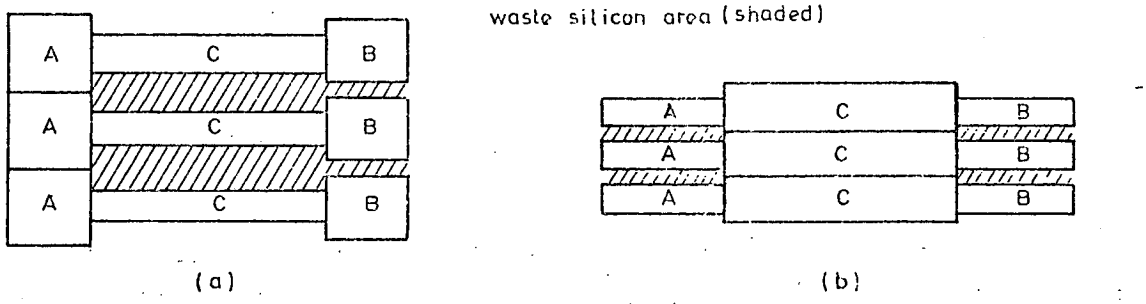


Fig. 2.7

To minimise the waste silicon, therefore, it was as important to get the height of the logic blocks to be as near as possible to 18.3 thou as it was to use the minimum area of silicon.

The design of the clock drivers brought an interesting fact to light. Because the clock driver drives so many gates its output transistor has to be bigger than the usual transistors used in the layout and the necessary calculations showed it to require a gate that was 2thou wide and 0.6thou high. The transistor also had to have a high gain and consequently low resistances. To minimise this resistance, it was necessary to have metal tracks along the source and drain diffusions and so the resultant design was that shown in fig. 2.8.

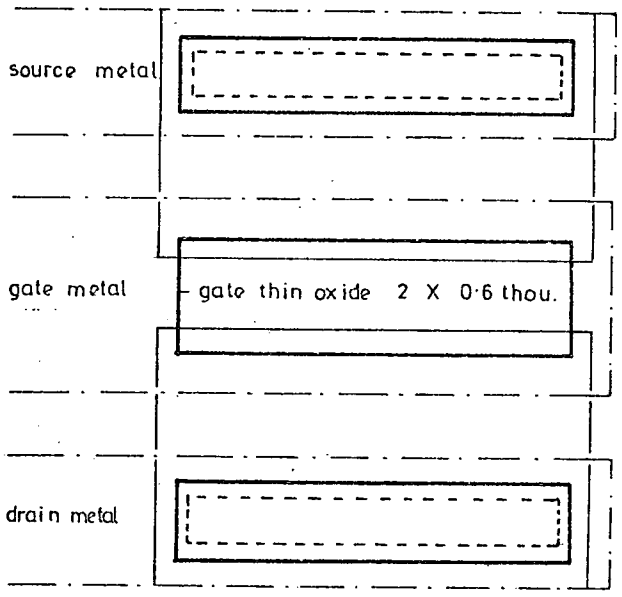


Fig. 2.8.

Here the total height of the transistor is 3thou i.e. 5 times the height of the gate. The effect of this size on the gate and diffusion capaciatances is surprising. The diffusion capacitance is 0.1pF per square thou for the MOS process used and so the drain capacitance becomes 2.2*1.3*0.1pF = 0.29pF . The thin oxide capacitance is 0.3pF per square thou i.e. three times the diffusion

capacitance, however, because of its size the gate capacitance is 0.6*2.0*0.3 = 0.36pF assuming the capacitance is as drawn i.e. ignoring the sideways diffusion. In other words the drain and source capacitance are comparable with the gate capacitance and must be taken into account in performance calculations.

In the logic block B, there is an output shift register which is clocked in the same way as the main shift register used in the counter C, and which is connected to the corresponding shift registers on the other logic blocks. It is therefore sensible to place two stages side by side so that the block B of one stage is adjacent to the block B of the second stage thus minimising the metal between them. However, if the four clock lines are sent down one column of shift registers and then up the next, then width for 8 clock lines must be allowed. To avoid this problem, the shift register stages were interdigitated and consequently only 4 clock lines are required and this gave a considerable saving in area as shown in fig. 2.9.

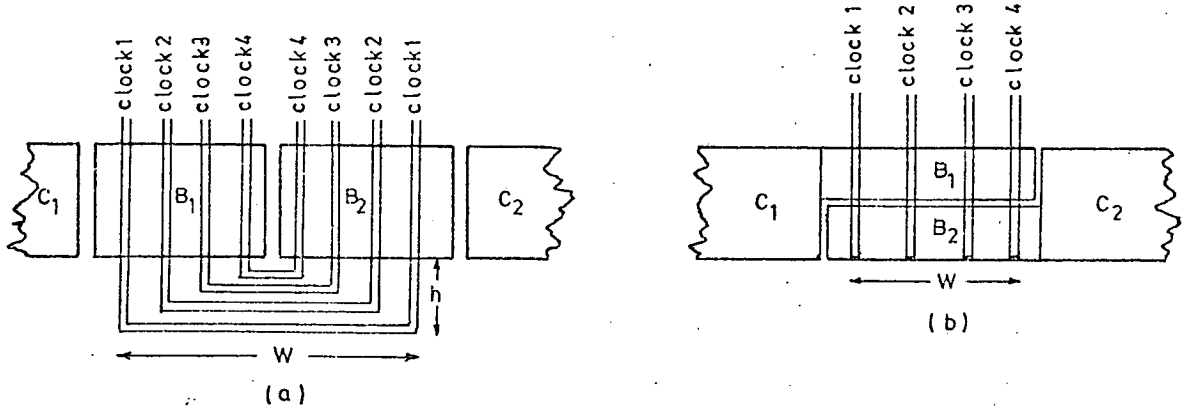


Fig. 2.9

This technique did, however, have some minor problems as the sizes of the transistors had to be increased for shift registers adjacent to the output driver stage and the actual layout of the final shift register stages became slightly cramped.

2.2 Methods of Producing Mask Masters

The early mask masters were produced on large sheets of gridded blockboard, by drawing the outline of each shape on each mask in turn on individual sheets of board and then filling in the individual outlines with black paint. These masters are finally photo-reduced to produce the actual masks. This method was reasonably successful for the early integrated circuits, which consisted of perhaps a couple of transistors and a few resistors. The method did however have several problems associated with it. Each mask had to be designed in isolation, which is not only a difficult task, but is also one that is very prone to error. Integrated circuit components are formed by the diffusion of specific substances into selected areas of a slice of silicon: the selected areas defined by one mask must align extremely accurately with the corresponding areas defined by another. Accurate alignment can best be achieved by designing all the masks at one and the same time. The second problem was that it was difficult to modify the designs, as the black paint could not easily be removed. The last problem was one

that we shall meet again and again; that is the difficulty of obtaining accurate grids.

As integrated circuits became larger the designs naturally became more complex and the limitations caused the method to be superseded by new methods. The first of these was known as 'taping' and basically consisted of using black tapes of different widths to define the shapes. Narrow tapes were used to define the outline of the large shapes, and then the interiors were filled in with wider tapes. This method had the advantage that the narrow tapes could define shapes with curved outlines in addition to the normal rectilinear outlines. The gridded block board was originally used for the layout and this new method had the major advantage that the tapes could be moved after initial placement. The designer, therefore, had the ability to correct or modify the layout. Unfortunately there remained the problem of designing the individual masks in isolation as the mask masters still had to be produced individually. There was also the problem of obtaining accurate grids and as the size of the layouts increased, there was the additional problem of obtaining gridded blockboard in sufficiently large sheets. There was a minor problem in moving or removing tapes as they tended to remove the surface of the blockboard at the same time.

To overcome this last problem, gridded mylar sheets were introduced instead of blockboard. In addition to being able to move the tapes easily, there were other less obvious advantages. The mylar was more stable dimensionally and so the grid was more accurate and the mylar sheets could be obtained in either translucent or transparent form. The translucent mylar had a sandblasted surface which could be drawn on using pencil or ink and thus the outlines of all the shapes could be drawn on the same sheet, different line textures or colours being used to distinguish between the various masks. Thus it was possible to design all the masks simultaneously, thus overcoming one of the earlier problems. A sheet of transparent gridded mylar was then placed on top of this composite drawing and the shapes for one mask taped using the grid on the transparent mylar to give the required accuracy. This was repeated for each mask in turn and gave an extremely fast and efficient method of producing the 'mask masters' with an easy system for modifying the designs. It also helped with the problem of checking the masks as two mask masters could be superimposed and the clearances and overlaps checked. Of course when it came to checking overlaps the overlaping mask had to be placed underneath and so a logical order of checking could not be maintained. However, the only time this became troublesome was when checking that the metallisation correctly covered the contact holes. There were of course certain disadvantages:

1) the width of the black tape tended to vary from roll

to roll,

2) the width of the tape varied with the amount by which it was stretched, and

3) the tape tended to creep after placing on the mylar.

The last two disadvantages are somewhat related: to obtain a long thin track for example a metal interconnection between two components, the tape is stretched before placing on the mylar to ensure that it is straight. Because of the elastic properties of the tape its length is increased and its width is reduced slightly and after placement the tape tries to return to its original shape. The tape has the elastic properties because it is designed to be laid in smooth curves as well as in straight lines.

The method, with minor modifications is still used by certain integrated circuit manufacturers who use only paraxial shapes with an MOS technology. Instead of using the black flexible tape, they use rigid mylar tapes that can be obtained in a range of accurate widths and in a range of colours. Different colours are used for the different masks. Providing a suitable colour is used for each mask they can be superimposed on the same sheet of mylar. This gives an extremely flexible method of mask design. For simple masks it is theoretically possible, given the right choice of colours, to photograph the composite directly again and again using different coloured filters to produce the actual masks. As far as is known this is not done in practice, and it is more

usual to digitise directly from this composite. This method is not suitable for bipolar circuits when nine or more different masks are required as:

1) it is not possible to obtain such a large range of colours, and

2) superimposition of nine or more layers gives parallax errors and the composite becomes fragile, losing the top layers of tape extremely easily. Most designers using the coloured tapes regard the technique as a way of designing layouts rather than as a method of producing the mask themselves.

Before the more accurate mylar tapes were available, there was a demand for a higher accuracy than the taping method was capable of producing and attention turned to manual coordinatographs. The manual coordinatograph consists of a table, typically four feet square, on which is mounted a gantry. This is constrained to move in one direction only, say the y direction. On it is mounted a tool holder, which is constrained to move along the gantry axis, ie. in the x direction. The tool holder can, of course, be moved to any point on the active surface of the table. By using the locks provided to prevent either the gantry moving or the tool holder moving along it, the tool holder can be constrained to move in only the x direction or the y direction. Movement can be accurately calibrated by means of scales and vernier dials and so any tool in the holder can be positioned to an accuracy of approximately two thousandth of an inch (2 thou). Two

different methods of using the manual coordinatograph have been tried. Both start by producing an accurate composite layout on gridded paper or mylar and both produce photographic masters in 'cut-and-peel' material. The cut-and-peel material, known by various trade names such as 'Rubylith' and 'Stabiline', consists of a translucent mylar base approximately 5thou thick on top of which is is a thinner (approx. 2thou) layer of photographically opaque mylar, usually red or orange. The top layer adheres to the translucent base and in normal circumstances, the two layers do not separate. If, however, shapes are cut in the top surface, the top layer can be easily removed, thus producing a series of shapes that are translucent.

The first method of using the manual coordinatograph consisted of counting increments from the origin of the composite to each shape in turn on a given mask, moving a knife in the tool holder so that the dials register the required coordinates, and then lowering the knife and moving it to the corresponding next coordinates on the shape. The pen was then lifted and moved to the start of the next shape and the process repeated until all the outlines of all the shapes on the one mask had been cut. A new sheet of 'cut and peel' was put on the table and the shapes on the next mask similarly cut. The method should have produced extremely accurate mask masters but unfortunately it relied on an operator counting squares on the original composite drawing. Counting squares is an

extremely tedious process and is very error prone and so it not only takes a long time to cut the masks but they also contain errors that have to be found and corrected.

The second method, although theoretically less accurate due to parallax errors and inaccuracies in the mylar grid was preferred and was known as 'overcutting'. The composite drawing was fixed onto the table of the coordinatograph taking care to ensure that it was parallel to the axes and that the origins were coincident. A sheet of 'cut and peel' was then fixed on top of the composite and the knife placed over the start of the first shape of the first mask. The knife was then moved until the dials indicated that it was on the nearest increment e.g. if the smallest movement used on the drawing was 50thou then the knife position was adjusted until the dials read an exact number of 50thous. The knife was then lowered and a cut made to almost the end of the first line segment of the shape; again the knife was accurately positioned by means of the dials. This process was repeated for each line segment in turn lifting the pen at the end of each shape, until the complete mask has been cut. The 'cut and peel' material was then changed and all the shapes on the next mask cut. This process was far more efficient and appeared to have a fair amount of job satisfaction associated with it, which was obviously very important for this type of work. It had two disadvantages, firstly the size of the composite became so large that it could not fit on the table, and secondly the accuracy of the gridded

mylar. The first disadvantage should theoretically have been overcome by either using coordinatographs with bigger tables or drawing the composite at a smaller scale. The first alternative was expensive as the camera used to photographically reduce the mask master as well as the coordinatograph would have to be replaced. A secondary disadvantage was that the operator could no longer reach the entire table from the one side and so duplicate dials would have been required and the operator would have had to do more walking. The disadvantages of the second alternative were a little more obscure as they were concerned with accuracy of the grids and with job satisfaction. The draughtsman when producing the drawing would work very rapidly provided that he was working with a grid of not less than a twentieth of an inch and the girl operating the coordinatograph could easily work out which grid line was intended when she was overcutting. However, if the grid was reduced to less than a twentieth of an inch, the draughtsman found it frustrating to draw lines accurately enough to enable another person to realise which grid line was intended. Any attempt to get a higher accuracy resulted in a high error rate and a constant stream of complaints about what was previously a very satisfactory job. Again the comparative accuracy of the grid on the mylar and the built in grid on the coordinatograph gave troubles for, although near the origin it was possible to tell which grid line was intended, on the other side of the drawing the grid line on the coordinatograph often appeared between two grid

lines on the drawing. At this stage it also became apparent that the effects of temperature and particularly humidity had an appreciable effect on the accuracy and so these had to be controlled.

The answer to the problems of producing the mask masters was to use tape controlled coordinatographs and most semiconductor manufacturers have adopted this solution. The tape controlled coordinatograph essentially consists of an accurate flat table similar to that used on the manual coordinatograph with a gantry and a tool holder. The movements of the gantry and the tool holder are performed by stepping or servo controlled motors. The motors themselves are controlled by data fed to the tape controlled coordinatographs by means of paper or magnetic tape. This tape also contains information which controls the solenoid which raises or lowers the knife and the stepping motor which rotates the knife. There is obviously some logic circuitry and sometimes even a small computer built in to the coordinatograph to sort all this data on the tape and route it to the appropriate motor. Consequently the cost is far higher than for the manual coordinatograph and is in the range 20,000 to 80,000 pounds. The basic input data required by these coordinatographs consists of the coordinates of every corner of every shape on each mask in turn. For a typical integrated circuit, it requires of 200,000 pairs of coordinates i.e. 400,000 numbers. Producing this type of data by hand with an error rate 0.1% means 400 errors to

be detected and corrected. This is virtually an impossible task. It is therefore essential to find an efficient way of producing the input data. The tape controlled coordinatographs produce mask masters that are more accurate and at a smaller scale than those that can be produced on a manual coordinatograph.

The cut-and-peel masters are now unfortunately approaching their limit as the size of the completed integrated circuit chips approach 250thou (0.25 inches) square. Work has been in progress for some time on another method of producing masks known as a photo-plotter. The input data requirements are, in general, similar to those for a tape controlled coordinatograph and so the same problems exist in producing correct input data.

Fortunately there is a great deal of redundancy in this input information and if this can be exploited by the use of a computer then the amount of data required can be considerably reduced. Most shapes used in integrated circuit layouts are paraxial i.e. have all their sides parallel to the axes of the drawing and therefore it is only necessary to specify the alternate corners of these shapes. This means that a paraxial rectangle is specified by the coordinates of a pair of diagonal corners. This simple expedient reduces the amount of data that needs to be entered by approximately half. Another characteristic feature of integrated circuit layouts is the number of shapes or series of shapes that are to be found in more

than one place in the layout. This repetitive feature appears in two forms, the first is where a series of shapes are repeated on a matrix, for example a single bit of a shift register is repeated many times to create a large shift register. The second is where the same series of shapes occurs in various random positions on the layout sometimes with different orientations. It is, therefore, desirable to derive a method of inputing the data for the series of shapes once and then arranging for the series to be 'repeated' or for an instance to be 'called' in various positions at various orientations. This again makes a substantial saving in the amount of input data, as can be seen from the following example. Consider the piece of integrated circuit layout shown in fig.2.10 which has a large proportion of repeated and grouped shapes. Using a group and repeat facility, the input data consists of 2000 words whereas just using the basic shapes where all the corners are specified required 20000 words.

The input data to the computer can be arranged to exploit all this redundancy and the computer can be programmed to produce the input tape for the coordinatograph. There are certain computer programs and computer systems that perform this function and also assist the layout design process in other ways and some of these programs will now be discussed.

2.3 Possible computer aids to mask design

There are several ways in which the computer can assist in the design and production of integrated circuit masters. These range from computer programs that simply expand the condensed input data into the large volume of output tape required to drive a tape controlled coordinatograph to fully automatic programs that will produce these drive tapes from a schematic diagram of the circuit. This fully automatic method would appear to be the ultimate objective as so is worth considering in detail first of all.

2.3.1 Automatic Computer Methods

The initial work in this area was carried out by an unknown research organisation and was financed by the American government. This was a placement and routing program based on printed circuit board techniques and was released to American government contractors in about 1967. Bardsley [ref 2.1] claims this as the reason why Collins Radio, Fairchild, Motorola and Texas Instruments all announced similar systems for automatic integrated circuit design simultaneously. The programs take a series of previously defined standard components or 'cells' and places them side by side as 16 lead dual in line packages are placed on a printed circuit board. They then route all the interconnections between the packages. The programs do actually produce drive tapes for tape

controlled coordinatographs and would therefore appear to be the answer. However the amount of silicon that was required for the circuit was up to 300% more than a corresponding manual design. There were three reasons for this: firstly it relied on a set of previously designed standard cells which were not necessarily optimum for the circuit being produced, secondly the placement being completed before the routing was started means that silicon must be reserved for possible use by the interconnections and thirdly the system followed the schematic diagram too closely. A schematic diagram is drawn with inputs on the left hand side and outputs on the right and this usually creates a long thin drawing and hence a long thin integrated circuit. It was realised that integrated circuits should be square and so the the strip was folded over to give the final integrated circuit. This waisted a large amount of silicon on the fold as shown in fig 2.10. The increase quoted for the chip size [B.R. Kirk private correspondance] indicates approximately a 70% increase in the side of the chip which can have a disastrous effect on the yield. A similar approach was tried by Fletcher [ref 2.2] using existing printed circuit board programs and was found to suffer from the same problems.

Radley [ref 2.3] has used a different approach to the problem where he places components one at a time and then does as much of the interconnection as possible. The components are selected in an order that keeps the length

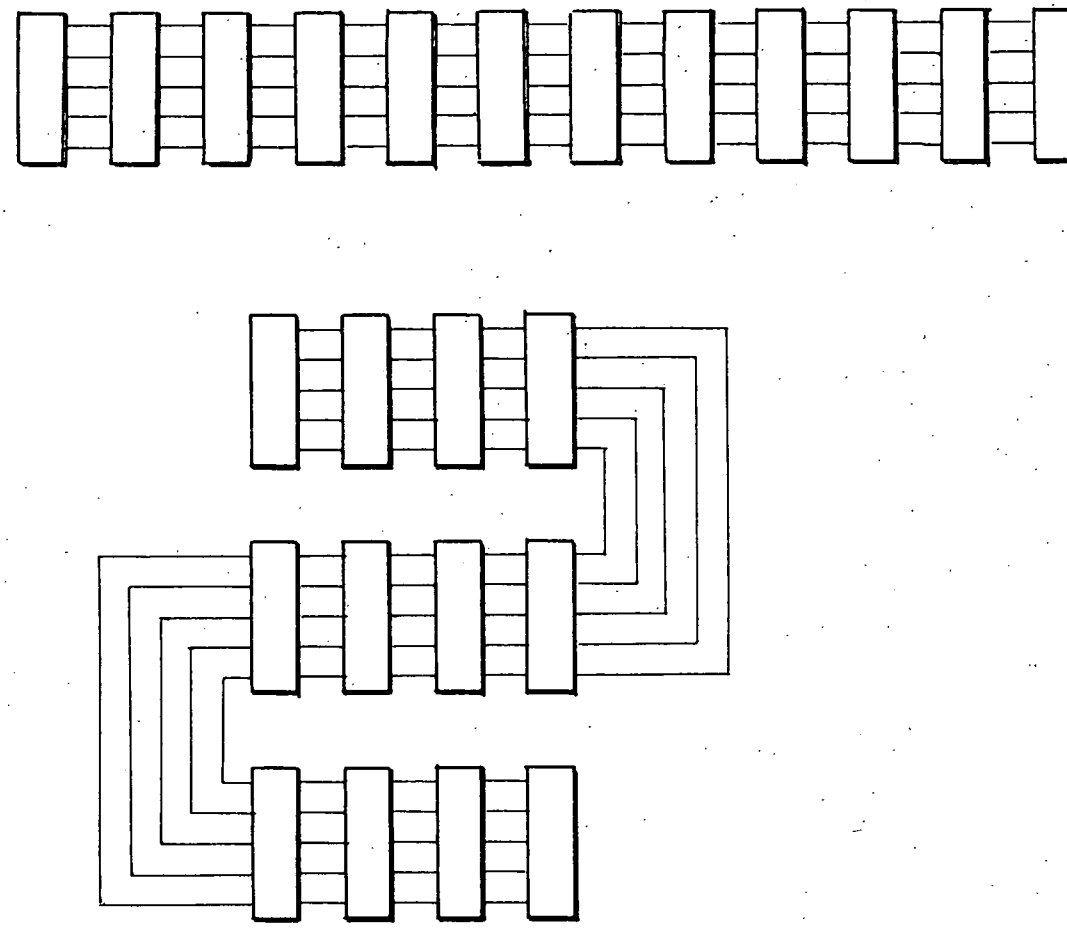


Fig. 2.10

2.3.2 Batch Methods

A batch program reads in a set of input data and performs various calculations on this data to produce a set of results. It does not allow the user any interaction with the program as all the steps are specified in advance. Batch processing is therefore of limited use in layout design but can be useful in the actual production of masks. For example, it can be used to expand the compressed input data describing a layout into the drive tapes for a coordinatograph.

2.3.3 Interactive Methods

An interactive program is one where the user controls the steps performed by the computer. Once the program is running the selection of the next step to be performed is usually based on the results of the previous steps or steps. It is the ideal type of program for design work as it allows the designer to exercise that skill which can never be programmed into the computer ie. his ability to realise that something is different, to think out a new course of action and to proceed on that course.

All the mask design programs except the automatic ones allow interaction of one form or another and these programs will be discussed in more detail.

2.4 Available computer programs

At the start of the research work there were several computer programs available in this country that assisted in the design of integrated circuit layouts and these are described in some detail below. Since the work started other systems have appeared on the scene all of them turn-key systems ie. a complete system of hardware and software which could be just switched on and used. These are commercial systems and although it is relatively easy to find out what they do, the techniques used are a closely guarded secret. They are described here briefly and what technical information that can be obtained on their operation is given in chapter 4.

2.4.1. CAMP System

The CAMP (Computer Aided Mask Production) system for assisting the production of integrated circuit mask masters was written at RRE Malvern by J. Wood, R. Newton, D. Snell of RRE and M. Walmsley of Plessey [ref 2.5]. It was developed as part of the activities of the consortium of British semiconductor manufacturers and RRE. It was written in Algol to work on what was then an Elliot 4130 with a refresh graphics display.

It was conceived as a method of producing the drive tapes for coordinatographs rather than a design aid but nevertheless did have certain facets that were useful as a

design aid and for this reason and the fact that it was one of the first systems to be produced, it is worth considering in detail.

The input data for CAMP exploits the redundancy described earlier by allowing for paraxial shapes and by providing a group and a repeat facility. The input data consists of a series of order words, mask specifiers, names, numbers and punctuation marks. The order words are enclosed in double quotes ("); some describe the various shapes used such as RECTANGLE, POLYGON, and LINE. There are other words that allow the group and repeat facilities and these include GROUP, NEWGROUP, ENDGROUP, REPEAT and ENDREPEAT and finally there are a series of order words that reduce still further the amount of input data that need to be entered: these are words like DITTO and SCALE. The mask specifier indicates the mask or masks on which each individual shape occurs. The names are those given to a series of shapes when they are defined as a group and the numbers are used for the coordinates of the corners etc. The punctuation is used to separate the various parts of the data. The GAELIC manual input language is based on this language and consequently a full description of the facilities of the language is given in the GAELIC Users Manual.

Because of the formal nature of the input language, it is possible to do 'syntax' checking on the input data to make sure the data obeys the rules of the language.

This checking detects many of the errors present in the input data and so reduces the number of errors that can possibly occur on the final masks.

The data is then converted into a ring data structure and then to a 'coordinate file' which contains the necessary information for the tape controlled coordinatographs i.e. every coordinate of every corner of every shape on each mask in turn. This coordinate file can be subsequently post-processed to give drive tapes for various tape controlled coordinatographs, an incremental plotter or a display file for the refresh graphics display on the 4130. On the graphics display, a window of the layout can be plotted out and the light pen used to find the coordinates of any errors. Unfortunately, it was not possible to modify the layout at this stage: modifications had to be made at an earlier stage either the manual input language or the dump code file.

This lack of interaction was one of the major drawbacks with the system; the other problem was the size of the ring data structure. As it was core resident, it restricted the size of circuits that could be handled.

The decision not to have any interactive facilities was one made by the consortium management committee who felt that any interactive facilities would make the system too dependant on a particular hardware configuration.

2.4.2. REDAC system

The REDAC system was also originally designed to run on an Elliott 4130 with refresh graphics system to help in the design of MOS integrated circuit layouts. The original concept was that the designer sat in front of the graphics terminal and with the aid of the light pen called up a series of paraxial rectangles and placed them on the screen. The rectangles could be on one of six masks controlled by six function keys on the display and rectangles on the same mask that touched could be joined together with the aid of the light pen to form paraxial polygons. Facilities were available for modifying and deleting shapes and for grouping a series of shapes together in order to repeat them. This grouping facility was nowhere near as comprehensive as the group and repeat facilities in the CAMP system. The user could also 'zoom in' to a small window of the drawing and move the window round the drawing. The original data structure was held entirely in core.

This initial system had several disadvantages some of which have been subsequently been removed though others for some reason are still present. The major disadvantage was the restriction to six masks which precluded its use for bipolar integrated circuit design. The data structure being held in core was a severe disadvantage as it restricted the size of the drawing. Sitting a designer at a graphics terminal costing 50 pounds an hour and telling him to design was not ergonomic or economic sense. It is

also difficult to design complex interconnection tracks using rectangles.

The subsequent system based on a PDP15 computer with VT15 graphics display does not have the size restriction as the data structure is disc based. An input language has been added which allows the layout to be designed at the designers own speed and then entered into the computer and stored in the data structure. The display terminal is now used to check and correct the design and thus forms a much better method of using the computer.

There is a program in the suite which checks if the layout description stored in the data structure breaks certain of the 'layout rules' issued by the semiconductor manufacturers and this is described in a paper by Treble [ref 2.6]. The program although fulfilling an extremely important function does appear to have two main disadvantages in that it requires a large amount of computer time to perform the checks (typically 3 hours on a PDP15) and that it gives a large number of possible errors that have to be checked manually by the designer.

The Redac system finally produces drive tapes for several tape controlled coordinatographs which produce 'cut and peel' masters.

2.4.3 Marconi System

The Marconi integrated circuit design system is based on the general purpose drawing program that runs on their Myriad computer with their X2000 graphics system written by S Bird [ref 2.7]. It was originally used to design layouts in the same way as the Redac system where the designer sat in front of the screen and drew shapes. It has, however, advantages over the Redac system because of its general drawing program origins, it has a far more flexible drawing system using either a light pen or tracker ball. Virtually everyone who used the system prefered the tracker ball to the light pen, usually because the designer had an uninterrupted view of the screen and did not lose the tracking cross.

Shapes that had line segments at angles as well as the normal paraxial segments could be drawn, modified and moved with constraints to keep the required lines paraxial.

A series of shapes can be defined as a group at the screen and instances of the definition called in many places in other group definitions, in the main layout or in any subsequent layouts.

The modifications made to the general purpose drawing program were to allow shapes to be allocated to specific masks, to take input data in the CAMP manual input language and to produce drive tapes for various coordinatographs.

It is an excellent design system but has certain disadvantages due to its origins as certain time consuming features are provided that are not used in integrated circuit design. Typical of these is the ability to join a line onto the middle of another line. It was also slow in use as all the layouts and group definitions ever created were kept in the same data structure and all these layouts and groups must be searched when a new definition is created. Another major problem is the high capital cost of the hardware which was in excess of 100,000 pounds. There are also minor problems with flicker when large amounts of layout are displayed.

2.4.4 Subsequent Systems

Since the commencement of the work on GAELIC, further systems to assist in the design of integrated circuit layouts have come onto the commercial market from the United States. The best known of these systems are probably the Applicon, Calma and Computervision. All three are known as 'turnkey' systems which means that they are complete hardware and software systems which once installed can be set into motion by just turning a key. They are all based on the use of minicomputers with disc storage and both the Applicon and Calma systems use a storage tube terminal.

The Applicon design assistant [ref 2.8] is based on a PDP11/05 computer, a Tektronix 611 storage tube display and a version of the Rand tablet. It has a small fixed head disc to hold the program and data and uses cassette tapes for the offline storage of designs. Again the use of the system is based on the philosophy of the designer sitting in front of the screen to design his layout from scratch. In a similar way to the Redac system, it allows the designer to build up the layout from rectangles but it does not allow these rectangles to be merged into polygons. Recent modifications to the software do allow the direct insertion of polygons with up to 127 corners. The rectangles can be in three forms, fixed, stretchable in one direction and stretchable in two directions. It also has a basic grouping facility which includes the ability to fix components to certain points on other components. This feature can be useful in a bipolar technology where the one contact hole can be fixed to one end of a 'stretchable' resistor and the other contact hole fixed to the other and so as the resistor is stretched, the contact holes stay in their correct positions. The main feature of the system is the clever use of the tablet where instead of typing commands, figures are drawn on the tablet which are interpreted as commands. The pattern recognition system is extremely impressive to see working but does not appear to be any faster to use than other more conventional systems. There is a system for using a digitiser to input a completed drawing and this again works with rectangles. The system provides drive tapes

for a Gyrex pattern generator which flashes rectangles in various positions on a photographic plate and this is the main reason for the restriction on the types of shapes available.

The Calma system consists of a Nova 1200 computer a moving head disc, a digitiser connected on-line, a Tektronix 611 tube, a tablet and keyboard. Data can be entered either via the digitiser or the screen and tablet. The main emphasis is on the digitiser which being connected on line is capable of being constrained to move in first the x and then the y directions thus ensuring shapes are paraxial. Non-paraxial line segments are possible by over-riding the locks. The fact that a polygon does not close or that there are rounding error problems can be brought to the users attention immediately and corrected.

The screen input is similar to that used in the Marconi system in that polygons and rectangles can be drawn. It uses a menu on the tablet to select commands. It appears to be just as fast, if not faster than the Applicon system to use.

The main disadvantage is that although it uses a standard Nova computer the design and interfaces are non standard and so the standard Nova disc compilers cannot run and so the computer cannot be used for other purposes.

The Computervision system is very similar to the Calma system but does not put so much emphasis on the use of a storage tube for either data input of data modification but rather relies on a digitiser plotter. This, as the name implies, is a digitiser and plotter combined. As data is entered via the digitiser, the shape can be immediately plotted, superimposed on the original sketch to give an immediate check and thus ease the problem of identifying and correcting errors. However, despite the extremely fast plotting time available, it is probably too time consuming to replot large areas of an integrated circuit layout.

2.5 Requirements for an Interactive C.A.D. system

As interactive graphic equipment is expensive, it is not possible to allow every designer his own graphics terminal. Each terminal must therefore be shared by several designers and consequently must only be used for interactive work. The individual designer should not, for instance, sit in front of the terminal and design from scratch as he will spend most of his time thinking and only a small part 'drawing'. While he is thinking, the terminal facilities are obviously wasted. The designer must therefore plan out exactly what he wants to do before going to the terminal.

It is therefore desirable to have a way in which the designer can sit at his desk, design part of his layout and code it up for the computer. He should then be able to feed this data into the computer and then use the interactive facilities to check and modify his design. To do this it is essential to have a manual input language that the designer can use at his desk to code up his rough sketch. Because this input language is to be used by a layout designer rather than by a computer specialist, it must be extremely easy to use. This in turn means that it is easy to understand and should not entail remembering a series of codes for the various shapes that he uses, instead it should use words that are easily remembered and recognised. The amount of data that he has to prepare must be kept to a minimum. Often a designer uses the same set of shapes over and over again in a layout; the input language must take account of this and allow him to define the set as a 'group' and then call up 'instances' of the group in various positions and various orientations in his layout.

One designer may be quite happy producing rough sketches and working from them, whereas another designer may prefer to produce an accurate scaled drawing of part of the layout before approaching the computer. This latter approach tends to produce a more complex drawing which takes longer to code up using the manual input language. Alternatively it may be necessary to modify an existing design that only exists as a large composite

drawing. For both these requirements it is essential to have a method of entering the data directly from the drawing into the computer without resorting to the manual input language. This effectively means using a digitiser or a similar device. It is therefore necessary to have a method of using a digitiser to extract the information from the drawing. The digitiser language like the manual input language must minimise the amount of data that has to be entered and must handle the group facility.

The integrated circuit designer, like any other human, can make mistakes, especially when coding up data for the computer if it is a new experience for him. The computer program that reads in the data, therefore, must check it as thoroughly as possible and when errors are detected, the program must give meaningful messages that tell the designer exactly what he has done wrong. This 'syntax checking' as it is usually called will detect shapes that have been incorrectly specified but will not detect that a shape is in the wrong place. This is usually done by visually checking the layout.

The interactive graphic terminal provides an excellent method of visually checking the layout and for correcting any mistakes found. The screen of the terminal is not big enough to display all of a typical layout at a scale at which modifications can be made. Facilities must therefore be provided in the interactive program to allow the user to 'zoom' in and plot out an area of the layout at a much larger scale. There must also be facilities for

plotting an adjacent area ie. 'windowing'.

Having plotted out a suitable 'window', the user will need to interact with the drawing; he will need to identify a shape containing an error and either correct it or delete the shape. Correcting an error involves either moving the whole shape or just a point on the shape; the movement is generally required in a direction parallel to one of the axes ie. orthogonal movement but is occasionally required at an angle.

Any group instance can be in the wrong position or can be drawn at the wrong orientation. The user does not wish to correct every individual shape in the instance and so the group structure must be kept in the data for the interactive program. Facilities should exist to allow the user to identify one point in the instance, the origin, and then move the instance or change its orientation.

There will obviously be times when mistakes are made in the shapes of a group definition and so facilities must be provided for correcting them. This must be done with care, however, as instances can occur in several places on the drawing and a shape that appears wrong in one instance may appear correct in another. It is therefore essential that the user knows that he is modifying a group definition ie. he cannot modify the shapes in an instance by mistake.

Often shapes are missed from the layout and have to be added interactively. Facilities must therefore exist for adding rectangles, polygons, lines and even group instances. These new shapes must be plotted on the screen so that the user can check that they are in their correct positions.

When the design is complete and free of errors, then tapes to drive the coordinatograph are required. These tapes can be used to produce extremely accurate large scale check plots. Unfortunately producing these check plots is expensive in both time and money. A rapid cheap plot is required to check the layout design at the designers leisure. This frees the graphic terminal for more interactive work.

CHAPTER 3: The use of GAELIC in mask design

This chapter is intended to give an insight into the use of GAELIC in the design of integrated circuit masks. The first part is devoted to input to the GAELIC programs describing the philosophy behind the two main methods of inputting the layout description into the computer. This is followed by a simple example of how the manual input language is used to enter data and how the syntax errors are detected. The next part describes the features of the interactive program and using the same example, shows how it is used to plot out all or part of the layout, how errors are corrected and how missing shapes are added. The final part briefly describes the operation of the other programs including the post-processors which produce the drive tapes for the tape controlled coordinatographs and mask making machines.

3.1 Input to the GAELIC suite of programs.

There are actually three different methods of entering the description of all or part of an integrated circuit layout into the GAELIC programs. These methods are:

1) by the use of a digitiser and the GAEL1A program,

2) via the manual input language and GAEL2A and

3) by use of the crosshair cursor and the keyboard of the Tektronix terminal using GAEL4A.

The third method, which is described in detail in section 3.2, is mainly used for interactively adding shapes inadvertently missed when the original layout description was entered. However, it does allow all or part of the layout to be designed on the screen. The first two methods of input are usually used to enter the original layout description into the computer and the obvious question that must be answered is why two methods are required. Essentially the digitiser is used to enter the large quantities of data required for a complete integrated circuit design while the manual input language is used for entering the data for a small part of of a large design or all the data for a small design. These two methods will now be considered in more detail.

3.1.1 Using a digitiser

The digitiser method is ideally suited for entering the description of a large fully designed layout. The design must be drawn on an accurately gridded mylar sheet which is securely fastened to the digitiser. The digitiser is used to accurately and quickly record the coordinates of the corners of the shapes without the manual counting of increments which we shall see is a feature of the manual input language. The method does, however, have the obvious prerequisities of a digitiser and someone who can use it.

The design and drawing of an integrated circuit layout is a creative function which is generally enjoyable. It does have certain problems when it comes to redrawing large areas of the layout in a slightly different position for example to insert an extra metallisation track, but this is not sufficient to detract from the overall enjoyment. The actual digitising, however, although requiring a continuous high degree of concentration, is extremely repetitive and tedious and gives no job satisfaction to the layout designer. It is consequentially desirable to use a different person for this task. The time required to digitise a layout is much less then that required to design the layout and so one digitiser operator can cope with designs from several designers. For a large integrated circuit design team say 5 or 6 designers, the economics of buying a digitiser and hiring an operator are favourable. However, with only one or two designers, the economics dictate that another method of designing and/or entering data into the computer is desirable.

Any method of using the digitiser must be as easy and straight forward as possible in order to minimise the number of errors that are made. Consequently certain minor modifications were made to the keyboard of the digitiser. The standard Metrograph digitiser like many others has a READ button which, when pressed, records the coordinates of the digitiser cursor on the output tape; it also has a small keyboard which enables the characters

marked to be added to the output tape when the respective keys are pressed. It was therefore possible to devise a system, using the digitiser, where pressing the various characters on the keyboard indicated the start of a shape and the READ button recorded its coordinates. However, remembering which character was used for which shape proved difficult so the labelling on the keys was changed to give more meaningful abbreviations such as RECT, POLY and LINE. Each key, however, still only produced a single character on the output tape and so the standard digitiser could be used in an emergency to digitise a layout. The modified keyboard is shown in fig. 3.1.1.

Often when digitising a layout, an operator will realise that a mistake has been made and will wish to correct it. This mistake may well be that a shape is on the wrong mask or the wrong name given for a group definition. It is always possible to write down notes regarding these errors and to subsequently edit the tape or edit the data once it is entered into the computer. This in practise turns out to be very disruptive to the operator and so a method of immediately adding corrections to a tape is required. The best solution would be to have a series of keys that can be pressed to correct the various mistakes that can be made. Unfortunately, however, the cost of extending the keyboard size to allow for this was prohibitive and so only one ERROR key was allowed. A method of correcting errors had therefore to be designed in which the instant the error button was

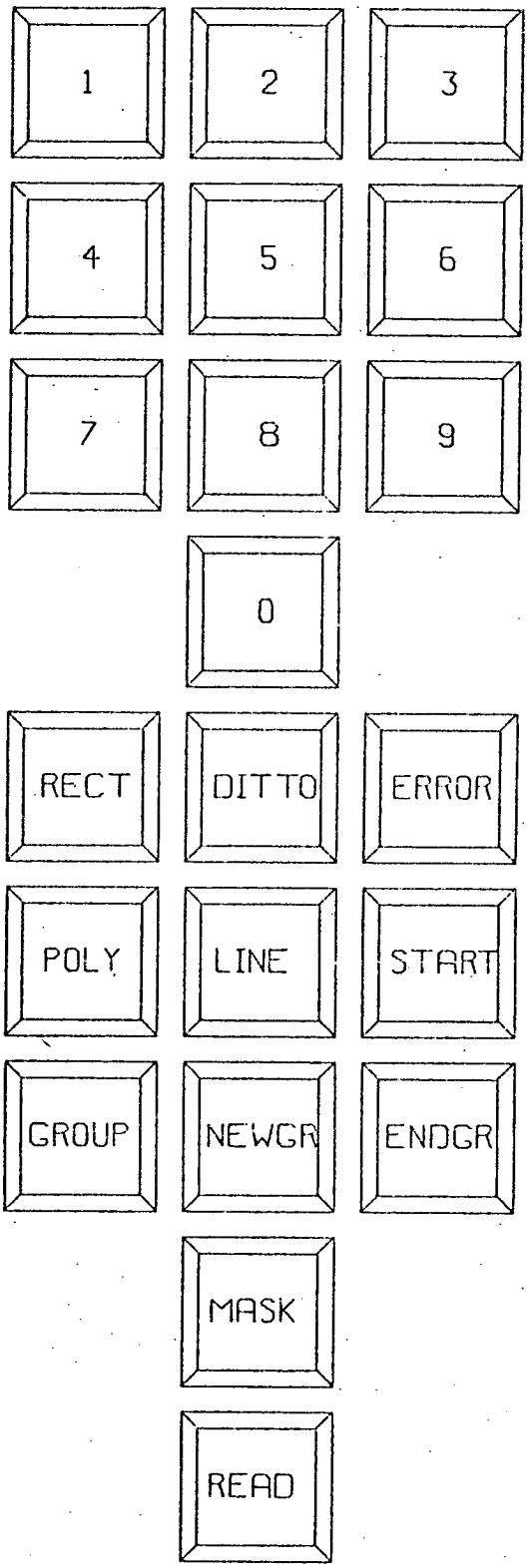


FIG 3 1.1 LAYOUT OF KEYBOARD

pressed and the number of times that it was pressed specified the error to be corrected. For example if a wrong point was digitised in a shape i.e. the cursor was in the wrong position when the READ key was pressed then the coordinates can be ignored by pressing the ERROR key immediately afterwards. However if the whole shape is to be ignored then the ERROR key is pressed twice. The use of the ERROR key is described fully in the GAELIC users manual [ref 3.1].

Another problem that must be allowed for when processing data from a digitiser is that of rounding. There are actually two problems, 'paper' distortion and 'paper' position. The word 'paper' is used here to mean the material on which the layout design is drawn. If a large layout is to be digitised then it is essential to use a stable material such as a translucent mylar sheet at least 5 thou (0.005 inches) thick, whereas if a small part of a layout is to be digitised, actual graph paper could be used. The 'paper' distortion is the distortion in the paper grid due to inaccuracies in the actual manufacturing process. Most grids on paper are printed from a roller and, with slight variations in speed of the roller and slip on the paper, a different scale is sometimes obtained in the x direction to that in the y. There is also a problem of calibration between the digitiser and paper e.g. a line that is nominally 10 inches exactly on the paper, may have a length of 10.12 inches according to the digitiser. The paper position problem exists because it

is impossible to place the paper exactly horizontally on the digitiser: consequently the paper axes are always at a slight angle to the digitiser axes. The layout is drawn with respect to the axes and grid scaling on the paper. However the digitiser will obviously output digitiser coordinates so the computer program must do the necessary coordinate transformations to give the corresponding paper coordinates, allowing for the errors due to paper distortion and paper position.

3.1.2 Using the manual input language

The manual input language is an extremely useful method for entering the description of a small part of a layout into the computer. A typical part of a layout would be a single bit of a shift register. The designer can quickly code up his design using this manual input language and enter this data into the GAELIC suite. The part layout can be quickly plotted on the screen of the Tektronix terminal and any mistakes discovered can be interactively corrected. This part layout can be stored in the computer while the designer works on another part. The process can be repeated until he has designed all the component parts of his layout, when he can use the interactive facilities to join them together to produce a complete design.

This is obviously a different method of design from that using the digitiser and is one in which the designer plays a much larger part. Some of the tedious repetitive work can be taken over by the computer, for example it is possible to get the computer to redraw large sections of the layout in slightly different positions. This is one of the tedious parts that has to be done manually when producing the finished layout drawing prior to digitising.

In order that this new method of design can be fully exploited, the manual input language should have the following characteristics.

a) it must be easy to use the language.

b) the language must minimise the amount of data that has to be entered.

c) the language must be easily processed by the computer.

There are unfortunately conflicts between these requirements: for example, for the data to be processed most easily by a program written in Fortran the data must be in a fixed predefined format. Fixed format input is extremely difficult to produce, mainly because the user cannot understand why the extra spaces are so critical. Secondly it is far easier to handle a purely numeric input data using Fortran so that a number 1 is used to specify that data for a rectangle is about to be entered, number 2 to specify data for a polygon etc. Unfortunately there are more to integrated circuits then polygons and rectangles so the user would have to make extensive use of

a crib sheet to decide which numbers to use. The use of numbers also makes it extremely difficult for the computer to check the input data for errors e.g. number 1 could be the code word for a rectangle or a coordinate value.

The input language chosen therefore is a compromise between the three requirements and is based on the language used in the CAMP programs [ref 3.2]. The input language is fully described in the GAELIC users manual and two examples of its use are shown below:

"RECT" (1) 10,5:20,10;

and

"POLY" (4) S,30,64:20,4,10,4,-12,-4,-18,-4;

These describe a rectangle, on mask 1 only, which has its origin at x=10, y=5 and is 20 units long and 10 units high and a polygon with all its sides parallel to the axes which starts at x=30, y=64 and has a line segment 20 units long in the x direction followed by a segment 4 units long in the y etc.

In order to code up a shape description from a drawing on gridded paper, the user must count the increments from the origin of the drawing to the origin of the shape and then count the increments along each line segment. This counting is a potential source of error and must be done very carefully.

Certain modifications were made to the CAMP input language in order to make full use of the facilities available and to give a more flexible approach to layout design. Probably the main modification was to omit the commands concerned a) with file storage and b) with the running of the CAMP program. The former is automatically handled by the operating system of a time-sharing computer and the latter is controlled by the user in GAELIC.

Another major modification to the CAMP programs involved the use of LINES. The basic philosophy behind the choice of shapes in the CAMP language was based on the idea that all shapes on an integrated circuit are closed shapes. Most integrated circuit masks are still made using a cut and peel material on a tape controlled coordinatograph. Here the knife cuts round the perimeter of the shape and the material inside can subsequently be peeled away. Obviously this is only possible if the line segments defining the perimeter form a closed shape. Consequently the LINE in CAMP describes a closed shape of a fixed width and was intended for the fixed width Aluminium interconnections. Unfortunately there are many violations of the concept of a fixed width interconnection; for example, when the interconnection passes over a contact hole the width is temporarily increased and so in practise the LINE was of limited use. However it is extremely useful to have a mathematical line (i.e. a line with finite length and zero thickness) as an intermediate shape which can subsequently be joined with

other mathematical lines to form closed shapes on the cut and peel material. Consider the aluminium interconnection of a single stage of a shift register defined in the CAMP language as a GROUP using closed shapes as shown in fig. 3.1.2a. Several instances of the GROUP are called to form the complete shift register as shown in fig. 3.1.2b.

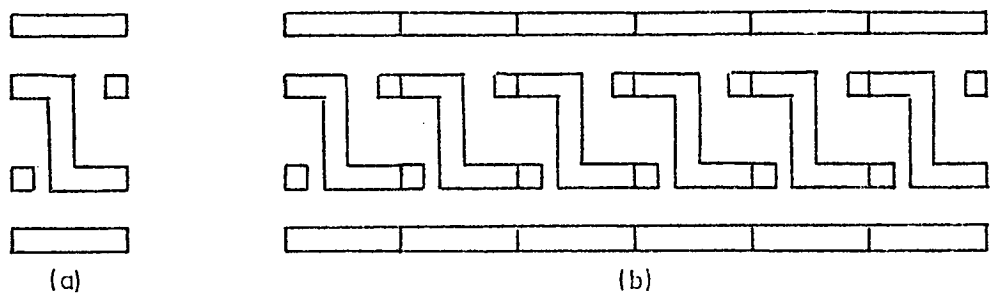


Fig. 3.1.2 Interconnections using closed shapes

The alternative method used in GAELIC involves a LINE with zero thickness and the single stage of a shift register is as shown in fig. 3.1.3a.

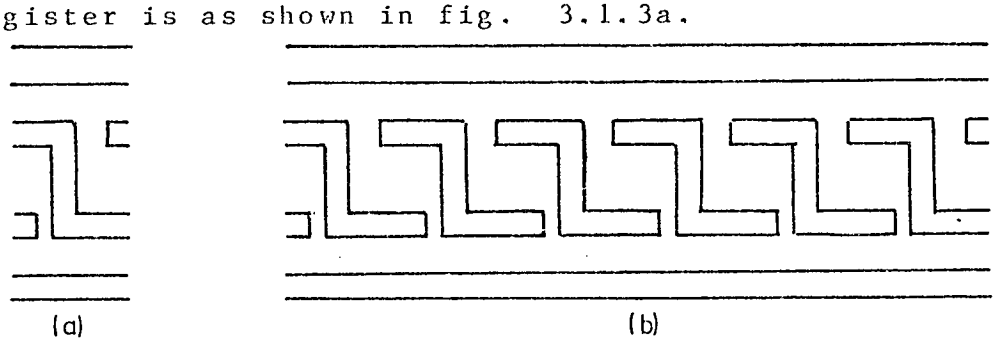


Fig. 3.1.3 Interconnections using open shapes

The several stages of shift register are shown in fig. 3.1.3b and it can be seen that the complete metallisation tracks can be easily checked for continuity and when the final cut and peel master is produced, the number of pieces that must be hand peeled is minimised.

Usually an instance of a group definition is required to contain all the shapes on all the masks. However, there are circumstances when only certain masks are required. For example, consider a group definition of a shift register stage; when an instance is called to form one bit in the middle of the stage then the shapes on all the masks are required. However, the instances forming the first and last bits may well require different shapes on the metallisation mask due to the interconnections joining the bits to other circuit components. The method used in CAMP was to define the first and last bits either as separate group definitions or as individual shapes in the main layout definition. However, in GAELIC an instance can be called so that only shapes on a specified series of masks are produced. Hence the metallisation can be ignored in the instances used for the first and last bits of the shift register stage and the special metallisation required to interconnect to the rest of the circuitry can be added to the main layout. The GROUP call in GAELIC therefore comes in two forms:

"GROUP" ONE,10,10,1;

and

"GROUP" ONE (1,2,3) 40,70,1;

The first call produces the shapes on all masks while the second only produces the shapes on masks 1, 2 and 3, any shapes on mask4 etc. are ignored.

3.1.3 The inputting of data for a small example

For small layouts such as the one shown in fig 3.1.4 which is a test chip to investigate the effects of changes in the semiconductor processing, it is convenient to use the GAELIC manual input language. The input language necessary to describe this layout is shown in fig 3.1.5. The input language contains several errors inserted to show how the error diagnostic system works. Most of these errors are trapped by GAEL2A the syntax checker as can be seen in fig 3.1.6 which shows the teletype printout obtained when running the program. The initial Dump code file created does not contain the descriptions of the shapes containing the syntax errors, the corrected descriptions of these shapes, however, can be typed in via the keyboard immediately after the input file has been processed and these corrections are then added to the Dump code file. The teletype printout obtained when doing this is also shown in fig 3.1.6. The Dump code file is then compiled into the Ring Data Structure using GAEL3A and this data structure is used to store the description of the layout throughout its design. Several programs interact with this ring data structure to allow for modification and addition, to produce large scale drawing and to produce drive tapes for tape controlled coordinatographs and mask making machines.

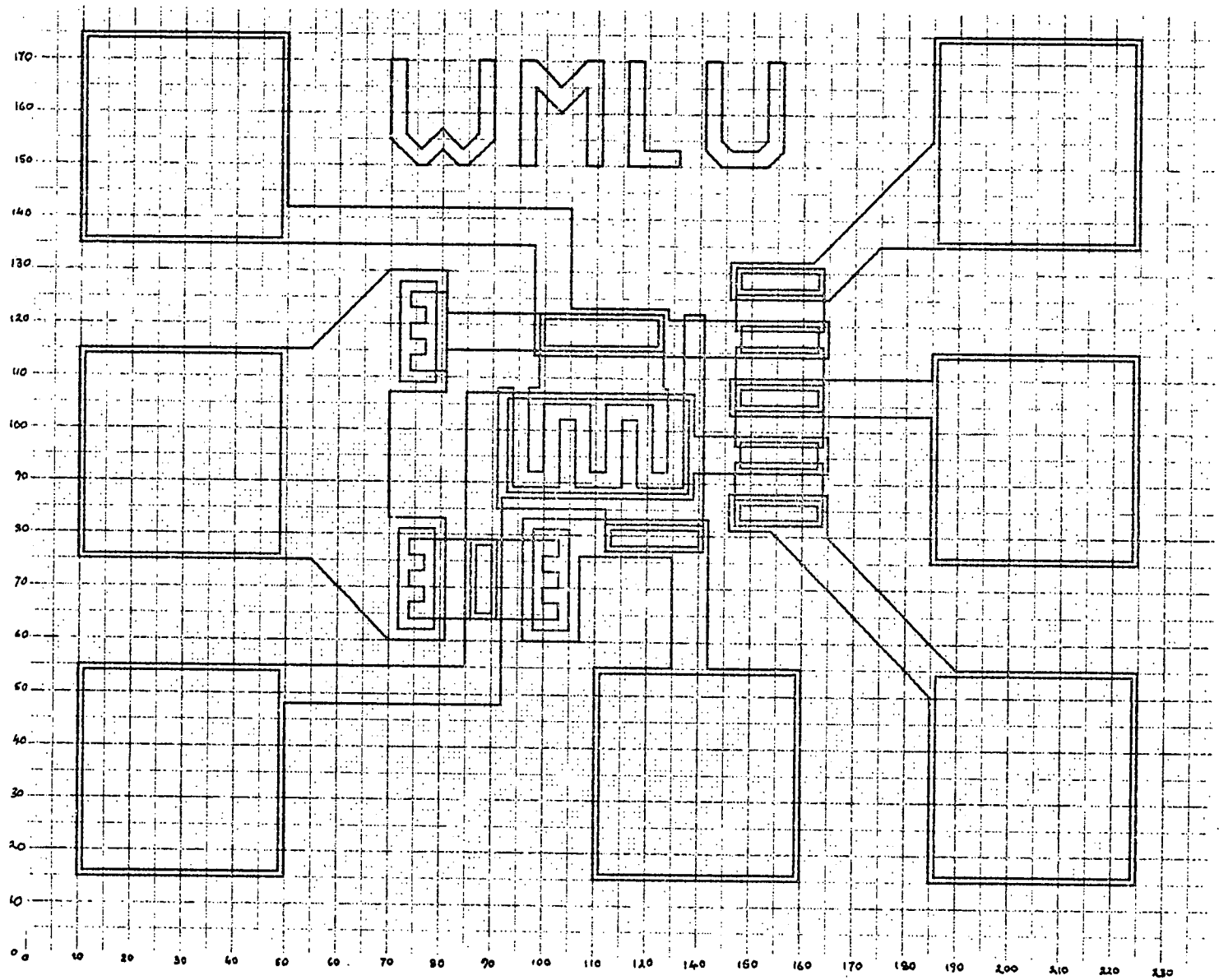


Fig. 3.1.4 Layout of small test circuit

```
"POLY" (1:4) L,75,150:2,0,3,3,3,-3,2,0,5,5,0,15,
                      -3,0,0,-14,-3,-3,-4,4,-4,-4,
                      -3,3,0,14,-3,0,0,-15,5,-5;
"POLY" (1:4) L,95,150:3,0,0,15,5,-5,5,5,0,-15,3,
                      0,0,20,-3,0,-5,-5,-5,5,-3,
                      0,0,-20;
"POLY" (1:4) S,116,150:10,3,-7,17,-3,-20;
"POLY" (1:4) L,134,150:9,0,3,3,0,17,-3,0,0,-15,
                       -2,-2,-5,0,-2,2,0,15,-3,
                       0,0,-17,3,-3;
"POLY" (1) S,74,64:29,3,-3,3,3,3,-3,3,3,3,-29,-3,
                   3,-3,-3,-3,3,-3,-3,-3;
"POLY" (1) S,91,85:21,-8,19,45,-4,-33,-9,13,-3,
                   -13,-9,13,-3,-13,-9,19,-3,-23;
"POLY" (1) S,97,32:3,13,9,-13,3,13,9,-13,3,16,-1,
                   14,-42,4,-7,-3,3,-3,-3,-3,3,-3,
                   -3,-3,7,4,18,-7,-2,-16;
"RECT" (1) 137,119:17,12;
"RECT" (1) 13U,97:17,19;
"RECT" (1) 137,82:17,12;
"RECT" (2) 72,62:7,19;
"RECT" (2) 93,88:35,18;
"RECT" (2) 72,109:7,19;
"RECT" (2) 99,115:24,7;
"RECT" (2) 137,126:17,5;
"RECT" (2) 138,115:15,5;
"RECT" (2) 137,104:17,5;
"RECT" (2) 138,93:15,5;
"RECT" (2) 137,82:17,5;
"RECT" (2) 112,77:19,5;
"RECT" (2) 98,62:7,19;
"RECT" (2) 86,64:5,15;
"RECT" (3) 87,65:3,13;
"RECT" (3) 113,78:17,3;
"RECT" (3) 138,83:15,3;
"RECT" (3) 138,105:15,3;
"RECT" (3) 138,127:15,3;
"POLU" (4) S,10,15:40,33,42,39,37,5,26,7,-26,8,
                   -44,-52,-75,-40;
"POLY" (4) S,110,15:40,40,-18,28,-36,-23,11,16,
                    18,-21,-15,-40;
"POLY" (4) L,175,15:40,0,0,40,-35,0,-25,25,0,8,
                    -19,0,0,-7,8,0,31,-31,0,-35;
"POLY" (4) S,175,75:40,40,-40,-5,-39,-7,39,-28;
"POLY" (4) L,136,125:19,0,10,10,50,0,0,40,-40,0,
                     0,-20,-23,-23,-16,0,0,-7;
"POLY" (4) S,10,135:88,-21,57,7,-31,2,-19,19,-55,
                    33,-40,-40;
"POLY" (4) L,10,75:45,0,15,-15,11,0,0,23,-11,0,
                   0,24,11,0,0,23,-11,0,-15,-15,
                   -45,0,0,-40;
"RECT" (5) 11,16:38,38;
"RECT" (5) 111,16:38,38;
"RECT" (5) 176,16:38,38;
"RECT" (5) 176,76:38,38;
"RECT" (5) 176,136:38,38;
"RECT" (5) 11,136:38,38;
"RECT" (5) 11,76:38,38;
"FINISH";
```

Fig 3.1.5 GAELIC input language

```
RUN GAEL2A

                          'GAEL2A'
PROGRAM TO CONVERT GAELIC LANGUAGE INTO DUMP CODE

DOES THE FILE HAVE LINE NUMBERS - YES OR NO
NO

ENTER NAME FOR NEW DUMP CODE FILE
TESTD

ENTER MASK NUMBERS USED IN LAYOUT
1 2 3 4 5


"RECT" (1) 13U,97:17,19;
-------------^
<ERROR NUMBER 23 IN STATEMENT NUMBER  9>
ILLEGAL TERMINATOR PRIOR TO COLON - SHAPE IGNORED

"POLU" (4) S,10,15:40,33,42,39,37,5,26,7,-26,8,
-----^
<ERROR NUMBER 23 IN STATEMENT NUMBER 28>
ORDER WORD NOT RECOGNISED - SHAPE IGNORED


ENTER NAME OF NEXT GAELIC LANGUAGE FILE OR TTY FOR
KEYBOARD INPUT OR PRESS RETURN TO FINISH
TTY

KEYBOARD INPUT WITH NO LINE NUMBERS EXPECTED
ENTER INPUT DATA
"RECT" (1) 137,97:17,19;
"POLY" (4) S,10,15:40,33,42,39,37,5,26,7,-26,8,
-44,-52,-75,-40;
"FINISH";


ENTER NAME OF NEXT GAELIC LANGUAGE FILE OR TTY FOR
KEYBOARD INPUT OR PRESS RETURN TO FINISH

DUMP CODE FILE SAVED AS :- TESTD

END OF EXECUTION
```

Fig 3.1.6 Running GAEL2A the syntax checker

3.2 Interaction with the layout

The user can interact with the layout using GAEL4A. This program uses one of the Tektronix 4010 series of storage tube terminals to interactively modify and correct an integrated circuit layout description held in a ring data structure file. The user can select which ring data structure file is to be processed and can select the part of that layout to be plotted, i.e. a particular group definition and window size. That part of the layout within the window is then plotted out on the storage tube screen. The plot remains or is stored on the screen until it is cleared. A non-storing cross-hair cursor can be displayed on the screen and its position controlled by a pair of thumb wheel potentiometers. Various character keys can be pressed when the cross-hair cursor is displayed which causes the terminal to not only send the character pressed to the computer but also four other characters which define the position of the cursor. This information governs the running of the program e.g. pressing 'R' indicates the starting coordinates of a rectangle to be drawn on the screen and added to the data structure whereas pressing 'F' finds the nearest point in the layout to the cursor and prints out its coordinates and whether it is in a group definition or a set of repeated shapes.

The storage tube screen is divided into two parts: the right hand edge of the screen is used for messages and is called the 'menu area'. It contains such information as the list of masks plotted on the screen, the mask number being modified and the name of a group when an origin is identified. The remainder of the screen except for a small area at the top which contains the window size, is used for plotting and is known as the 'plotting area'.

The program operates in a hierarchical manner in that it gives the user a choice of options at one level and when one of these is selected, the program drops down to a lower level where the user has a different choice of options. The first level is known as the 'program command level' and the second as 'cursor command level'.

The program command level options are concerned with selecting the group definition to be processed, the size of the window, the mask numbers to be plotted, modifying or drawing on an existing data structure etc. Two of these options MODIFY and DRAW allow the cross-hair cursor to be displayed and this can be used to identify existing shapes plotted out on the screen, to indicate the coordinates of new shapes to be added or to change the window being plotted etc. This lower level is the 'cursor command level'.

Whenever the cross-hair cursor is on the screen there are certain options that are available mainly associated with the window plotted, these are known as the 'permanent

cursor commands' and are selected by pressing one of the following character keys:

1, 2, 3, 4, 5, 6, 7, 8, 9, J, Q, U, V, W and Z.

The results of pressing these keys are described in detail in the GAELIC users manual.

When the cross-hair cursor is first set up by the MODIFY or DRAW options there are certain options available which can perform such functions as identifying the nearest point in the definition, indicating where a polygon should start etc and these are known as 'main cursor commands' and are selected by pressing one of the following character keys:

F, G, I, L, M, P, R, \,], ^ and SPACE

After several of the 'main cursor command' options have been selected, further information is required. For example, when a point on a shape has been identified, the user needs to tell the program if the point or the shape is to be moved and its new position. This is accomplished by using options known as the 'subsequent cursor commands' and are selected by pressing one of the following character keys:

A, D, E, H, N, O, S, X, Y, [, # and SPACE.

By using the cursor commands at the various levels, shapes can be identified and modified, new shapes can be added and the window changed. This process is illustrated in the following example.

3.2.1 Interactively modifying the example

GAEL4A is used to plot out all of the layout on the screen. the plot is shown in fig 3.2.1; the different line types are used to distinguish between the various masks in the layout. There are two errors shown in the plot, a contact hole on mask 3 is missing and part of the thin oxide (mask 1) is in the wrong position.

These errors are corrected by first selecting the MODIFY option and asking to modify mask 1. The cross hair cursor is positioned over point A of the thin oxide shape in the wrong position. By pressing the character 'I' the coordinates of the cross hair cursor are sent to the computer and a search is made for the nearest point in the data structure, the point is illuminated briefly and then the cross hair cursor is returned. The cursor is then positioned at the correct position for the point on the shape, ie. point B, and the character 'H' pressed. This moves the whole of the shape into its correct position in the data structure so that next time the drawing is replotted, the shape will be in its correct position. The cross hair cursor is then returned to the screen ready to initiate further modifications. However, if the user wishes to immediately check that the shape he has just modified is in the correct position he can just replot the particular shape by pressing the character 'D'. When the cross hair cursor is returned, it is positioned at point C and 'Z' pressed. This causes the program to 'zoom in' and replot the window at twice the original scale as shown in

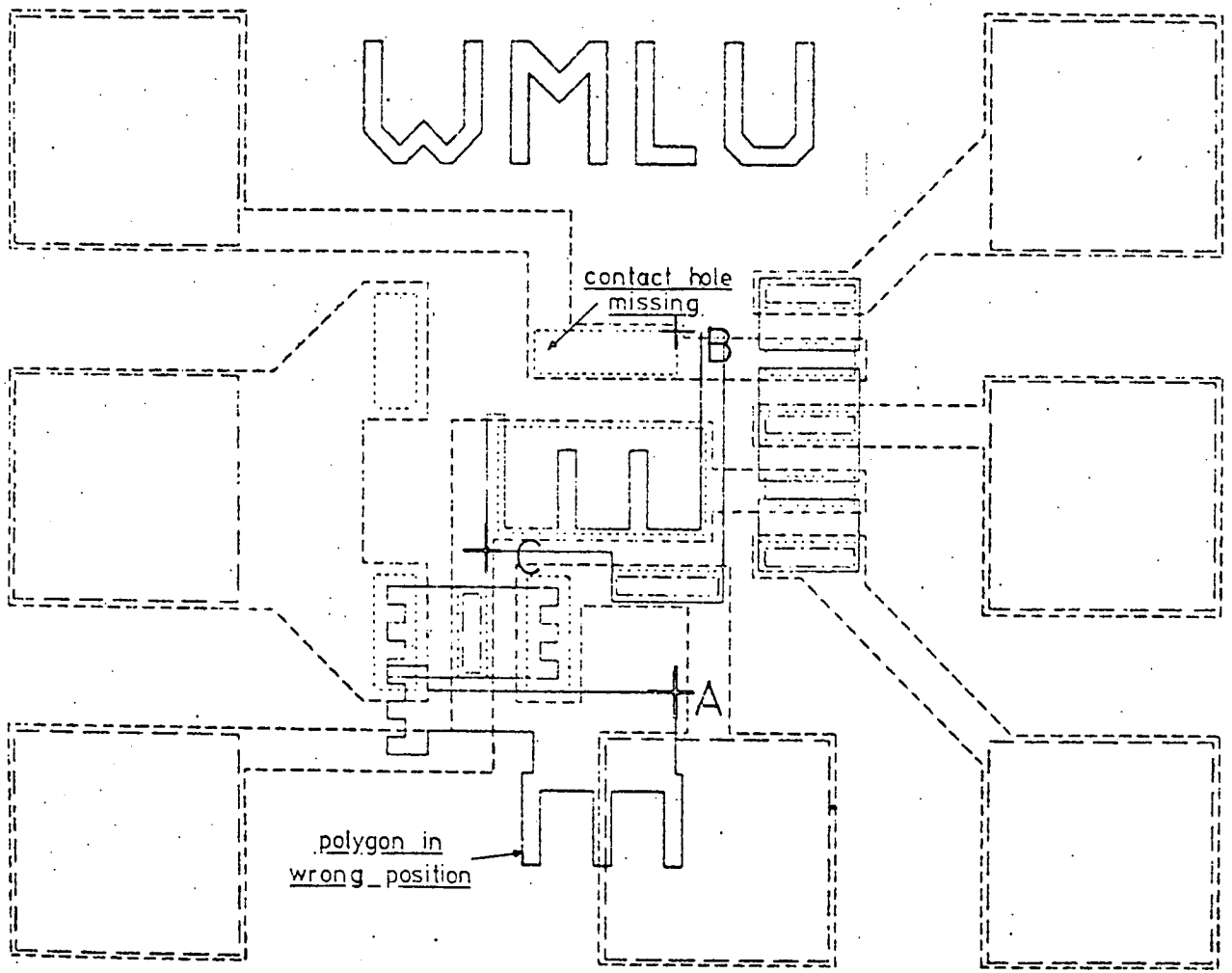
WINDOW SIZE IS 10 15 215 220

MASKS PLOTTED

1 2 3 4 5

MASK NUMBER

1



WHAT NEXT

MODIFY

Fig. 3.2.1

fig 3.2.2. Note that the thin oxide shape is now in its correct position. Pressing the character 'M' when the cross hair cursor is displayed allows the user to change to operate on a different mask number and this is obviously necessary to add the contact hole missing from mask 3. The cross hair cursor can then be positioned at point E, the bottom left hand corner of the rectangle and 'R' pressed. A dot will appear at the nearest grid point to the cross hair cursor indicating the position for that corner of the rectangle. The cross hair cursor is returned and is positioned where the top right hand or opposite corner is required, ie. point F, and 'O' pressed. Again a dot appears at the nearest grid point and the cursor returned. Pressing 'D' causes the rectangle to be drawn on the screen before the cursor is returned. To check that all the modifications have been made the user may wish to have a final look at the complete layout, this can be done by pressing 'J' and the result is the plot shown in fig 3.2.3. If space ' ' is pressed 'WHAT NEXT' will appear at the bottom of the menu area. Answering 'END' to this question will exit from the program with the corrected data structure.

The description of the layout in the Ring Data Structure is now correct and can then be post-processed using GAEL9F or any of the other post-processors, to produce a series of files containing drive tapes for a tape controlled coordinatograph or mask making machine.

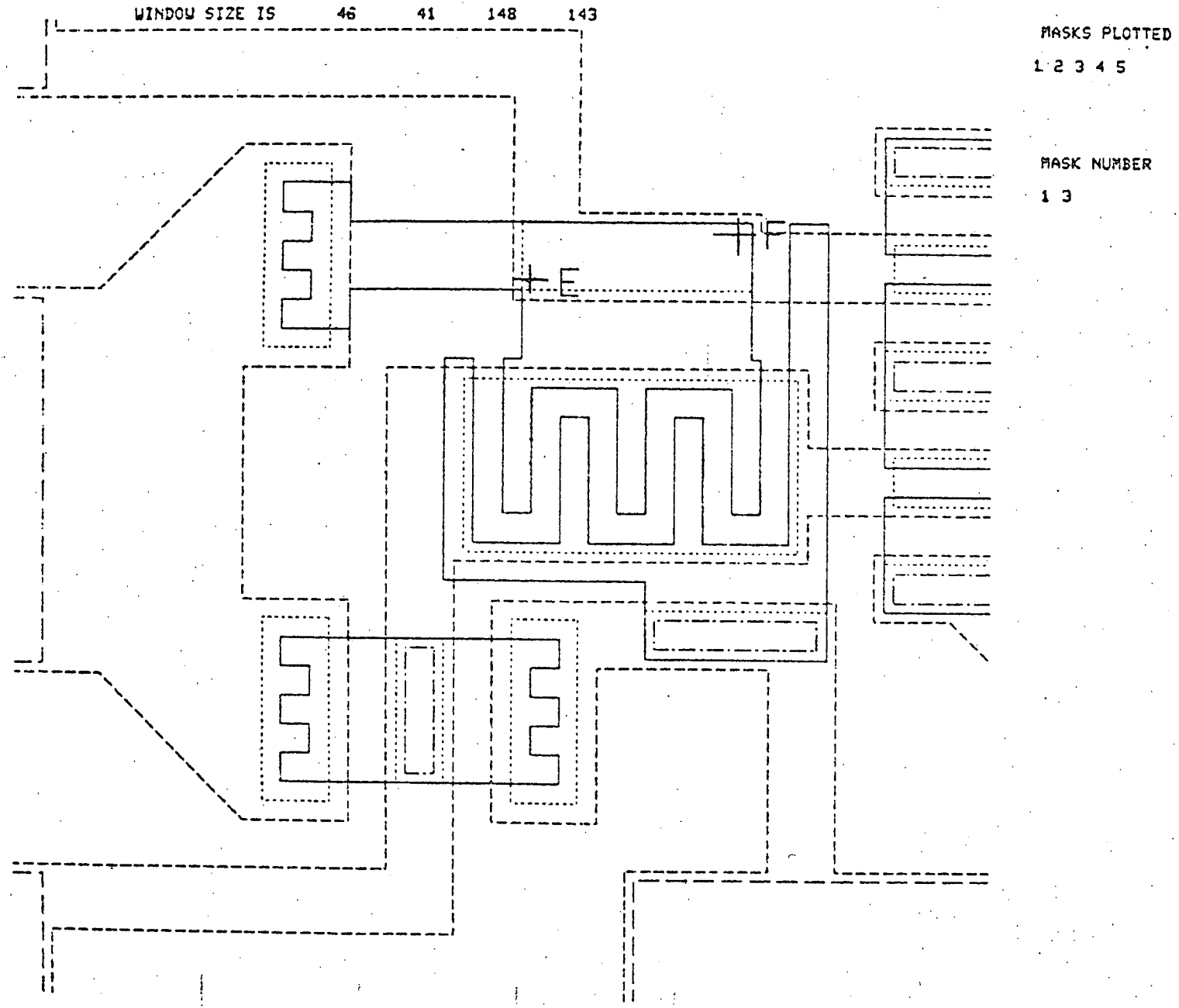

WINDOW SIZE IS 46 41 148 143
MASKS PLOTTED
1 2 3 4 5
MASK NUMBER
1 3
E

Fig. 3. 2. 2.

WINDOW SIZE IS 10 15 215 220

MASKS PLOTTED

1 2 3 4 5

MASK NUMBER

3

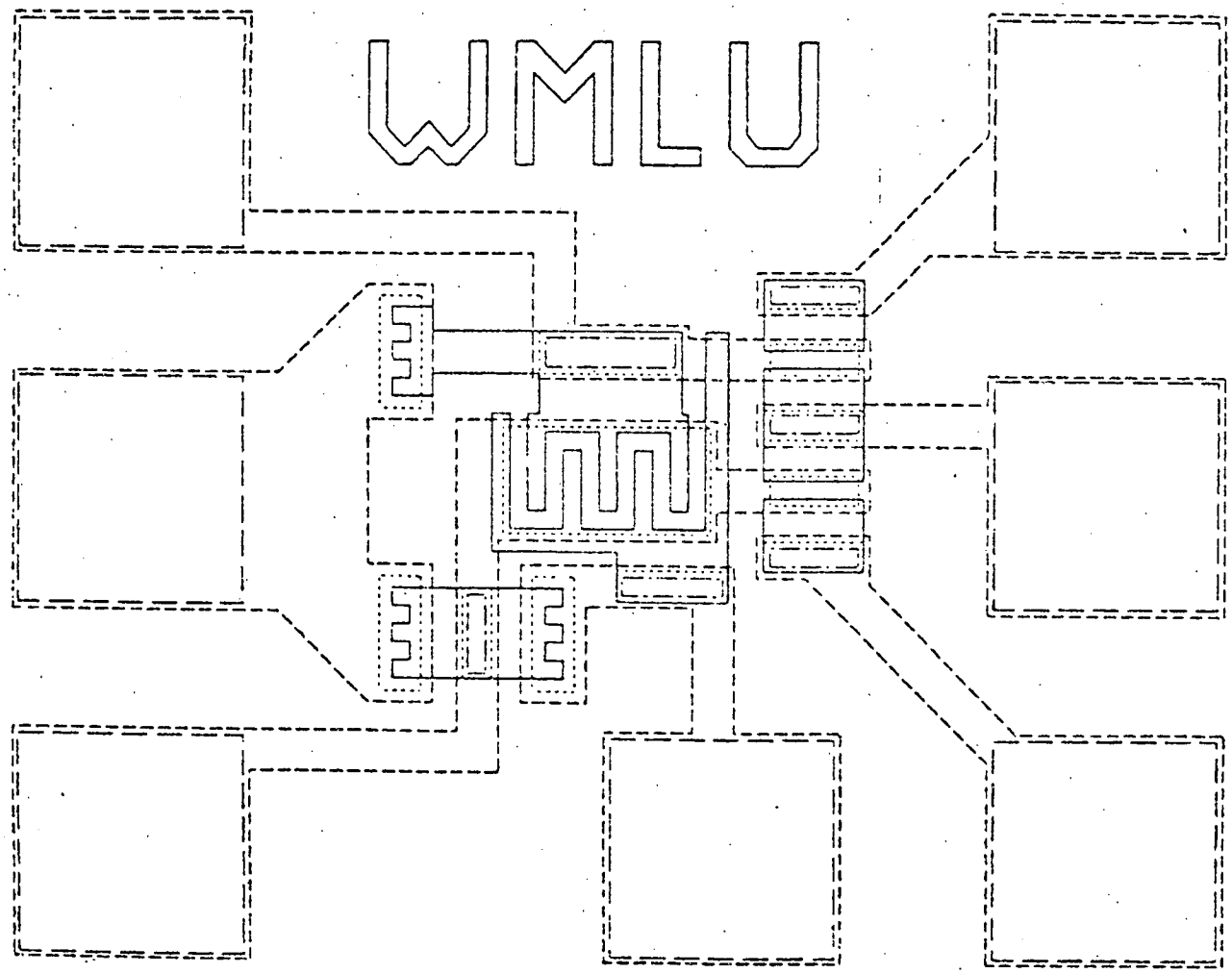


Fig. 3. 2. 3.

3.3 Extra features of GAELIC

Because the GAELIC system running on a time-sharing computer is always available i.e., does not have to be booked for a number of hours, say a fortnight in advance, it is possible to use it for short periods at a time. Good use can be made of such short periods to design smaller sections like group or repeat definitions.

A definition can be roughly drawn on squared paper or just a dimensioned sketch made which can in turn be coded up in the GAELIC manual input language without the repeat or group headers and entered into the computer. It can then be converted into the ring data structure correcting any syntax errors by either editing the input language file or by adding the corrected shapes at a later stage. The layout can then be plotted out, and any obvious corrections made on-line. More obtuse corrections can be made off-line and correcting tapes typed up before going on line to the time-sharing computer to process the correcting tapes and modify the data structure. When the part layout has been corrected and checked thoroughly, GAEL7A can be run to convert the ring data structure back into GAELIC manual input language. Here the necessary repeat or group headers and trailers can be inserted and the design stored away for future use. This method builds up a library of subpictures that can be used when required to design the full circuit and can, if suitable, be used in future circuits.

Another useful technique is to produce on say mask 15, the outline of the group and then when it comes to fitting the groups together to form the complete mask most of the work can be done with just the outline, which saves a lot of drawing time. This technique of using the outline can be taken further still by using the REPLACEGROUP order word (see GAELIC users manual). The outline in this case can be defined as the group and used in the main layout until the positioning of the groups and interconnections have been completed and then just prior to running GAEL9F etc. to produce drive tapes for the Ferranti Masterplotter, a series of REPLACEGROUP's can be entered into the ring data structure replacing the outlines with the full group definitions. This technique saves not only plotting time but also computing and storage costs.

3.4 Other GAELIC programs

There are several other programs in the GAELIC suite that interact with the ring data structure. These programs are briefly described below.

3.4.1 GAEL5A

This program plots all or part of a layout on a CALCOMP incremental plotter. This provides a permanent hard copy drawing of the layout that can be studied at leisure. The data to drive the plotter can also be written to a disc file or to magnetic tape and plotted later using a very small program which uses the minimum computer resources.

3.4.2 GAEL6A

This program extracts all the lines from a Ring Data Structure, joins them together to form polygons and returns these polygons to the Data Structure. This provides a layout description that only contains closed shapes and can subsequently be post-processed to produce drive tapes for photo-plotters.

3.4.3 GAEL7A

This takes the contents of a corrected layout in its Ring Data Structure and converts it back into the GAELIC manual input language. This is an extremely useful program as it provides a method of setting up a library of frequently used components.

3.4.4 GAEL8A

This program fulfils two functions, it removes all the discarded sections of the ring data structure and rearranges the data for most efficient processing.

3.4.5 GAEL9F

This program takes the data from the ring data structure and converts it into drive tapes for the Ferranti Master-Plotter. These tapes are produced on a high speed paper tape punch and can give either a MICROFILM plot of the layout or a set of 'cut and peel' masters.

3.4.6 Other post-processors

There are a number of post-processors that are very similar to GAEL9F which take the data from the ring data structure and convert it into drive tapes for various other tape controlled coordinatographs and mask making machines. Most of these have been written by students during vacational employment at Edinburgh University.

CHAPTER 4: Data Structures

4.1 The need for a data structure.

It is possible to write computer programs that will only process the one set of data built into the program and examples of these programs are often written as exercises during programming courses. A typical example is a program that prints out all the prime numbers between 1 and 100. The finished program, however, is of limited use. Most computer programs use a different set of data each time they are run and this data is read in by the program from a deck of cards, entered via on on-line terminal or read from disc or magnetic tape. In our simple example the program could be modified to read in the range of prime numbers to be printed out and so on one run the prime numbers between 100 and 200 could be produced and on another the numbers between 1000 and 1800. This data i.e. the range, must be entered into the computer and stored in the correct order.

The data that is stored in computer memory usually consists of numerical values e.g. the value of a resistor or the number of hours that an employee has worked during a week. It can, however, consist of strings of ASCII characters forming names or text, or can consist of bit patterns that form codes or symbols. Each item of data is usually referred to as a 'data element'. Data elements are stored in a computer memory in an organised way such that

the logical relationship between the elements is preserved and this organisation is known as a 'data structure'.

Data structures can vary in complexity from the very simple to the extremely complex depending on what the data represents and what processes must be carried out on that data.

Consider the data for a graph where the computer memory contains a simple series of y coordinate values for certain known x values. The word 'series' is used here instead of the more usual word 'list' because 'list' is used by the computer scientist to describe a particular type of data structure that will be introduced later. The correct graph will only be obtained from the data if the coordinates that are stored in the computer are plotted in their correct order: any other order would give a different graph. Thus the data is structured in a sequence of y coordinates and is usually stored in the computer in an array. This forms what is probably the simplest data structure. Programmers use arrays without realising that they are actually data structures, consequently the term data structure is often reserved for the more complex structures that allow for more flexibility when processing data. The processing of data for the graph is simple and straightforward: data is read into the computer in sequence, stored in the same sequence in an array and then processed to produce the graph. It is possible to change the values of certain coordinates but coordinates cannot be added or deleted.

However, the processing required on the data for other applications can be far more complex, for example, it may be necessary to preserve the hierarchical nature of the data or to delete from or add to specific positions in the structure. A simple array will not hold a structure capable of handling these facilities and more complex structures must be used. Any data structure that holds the description of an integrated circuit layout must be capable of efficiently plotting out the layout and of modifying it by adding, deleting and changing shapes. This must be done without making the data structure too big and a compromise must be obtained between the size of the data structure and the efficiency of the various operations. The structure must also maintain the heirarchical nature of the layout. The specific requirements of an interactive system for the design of integrated circuit layouts were described in Chapter 2 and it will be realised that most of these requirements are common to many other interactive graphic systems except that the amount of data required to describe an integrated circuit layout is so large that it cannot normally be held in the core memory of a computer. The various types of data structure will now be described and it will become obvious how well the requirements can be met.

4.2 Types of Data Structures

Excellent introductions to the subject of general data structures are given by Knuth in his book [ref 4.1] and in the paper by Dodd [ref 4.2]. These introductions are for general data management and are too broad based to be considered in detail here. However, there is a paper by Williams [ref 4.3] which deals specifically with data structures for computer graphics systems. As the interactive design of integrated circuits is mainly concerned with computer graphics, his paper is worth discussing in greater depth. It is an excellent review of the types of data structures that exist, concentrating on those used in computer graphics systems and of the various computer languages that have been used to handle the structures. The various languages used are dealt with later in this chapter and the present discussion concentrates on the various types of data structures that can be used.

Because of the differing terminologies used by the various workers in the field, it is necessary to define the terms that will be used. We have already met the terms 'data element' and 'data structure' and their meanings. However, when a data element occupies one word of computer memory, it is often loosely referred to as a 'word'. If a word of computer memory is used to contain more than one data element then the word is said to be split into 'fields', each field is therefore a data element. A 'record' is used by Williams to describe a collection of data elements that are stored in contiguous

(consecutive) memory locations but the term is usually used in data structures associated with input/output processes and only occasionally in more general data structures. The two terms used to describe a collection of data elements in contiguous memory that are in more general use are 'block' and 'bead'.

Dodd has postulated that all data structures can be constructed from three basic types. These are the sequential, random and list data structures and consequently it is worth considering these three types in detail.

4.2.1 Sequential Data Structure

This type of structure consists of a sequence of records or data elements. Any particular record or element is accessed by searching sequentially through the structure until the appropriate information is found.

As Williams points out, present computer memories are one dimensional in access, memory locations are sequentially numbered and the computer hardware is designed to access data serially (this last process is obviously interrupted by software when necessary). This means that processing data in a sequential data structure is particularly efficient as the mechanisms to do it are already built into the computer. However, there are unfortunately some disadvantages. If all the records in

the data structure need to be processed each time the structure is accessed then this can be done extremely efficiently. If, however, certain records are to be ignored when processing data then the efficiency of the process will fall, depending on the number of ignored records.

Probably the main feature of interactive graphic systems is that of regularly adding or deleting records from the data structure and any structure used must be able to cope with this feature. This is again an area where the sequentially data structure has problems. There are two possible methods that can be used to efficiently delete a record. The first is to change the initial data element in the record to indicate that it is to be ignored and the second is to re-create the structure without the particular record. The first method has the pre-requisite that there is a number that can be entered into the first element of the record to indicate that the record is to be ignored. This number must be unique i.e. outside the range of coordinates and other markers. The second method is time consuming as the new data structure is created by copying the original until the record is reached, ignoring the record and then copying the remainder of the structure.

If a new record is required to be added then there are again two possible approaches, namely to add the record at the end of the structure or to re-create the structure with the record in the middle. The first method

is an easy operation but the position may not be a possible one for the record. It may for instance have to be near other records sharing a common attribute. In this case the second method must be used: this consists of copying the original structure until the required position is reached, adding the record and then copying the remainder of the data structure. This second method is obviously time consuming.

The time taken for the methods requiring the re-creation of the data structure may not be not critical providing the two structures are held in core but the time will be significant if secondary memory has to be used.

These problems with the deletion and addition of records were sufficient for Williams to discount the sequential data structure as one to use in an interactive graphics system, however, as will be seen later there are at least two integrated circuit design systems that use a form of this structure.

4.2.2 Random Data Structure

In a random data structure, an address in core or on disc is allocated to each block of data, and each block is stored in memory starting at that address. The data can be subsequently retrieved from that address.

The simplest way of assigning the address is for it to be supplied by the programmer and specified by the program each time that it is required. This method is not practical for variable size data structures with variable size blocks as the amount of memory allocated for each block must be the maximum that any block could possibly require and the maximum number of blocks must be accomodated. The method is, therefore, extremely extravagant in the amount of space required.

A more flexible system of assigning the address of each block is required. The usual system is to create a table or array of block names and associated addresses. Each time a particular block is required, the table is referenced and the corresponding address is found. The data for the block is then retrieved from that address. In this system, the addresses are calculated by the program as required and not specified initially. Consequently the addresses are allocated so that they just leave room for the data. The number of entries in the table are the same as the number of blocks actually used. This table is more formally known as the 'symbol table' or 'dictionary' and is the most general method of using random data structures.

Present day integrated circuits are very complex and contain many components and consequently require very large amounts of data to specify the layout. If a random data structure is used to hold the data then the symbol table becomes very big and considerable computing time is

expended finding the address of any particular block from the table.

Another problem with the random data structure using a basic table is that it is not ideally suited for interactive graphics because of the problems of up-dating the structure. It is, however, easier to update than a sequential data structure as only the symbol table need to be updated instead of the whole data structure.

There is another method of constructing the symbol table which incorporates the storing of the position of the entry in the table for the following block as well as the address of the present block. A given block can then be found by following this chain of table positions and their respective addresses. This speeds up the search through the table as only table entries for blocks with a common attribute need be on the same chain, other entries for other blocks with differing attributes being held on different chains. This method also has advantages with updating, as the position of the next entry in the table can be altered so that a particular block is bypassed or included in the correct position. This method is therefore far more flexible and results in shorter search times than the sequentially ordered table. It has the disadvantage that the table is bigger because of the extra table positions stored.

Another method of using random data structures is to use a technique known as 'hash coding'. Here instead of using a table to contain the name of a record and/or its address, the program treats the name of the record as either a number or series of numbers and performs some arithmetic operation on these numbers to give a result. This result is then used as the starting address at which the block is stored. The arithmetic operation is known as 'hashing' and one of its problems is that two or more names can hash to the same address. This is called a 'collision' or 'conflict' and there are many methods of dealing with these conflicts which are described by Morris [ref 4.4]. The problems of conflicts and the sizes of the blocks make this method of using random data structures difficult to program in Fortran or Algol. There have been users of hash coding in graphical applications reported by Feldman and Rovner [ref 4.5] that use the language LEAP which is based on ALGOL.

These random data structures do have limitations at the moment like relying on software to associate names with addresses. However, if computers are eventually built with large associative memories then these data structures will come into their own.

4.2.3 List and Ring Data Structures

These data structures are similar to random structures using symbol tables in that they built up using a series of blocks of data which are located at specific addresses and these addresses are stored elsewhere in memory. However in list structures the address is kept in a data element in the previous block and the address is usually refered to as a 'pointer'. Hence blocks of data having similar attributes are 'chained' or joined together by means of pointers, the individual blocks, however, may be randomly scattered throughout the memory. A series of blocks chained together by means of pointers is known as a 'list'. A block of data can be on several lists and will consequently have several pointer chains passing through it. This is using the term 'list' in its broader sense, in 'list processing' using 'Lisp' [ref 4.5] where the block is just one computer word long.

The main advantage of list data structures is the speed in which they can be modified or updated. If a new block is to be added in a specific position in a list, the block itself can be added at any convenient place in memory then the pointer in the previous block is changed to point to the new block and the pointer in the new block set to point to the next block in the sequence. Similarly if a block is to be deleted then the pointer in the previous block is set to point to the block after the redundant one.

An example of a list data structure is shown in fig 4.1. which shows three separate lists combined together. The first list 'A' can join together the data describing a series of triangles, the second list 'B' may join together a series of rectangles and the third list 'C' may join shapes whose area is greater than a given value. The actual layout in memory may be fragmented as shown in fig 4.2. Deleting the block for triangle 'T2' requires the simple process of changing the pointer in block 'T1' to point to block 'T3'.

A more difficult block to delete is one that is situated on more than one list such as 'T3'. Here not only must the block be deleted from list 'A' by changing the pointer in block 'T2' to point to block 'T4' but it must also be deleted from list 'C' by changing the pointer in block 'R2' to point to block 'R4'. This is difficult as it means noting which block is deleted when traversing list 'A' and then checking each block in turn on list 'C' to see if it is the deleted block. One list is obviously being traversed when the block to be deleted is detected and so by keeping the address of the previous block on that list, the block can easily be removed by changing the pointer values. The other list, however, will have to be specially scanned from the beginning to find the block to be deleted and this could be a time consuming process if there are many blocks in the list. There are also problems if the program does not know where the list starts. In our example blocks could be on list 'C' or on

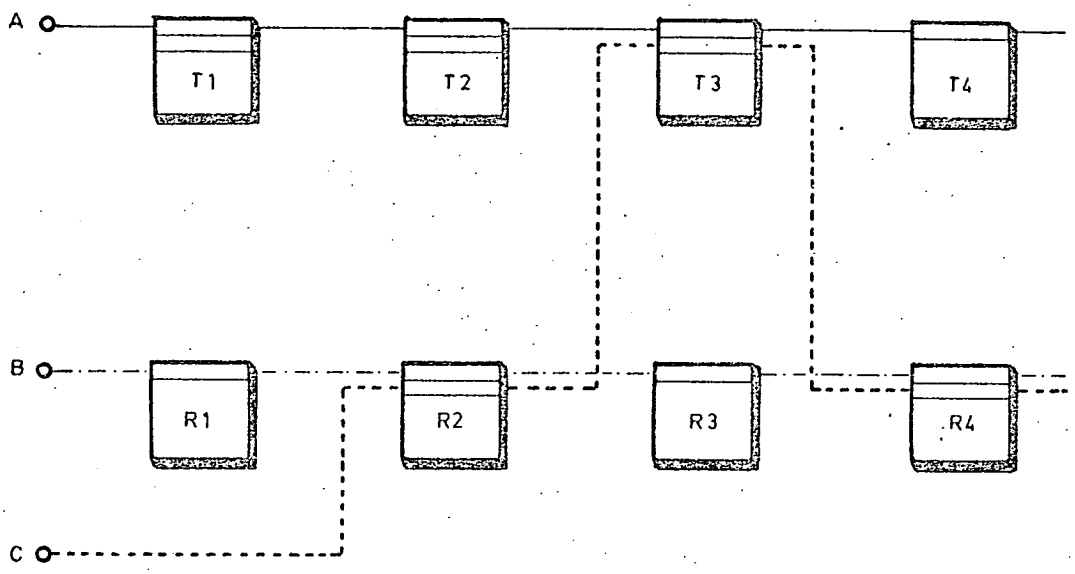


FIGURE 4.1 example of list structure

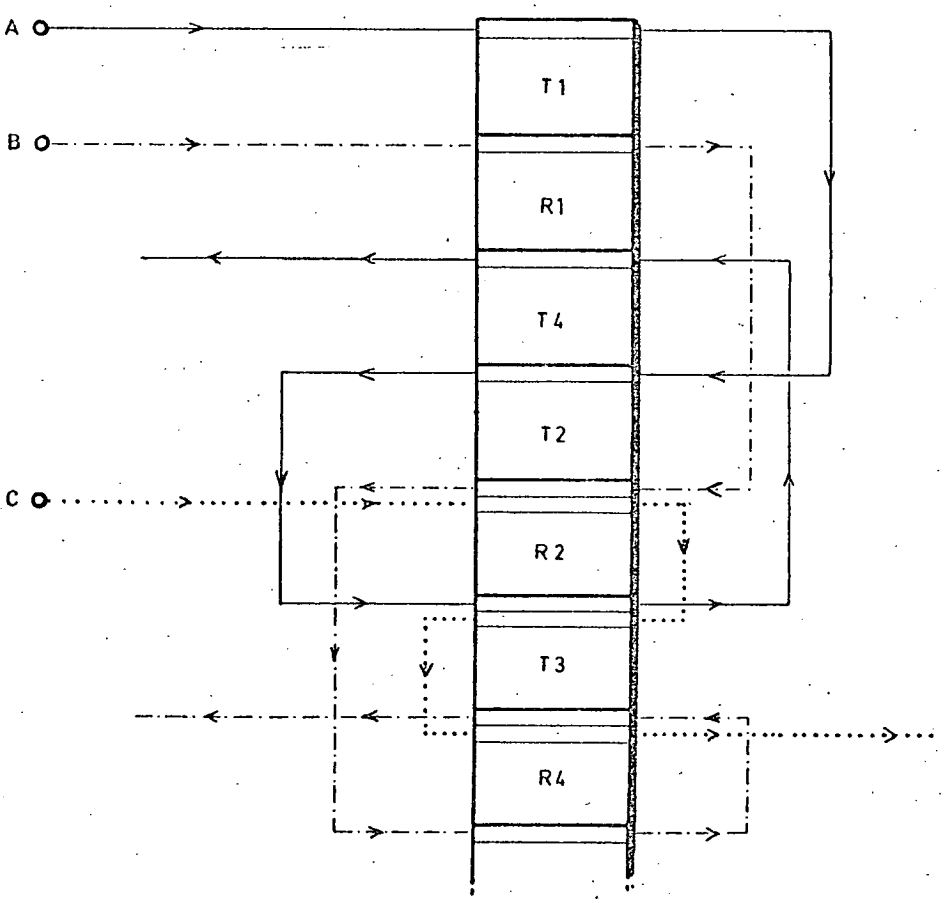


FIGURE 4.2 layout of structure in memory

other lists depending on the area of the shape and so it is conceivable that the whole data structure would have to be searched.

This problem can be overcome by use of 'forward' and 'backward' pointers between blocks so that not only does a block have a pointer to the following block, it also has one to the preceding block. This speeds up deletion but does have the disadvantage that it requires bigger data blocks to store all the pointers and hence requires a bigger data structure.

A special type of list data structure can be formed by arranging that the pointer in the last data block points back to the first. Thus the pointers form a 'ring' or 'circular list' and structures using these are referred to as 'ring data structures'. Ring data structures have been used in several different applications by different people and consequently has acquired several different terminologies to describe it. The term 'bead' is used instead of 'block' by some people presumably because of the similarity of a drawing of the blocks on a ring of pointers to beads on a necklace. However terms like 'keys', 'chickens', 'hens', 'mothers' and 'daughters' have been used for the same blocks. It is therefore essential to define the terminology to be used before proceeding further.

A 'bead' is a series of consecutive memory locations that are joined by 'ring pointers' to form a ring. One of beads has different attributes to the others and corresponds to the first block on a list and is known as a 'head bead' or 'ringhead bead' and the pointer in the head bead is known as the 'ring head pointer'. The first word in a bead is called a 'bead head' or 'head word' and usually contains data elements or 'fields', which identify what the bead contains and how big it is. Another type of pointer is used which instead of pointing to the next bead in the ring points to the head word of another bead, this type of pointer is known as a 'direct pointer'. A simple ring data structure is shown in fig 4.3 which illustrates many of these terms.

Usually a bead is divided into three parts, the head word itself, the ring pointers and the data. The head word usually contains the number of pointers words and the number of data words used in the bead. The data words only contain data such as numeric values and codes but it will be shown later that under certain circumstances direct pointers to other beads can be included thus reducing the size of the data structure.

Just as in the standard list data structurei where blocks can be on more than one list, beads in a ring data structure can be on more than one ring. The fact that rings are used rather than lists facilitates deletion of beads without having to resort to forward and backward pointers. Fig 4.5 shows the ring data structure to hold

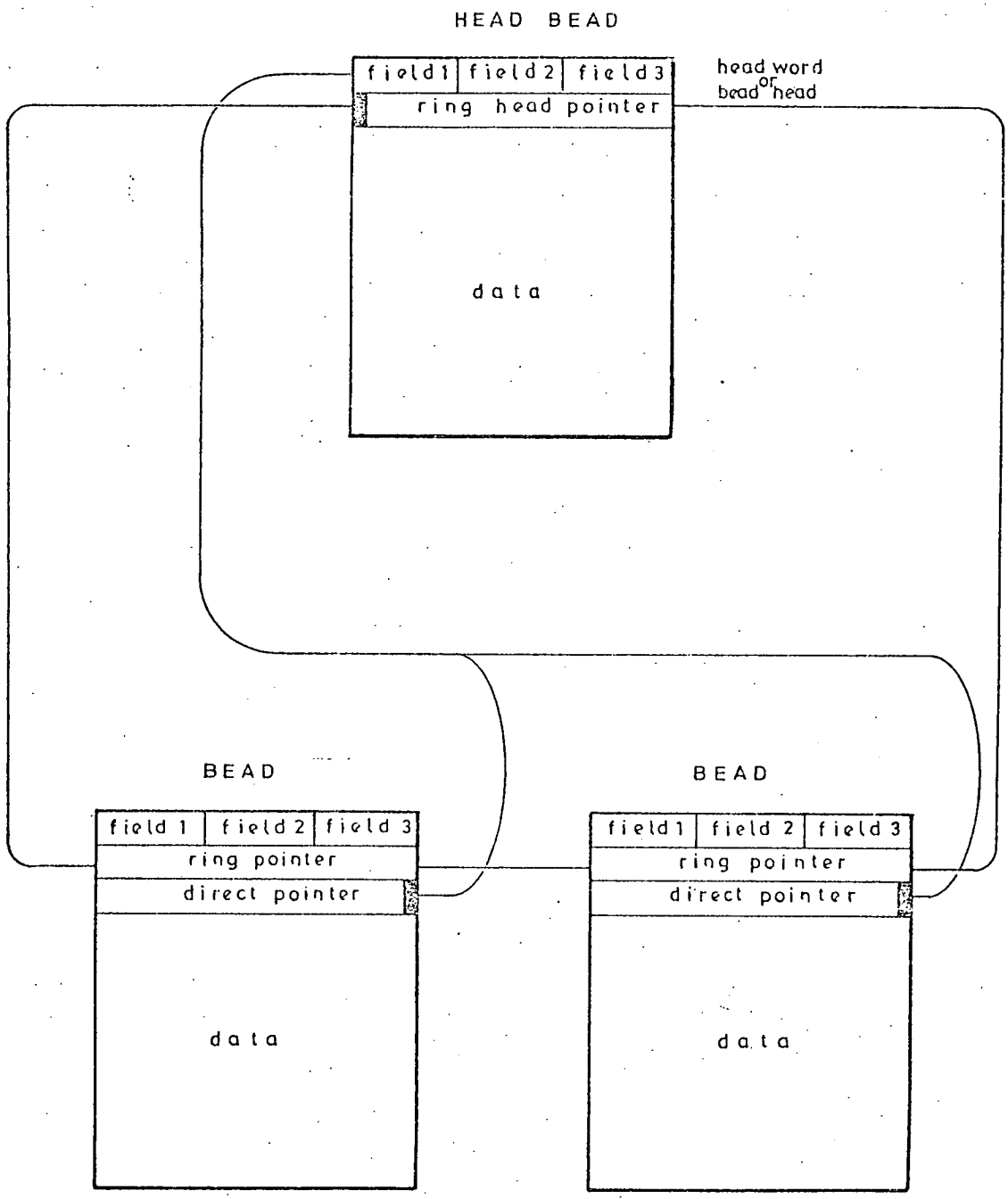


FIGURE 4·3 simple ring data structure

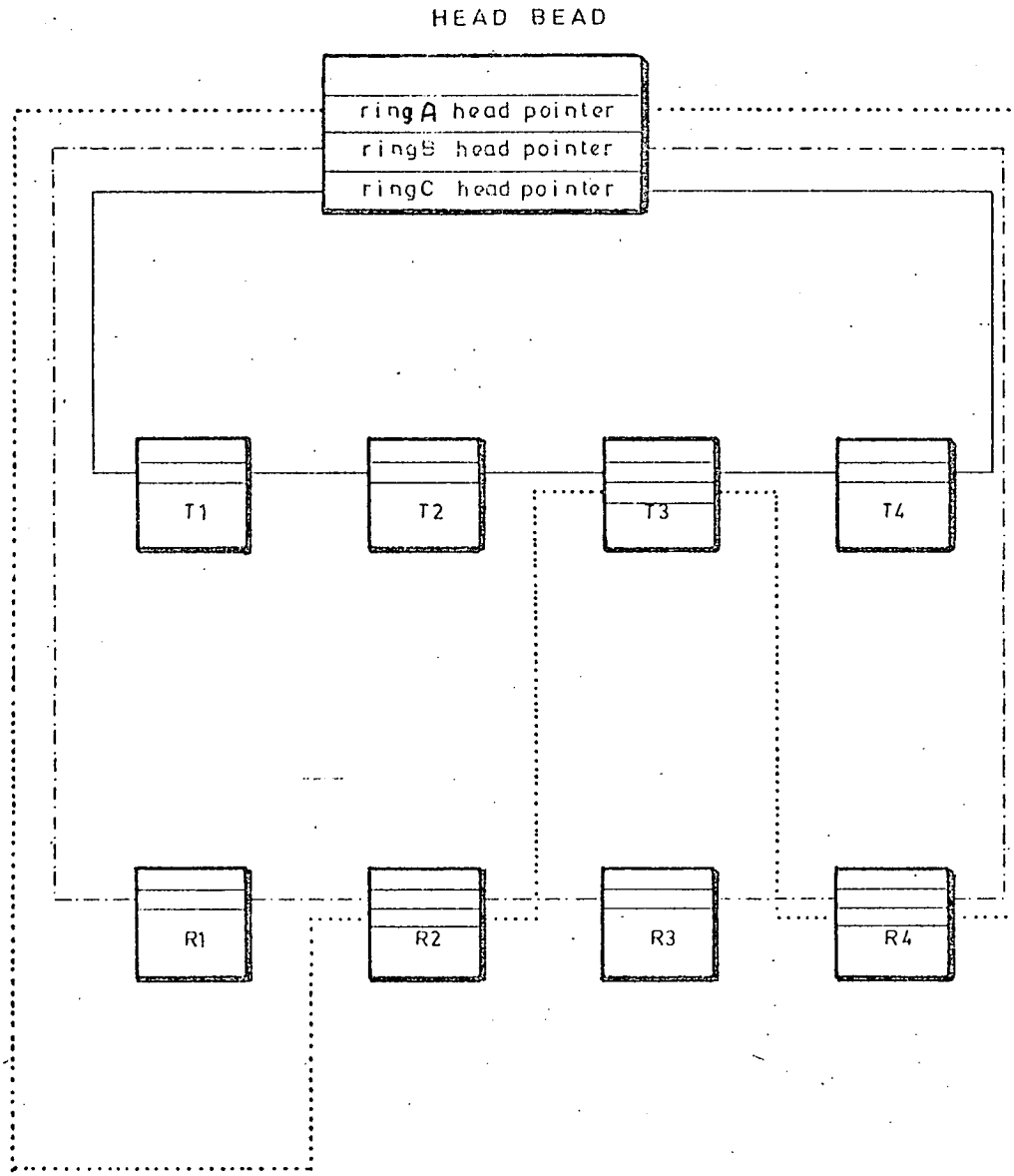


FIGURE 4·5 ring data structures for rectangles and triangles

the description of the triangles and rectangles used in fig 4.1. Now if bead T3 is to be deleted and it is detected by following ring A it can be deleted from ring C by examining each bead in turn i.e. R4, the head bead and R2 until the pointer to T3 is found. This can then be modified to point to R4 and the deletion is complete. Only one ring had to be processed. The other advantage of the ring data structure over the list is that it is possible to find the head bead of a ring by following the pointers round the ring until the ring head pointer is reached.

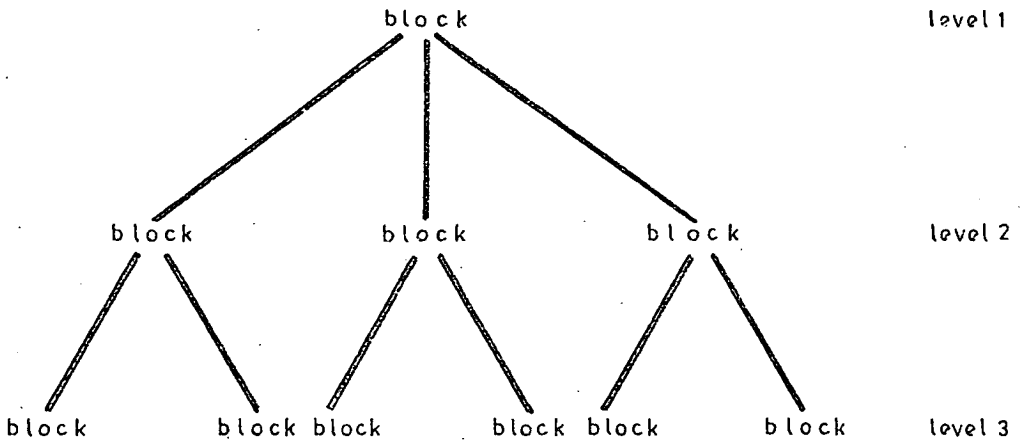
An interesting psychological point arises with ring data structures. Because of the varying terminology used in describing them, there is an assumed air of mysticism about them. This appears to affect programmers in one of two ways, they will either fully accept ring data structures and use them regularly even if their use is not fully justified or they will avoid using them at all costs. However, ring data structures are extremely powerful under certain circumstances particularly for interactive applications where data in continually added or deleted. They also have advantages when it comes to handling a large number of different types of data e.g. the data for an electronic circuit analysis program consists of resistors, capacitors, transistors, voltage sources etc. Normally if these are held in Fortran arrays, there is a limit on the number of resistors, a limit on the number of capacitors etc. It is therefore

possible to have a circuit that is too big to analyse just because there are too many transistors even though the space for capacitors is empty. Using a ring or, for that matter, list data structures, the program can be written so that it is the total data size that matters, not the size of the individual components.

4.2.4 Complex Data Structures

The sequential, random and list data structures just described can be used or combined to form more complex structures. The best known of these complex structures are the 'tree' and 'hierarchical' data structures, both of which have been used in graphic applications.

Graph Theory describes a 'tree' as a graph which has no circuits (or rings in our terminology). The computer form of a tree consists of a series of blocks spread randomly throughout the memory and a series of pointers chaining them together and an example is shown in fig 4.6.

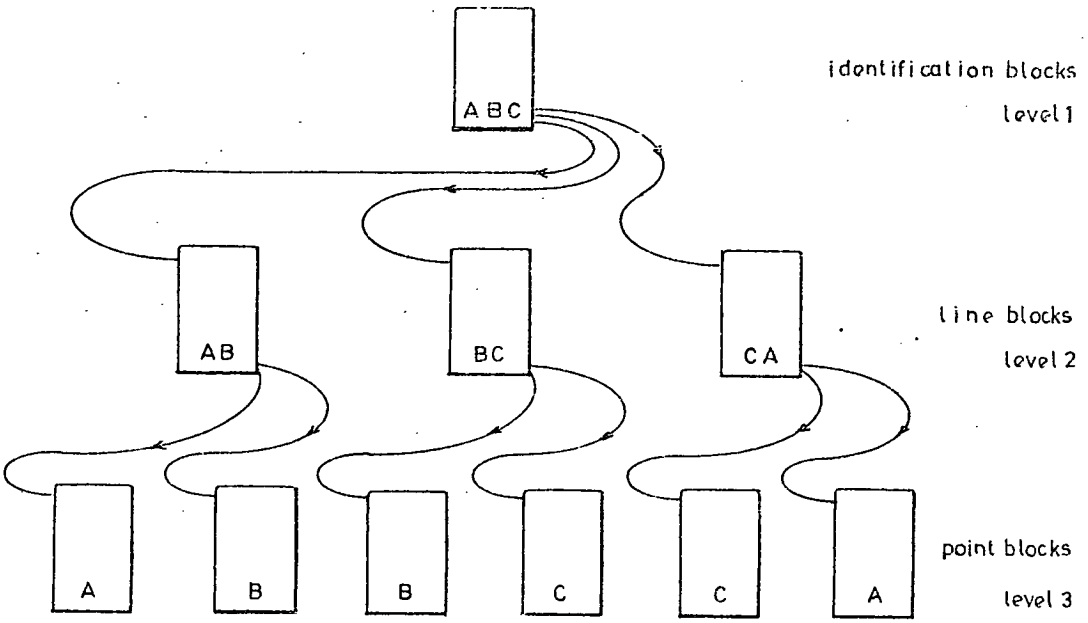


a tree structure

Fig 4.6

The structure starts with an identification block at the top of the tree: this block contains certain parameters describing the structure such as its name or size and also pointers to shapes at a second level. These second level blocks contain certain parameters describing their function etc and also a series of pointers to blocks at a third level and so on.

The data to describe a triangle could be held in the basic tree structure shown in fig 4.7.

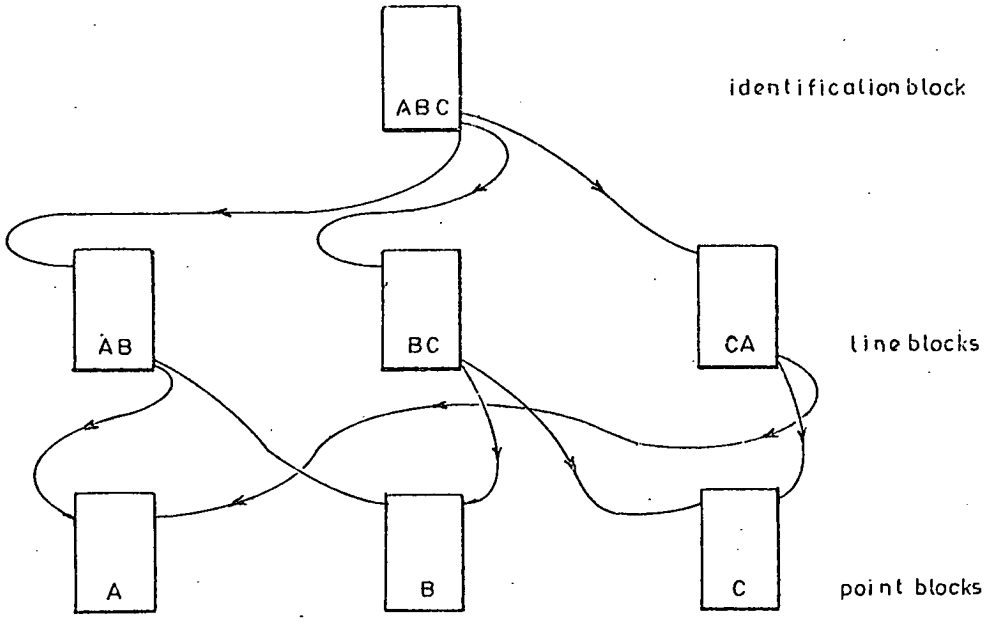


tree structure for triangle

Fig 4.7

The identification block would hold the name of the triangle ABC and pointers to the blocks for the lines AB, BC and CA. These line blocks in turn would contain the name of the line e.g. AB and pointers to the coordinate blocks A and B. The coordinate blocks contain the coordinates of the appropriate point.

Probably the most important point to notice in fig 4.7 is that the coordinates of each point are recorded twice. This is obviously wasteful of storage space and a simple modification of the basic tree structure is made that allows more than one block at a high level to point to the same block at the lower level and the data structure is changed to that shown in fig 4.8.



modified tree structure for triangle

Fig 4.8.

The flexibility with the pointers can be taken a stage further where the pointers in a block at a given level can point to blocks on the same level. They cannot however point to blocks at a higher level or this would give the possibility of forming rings.

The system of different levels provides a grouping or subroutining facility so that the definition of subpicture can be stored in the structure starting at a given level and instances of the subpicture can be called by inserting blocks at a higher level which contain pointers to the definition and the coordinates of the origin of the subpicture.

The basic version of the tree data structure is difficult to modify particularly if blocks are to added or deleted. If an extra block is to added at a given level, then the block at the higher level must be replaced by a

bigger one to accomodate the pointer to the new block. This means that the value of the pointer in the block at the next level above must be changed to point to the replacement block and so the process of adding shapes becomes extremely complex.

The tree data structure can be made more flexible by using list structures rather than random structures as shown in fig 4.9.

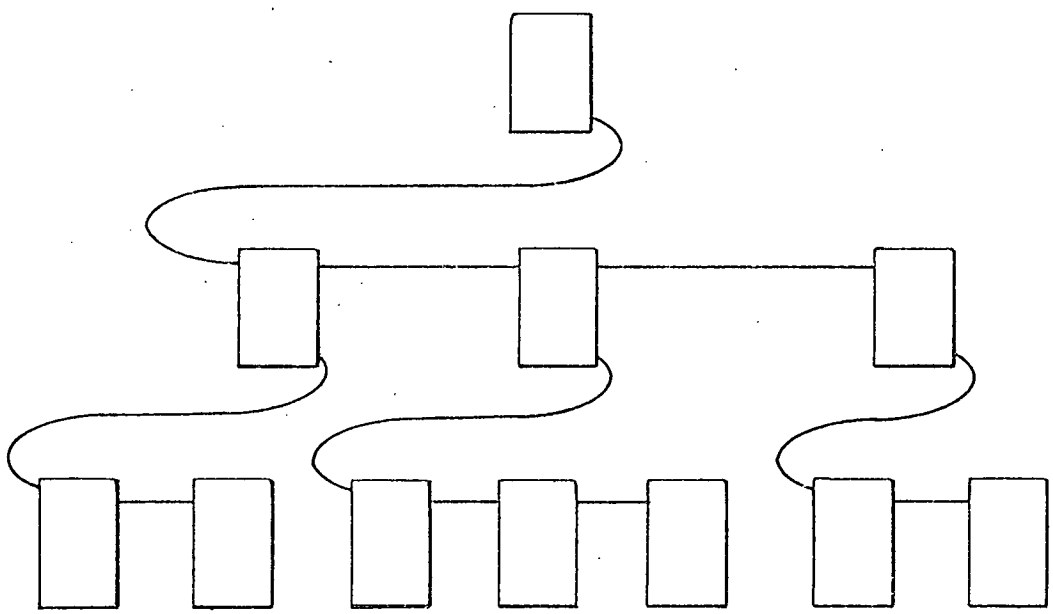


Fig 4.9 Tree Structure using Lists.

Here the identification block contains only a pointer to the first block on the second level, the first bead, however, contains a pointer to the second bead on that level and so on. Now all that is required to delete a bead is to change the pointer in the previous bead.

No description of this version of the tree structure has been found in the literature though it is obviously a far more flexible system. This is probably because it is

so similar to the 'hierarchical' data structure which uses rings rather than lists and which consequently has certain advantages.

The 'hierarchical' data structures like the 'tree' data structures are created from beads or blocks on different levels but in this case the beads are on rings. The term 'hierarchical' is not often used and most hierarchical data structures are simply refered to as 'ring data structures'. Consider the drawing shown in fig 4.10 which consists of a triangle BAC sharing two line segments AB and BC with two other triangles

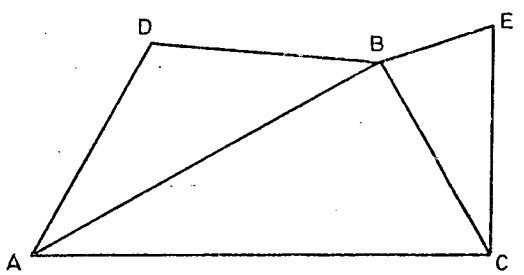


Fig 4.10.

ABD and BCE respectively. The hierarchical structure to hold the data for this drawing could be as shown in fig 4.11.

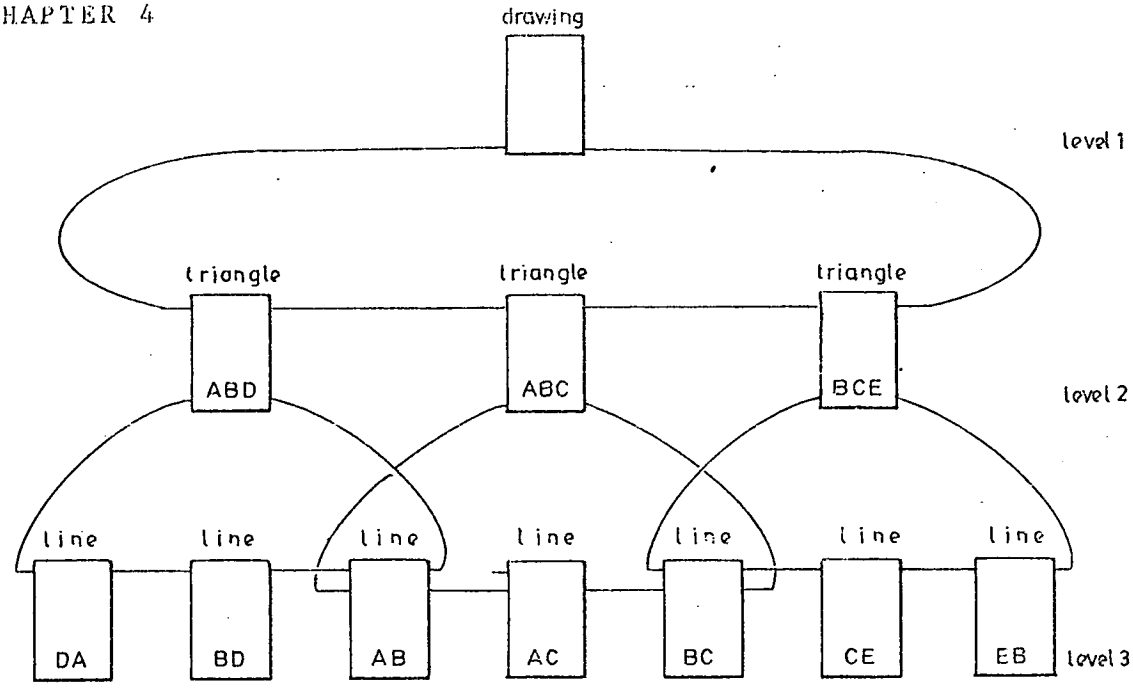


Fig 4.11.

Here the identification bead on the first level contains the name of the drawing and perhaps its size. It also contains the ringhead pointer to a ring of 'triangle' beads at a second level. Each triangle bead could contain the name of the triangle e.g. ABC, the ring pointer to the next triangle bead and also the ringhead pointer to a ring of 'line segment' beads that are used to create the triangle. Each line segment bead contains the ring pointer to the next line bead and the coordinates of the line segment.

Certain line segment beads are common to two triangles, for example AB occurs in triangles ABC and ABD and so the line segment bead must be on both rings. In order to access the coordinates of the line segment, it is essential to know which ring is being traced or rather where the coordinate data is with respect to the ring pointer. To accomodate this feature the pointers are

numbered and each pointer word is consequently divided into two fields, the first field contains the pointer number or offset from the headword, the second the address of the next bead. This obviously restricts the number of bits available for the address and hence the maximum size of the data structure.

This numbering of the pointers was a feature of the first ring data structure used by Sutherland [ref 4.6] in the Sketchpad program and the feature was also used by Evans and Katzenelson [ref 4.7] when they applied complex ring data structures to electrical circuits and the feature now appears as a matter of course in ring data structures. One of the novel features of the GAELIC data structure is that it is designed so that the pointers need not be numbered. This has the advantages that either larger data structures can be handled or else the same size structure handled on a computer with a shorter word length. This technique has subsequently been used by McGuffin [ref 4.8] in the automatic routing of P.C. boards. Dr. P.F.A. Reilly did not use use numbered pointers in some of his data structure design work [ref 4.12]. Instead he arranged that all the pointers on a ring were in the same position in their respective beads regardless of the bead type. This was found to be restrictive and made data structure design difficult.

4.2.5 Other Data Structures

There are other data structures that have not yet been considered for graphics applications and probably the most imortant of these is the Set Theoretic Data Structure (STDS). This data structure was described by Childs [ref 4.9] and has been partially implemented. The data is sorted into mutually disjoint sets which are known as 'generator sets'. Generator sets can be joined together to form 'composite sets'. There are no explicit pointers used between the various sets of data and so sets can be moved about in memory independent of each other. This technique of not having explicit pointers could be applied to other data structures and is inherent in the sequential data structures.

Certain set operations are used in the STDS to retrieve the data and set theory questions can be answered about the data. It appears to be a useful data structure for such applications as statistics and personel management. For example, if certain standard facts about each person employed by a company are stored in the data structure, say whether a person is married and how many children he has: then it is possible to answer such questions as 'how many men are married with two children?'. However, the data stored must be amenable to arrangement in set form and this restricts the range of applications. It cannot, consequently, be easily used for interactive graphics work.

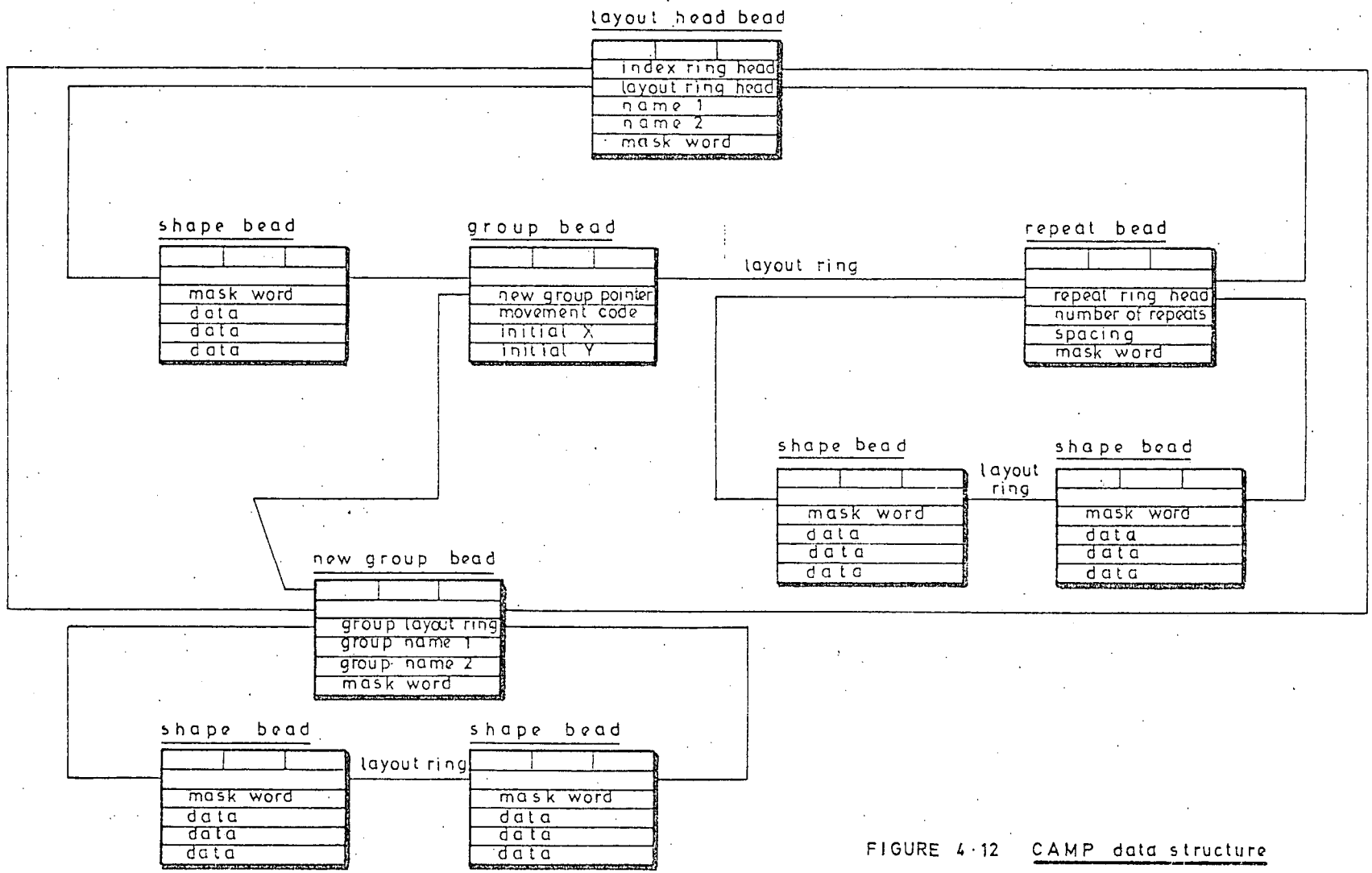


FIGURE 4·12 CAMP data structure

4.3 CAMP Data Structure

The CAMP data structure is mainly based on the work of Evans and Katzenelson [ref 4.7] and was designed by Wood [ref 4.10]. It is a hierarchical structure shown in fig 4.12. The input data to the CAMP programs can be descriptions of rectangles, polygons, circles and lines as well as the group and repeat structures and is described in detail in Chapter 3. The basic shapes are, however, all converted into polygons when stored in the data structure. A typical polygon bead is shown in fig 4.13.

2	1	n
Shape	Ring	Pointer
Mask Word		
Initial X Coordinate		
Initial Y Coordinate		
	"	
	"	
	"	
	"	
Final Y Coordinate		

Fig 4.13 Typical 'CAMP' Polygon Bead

The first word in the bead is divided into three fields, the first field contains a number to indicate that it is a polygon rather than a group call or repeat bead. The second field contains the number of pointers which for a polygon is always 1 and the third field contains the number of data words.

The next word contains the ring pointer to the next shape on the ring. It again is divided into 3 fields, the pointer number, the type of pointer and the actual address.

The next word contains the 'maskword' which is a bit pattern indicating on which masks the polygon appears.

The remaining words contain the actual coordinate data describing the polygon.

The head bead for the complete layout shown in fig 4.12 has the bead pointers of two rings, an index ring and a layout ring. The layout ring contains the beads for all the shapes in the main layout. It also contains 'group' beads which are calls to an instance of a group. This contains a direct pointer to the definition of the group and also contains the orientation of the group and the coordinates of the origin of the instance. The ring also can contain repeat beads, which contains the number of patterns, the spacing between them, and a maskword indicating which masks contain shapes to be repeated. It also contains a head pointer to the ring of shapes to be repeated.

The index ring contains all the new group beads, one bead for each group definition used in the layout. Each bead contains the name of the group and a maskword. This time the maskword indicates which masks contain shapes in the definition. It also contains the head pointer of a ring containing all the shapes in the definition.

The process of plotting out the shapes on mask 1 consists of starting from the head pointer of the layout ring and examining each bead in turn, by following the layout ring pointers. When a bead is found, the contents of the bead head are examined to check that it is a polygon and to find the number of data words. The maskword is then examined to check whether the polygon is on the required mask i.e. mask 1: if it isn't, the shape ring pointer is followed to the next bead. If it is on mask 1, then the shape is plotted out. This process is repeated until the shape ring pointer points back to the ringhead.

It should be noted here that every shape is processed to an extent regardless of whether it is on the required mask or not. If we assume that an integrated circuit consists of 5 masks and that the shapes are distributed equally between the masks, then the time spent processing 80% of the shapes when plotting a given mask is unproductive. This obviously is a waste but as the CAMP data structure was designed to be core resident, the actual time taken to process the extra shapes is small. However, if the data structure were disc based with only a

few pages in core then the time required to do all the extra disc reads would be appreciable.

The layout ring contains two other types of bead besides polygons, these are the group call bead and the repeat bead. If a group call bead is encountered when plotting a mask, the direct pointer is followed to the group definition. The maskword in the definition bead is examined to see if the group contains any shapes on the required mask. If there are no shapes present, the program returns to process the layout ring. If there are shapes, then the program examines each shape on the group layout ring, plotting out those on mask 1 before returning to the main layout ring.

When a repeat bead is found, the maskword in the repeat bead is examined to see if any shapes on mask 1 are repeated. If there are, then the number of repeats and their spacing are obtained from the bead. The shapes on the repeat layout ring are processed the required number of times with the appropriate modifications to the coordinates.

The data structure has three disadvantages:

1) the processing of redundant shapes as described above.

2) The fact that all shapes are stored as polygons. It only requires two pairs of coordinates to uniquely specify a paraxial rectangle but requires at least three pairs of coordinates to store the same shape as a polygon. An

integrated circuit design is typically made up of 30% of rectangles so this can cause an appreciable increase in the size of the data structure.

3) As the structure is core resident, the size of the layout that can be designed depends on the amount of core available. The average computer, therefore, is not capable of handling the large integrated circuits designs that are now being manufactured.

4.4 Marconi Myriad Data Structure

A sophisticated hierarchical data structure was implemented on a Marconi Myriad computer equipped with an X2000 graphics system by S Bird [ref 4.11]. This data structure was initially designed for a general purpose drawing program and was later modified slightly for use in integrated circuit layout. The structure is based on the work done by Sullivan [ref 4.6] for the Sketchpad system and includes Sullivans 'constraints' which ensure that certain shapes in the layout maintain a certain displacement from other shapes. The Myriad Data Structure takes the hierarchical principal to its logical conclusions. Instead of the line segments bead containing the values of the end coordinate as shown in fig 4.11, the bead contains the ring pointers to two 'point' beads one for each end of the line. These point beads in turn each contain the ring pointers to two 'value' beads, one for the x coordinate value and one for the y.

All the point beads and all the value beads are also on rings with their head pointers in the drawing head bead.

It does, therefore, create rather a complicated data structure for a simple drawing. For example consider the drawing shown in fig 4.14 which shows a horizontal line joined to a vertical line.

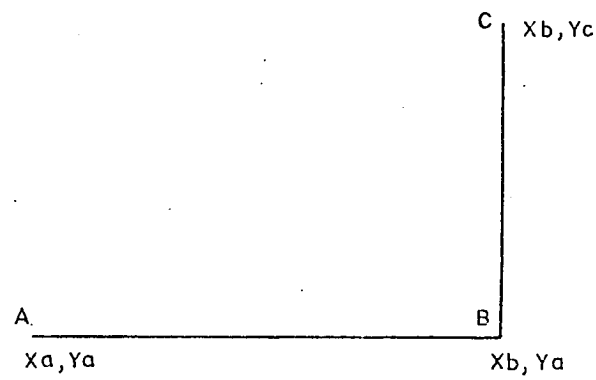


Fig 4.14

The data structure for the two lines is shown in fig 4.15. It can be seen that the line AB is constrained to be horizontal by making point A and point B share the same y coordinate value. Line AB is constrained to be joined to BC by making both line segments share the same point bead.

The initial Myriad Data Structure was modified by S. Bird to cope with the different masks encountered in integrated circuit layouts. This was accomplished by adding an extra 'maskword' at the end of a line bead. The data structure for the rectangle on mask 1 shown in fig 4.16 is shown in fig 4.17.

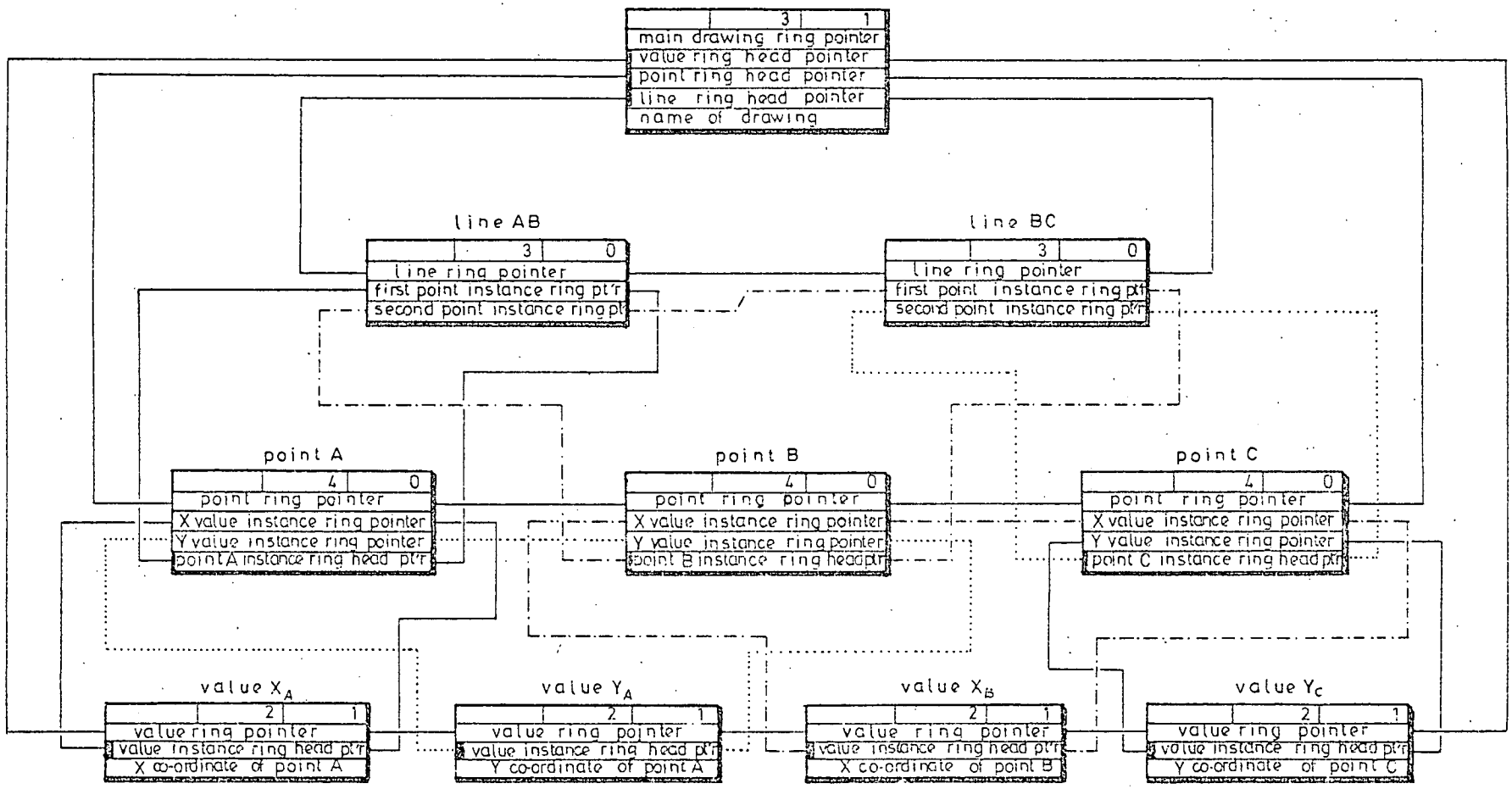


FIGURE 4·15 original myriad data structure for two line segments

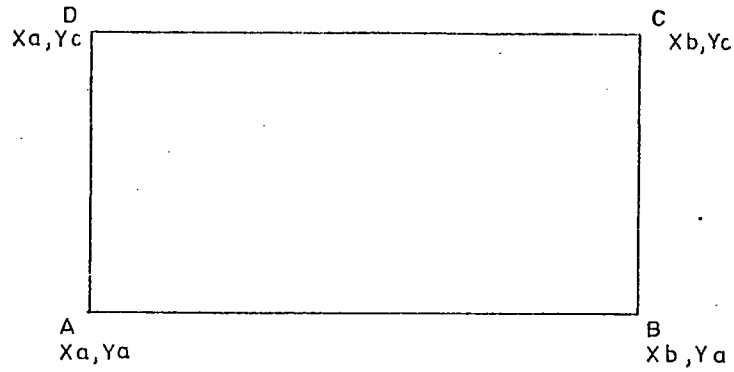


Fig 4.16 Rectangle on Mask 1

The structure does contain group facilities and so a series of shapes that are used frequently need only be defined once and then instances are called in the required positions on the layout. The group structure is very similar to that used in CAMP except that there is no special main layout bead; everything is regarded as a group including not only the main layout of the circuit itself but also the main layouts of any other circuits held on disc. It does, however, have facilities for deleting a group definition and all the instance or call beads. This is done by having a group instance ring whose ring head is in the group definition that joins all the instances of the definition.

The data structure is disc based with certain 'pages' of the data actually held in core at any one time and so is therefore capable of handling very large circuits.

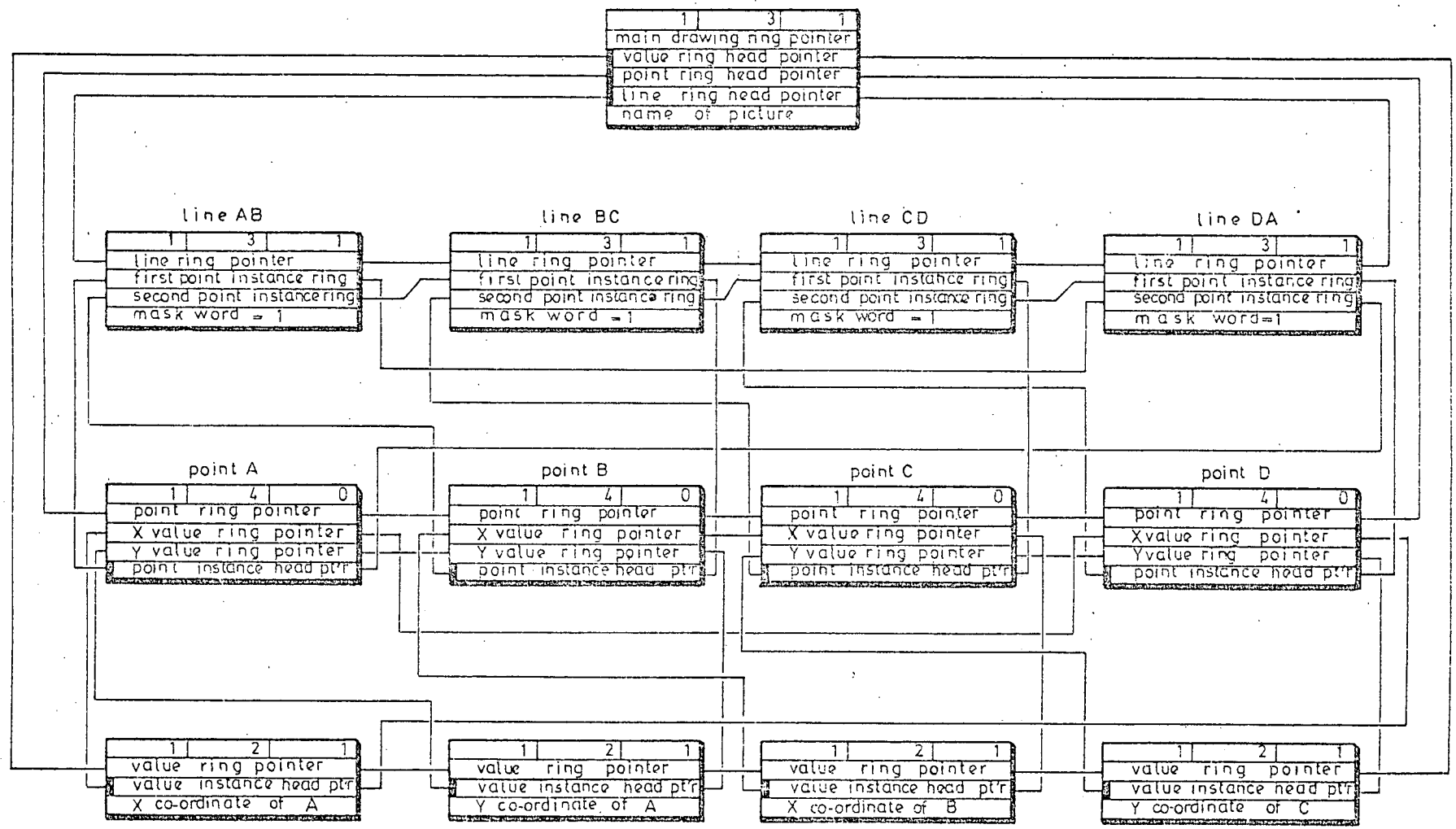


FIGURE 4-17 modified myriad data structure for rectangle on mask 1

The structure is extremely flexible but does suffer from problems of having to number the pointers and of having a high proportion of pointers to actual data, requiring very large data structures.

4.5 Other Data Structures

The CAMP and MARCONI data structures were the only ones that were known that had been applied to integrated cicuit layout design when the initial GAELIC data structure was designed. Since that time other integrated circuit layout design systems have become commercially available but in general their data structures have not been described in the literature. However, by talking to the people using these systems a certain amount of information has been obtained and this is given below.

The Redac integrated circuit design program uses a sequential file for its main data structure and apparently the whole data structure is searched each time a new display file is created.

The Calma system also uses a sequential file approach but subdivides it into disc segments. The bounding rectangle of the shapes in each segment is stored as a segment header and is examined each time a window is plotted to see if shapes within the segment overlap the window. It uses similar sequential structures for the group definitions. This system can be reasonably

efficient on disc transfers if the data is entered in the correct order ie. all shapes for a particular area of the layout entered one after another.

The Applicon system makes more effort by sorting shapes according to their bottom left hand corner and writing them to specific segments on disc. Again the bounding rectangle of the shapes in the segment is stored and examined to see if the segment need be processed. It appears to have problems with large shapes i.e. large rectangles and polygons. A large shape in a segment causes the segment to have a large bounding rectangle and hence it is processed for most window sizes. Because the data is divided into fixed sized segments, on the Calma and Applicon systems there are restrictions on the sizes of polygons allowed.

The final GAELIC data structure described in the next chapter makes extensive use of the area concept to minimise the number of disc transfers. Since the work started, two other organisations have been found to be using area associations in their data structures. Bell Telephone Laboratories have produced what is effectively an area associated display file which at present is restricted to rectangles. The rectangles are sorted into areas depending on their size. There are, of course, large rectangles that overlap more than one area and these are catered for by entering them in each area. This approach was considered for GAELIC when the final data structure was being designed but was rejected as so many

areas had to be accessed when a large shape was moved.

I.B.M. research laboratories at Hursley have taken a similar approach to GAELIC and B.T.L. in sorting shapes into areas but have found another solution to the problem of the large shapes. They are split up into a series of smaller shapes by cutting the shapes along the area boundaries. This makes for efficient processing but does give a layout that is difficult to check as it is different to the layout actually entered.

CHAPTER 5: GAELIC Data Structure

This chapter is mainly devoted to the three data structures that have been used during the development of the GAELIC programs. It starts, however, by summarizing the requirements for the data structure that were developed in Chapter 2.

To minimise the amount of data that is held in the computer memory, all the redundancy that exists in the input data for the tape controlled coordinatographs must be eliminated. Rectangles must be described by the coordinates of a pair of diagonal corners and other shapes that are paraxial must be described by every other pair of coordinates. Also there must be facilities for repeating a series of defined shapes individually or on a matrix.

As the main feature of the GAELIC programs is the interactive phase, it is also essential to be able to perform all the interactive operations as quickly as possible, the interactive requirements, therefore, must be born in mind during the design of the data structure. The requirements were given in detail in Chapter 2 but are briefly:

1) To plot out all or part of the layout.

2) To be able to identify a point on a shape and either modify or delete the shape.

3) Identify the origin of the instance of a group and either delete it or change its orientation.

4) Identify that a shape or series of shapes are

repeated and be able to modify the number of patterns or their spacing.

The Wolfson Microelectronics Liaison Unit received a contract from General Instrument Microelectronics Ltd. to write a suite of computer programs to produce drive tapes for a tape controlled coordinatograph from either data tapes from a digitiser or from manually prepared data. These programs were to run on a particular commercial time sharing service. This service did not at the time have any random access facilities for data files used in Fortran programs and so it was not possible to use sophisticated data structures. This restriction resulted in a suite of computer programs known as PAELLA (Plotter Aided Engineering Layout of Linear Artwork) which uses the version of the sequential block data structure described below.

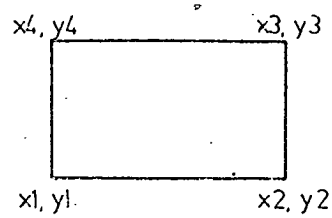
5.1 The Sequential Block Data Structure:

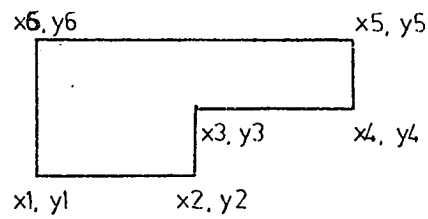
The sequential block is about the simplest data structure that can be used for this type of work. The structure is simply created by sequentially writing the blocks of data to arrays or disc files. In our case each block contains the data describing a shape and is written to an array or file in the order in which it appears in the input data. The blocks vary in length for different shapes e.g. a rectangle requires 6 elements and a 6 sided paraxial polygon requires 10 elements. An example of the

data for several shapes in a sequential block structure is shown in fig.5.1.

The basic sequential block data structure is not very efficient unless all the data can be held in an array in core. This is because the whole of the data must be searched sequentially to find a particular item rather than searching through the series of items having similar attributes. For example, if a particular shape on mask 1 is required then it is desirable to sort through the shapes on mask 1 and ignore the shapes on masks 2, 3 and 4. The main advantage of the data structure is that it is compact and so can often be held entirely in core when a more sophisticated data structure would have to be held on backing store. To search the sophisticated structure will therefore require data transfers to and from the backing store and this obviously takes time. With integrated circuit layout designs, the amount of data is so large that it cannot possibly be held in core and so must be held on disc or other backing store. Any use of the basic sequential block data structure for integrated circuit layout design must, therefore, be inefficient. There are two other problems with the basic data structure: firstly the data must be processed in the order in which it is entered and this is not necessarily the best order for subsequent processing. Secondly it is difficult to handle the group and repeat facilities. If these facilities are to be used then a count must be made of the number of data elements processed in the file when a group instance is

1	rectangle marker
2	mask number
3	x1 value
4	y1 "
5	x3 "
6	y3 "
7	polygon marker (s)
8	mask number
9	x1 value
10	y1 "
11	x2 "
12	y3 "
13	x4 "
14	y5 "
15	x6 "
16	y1 "
17	polygon marker (l)
18	mask number
19	x1 value
20	y1 "
21	x2 "
22	y2 "
23	x3 "
24	y3 "
25	x4 "
26	y4 "
27	x5 "
28	y5 "
29	x1 "
30	y1 "
31	rectangle marker





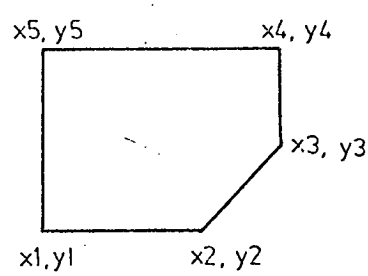


Fig 5.1 Sequential block data structure

encountered. The file must be rewound and searched from the beginning for the appropriate definition. When this is found, then the shapes contained are processed after being suitably modified to account for position and orientation of the instance. The program must then return to the beginning of the data structure and all the elements skipped until the count is reached and the main processing continued. This necessity to return to the beginning of the file whenever group instances or repeats are met is extremely inefficient and not a practical proposition for large amounts of data.

Certain of these disadvantages can be overcome by the use of extra sequential block data files and the methods by which this can be done are discussed below. However it must be remembered that in the commercial time service used, a maximum of only four disc channels i.e. four disc files were allowed to be open at any one time and this necessitated the restriction that repeats cannot be nested i.e. a series of shapes to be repeated cannot contain another series of repeated shapes.

To enable each mask in turn to be plotted out quickly, it is desirable to have the shapes for each mask on a different file. Unfortunately the number of masks can vary between 4 for a simple MOS process and 16 for a complex Bipolar process. The maximum number of masks must be catered for even though on average less than half will actually be used. This results in a requirement for 16 disc files and preferably 16 disc channels. This is

impractical and so all data for the shapes in the main layout definition have to go into the one file to be scanned in toto for each mask in turn.

The group definitions, however, need to be separated from the main definition, and again each definition should theoretically go into a separate array or file to enable it to be found and processed quickly. Again because of the variation in the number and size of the group definitions it is impractical to use separate arrays and the number of disc channels available limits the number of files allowed. Consequently all the group definitions must also go into one file.

This restricts the system to the one whose block diagram is shown in fig. 5.2. Here the order of entering the data is flexible i.e. the shapes on mask 3 can be entered before the shapes on mask 2 or a shape on mask 2 can be preceeded by a shape on mask 4 and can be followed by another shape on mask 4. Also group definitions can be entered in any order. The process of converting input data into drive tapes for tape controlled coordinatographs is as follows. After checking the data for syntax, it is converted into the purely numeric form known as the 'dump code file'. (This is a basic sequential block data structure) This file is subsequently sorted into two separate sequential data files, the first contains all the main shapes and main repeats and the second contains all the group definitions. The program now returns to the beginning of the main file and each shape is looked at in

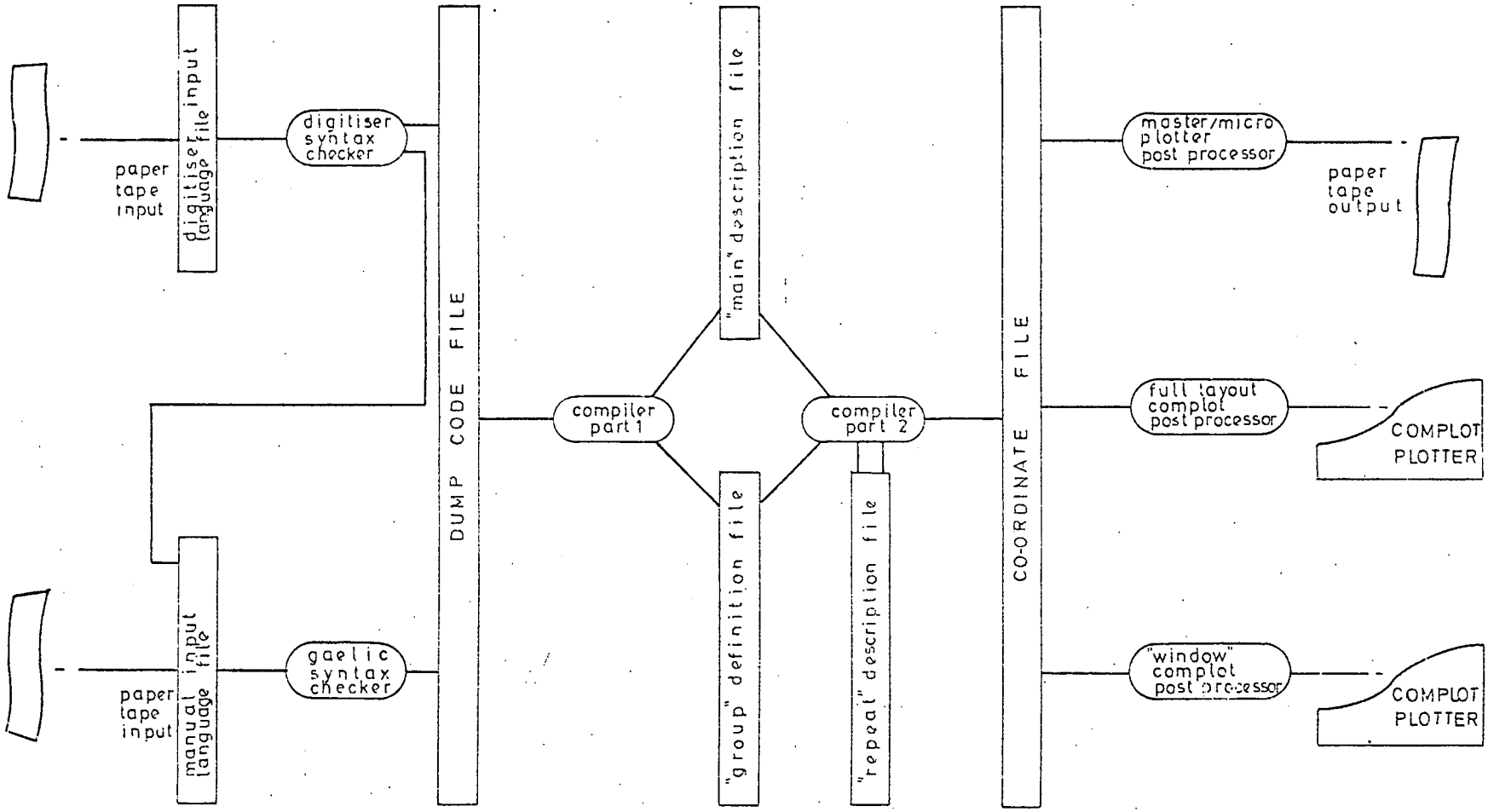


FIGURE 5·2 block diagram of PAELLA system

turn to see if it is on the required mask. If the shape is required, then it is written to a new file, if not, it is ignored. This new file is called the 'coordinate file' and contains the basic information required by most tape controlled coordinatographs, that is the coordinates of every corner of every shape of each mask in turn. This co-ordinate file is subsequently post-processed to either drive an on-line plotter or to give the drive tapes for a particular coordinatograph.

When a group call is encountered then the group definition file is searched from the beginning for the appropriate definition. When the definition has been found, the shapes that are on the required mask are written to the coordinate file, taking into account the position and orientation of the call. The program then returns to continue reading from the main file.

When a repeat header is found, all the shapes to be repeated are first written to a separate 'repeat' file. The repeat file is then rewound and the shapes written to the coordinate file the required number of times with the appropriate increments on the co-ordinates.

As stated earlier the system is capable of providing check drawings from the co-ordinate file and can conceivably be modified to alter the position of shapes but it does have problems when it comes to deleting shapes and drawing new ones. This can only be done by copying the file up to the header of the shape to be deleted,

skipping the marker and coordinates of the shape and then copying the remainder of the file. Adding new shapes at the end of the file may be feasible on certain computer instalations that allow extra data to be subsequently appended to a file, but adding a shape to a group definition again requires copying the file, and this is obviously time consuming. Corrections on the system implemented are, therefore, always made to the manual input language.

When additional facilities were added to the commercial time sharing service which enabled the user to start reading from the sequential file at some point other than the beginning of the file, the process was speeded up. This was done by the program storing the starting address of each particular group definition as it was written in the group definition file and then going directly to that position on the file when the definition was required. It was a system with this facility that was used in the comparative tests discussed in Chapter 8.

5.2 The Initial Ring Data Structure:

A ring data structure provides a more versatile method of storing or handling the data for a layout. It does not have the restrictions on the repeat nesting nor the problems of processing unnecessary information that are present in the the sequential block structure. The block diagram of the system using a ring data structure is

shown in fig.5.3 and the actual data structure used is shown in fig.5.4.

It is a hierarchical structure in that it has beads which hold the head pointers of rings of beads of the next hierarchical level e.g. the main definition bead contains the head pointers to the rings of the group definition, the repeat definition and the main mask beads. These beads in turn have rings of beads of the next level. The program works at one level going round a ring checking each bead in turn until the required bead is found and then descends to the lower level and goes round the next ring, it does not need to descend to this lower level unless it requires data e.g. the program will go round a mask ring, (the ring containing the mask beads) until it finds the required mask number and then will descend to the lower level and process the shapes on that mask. It does not have to process shapes on any mask other than the one required.

The full data structure at first sight appears to be complicated but can be understood by considering first of all the main definition on its own as shown in fig 5.5. The whole of the ring data structure is built up on a main definition bead which is shown in fig 5.6.

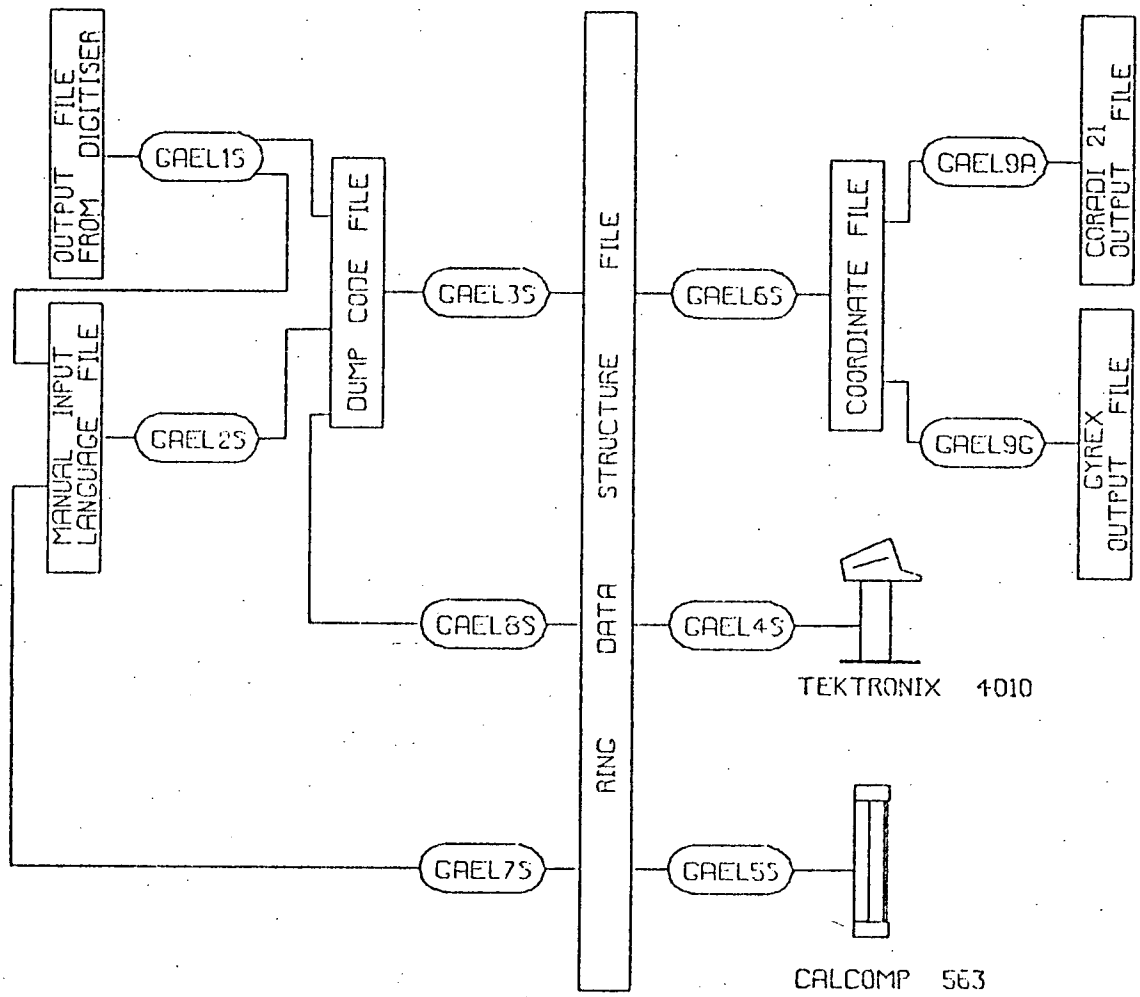


FIG 5 3 BLOCK DIAGRAM OF INITIAL GAELIC SYSTEM

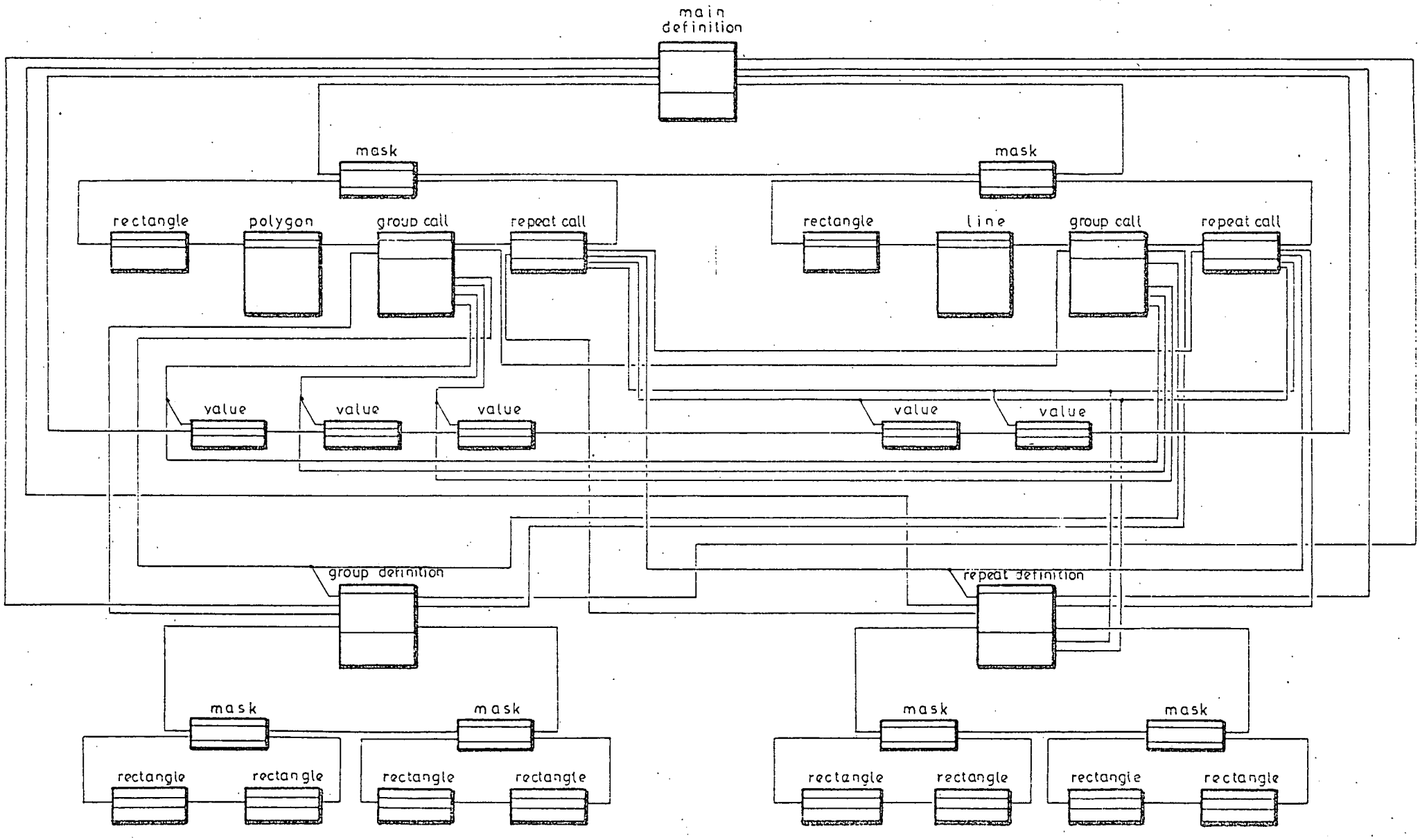

main definition
mask
mask
rectangle
polygon
group call
repeat call
rectangle
line
group call
repeat call
value
value
value
value
value
group definition
repeat definition
mask
mask
mask
mask
rectangle
rectangle
rectangle
rectangle
rectangle
rectangle
rectangle
rectangle

Figure 5.4

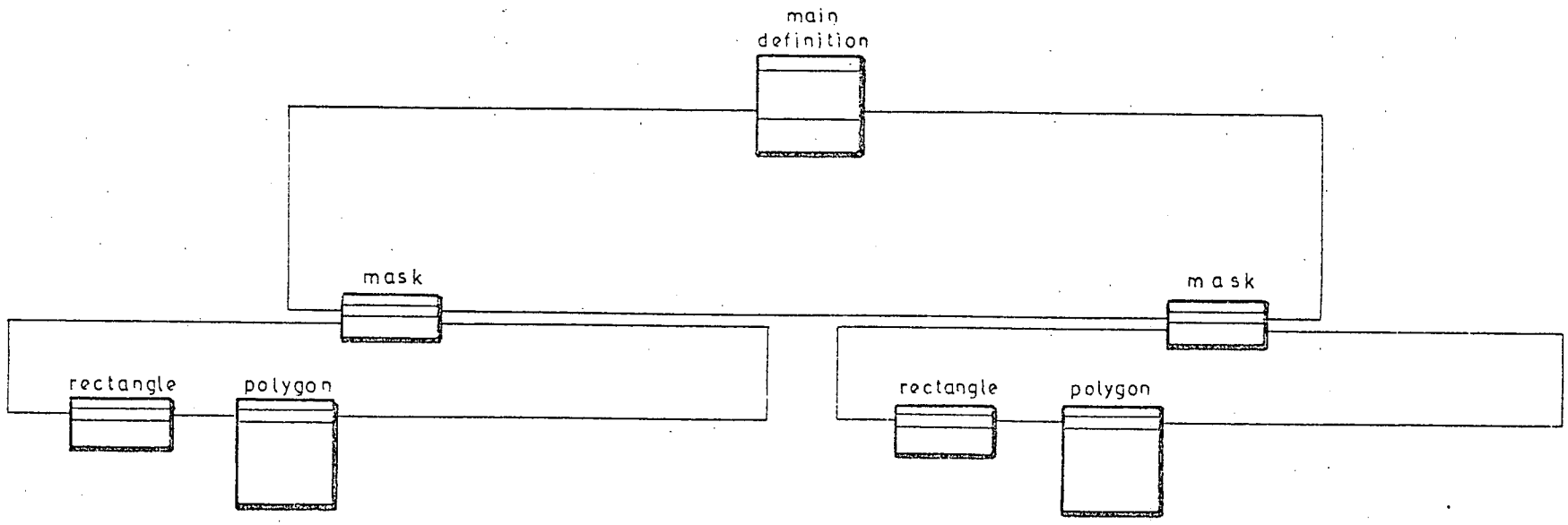

main
definition
mask
mask
rectangle
polygon
rectangle
polygon

Figure 5·5

1	5	4
Garbage Ring Head Pointer		
Spare Ring Head Pointer		
Group Definition R. H. Pointer		
Repeat Definition R. H. Pointer		
Main Area Ring Head Pointer		
Minimum X Coordinate		
Minimum Y Coordinate		
Maximum X Coordinate		
Maximum Y Coordinate		

Fig 5.6 Main Definition Bead

The bead contains the head pointers of several other rings that are used in the data structure and their use will become clear as the structure is developed. The last of these ring head pointers is the start of the mask ring and fig 5.5 shows how this ring contains a series of mask beads, one for each mask used in the layout. Each mask bead is similar to the one shown in fig 5.7 and contains the pointer to the next mask bead, the head pointer of the appropriate shape ring and the number of the mask.

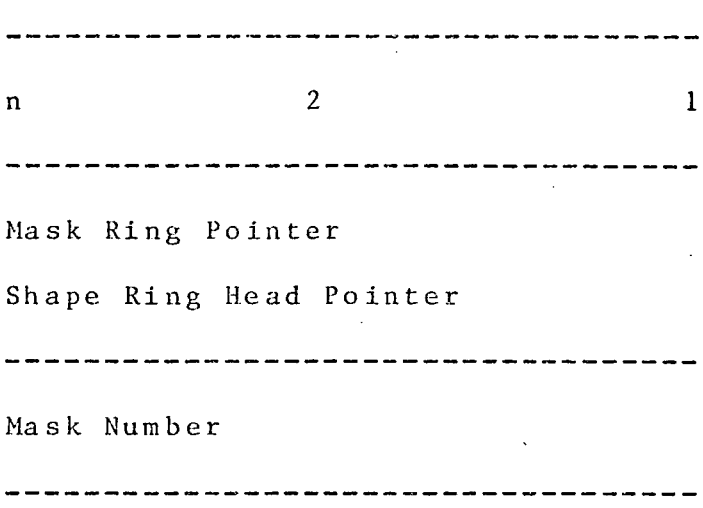


Fig 5.7 Mask Bead

The shape ring contains the shape beads holding the description of shapes on the particular mask. There are three basic shapes that can be described these are the RECTANGLE, POLYGON and LINE. An example of a polygon bead is shown in fig 5.8.

2	1	n
Shape	Ring Pointer	
Minimum X Coordinate		
Minimum Y Coordinate		
Maximum X Coordinate		
Maximum Y Coordinate		
Format		
Initial X Coordinate		
Initial Y Coordinate		
"		
"		
Final X Coordinate		

Fig 5.8 Polygon Bead

The head word is as usual split into three fields, the first field contains the 'type' of shape bead in our case the number is 2 for a polygon (number 1 indicates a rectangle and number 7 indicates a line). The second field contains the number of pointers in the bead and the third field contains the number of data words. The data of the polygon bead consists of the coordinates of the bounding rectangle of the polygon, the format number (8388527 for a short format, and 8388526 for a long) followed by the actual coordinates of the polygon.

A polygon or line can have up to 1000 corners and so it can take a long time to go through the data of the shape only to find that none of the shape appears within the window. For this reason the bounding rectangle of every polygon or line is computed as the data is entered and the co-ordinates of this rectangle are stored in the the first four data words of the bead. Each time a shape is processed, an initial check is made to see whether any of the shape appears within the window, before processing the actual co-ordinate data.

The data structure is built up by initially creating the main definition bead and setting all the ring pointers to point to themselves. As the first shape is read in, the appropriate mask bead is created and added to the main mask ring. The appropriate shape bead is then set up and added to the shape ring of the new mask bead. When subsequent shapes are read in, the mask ring is searched each time for the appropriate mask bead. if the bead is found then the new shape bead is created and added at the beginning of the shape ring. However, if the mask bead does not exist a new bead is created and inserted at the beginning of the mask ring. The shape bead is then created and added to the shape ring. The shapes and masks are added at the start of the rings for speed, as the value in the head pointer is simply transfered to the pointer in the new bead and the address of that pointer entered into the ring head.

The process of plotting out a main data structure consists of going round the mask ring until the appropriate mask bead is found. The area ring of the mask bead is then processed, plotting out each mask in turn. If a window is to be plotted, the bounding rectangle of each shape is checked against the window and shapes outside the window are ignored and the next shape processed. This has very little saving for a rectangle where the bounding rectangle consists of the actual coordinates, but has considerable savings with polygons and lines, where there can be up to 2000 coordinates.

Identifying the nearest point in the layout is very similar to plotting, the mask ring is again searched for the appropriate mask and then the shape ring is processed shape by shape checking each pair of coordinates within the window in turn.

The above description applies to the main part of the layout and does not use any of the group and repeat facilities. The methods of handling the group and the repeat structures are basically the same and consequently only the group structure will be dealt with in detail. A group call or instance bead is shown in fig 5.10.

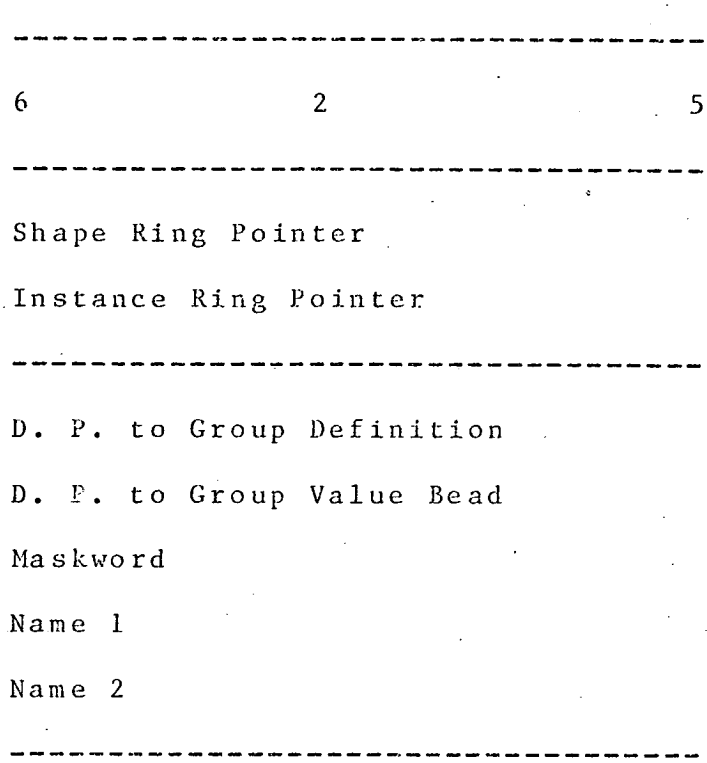


Fig 5.10 Group Call Bead

Group call beads appear as a 'shapes' on the shape rings of the various masks used in the definition e.g. if a group definition contains shapes on masks 1, 2 and 3 and there are instances of the group called in the main definition, then there are group call beads on the shape rings of masks 1, 2 and 3 of the main definition as shown in fig. 5.9.

When a group call is processed the appropriate group definition is found by means of a direct pointer and the the shapes on the appropriate mask of the definition are then processed. The use of the direct pointer may appear redundant as there is a group instance ring joining all the group calls to a particular definition and whose head pointer is in the definition. The program could obviously

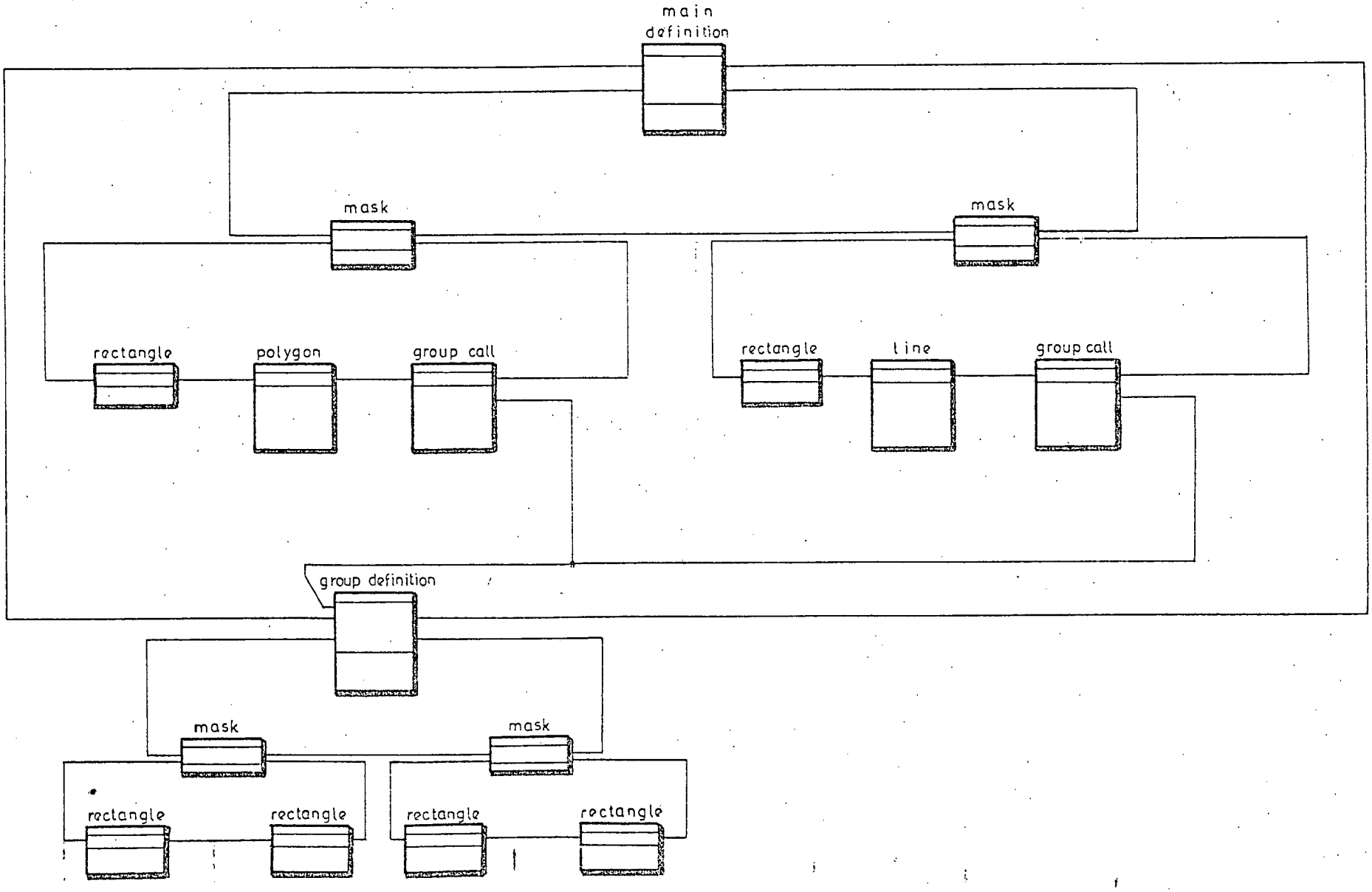

main
definition
mask
mask
rectangle
polygon
group call
rectangle
line
group call
group definition
mask
mask
rectangle
rectangle
rectangle
rectangle

Figure 5·9

trace its way round this ring to the definition. It must be remembered, however, that there can be of the order of 100 group calls on certain layouts and on these layouts the program would, on average have to pass through 50 group calls before reaching the definition and the process is therefore time consuming. The group instance ring is actually present so that individual group calls or group definitions can be deleted. The position of the group call and its orientation could be stored in the group call bead as shown in fig 5.9. This has the severe disadvantage that if the position of the call or its orientation are modified on say mask 1, then the same modification must be made on all the other masks that contains a group call bead. The designer can very easily forget to do this especially if many modifications are performed on mask 1 before modifying the other masks. This would create errors in the layout which are not easy to detect. This problem is overcome by setting up 3 'value' beads which are inserted onto a special value ring. These beads contain the values of the x and y coordinates of the group origin and its orientation. The group call bead as shown in fig 5.10 contains direct pointers to the value bead heads instead of the actual values and this gives the data structure that will handle the group facility shown in fig 5.11.

The way in which the group facility is built up in the data structure is a little complicated in order that the input data can have calls to a group before the group

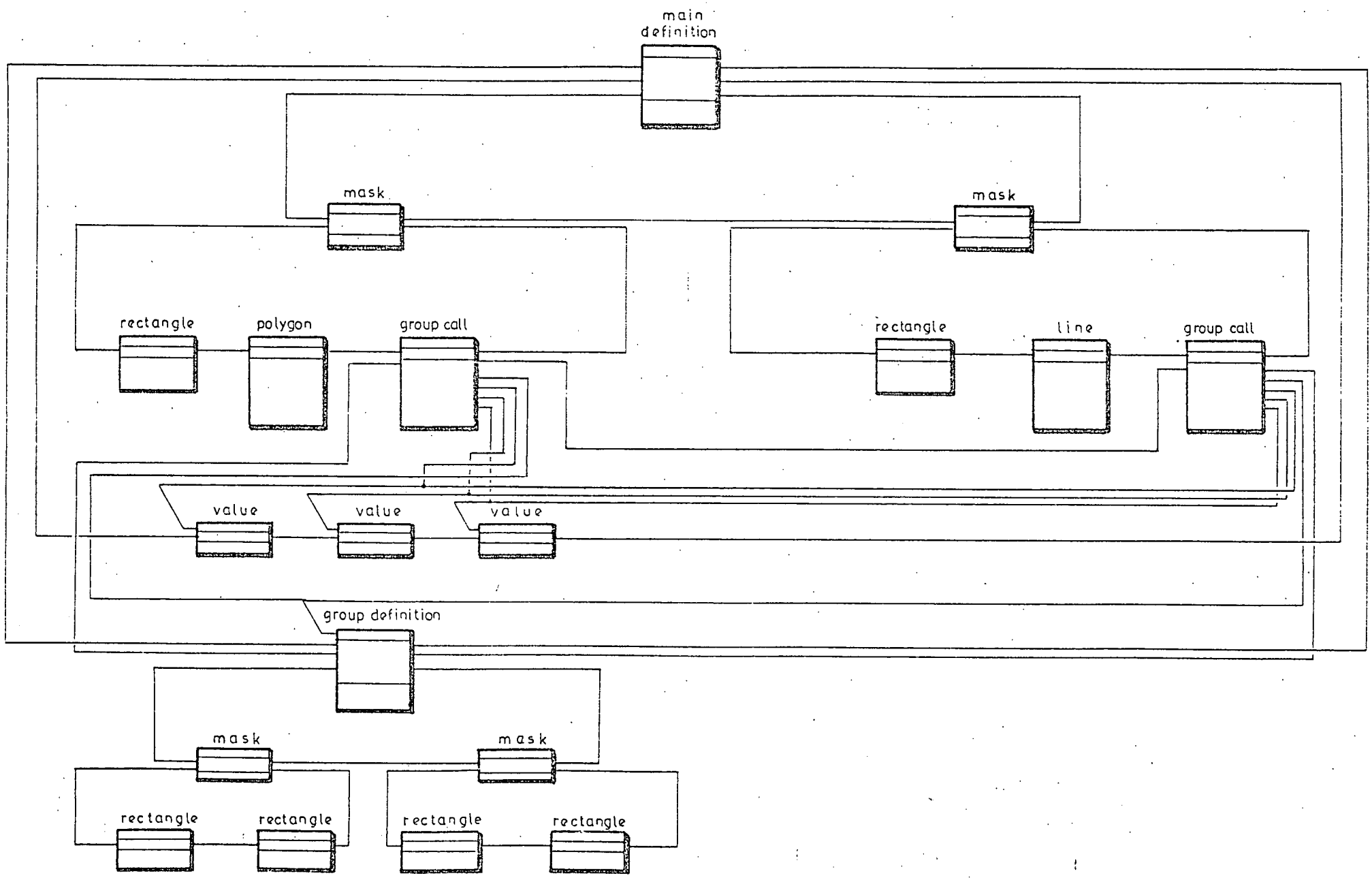

main definition
mask
mask
rectangle
polygon
group call
rectangle
line
group call
value
value
value
group definition
mask
mask
rectangle
rectangle
rectangle
rectangle

Figure 5·11

is defined and vica versa.

When a group definition is encountered first in the input data, the appropriately named group definition bead is created and added to the group definition ring. This group definition bead is very similar to the main definition bead shown in fig 5.6 except that:

1) The first field in the head of the bead, the type, contains the number 2.

2) The group definitiion ring pointer is the first pointer in the bead

3) The group instance ring head pointer is the second pointer and

4) Two extra words appear at the end of the data words and these contain a numeric representation of the name of the group.

The various mask and shape beads are then added to the group definition bead in exactly the same way as they are added to the main definition bead until the end of the group is reached.

When a call to a particular group is entered before its definition, then the group definition bead is again set up but this time there will be no shapes to be added. Three value beads are set up and added to the value ring. Each value bead consists of 3 elements: the bead head which contains the usual 3 fields, the type, the number of pointers and the number of data words, the value ring pointer and the actual value. In this case the first ring

has the value of the x origin of the group instance, the second has the value of the y origin and the third has the orientation. The group call beads are then set up and added to the shape rings of the appropriate masks and direct pointers to the head of the group definition, the x origin, y origin and orientation beads are added as shown in fig. 5.11.

The way the program discovers whether the group definition or group instance have already been encountered is to search the group definition ring for the appropriate definition bead. If the definition bead is present then the address of the bead head is noted and the group call or group definition processed as described above. If a definition for a particular group is encountered a second time, an error message to that effect is printed out and the initial definition is overwritten. More than one group call to the same definition are of course legal and so are added to the data structure, each new call having its own value beads and group call beads. Group calls to one definition can occur in another group definition, or in a repeat definition, as well as in the main definition. These are processed in the same way except that the value beads and group call beads are added to the appropriate definition. The 'type' of the value bead also reflects the type of calling definition and is set to 1 for a call from the main definition, 2 for a call from another group definition and 3 from a repeat definition. This is not essential for the program operation but makes debugging

the program a lot easier.

When processing the data structure to produce a plot or identify an actual point, the group call bead is obviously encountered. As explained in Chapter 3, the user is not allowed to modify shapes in the instance of a group but can modify the position or orientation of the instance. When the origin of an instance is identified, the direct pointers to the value beads are then followed and the values of the x and y origin compared with the coordinates of the cross hair cursor and then the next shape is processed. If plotting then as well as following the direct pointers to find the position of the call and its orientation, the direct pointer to the definition is followed. All the shapes in the definition are then processed, transforming all the coordinates to account for the position and orientation of the instance. When all the shapes have been processed the program returns to the next shape bead after the group call bead.

The method of handling the repeat facility is very similar and the data structure with repeated shapes is shown in fig. 5.11 The value beads in this case contain the number of patterns and the spacing between them.

The repeat call is contained implicitly in the repeat definition and so we have a simpler system for building up the data structure, the only slight complexity is the fact that 'repeats' can occur in the main definition, in group definitions or even nested in other repeat definitions.

This means that when the repeat definition is encountered in the input data the program must assertain which value ring and repeat definition ring must be used.

Processing the repeat calls is again similar to processing group calls, the first pattern only is processed during modification but all the patterns being processed during plotting.

This bounding rectangle concept that is used on the polygon and line beads is taken a stage further by calculating the bounding rectangle of each definition as the shapes are entered, and storing the co-ordinates of this rectangle in the definition bead. This has two advantages: when a group or repeat call is processed the bounding rectangle of the definition, modified by the position and orientation of the call, is checked against the window and if outside, the definition is ignored. The other advantage is that when a definition, main, group or repeat, is being plotted the user can be given the minimum window size that will allow a plot of the whole definition.

Beads that are deleted are put onto a 'garbage' ring ready for re-use if required and the pointer in the previous bead changed to point to the following bead and so once deleted the bead is not processed again.

5.3 Problems with the Initial Data Structure

The information held in the initial ring data structure is not necessarily in the most efficient form for subsequent processing because the information can be fragmented over the disc. The effect of this inefficient storage of data is not normally noticed on the Decsystem 10 because of the low data rate (1200 baud) available to the Tektronix storage tube terminal and the fact that the users program is being continually swapped in and out of core by the time sharing executive. However when designing large integrated circuit layouts, say above 150thou square, delays can be noticed, usually during modification, that are due to the number of disc transfers required.

The reason for the large number of disc transfers can be understood by considering the following example. Assume that a plot of a window on mask 1 is required. To do this the mask ring is searched, examining each mask in turn until the appropriate mask bead is found: the shape or contents ring of the required mask bead is then traversed examining each shape in turn and plotting those within the required window. Assume that the program can only have three pages of the data structure in core at any one time and that initially these are pages 1, 2 and 3. If the first page containing a shape on the required mask is page 27 then one of the pages presently in core say page 1 must be overwritten by page 27. The shape on page 27 can then be processed. The next shape could well be on page

28 and so page 2 must be overwritten by page 28 and that shape processed. The next shape may be on page 1 which will have to be brought back into core again, this time overwriting page 3. If the data is awkwardly fragmented, the next shape may be on page 3. i.e. the one that has just been overwritten, and so page 3 must be brought back in again this time overwritting page 27. This arrangement is obviously extremely inefficient and occurs when each shape is in virtual isolatiion i.e. is apparantly on a separate page. If therefore, all the shapes on a given mask were arranged to be on the same page then once this page was brought into core then no other disc transfers would be required to plot or modify that mask.

The obvious solution is therefore to arrange that all the shapes on a given mask are on the same or consecutive pages and this raises the obvious question 'why isn't it done?' This is a question that is much easier to ask than it is to answer. Chapter 2 shows that the average designer will produce the layout of all the masks of a given section simultaneously. His natural reaction is therefore to specify the input data for the section as soon as he has designed it, and then, after checking and modifying the layout, will proceed to design the next section. The pages of the ring data structure are written consecutively i.e. page 1 is filled before page 2 is started. Thus the data for one section of the layout will go on the same page or consecutive pages i.e. the shapes on mask 1 for the section will be near the shapes on mask

2 for the same section. However shapes on mask 1 for another section will probably be on another page.

If the designer designs and draws the complete layout and then and only then passes it over to a tracer or similar grade of staff to be digitised, then the data can be entered into the computer mask at a time. All the data for mask 1 will therefore be entered onto adjacent pages of the data structure. This appears at first to be the obvious solution to the problem but does assume that the designer is prepared to design the complete layout before the data is entered into the computer. This method also means that the full facilities of the GAELIC system, for example the group and repeat facilities, cannot be exploited and therefore involves the designer in a lot of unnecessary work. Exactly the same argument applies to coding the completed layout using the manual input language and so this is not a viable alternative.

It is therefore inevitable that the data is fragmented onto different pages if the designer is to be allowed to design in the way that is most natural to him. A designer always works best when as few constraints as possible are put upon his method of working and it is essential that those constraints that are absolutely necssary are easily understood.

There is another reason why the data is fragmented on the disc which is concerned with adding shapes directly by means of the cross hair cursor on the Tektronix screen.

All shapes that are added during the interactive phase are placed on the last page of the data structure regardless of where other shapes on the same mask are situated. If the complete layout is designed on line, then the resultant layout can be fragmented throughout the disc. however, the occasional shape that was missed from the input data can be added without any noticeable deterioration in response.

In order to allow the designer to have the necessary flexibility in the input data and to cope with large numbers of shapes added interactively, it is essential to be able to order the information in the data structure after it has been initially created rather than ordering it on input. There are three ways in which this ordering can be accomplished. Firstly a new ring data can be constructed from the old fragmented structure. Secondly a new dump code file can be created from the old ring data structure and thirdly a new manual input language file can be created from the ring data structure.

The first option is perhaps the most elegant but does have core store and programming problems as two ring data structures must be handled simultaneously. The second option requires only one ring data structure and one sequential file and is therefore easier to program and requires less core store. This approach was programmed successfully for the early GAELIC software but was not used in practice by integrated circuit designers. The designers had no confidence in the method as they could

not manually check the binary dump code file before it was 'recompiled' back into the new ring data structure. This is a difficulty that is not always realised by the applications programmer: the designer has to undergo a traumatic change in his design technique when he starts using a CAD facility and is naturally very sceptical. He is having to put his design into the hands of a computer and a computer as far as he is concerned is the cause of mistakes in his gas bill and is the reason why his queries about car insurance take so long. If he can be reassured at intervals that everything is alright and completely under his control, then he will settle down to the new technique that much quicker. The ability to quickly plot out part of his design is one reassuring feature and the ability to do spot checks on the manual input language is another. People using the programs therefore, preferred the third alternative method of creating a new ring data structure i.e. converting the fragmented data structure back into the manual input language, even though this required an extra stage of processing (converting the manual input language into a dump code file). This third alternative has the additional advantage that it allows the use of 'library' components. The designer designs a section of a layout that performs a specific function e.g. an R.S flip-flop, enters the description into the computer and interactively checks and corrects his design. He then produces a corrected version of the input language file which is stored on disc or magnetic tape and is called up whenever the component is

required. There is yet another advantage in the ability to create a manual input language file from a ring data structure. As the input language file consists of ASCII characters the file can easily be transfered from computer to computer.

The program (GAEL7) that converts the ring data structure back into the manual input language is arranged so that it processes all the shapes on one mask before it processes the shapes on the next. Hence the manual input language has the shapes in this same order. When recompiled back into a new ring data structure, the shapes on one mask are put on the same page or consecutive pages. Thus this new data structure will plot out all the shapes on one mask with the minimum number of disc transfers.

When designing large integrated circuit layouts, it is not practical to plot the whole of a mask on the Tektronix 4010 terminal because of its limited screen size and resolution. This does not detract from the use of the terminal as most the designer requires to look in detail and modify small sections of the layout otherwise known as windows. The user not only requires to plot or modify one mask at a time but also requires to examine several masks superimposed on the same plot. The time taken to plot out a window for a given mask can be appreciable as the data for all the shapes on the mask must be processed to find those within the window. Certain features of the data structure described earlier in this chapter allow instances of group or repeat definitions to be ignored if

the are outside the window. These features do reduce the amount of processing but neverthless a lot of unnecessary data will have to be processed especially if there are not many grouped or repeated shapes.

There is therefore, a requirement to modify the data structure so that the amount of information that must be processed for a given window is reduced to a minimum. There are four possible approaches to solving this problem that were considered, these were:

1) Shapes within a window are placed on a fixed size page.

2) Shapes within a window are placed on a variable size page.

3) Shapes within a window are placed on a fixed size page until it is full and then the remaining shapes are placed on consecutive pages.

4) Shapes within a given area are placed on special rings associated with that area and are periodically arranged to be on consecutive pages on the disc.

Let us now consider these four approaches in a little more detail.

The first approach is extremely rigid and has the following features:

1) A page on the disc must be provided for every possible window of the maximim size of chip used i.e. regardless of the size of chip being designed. This means

the data structure must always be the same size and must always be maximum size.

2) Each page must be big enough to contain the maximum number of shapes that are possible within the window regardless of the fact that the window only contains only one shape.

3) The window size is related to the page size and need not be related to the window size that user would wish to use.

4) When a shape is moved from one window to the next, the shape description must immediately be added to the new page and then deleted from the old and this can cause problems.

5) There are always shapes in an integrated circuit that start in one window and finish in another and these cannot be accommodated with this approach.

6) There must be the appropriate mechanism in the program to select the appropriate page and bring it into core. This is a fundamental problem associated with all three appproaches and is added mainly for completion.

7) The must be a garbage collection and re-use system operating on each page to re-use the space freed by deleting shapes.

The second approach is more flexible because of the variable size of page. A lot of work has been done by Hubbald [ref 5.1] on the variable page data structure. It has the following features:

1) the pages need only be provided on the disc when they

are required and so the data structure size is kept to a minimum.

2) There must be a mechanism in the program to expand or contract the page as shapes are added or deleted.

3) The window size is again fixed by the maximum size of page allowed and the number of shapes that it can contain.

4) There is still a problem with moving shapes from one window to the next and with shapes that start in one window and finish in another.

5) There must be a mechanism to sort out which page to bring into core and which page or pages to write back to disc to make room for it. For a variable length of page this is an extremely complicated algorithm.

The third approach is more flexible still and has the following features:

1) Pages are again only used when required and so for a small layout only a small data structure is required.

2) The pages can be made a convenient size for the computer and do not depend on the window size required.

3) The window size is still predetermined and cannot be changed by the user.

4) There are still problems with moving shapes from one window to another and with shapes that start in one window and finish in another.

The fourth approach has a fundamental difference from the other approaches in associating the shapes with areas of the layout and not with windows and has the following features:

1) The window size is determined by the user without any constraints from data structure size or page size.

2) The size of the data structure is dependent on the size of the layout. i.e. pages for windows need not be created unless they are required.

3) The mechanism for swapping pages can be the same as that already used in the initial data structure.

4) When shapes are moved from one area to another, only the pointer values need be changed to associate it with a new area.

5) The problem of shapes that start in one area and finish in another is still present.

6) There must be a mechanism for reordering the data structure on the disc so that shapes in a given area are on the same page or adjacent pages.

This last approach is extremely flexible and is therefore the one implemented. As described, it still has several problems associated with it that have to be solved. Probably the most important of these is how to decide with which areas shapes are to be associated. Fig 5.12 shows a section of integrated circuit layout with a grid superimposed which divides it up into areas. It can be seen that there are two main classes of shapes, those that lie entirely within an area and those that do not. The problem is what to do with the latter. There are three options:

1) Treat these shapes the same way as those lying entirely within the area and associate each shape with the

area in which it starts. This is a non-starter as this means that every area must be processed for even the smallest of window just in case it contains a shape that extends into the window.

2) Associate shapes that are entirely within an area with that area and treat all the remaining shapes as special cases. When plotting or modifying only the area or areas within the window and the special shapes need be processed. This is a far more practical approach but has the disadvantage that there are an awful lot of special shapes that must be processed for each window.

3) A closer look at fig 5.12 shows that these specials can be split into two subdivisions, those that start in one area and extend only to an adjacent area and those bigger shapes that extend further. This allows a modification of the second option so that shapes entirely within an area and shapes that only extended into adjacent areas are associated with that area and only shapes extended beyond the adjacent areas could be treated as specials. This modified option was chosen as it meant that there were only a few special shapes to be processed for all windows and the only areas that needed to processed were those contained within and adjacent to the window.

The numbering of the areas is also an interesting problem. The obvious solution is to number the areas on a raster as shown if fig 5.13.

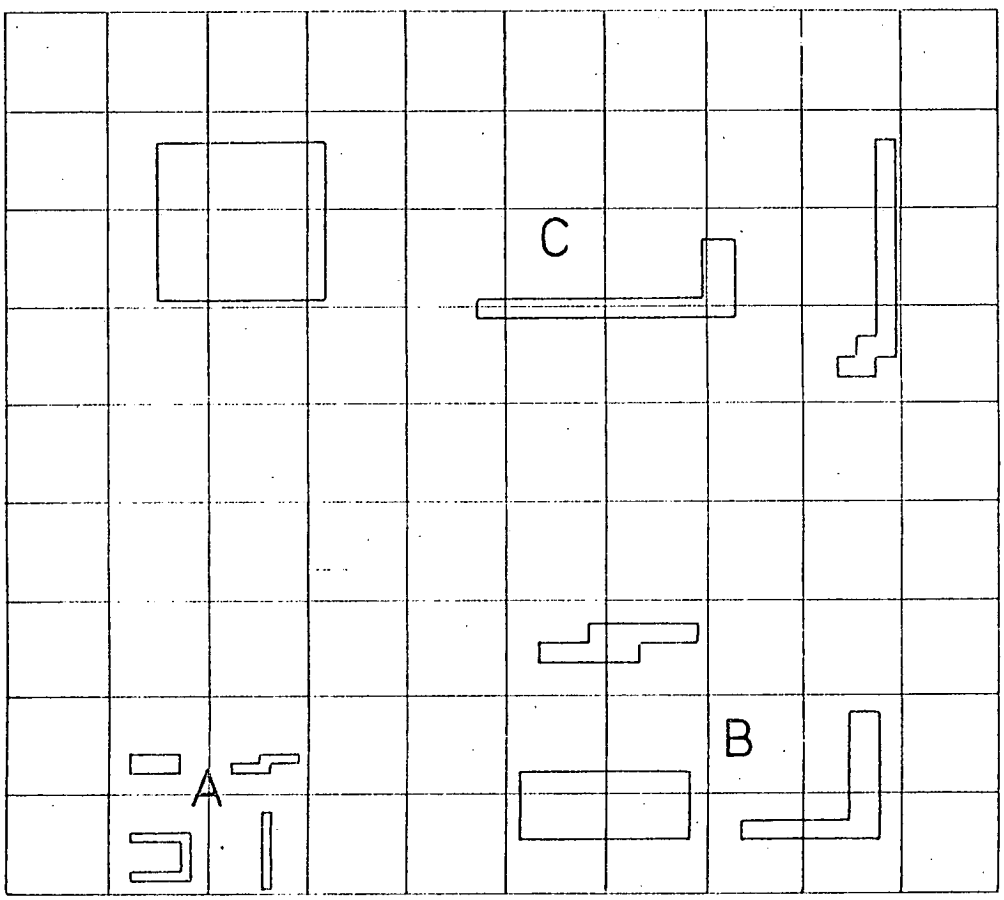


FIGURE 5·12 layout showing relationship between shapes and areas

25 26 27 28 29 30 31 32

17 18 19 20 21 22 23 24

9 10 11 12 13 14 15 16

1 2 3 4 5 6 7 8

Fig 5.13

The actual area number can be quickly evaluated. However, on giving the problem a little more thought, it can be realised that by modifying this numbering order, it is possible to overcome one of the common problems met when plotting large composite drawings and when actually cutting the cut and peel material. This problem is to minimise the distance traveled and hence the time spent with the pen or knife up. Only when the pen is down and drawing is it doing useful work. It is very difficult and time consuming to sort the information in the data structure so that it can produce a drive tape for the coordinatograph that has the data in the optimum order. It can be done whilst entering input data into the computer but that is contrary to the policy of putting as few constraints as possible on the input data preparation.

If the method of numbering area beads as shown if fig 5.6 is used and the areas are plotted out in sequence, then there is a distinct improvement over any random method. There is obviously very little distance between shapes in each area and very little distance between

adjacent areas so travel with the pen up is minimised. The main travel is during the 'flyback' e.g. when travelling from area 8 at the end of the first row to area 9 at the beginning of the second.

The optimum solution would be appear to be to arrange the area beads in a spiral starting in the middle of the layout as shown in fig 5.14.

17	16	15	14	13
18	5	4	3	12
19	6	1	2	11
20	7	8	9	10
21	22	23	24	25

Fig 5.14

This method does present certain implementation problems such as finding the middle of the circuit to start the counting when circuit sizes obviously vary and evaluating which areas are required for a given window.

A modification to the basic spiral can be made so that it starts at the bottom left hand corner of the layout as shown in fig 5.15.

17 18 19 20 21

16 15 14 13 22

5 6 7 12 23

4 3 8 11 24

1 2 9 10 25

Fig 5.15

Plotting from this type of numbering system is excellent for a full layout or a full mask. However, the algorithms required to evaluate the correct area for a shape as it is entered and to evaluate the correct areas to plot a window are extremely complicated.

Complication for its own sake is never worth while and the numbering sequence finally chosen is shown in fig 5.16.

32 31 30 29 28 27 26 25

17 18 19 20 21 22 23 24

16 15 14 13 12 11 10 9

1 2 3 4 5 6 7 8

Fig 5.16

It has the beauty of being a simple system to implement

using a fast algorithm to calculate with which area a shape should be associated and which areas should be plotted and yet minimises the travel with the pen up. The problems of further optimisation of plotting files are discussed in Chapter 7.

In this section we have discussed methods of speeding up the plotting of windows and indentifications of points in the data structure. Some of these require additional programs to rebuild the the original data structure and these programs are available in system using this structure. Others i.e. those involving the use of areas required a new data structure and this new structure is now dealt with in more detail.

5.4. The Final Data Structure.

There are two ways in which the area concept can be incorporated into the data structure and these are shown in figs 5.17 and 5.18. The first method (fig 5.17) has mask beads that instead of containing the head pointer of the shape ring contain the head pointer to a ring of area beads, usually known as the area ring. There is an area bead for each area occupied on the mask. Each area bead contains the area number and the head pointer of the appropriate shape ring. Once constructed the data structure can be reorganised to arrange all the contents of the area ring to be on one page or on consecutive pages. In this case all the shapes on the given mask will

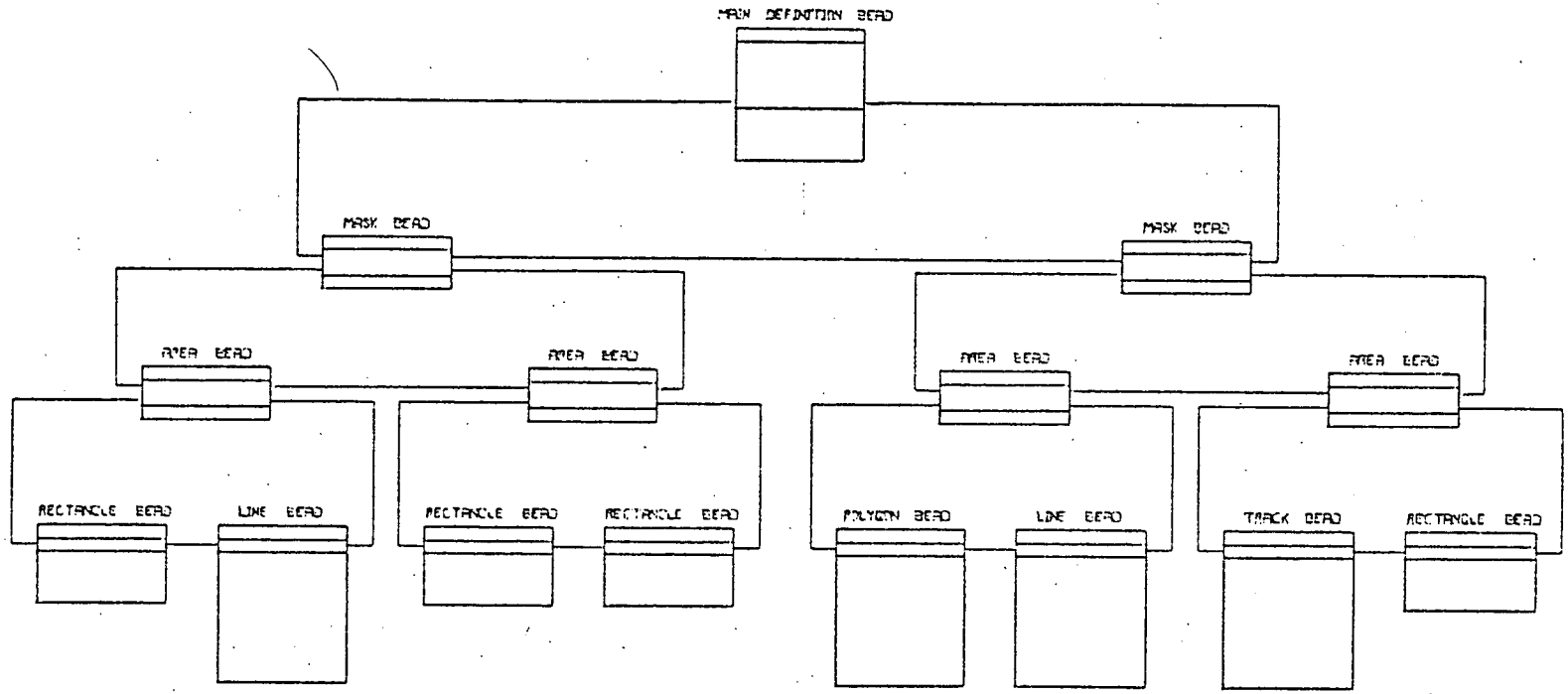


FIG 5.17 DATA STRUCTURE IN WHICH THE MASK BEADS CONTAIN THE AREA RINGS

be on the same page but not shapes on another mask. However shapes on the given mask that are in the next area are placed immediately after those for the first area and so will be on the same page or consecutive pages. This makes this particular system of implementing the area concept ideally suited for operations that involve one mask at a time and require consecutive areas. Modification immediately comes to mind in this context. It is not so well suited to operations that involve shapes on more than one mask in the same area as shapes on another mask will probably be on another page.

The second option (fig 5.18) has the main definition bead modified so that instead of having the head pointer of the mask ring, contains the head pointer of the area ring. Each area bead contains the area number but instead of the head pointer of the shape ring, contains the head pointer of the mask ring. Each mask bead is the same as in the initial data structure, i.e. contains the mask number and the head pointer of the shape ring: the shapes on this ring however contain only shapes within the appropriate area. This arrangement is preferable for operations that involve shapes on more than one mask within a given area and plotting is the first operation to come to mind. This is because when the data structure is reorganised, the contents of an area are put on the same page or consecutive pages i.e. the mask beads and all the shapes.

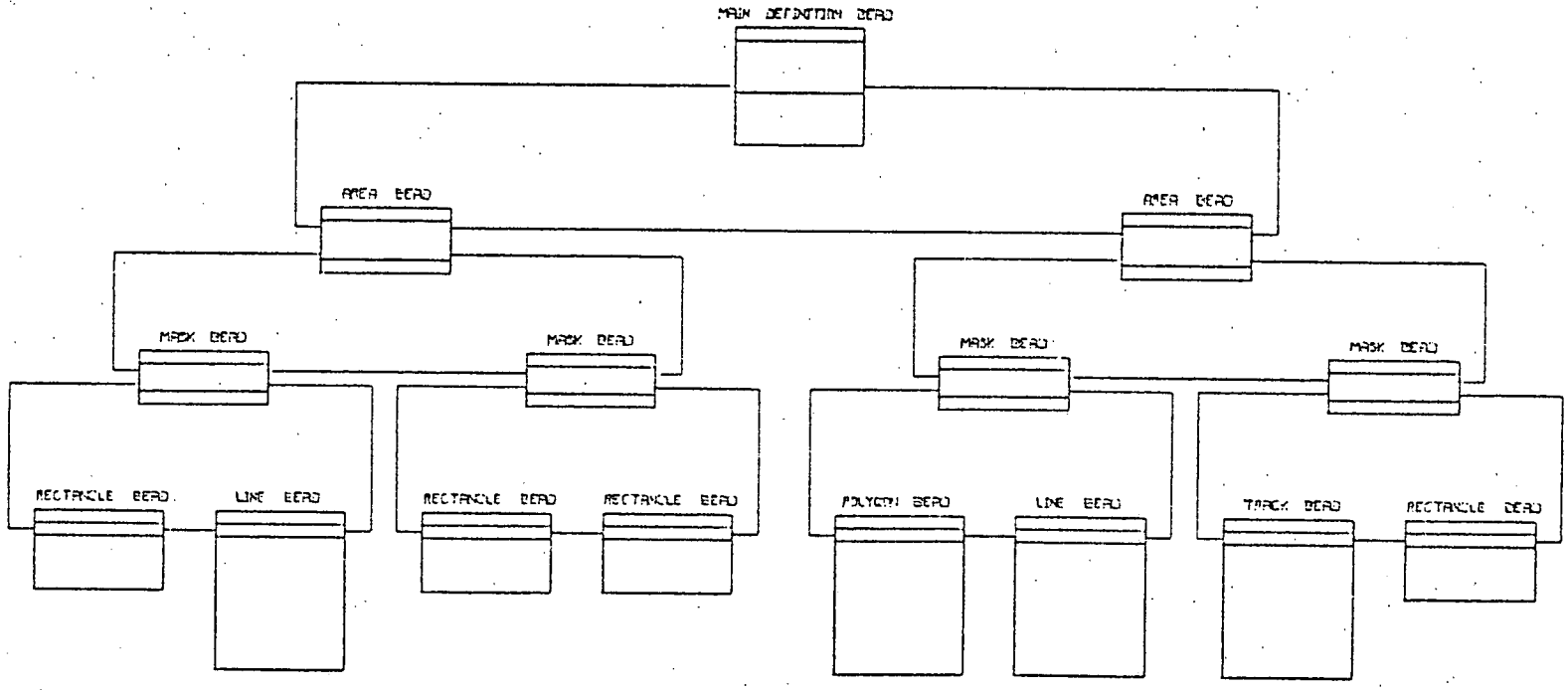


FIG 5.19 DATA STRUCTURE IN WHICH THE AREA BEADS CONTAIN THE MASK RINGS

The choice of which sytem to implement depends on which is more likely to be required , operations involving one mask and consecutive areas or those involving shapes on several masks in the same area. The user spends most of his time working on a window of the layout and only certain areas are required. These are by definition not always consecutive as can be seen in fig 5.19.

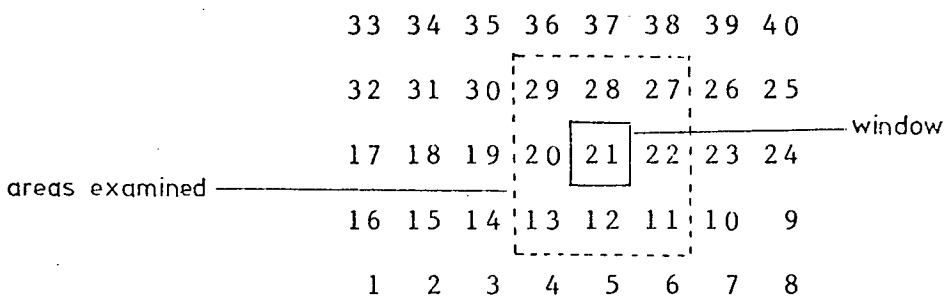


Fig 5.19

The areas required for the window are 11, 12, 13, 20, 21, 22, 27, 28, 29 which are some consecutive areas and some nonconsecutive. There are many area beads between the consecutive triplets that are not required for the window and so on balance the second option is the one to choose. There is another reason that substantiates this choice and this is discussed in detail in Chapter 7 in section 4 when the organisation of the data structure on disc is discussed. It is shown there that in order to process unwanted area beads quickly, all area beads should be on the same page. The fewer area beads the easier it

is to reach this objective.

The size of the grid that divides the layout into areas can obviously be varied giving different area sizes. The size of the area can have an effect on the program performance and it is necessary to find the optimum size. The reason for the variation in performance with size can be understood by considering the extreme cases. If the area is too big then most areas will have to be processed regardless of the size of the window used. This is shown in fig 5.20 which shows the complete layout divided into 9 areas, and shows a small window in the centre area. Shapes in the adjacent areas can extend into the centre area and hence into the window and so all 9 areas must be processed each time the contents of the window are plotted or modified.

The other extreme is to have so small an area that all shapes extend beyond their adjacent areas and so are placed in area 0 which is reserved for the special shapes. The shapes in area 0 are processed regardless of the size of the window and so the same data is processed for every window. There is also another problem in that the smaller the area the more area beads are required. The larger the number of area beads the larger the number of mask beads that must be in the data structure. The programs handling the data structure have to check each area bead in turn to check if it possibly contains shapes within the window. (the reason for this is discussed in chapter 7 section 5) There is therefore a large overhead in data structure size

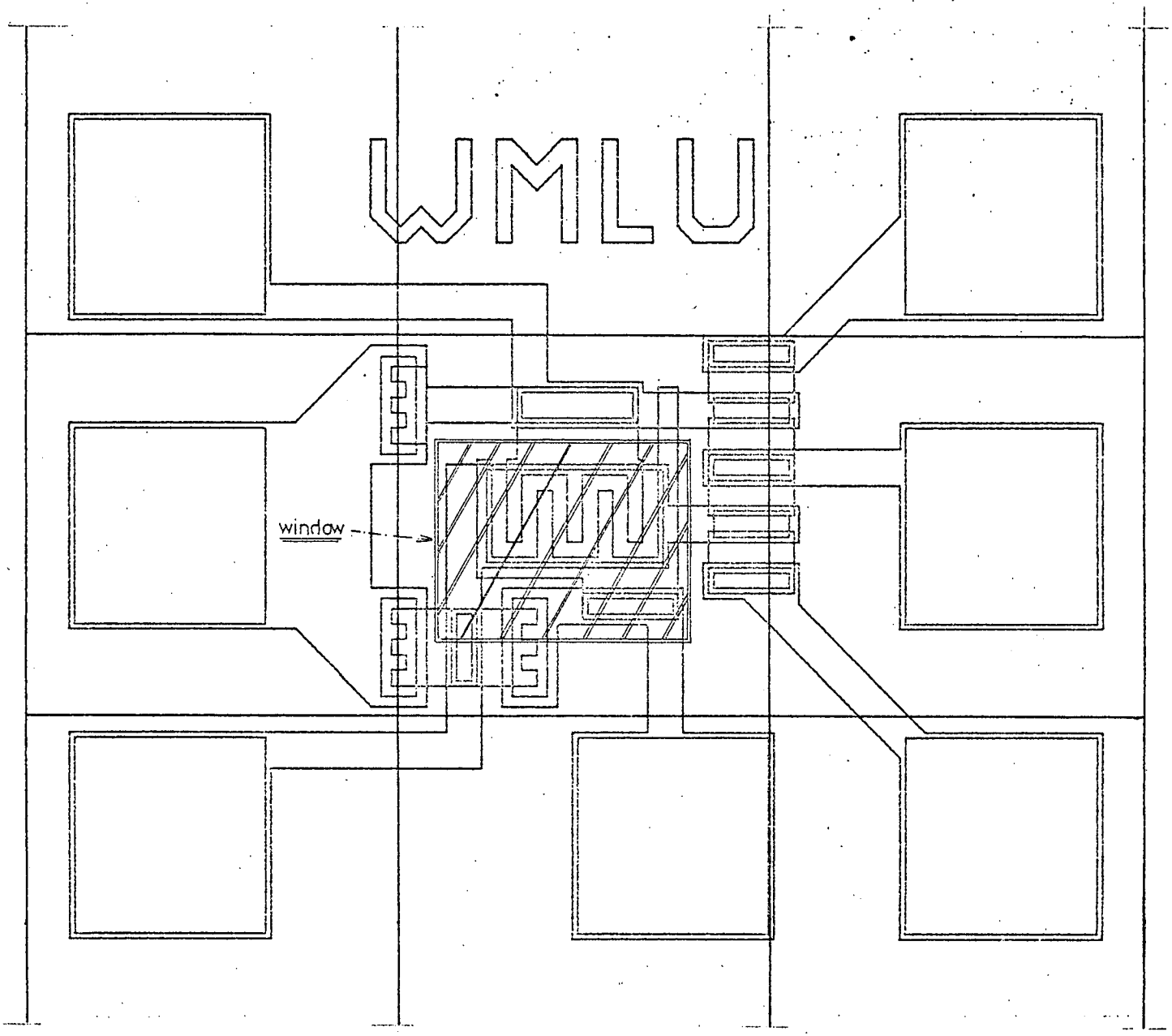


Fig. 5. 20 Small layout divided into 9 areas

and in CPU time to process all these beads. The optimum size must therefore lie somewhere between these two extremes and a thecretical value for this optimum size can be obtained by considering the problem from a different angle.

Let us consider the size of window that will be used most frequently and the effect of that size on the size of the area beads. Occasionally the user requires an overview of a large portion of the layout to identify sections that require closer examination or to check the interconnecting metallisation. However, most of the time he will require much smaller windows that will enable him to visually check the distance between two shapes and enable him to position a shape so that it a given distance from another shape. This means that the users requires a resolution of one increment. To enable this resolution to be obtained on Tekronix 4010 terminal this means a minimum of two screen units to one layout unit. The screen resolution is 760 by 1024 screen units and as the right hand side of the screen is used for messages this gives an active window area of 700 by 700 screen units. The window size is therefore 350 by 350 layout increments. Let us now examine the effects of various area sizes on this window.

If the area size is made the same as the window size i.e. 350 by 350 layout increments then the number of areas that have to be processed varies between 9 and 16 as can be seen in fig 5.21. Any smaller area size would require

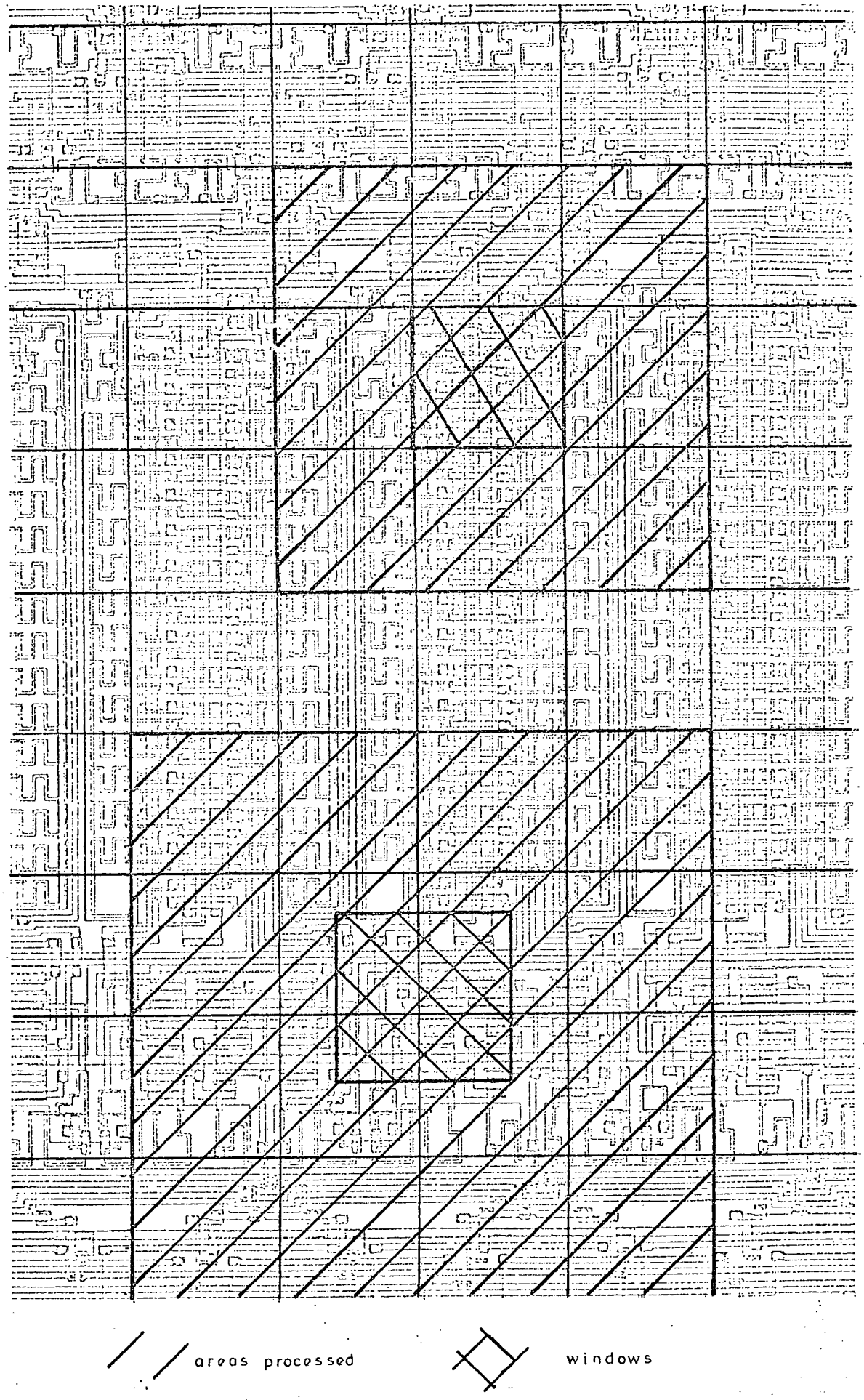


Fig. 5.21 Layout divided into areas 350 x 350 increments

more areas to be processed for example fig 5.22 shows an area size of 175 increments square and this requires between 16 and 25 areas to be processed for the window. It will obviously take longer to process the increased number of area beads and will mean more shapes in area 0.

Any larger area size than the window will still require between 9 and 16 areas to be processed as can be seen in fig 5.22. It is probable that the most commonly used window size will also contain a large number of shapes completely within the window: if many shapes extend beyond the window the user would use a larger window to see what is happening. It therefore appears to give a very strong argument for having the areas size exactly the same size as the most commonly used window and that window size is approximately 350 by 350 increments. Observing a colleague using the programs in anger to actually design an integrated circuit showed that his most frequenly used window was approx 250 by 250 increments which considering variations in human preferences showed a large measure of agreement.

The programs were written to handle a range of area bead sizes and the same program and same data were compared for differing sizes The results are given in Chapter 8.

The above discussion assumed that plotting was the most important process in layout design and this is not really the case. The user is reasonably patient when the

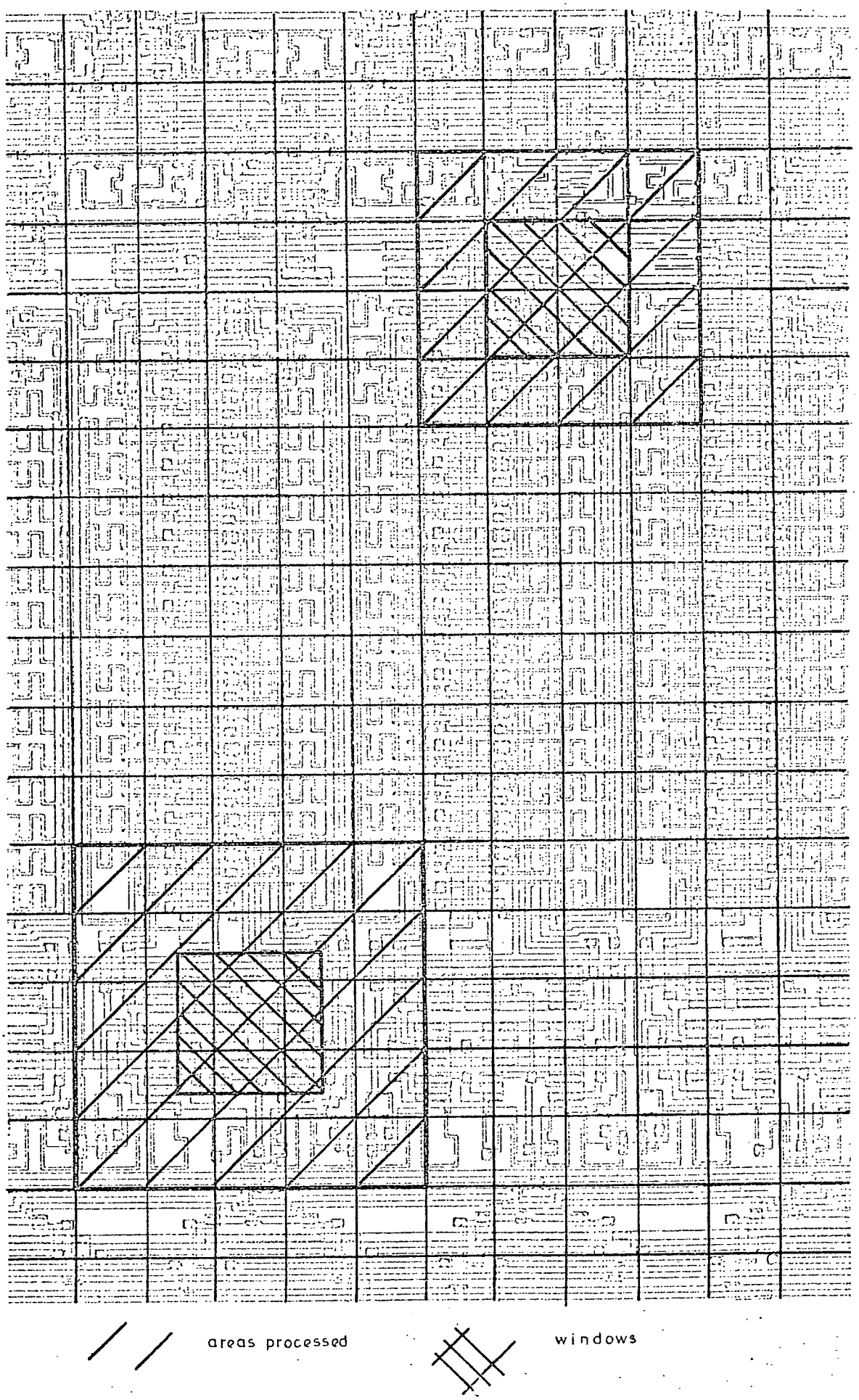


Fig. 5.22 Layout divided into areas 175 x 175 increments

terminal is plotting out a window of his layout as he can check the layout as it is being plotted. However, when waiting for the computer to find the nearest point in the data structure to the cross hair cursor during modification there is very little that the user can do and so he requires a virtually instantaneous response. It is therefore clear that the modification process is the more important. The cross hair cursor can be positioned within two or three increments of the point to be identified and so it is only necessary to check the shapes that pass within two or three increments of the cross hair cursor position. In other words the effective window is extremely small. It could therefore be argued that the area size should be made equal to the smallest window size when we were discussing plotting and at first it would seem logical therefore to have the area equal to the small window required during modification. Unfortunately as explained earlier, the number of shapes that would extend beyond the adjacent area would be astronomic and so the area beads would be wasted. Also the overheads in terms of data structure size and CPU time to process all the area beads would be excessive. An area bead would be required for every part of the layout of say 10 increments by 10 increments. As the maximum size of circuit is 32000 by 32000 increments this will mean 3200*3200 area beads i.e. approx. 10 million beads for a maximum size circuit. The situation is bad enough for an area size of say 320 by 320 increments when a maximum of 10000 area beads would be required plus the appropriate mask beads. An area bead

requires 5 words of storage and each mask bead requires 4 words. Assuming therefore, that the MOS process to be used for the circuit, required 4 masks and that there is at least one shape in each area on each mask, then there is a requirement for 5+4*4 = 21 words per area and a total of approx. 210000 words of storage are required for the area and mask beads.

From the above discussion it appears as if the area bead concept is a non starter because of the high storage requirements. However, there are several mitigating reasons why it is an extremely attractive concept.

1) Area beads and the corresponding mask beads are not entered into the data structure until there are actually required.

2) Most integrated circuit layouts use repeated components and grouped components and this reduces the number of area beads that are required.

3) The integrated circuit comparator designed by the Wolfson unit is approx 180 thou by 180 thou and only requires a coordinate range of 1760 by 2200 increments i.e. nowhere near the full coordinate range. The largest layouts presently being designed are approx. 250 thou square and assuming the increment is unchanged, require 2500 by 2500 increment (the maximum you will remember is 32000 by 32000). Photographic and semiconductor processing will doubtlessly improve so that finer geometry lines will be used. Assuming the increment size is halved in the future and the size of the layout increase to 350 thou

square, even then the coordinate range is only 7000 by 7000 increments.

4) The improvement in times to plot and modify windows due to the reduction in disc reads as discussed in Chapter 8 are extremely important for an efficient design system.

5) As circuits get bigger the time taken to process the whole of the data for each mask will increase and will therefore increase the need for a method of reducing the ammount of data processed.

6) There are the savings in plotting time due to the area beads as discussed earlier in this chapter.

7) There are tremendous advantages to be gained from the area bead concept when it comes to checking the layout design which will be discussed in Chapter 9.

8) Finally the increase in data structure size is nowhere near as large as expected for a typical layout. The layout shown in fig 8.24 required 40 pages with only 47 words on the last page for the initial data structure i.e. without area beads a total of 19859 words while the data structure with area beads for areas of 512 by 512 increments required 41 pages with 217 words on the last page giving a total of 20537 words. This is an increase of approx. 3 percent, which is a small price to pay for the advantages described above.

5.5 PAGING THE DATA STRUCTURE

THE GAELIC data structure is so large that it cannot be held entirely in the core memory of the computer and consequently has to be held on disc. The structure is divided into a number of parts known as pages, each page containing an equal number of words (at present 508). In order to interact with the data structure, copies of certain pages are held in core and information is transferred to and from the data structure via these core pages. In other words, if information is to be read from a particular location in the data structure, then a copy of the page containing the location is read into core from disc and the contents of the location read from core. Similarly if information is to be written to a given location, then again the page containing that location is read into core and the data written to it and at some time in the future a copy of the updated page is written back to the disc.

The process of reading pages to core and writing pages back to disc is known as 'paging' and the main problem that any 'paging' algorithm has to solve is how to arrange which pages are to be in core at any one time and which pages should be written back to disc to make room for the next page.

The simplest algorithm to do this uses the following stratagy:

1) read in a copy of a page containing the location

to be changed or examined,

2) change or examine the location and then

3) immediately write a copy of the page back to disc. This stratagy has the advantage of only requiring space for a single page to be held in core but has the distinct disadvantage that there are many unnecessary disc reads and writes. For example, if data is to be read from 4 consecutive locations on the same page, this simple stratagy dictates that the same page must be consecutively read in four times and written out four times.

There are certain modifications that can be made to this simple algorithm that will improve its performance:

1) it is worth checking if the contents of a location are only examined and not changed. In which case there is no need to write the page back to disc.

2) it is also worth checking whether the next location to be examined or modified is on the same page as the previous location. In this case there is no need to read in the same page from disc again.

However, if the pages containing locations are continually alternating, then there is still a large number of disc reads and it is worth considering a more complex system involving more than one page in core.

A more complex system of this type reduces the number of disc reads and writes because of the higher probability that the required page will be in core. However, it does raise the problem of what action must be taken if the

required page is not in core. Obviously one of the pages in core must be copied back to disc if it has been modified and then the required page copied from disc overwriting the previous core page. The problem is deciding which page to overwrite. The simplest system is to use a first in, first out algorithm known alternatively as FIFO or Round Robin. An alternative system that is often used is to count the number of times each page in core is accessed and then overwrite the least used page. This is known as a Frequency algorithm and was at one time considered for GAELIC. It was rejected because it was not efficient for the particular way that the layout description was arranged on the disc. The area beads for a layout are placed at the start of the data structure and for a medium size layout may well be all on page 1. The mask and shape beads for area 0 are then written next followed by the mask and shape beads for area 1 etc. Let us assume they are on pages 2-4 and 5-6 respectively. When plotting out a window of the layout for say mask 1, each area bead in turn must be examined to see if shapes associated with that particular area could lie within the window. When this is the case then the mask ring must be examined for the appropriate mask bead and the shapes on the shape ring examined in turn and plotted if necessary. The page containing the area beads i.e. page 1 is accessed to establish whether the shapes within an area could be in the required window and if so the page is not accessed again until all the shapes on the required mask have been processed. The processing of the shapes can require a

given page to be accessed many times in rapid succession, for example page 2, followed by page 3, followed by page 4, and only when all the shapes within the area have been processed does the program return to investigate the next area bead in the first page. Using a 'frequency' algorithm the first page would probably have been swapped out because it had not been accessed for so long and would have to have been brought back to core. The pages containing the shapes would remain in core for a long time as they had been accessed so many times but may not be required again as the shapes for the next area, i.e. area 1, may be on different pages i.e. pages 5 and 6. To avoid this problem a more complicated algorithm would be required and it was felt that the time spent computing which page to change would be prohibitive. The simple Round Robin system was therefore implemented in which the pages in core were written out in turn.

The Round Robin system could possibly overwrite the area bead page just before it was required and it appeared worth considering a more complex system to avoid this. The system consisted of two round robins superimposed on each other. This was implemented in a special version of GAEL4A and compared with the simple round robin. The method was as follows, each call in the program to the routines that examine or modify the data structure was given an extra parameter which indicated whether it was concerned with either the area beads or the masks and shapes. If the routine call contained a parameter value

from a shape setting then the routine would also only overwrite pages that were in the part of core reserved for the pages containing shape information. If on the other hand the call had the parameter setting for areas then only pages containing area information would be overwritten. Whenever a location was required then all the core pages were checked to see if they contained the appropriate disc page regardless of whether the particular core page was called in for area information or shape information, and so the pages are not restricted to having area information only or shape information only. [The results given in Chapter 8 show that it had no appreciable reduction in the number of disc reads and in fact used more CPU Time].

Another interesting facit in the handling of the data structure is the method of checking whether a copy of the required page is in core or not. Two different methods are in use in different programs of the suite. The methods rely on keeping either a list of the contents of each core page or a list of where each disc page is situated i.e. in core but not written to, in core and written to or not in core.

The first method requires a one dimensional array of length equal to the number of pages in core i.e. if there is room for 5 pages in core then the array is 5 words long. The array contains the number of disc page that is in the corresponding core page i.e. the first word in the array is the number of the disc page that is held in the

first core page etc. The number is negated if the page has been written to. When a given disc page is to be examined or modified, each word of the array is examined in turn to see if the corresponding core page contains the required disc page, and if not arranges for one of the core pages to be overwritten.

The second method requires an array of length equal to the number of posible pages that can be held on disc i.e. a much larger array. The number stored in each element indicates whether there is a copy of that page in core and which core page it occupies and whether it hs been modified since it was brought into core. Each time a specific page is required only the one element in the array need be examined. The second method is, therefore, a faster system as it requires only one array access rather than the possibility of 5 array access where 5 is the number of pages in core. However the second method does require a larger array and the original versions of the program working on the Systemshare time sharing service were severely restricted on the core available for the program and the first method with the smaller array had to be used. The restriction was so severe that certain of the programs used all the available core except for one or two words.

More core was available on the Decsystem 10 and a special version of the second algorithm was written by Dr. W.D.Hay in Macro 10 where use was made of the sophisticated indirect addressing features of the machine

code to automatically write out and call in the required disc page.

CHAPTER 6: Graphic Output and Input

The main forte of the GAELIC system is the ability to interactively modify a layout design. The ease with which this can be done depends to a large extent on the choice of data structure and this choice was described in Chapters 4 and 5. To a lesser extent, it also depends on the choice of hardware used for the graphical output and input. In addition the choice of the hardware has a distinct effect on the cost of the overall system. The various options that were considered are discussed in this chapter.

6.1 Graphical Output Devices

The types of device capable of producing graphical output vary from a sophisticated refreshed cathode ray tube terminal through to a simple X-Y recorder. Each type has its own advantages and disadvantages and these will be discussed below.

6.1.1 Refreshed CRT Graphic Terminals

This is without doubt the most well known graphics terminal in use and is the one that immediately springs to mind when the term 'interactive graphics' is muted. Essentially it consists of a high quality CRT tube with the necessary D to A converters, video amplifiers and drivers to convert the digital signals into either

movements of the electron beam or into various beam intensity levels. Usually it also has a display processor which takes instructions stored as bit pattern in the computer memory and converts them into the necessary input signals. This set of instructions is referred to as a 'display file'. Although the cathode ray tube has a relatively high persistence phosphor on its screen it is essential to refresh the picture approximately 50 times per second. The actual speed at which the picture can be redrawn or 'refreshed' depends on two factors: firstly the speed of the electronics, how quickly data in core can be converted into movements of the electron beam and secondly the size of the display file i.e. how much data there is to be displayed. If the electronics are too slow or there is too much data, then the picture cannot be redrawn quickly enough to give the impression of a continuous picture and a phenomenon known as 'flicker' is observed. This is when the drawing appears to flash on and off. This flicker is generally extremely disturbing to the user causing him to become tired and to lose concentration. The 'speed' of a refreshed graphics terminal is defined as the number of characters or the number of vector inches that can be displayed without appreciable flicker and typical values are 2000 characters or 3000 vector inches.

There are several ways of producing a picture on the screen, the best known is probably the raster scan which is used in television sets but in general is too slow for

many applications as the complete screen must be scanned regardless of how much of the screen is dark. Also complex scan conversion equipment is required to convert the data into a raster. The more usual method is to use a steered beam or vector generator when the only dark lines or vectors that are drawn re those between the light vectors. Thus the minimum amount of beam movement is is required to produce the picture.

The main advantage of the refreshed graphics CRT is the ease of interaction. It is comparatively easy to note the position in the display file when a shape is detected by a light pen (The light pen and its use will be described in detail later in this chapter). A shape or series of shapes can be moved across the screen dynamically so that the shapes follow the cursor or tracking cross. The tracking cross may be attached to the top right hand corner of the shapes but the positioning of the bottom left hand corner may be critical. With a refreshed graphics CRT the bottom left hand corner will always be on display at all the intermediate positions and so can be continuously moved until its correct position is reached when it can be fixed. This dynamic movement of shapes is not possible on other graphic output systems.

The main disadvantage of the refreshed CRT is the cost of the hardware which is typically of the order of 10,000 pounds (cheaper systems costing 5-6000 pounds are just starting to appear on the market). There are additional disadvantages when using a refreshed CRT for

integrated circuit layout because the display file is so large that an extremely large memory is required in the host computer to drive the terminal. Admittedly the full layout of an integrated circuit is not often displayed as even with the largest refreshed CRT, available, the resulting picture is at too small a scale for interaction. Nevertheless, it is occasionally required to identify areas of the layout that require attention. Usually a small area of the layout or window is drawn on the screen requiring only a small display file but of course, as soon as the window is changed a new display file is required. The time taken to produce this new file can be significant and the user has either a blank screen or a jumbled mixture of old and new pictures during that period. Certain modern CRT terminals, for example the Vector General 2D3 do have hardware windowing which allows only part of the display file to be plotted on the screen. This hardware windowing does alleviate the problem to an extent as often the data for the next window is in the display file but there will obviously be times when that is not the case and so the file must be recreated.

The refreshed graphics terminal cannot be used on its own with a time sharing computer because of the necessity to continuously refresh the picture. Even with the highest transmission speeds used in time sharing computer systems, the amount of data required to redraw a picture thirty times a second is prohibitive. There is also the additional problem that the user will have his job swapped

in and out of core at intervals giving pauses when redrawing the picture. The only way refreshed graphic terminals can be used is to use a satellite computer as is done on the Decsystem-10 at Edinburgh University [ref 6.1] and the system at the CAD centre at Cambridge [ref 6.2]. The satellite is faced with similar problems to a dedicated computer in that it must be able to hold the complete display file in core and must be possible to redraw the picture at least 30 times per second. This means that the satellite must be fast and have a large memory and is therefore getting very near to the specification of a stand alone computer system.

There are some minor advantages of the refreshed graphic terminal:

1) the line texture i.e. intensity and/or mark space ratio can be varied,

2) a shape can be flashed to indicate that it has been identified and

3) it is also possible to delete or selectively erase components without recreating the display file.

6.1.2 Storage C.R.T. Terminal

The main feature of this type of terminal is the storage cathode ray tube. This is similar in many ways to the conventional CRT but has an extra layer of a special proprietry material on the screen in addition to the conventional phosphor. Each individual molecule of this material can exist in one of two stable states: the first

will radiate light when exposed to low velocity electrons and the second will not. To provide a source of these electrons there is a flood gun assembly in addition to the standard assembly which 'floods' the whole of the screen with low velocity electrons. The material can be changed to the emitting state by means of the conventional electron beam and as the low velocity electrons are always present the parts of the material changed by this electron beam will immediately emit light and will hence store the picture. The material can only be returned to the non emitting state by a flash of high velocity electrons all over the screen. There is no mechanism for selective erasure of parts of the screen and so individual deletions are not possible. The picture can be built slowly and there can be pauses as the picture is built up and so it is an ideal terminal for direct connection to a time-sharing service. It does have the disadvantage that the interaction is slightly restricted. It is not possible to tow a shape or series of shapes across the screen as they will leave an permanent image at each position they are drawn. The new Tektronix 4014 Terminal which is just coming into production does have a 'write through mode' which will allow for non storing pictures but this will require a fast uninterupted data rate from the computer to allow the shape or shapes to be drawn instantaneously.

6.1.3 Incremental Plotters

These produce permanent drawings usually in ink on paper. The paper is fed from a roll over a drum which is controlled by a stepping motor. Over the top of the drum is a gantry containing a tool holder which can be moved up and down the axis of the drum by means of a second stepping motor. The tool holder normally contains a pen but can contain a knife or scriber. In effect therefore the pen can be moved in X and Y direction across the paper. There is also a solenoid built into the tool holder which lifts or lowers the pen to the paper. There is a small amount of logic associated with the plotter which converts the characters sent to the plotter into actual steps on the stepping motors or lifts and lowers the pen.

Because of the characteristics of the stepping motors, the incremental plotter although more accurate than the both cathode ray tube terminals is an order of magnitude slower. It can, however, produce large reasonably accurate drawings with different colours and different line thicknesses to distinguish between parts. This is a permanent hard copy that a designer or engineer can take away and study at leisure. It can be connected to a time-sharing service either as a common shared peripheral like the card reader or magnetic tape unit or by using a special controller via the time sharing teletype inputs.

The direct connection to the computer means that interaction with the drawing is impossible. A hard copy drawing can be produced and that is all. However by using the special hardware controller a restricted amount of interaction is possible. A drawing can be produced and the pen can be moved to a point on a shape in the drawing requiring modifiction. This cannot be done by moving the tool holder by hand as there is no way the plotter can send the new coordinates to the computer. Therefore the user must enter the necessary dat into the computer and the computer must move the tool holder. This produces a 'chicken and egg' problem as the user has to type in the coordinates to which the tool holder is to be moved, in order for the program to identify the coordinates of the nearest points in the data structure to the tool holder. It is possible to enter incremental moves which does make it possible to move the pen to the correct place without having to calculate the absolute coordinates and this method of interaction although slow is feasible.

6.1.4 Tape Controlled Coordinatographs

Tape Controlled Coordinatographs are similar to incremental plotters in that they are capable of producing large hard copy drawings. The paper, however, is fixed to a large flat table and is capable of producing larger more accurate drawings. The tool holder is again held on a gantry and is normally controlled by a stepping motor though certain models do use other techniques. The gantry

usually moves across the table or the table moves under the gantry. The table size is usually of the order of 4 feet by 3 feet though very large tables are available. The accuracy is usually of the order of 1 thou with repeatability of 0.5 thou. There is always a requirement for hardware to read the data from the input tape and convert it into pulses to the stepping motors or instructions to lift and lower the tool. The main differences apart from the size and accuracy of the drawing is the ability to take other tools such as a scriber knife or photographic projector. The latter two require extra facilities from the hardware in that extra information must be read from the tape that controls the angle of the knife or controls which apperture is used on the photographic projector. In order to reduce the amount of data on the type or to make the data on the tape readable the hardware can consist of a small computer.

The tape controlled coordinatograph is not usually thought of as a graphics output terminal but is capable of producing hard copy drawings just as the incremental plotter. Its more usual use in integrated circuit production is for producing mask masters but a mask making system with a tape controlled coordinatograph would not require an incremental plotter. The main advantage of the tape controlled coordinatograph is the accuracy and repeatability of the drawing, cutting or photo exposing. It is, however, expensive (between 20 - 80,000 pounds) and slow.

6.2 Graphic Input Devices

Any graphic input device must fulfill the two basic functions of identifying an object already displayed on the output device and pointing to a specific position. These are completely separate functions and are often refered to as 'picking' and 'pointing'. Some input devices are ideal for picking but difficult for pointing while the reverse is true for others.

6.2.1 Light Pen

The light pen is without doubt the most common graphic input device. It is a hand held light detector with a limited field of view which is usually pen shaped and is connected to the computer by means of an electronic cable. It usually contains a shutter with which the user may control whether light enters the pen or not. The 'pen' can be pointed at the screen and when it sees light, a signal or interrupt is sent to the computer and the process of plotting is interrupted. It will be evident that if the computer has already finished plotting when the light pen is pointed at the screen then there will be no interupt generated. Consequently, the light pen will not operate on a storage tube except during the actual plotting time. This effectively prohibits the use of a light pen on a storage tube as it is too slow and too inconvenient to replot every time an object is to be identified. The light pen is is therefore restricted to refreshed graphic systems. The displayfile is modified

slightly to contain an identification of each object displayed and as the display file is processed the identification of the present object is stored in a buffer or accumulator and is updated as each new object is processed. When an interrupt is generated by the light pen, the identification of the object being processed can be retrieved and the data describing the object itself which is stored in the main data structure can be modified. It is therefore an excellent method for 'picking' on a refreshed graphic system as it requires very little modification of the normal plotting facility and is extremely fast. Its main disadvantage is the increase in size of the display file required to store the identification of each object.

The light pen cannot, however, be used on its own for 'pointing' as an interrupt cannot be generated unless light is detected and in general there will be no light emanating from the point where a new object is to be inserted. The usual way of solving this 'pointing' is to use a tracking cross which is described in the next section.

There are ergonomic problems when using the light pen. The pen must be held in the hand and moved across the screen to the designed position. To avoid the possibility of detecting the wrong object, the pen must be held perpendicular to the screen and this results in the picture being probably obscured by the pen and the users hand. The pen is also held at an unnatural position

similar to writing on a wall and this can be very tiring if used for long periods.

6.2.2 Tracking Cross

This is usually a small cross that is displayed on the cathode ray tube which can be moved round the screen by a light pen, tracker ball, joystick etc. It can be used for both 'picking' or 'pointing'. It can also take the form of a crosshair cursor consisting of a pair of fine lines, one going from side to side of the screen and the other from top to bottom. They are normally found on refreshed tube terminals and on storage tube terminals but on the latter they have to be specially designed so that they are non-storing i.e. the beam intensity must be so low that it cannot change the extra layer on to the back of the storage tube screen.

When used with a light pen moving the cross is difficult. The light from the tracking cross must be detected and the fact that the pen is not central to the cross noted and the cross then repositioned. If the tracking cross and field of view of the pen are as shown in fig 6.2.1.

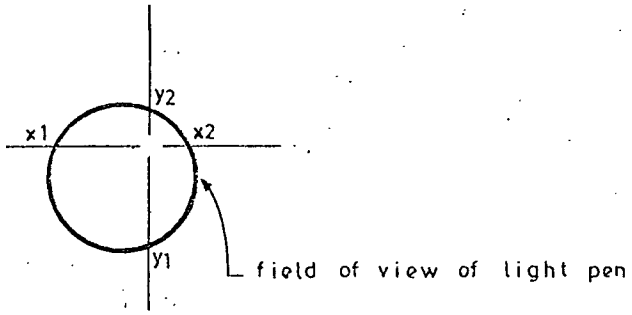


Fig 6.2.1 Detecting the Tracking Cross

The basic way of ensuring that the tracking cross follows the light pen is to reposition the cross at (x1+x2)/2, (y1+y2)/2. This simple scheme and more sophisticated interruption schemes that include the distance moved since the last interrupt all have problems with losing tracking e.g. the light pen has been moved so quickly that it does not detect any light from the tracking cross during a replot of the drawing. This means another mechanism must be used to find the position of the tracking cross. Two such mechanisms consist of spiral or raster search patterns such as those shown in figs 6.2.2 and 6.2.3.

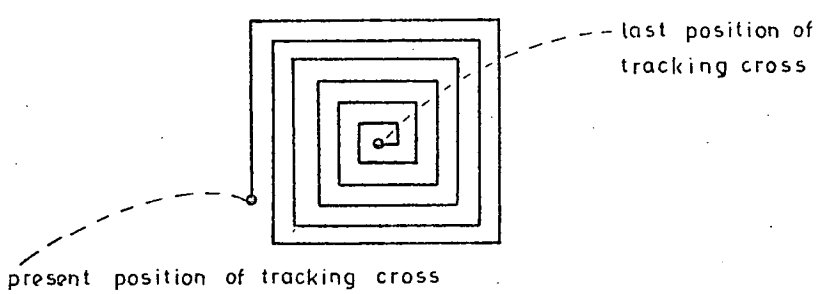


Fig 6.2.2 Spiral Search Pattern

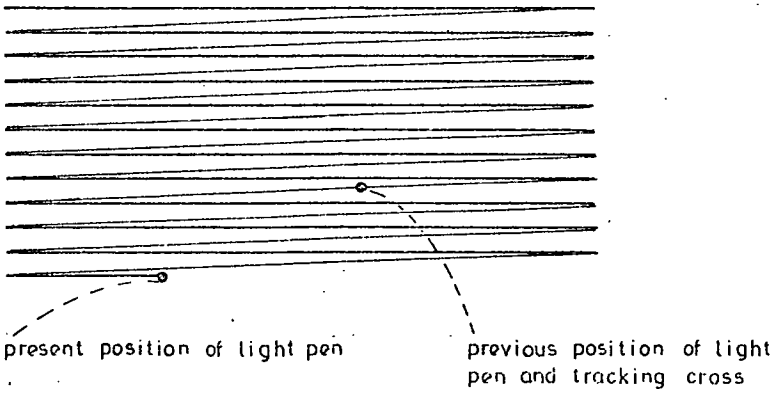


Fig 6.2.3 Raster Scan Search Pattern

This tracking of the cross inflicts a heavy overhead on the picture processing especially the spiral searching and it is common to loose the picture instantaneously as the controller relinquishes its refreshing duty to control the tracking cross.

The position of a tracking cross can be controlled by other input devices in addition to the light pen for example a tracker ball, joystick, a pair of thumb wheel potentiometers or a tablet. In these cases, the position of the cross is directly related to the x and y coordinate positions given by the ball etc. There are none of the problems of loosing 'tracking' and so the system using a tracking cross this way are pleasanter to use than systems using a light pen. There are, however, problems with these systems when it comes to 'picking' as the beautifully simple system involving light pen 'hits' is not available. The data structure must be searched for the object with the nearest coordinates to the coordinates of the tracking cross. Theoretically this data structure search could be through the display file with the same object indications as when using a light pen but usually

it consists of searching the main data structure. This can obviously be a time consuming operation and one of the novel features of the GAELIC system is the way in which this search time is reduced. The various methods of controlling the tracking cross will now be discussed in more detail.

6.2.3 Tracker Ball

This is a ball approx 3 inches in diameter which is recessed into a horizontal surface or table so that only approx the top third can be contacted. The ball can be turned by hand and the rotation in x and y direction is detected by optical shift encoders or potentiometers and translated into the x and y coordinates of a tracking cross. It is usually arranged so that several revolutions of the ball are required to move the cross from one side of the screen to the other. This makes the tracker ball an extremely accurate method of positioning the cross compared wth say the light pen but it does mean that moving the cross over large distances can be relatively slow. The ball can, however, be released and the cross will stay in the same position.

6.2.4 Joystick

The joystick as the name implies is functionally similar to an aircraft joystick, in that the movement of the stick in any direction is converted to movement of some other object in the same direction. In our case

movement of the stick forward causes the tracking cross to move up the screen. It is not quite as accurate as the tracker ball but is much quicker to use, as a much shorter movement of the stick is required to move the cross from one side to another. It is not always possible to release the stick and leave the tracking cross in position.

6.2.5 Thumb Wheel Potentiometers

There are usually two potentiometers, one controlling the x coordinate and one controlling the y coordinate of the tracking cross. The potentiometers may be single turn or multi turn: the former is capable of moving the cross faster but less accurately than the later. The system is ideal when all the movement required is in either the x direction only or the y direction only but are not as convenient when movement at an angle is required. Like the tracker ball, the potentiometers can be released without moving the tracking cross making any interactive graphics system ergonomically easy to use.

6.2.6 Tablet

This consists of a flat surface usually of the order of 12 inches square which has a grid of fine wires embedded in its surface and a scriber or pen that is capable of emitting signals. These signals are detected by the grid and the accurate x-y position computed. This system is ergonomically excellent as the user is virtually using what he has been trained to use since childhood, a

pen and paper. The pen can be quickly moved from one corner of the screen to another and so a menu of useful commands can be put on the tablet and pointed to when required i.e. 'menu picking'.

6.2.7 Digitiser

The digitiser is virtually the same as a tablet except that the working surface is bigger and that the coordinate position is only sent to the computer when a button is pressed. The working surface is usually about 3 feet long by 4 feet and can be at an angle or flat. It is extremely useful for extracting dimensional information from a scale drawing but is expensive as a graphical input device.

6.2.8 Other input devices.

There are other possible graphic input devices capable of driving a tracking cross or cursor. Probably the most well know, although not necessarily as an input device is the teletype when the user can type in the coordinate required. This sounds at first like a chicken and egg problem - to move the cursor it is necessary to type in the coordinates of the point to be identified and the point is being identified in order to find its coordinates. However, by using incremental coordinates, it is a possible system albeit slow and has been used to control the tool holder on a CALCOMP plotter.

Another system that was originally used on the ARDS storage tube terminal is called a 'MOUSE', this is a device that is moved over a flat smooth surface to indicate x or y coordintes. Underneath are two wheels at right angles, both are connected to shaft encoders or potentiometers, one indicating the x and the other indicating the y coordinates.

Other systems that are being developed are touch sensitive screens and ultrasonic transduces but these are not yet in production.

6.3 Tektronix 4010 Series Terminal

The decision to base the interactive part of the GAELIC suite of programs on the Tektronix 4010 terminal was due to the requirement of a minimum capital cost system.

The Tektronix 4010 series terminal consists of a storage cathode ray tube mounted on a stand, with a keyboard and two thumb wheel potentiometers. The stand contains all the control logic and the necessary interfaces to connect the terminal to the computer. The basic 4010 terminal has a screen size of approximately 8" by 6" and costs approx. 2500 pounds with the necessary interface to connect it to the computer in place of a standard Teletype.

The terminal can work in either alphanumeric mode when it will print lines of alphanumic characters and the graphic mode where it will draw vectors on the screen. There is also a graphic input mode when a non-storing cross hair cursor is displayed on the screen which enable existing items, i.e. text or vector to be identified and also allows the position of addition text or vectors to be indicated.

The alphanumic characters are produced by means of a hardware character generator employing read only memories. Once in alphanumeric mode, the codes or bit patterns for the characters are sent down the line to the terminal. The codes for carriage return and line feed do exactly as expected by ensuring the next character appears at the left hand edge of the screen or on the next line below respectively. Most other non printing characters are ignored: the main exception being the character usually known as 'GS' the receipt of which converts the terminal into graphics mode. Subsequent characters sent down the line are converted into vectors, the first vector after the 'GS' always being a 'dark' or hidden vector. Usually four characters are required to specify a vector and they are known as high y, low y, high x, low x characters. There are 1024 by 1024 addressable points on the screen with 1024 x 780 actually viewable i.e. it is possible to address points off the screen in the y direction. Vectors are drawn from the present beam position to the position defined by the four characters. The screen is divided

into a coarse grids of 32 addressable points in each direction; the high y and high x characters select the position on these coarse grids. Each coarse grid increment is divided into a fine grid of 32 points and the low y and low x characters then select the incremental moves on this fine grid to give the final absolute position. The number range for each character is therefore 0-31 only and so for the standard byte (8 bit) characters, there are 2 redundant bits plus the parity bit. These redundent bits are used to define which of the characters are being transmitted and so the various bit patterns are shown in fig 6.3.1.

Bits	7	6	5	4	3	2	1	
Low X	1	0	?	?	?	?	?	Byte 4
High X	0	1	?	?	?	?	?	Byte 3
Low Y	1	1	?	?	?	?	?	Byte 2
High Y	0	1	?	?	?	?	?	Byte 1

Identification Bits

Bit patterns transmitted to terminal to draw line

Fig 6.3.1

The identification bits allow for less than 4 characters to be sent under certain circumstances. The low x character must always be sent as this character is used to initiate the drawing of the vector. However, if the high y or high x character is not changed from the previous vector, then the character need not be sent. If the low y character is not changed then it need not be sent unless the high y is changed and then because there is no difference between the identification for the high x and high y bits the low y bit must be sent. The minimum number of characters that must be sent is therefore one, the low x character. This reduction on the number of characters is extremely important when connecting the terminal to the computer by means of a low speed line.

Certain pairs of characters when sent to the terminal have special effects, the most important is the pair which convert the terminal into graphic input mode. In this mode a non storing cross hair cursor is displayed, the x position of the vertical line is controlled by one thumb wheel potentiometer and the y position of the horizontal line controlled by another. The non-storing feature is obtained by rapidly switching on and off the beam. The cursor can be therefore moved to any position on the screen and one of the keys pressed, the terminal then transmits the character requested by the key, plus 4 characters that represent the coordinates of the cursor, followed by carriage return and or the end of tape character (EOT). The characters use the same method of

denoting coordinates i.e. high x, low x, high y, low y but all four characters have the same identification bits set.

Other pairs of characters clear the screen and arrange the next alphanumeric characters to be written at the top left corner of the screen.

There are two larger versions of the Tektronix terminal now available and these are known as the 4014 and 4015. Here the screen size is 15 inches by 11 inches and has a resolution of 4096 by 3120 increments. To address this in full an additional characters is sent to the terminal. The terminal also has additional facilities such as 'write through' or non storing mode and the capability of producing dashed lines by hardware. This terminal would give considerable increase in performance but unfortunately costs approximately 4,500 pounds.

The Tektronix 4010 series can be connected to a computer using a teletype driver interface with the clock rate increased from 110 baud to 9.6k baud, and so can be easily connected to most computers whether stand alone or time sharing.

The choice of the Tektronix terminal and Fortran as a programming language means that the software can quickly be implemented into different existing computers and thus provides an inexpensive method of obtaining a design system.

6.4 Graphics Software

This section is devoted to the basic software required to drive the Tektronix 4010 series of storage tube terminals. It will be remembered that there are three modes of operation of the terminal: alphanumeric, graphics output and graphics input and any basic software system must cater for all three.

When used in alphanumeric mode, the terminal behaves in virtually the same way as the ASR33 Teletype in that it receives the bit pattern for the various ASCii printing characters plus carriage return <CR> and line feed <LF> and displays the characters on the screen or moves to the beginning of the line or next line. The terminal also transmits the bit patterns, corresponding to the keys pressed, to the computer. If the Tektronix is connected to the computer via a standard Teletype interface, then no special software is required to drive the terminal in alphanumeric mode.

In graphics output mode, however, the terminal must obviously behave in a different way in order to draw the vectors. In this mode each character transmitted is interpreted by the terminal as part of the description of a vector. There are two main types of vector that are required in any graphics system: the dark vector where an invisible line is drawn from the present beam position to a new specified position and a light vector where a visible line is drawn. The light vector can be subdivided

into vectors of differing intensity or into dotted or dashed vectors. The storage tube terminals cannot display vectors of differing intensities because of the fundamental characteristics of the storage tube and so there is no need to cater for them. The Tektronix 4010 and 4012 terminals do not have any hardware facilities for producing dotted and dashed vectors and so these can only be obtained by drawing alternate light and dark component vectors. The 4014 and 4015 terminals do have hardware facilities for dotted and dashed vectors and the type of light vector to be drawn is set by transmitting a special character to the terminal. The Tektronix terminals distinguish between the light and dark vectors in an unusual way by arranging that the hardware treats the first vector after turning the terminal into graphics mode as a dark vector and treats all other vectors as bright. The terminal is turned into graphics mode on receipt of the character usually known as 'GS'. This method of defining whether a vector is bright or dark appears at first to be extremely restrictive until it is realised that:

a) most shapes to be drawn are made up of a consecutive series of bright vectors preceeded by a dark vector e.g. when drawing a rectangle, a dark vector is drawn to the one corner and then four consecutive bright vectors, one for each side, and

b) another dark vector can be specified by sending another 'GS' although the terminal is already in graphics mode. Vectors are always drawn from the present beam

position which is at the end of the previous vector or at the position for the next character to be printed.

The end of the vector is specified by sending four characters to the terminal, two of these characters specify the absolute screen coordinates in terms of a coarse grid. The other two characters specify the incremental coordinates from the coarse grid position. The coarse grid is every 32 increments and as the addressable range is 0-1023 increments in the x and y direction, the maximum range for the coarse grid is 0-31 i.e. 5 bits. The two coordinates specifying the coarse grid position are known as the high x and high y coordinates; the coordinates specifying the incremental position from the coarse grid are known as low x and low y coordinates and again are in the range 0-31. Only 5 bits are required to specify the value and so with the 7 bit characters used there are two bits available to identify the character i.e. high x, low y etc. The actual bits that are sent for each character are shown in fig 6.4.1.

Bits	7 6 5 4 3 2 1	
Low X	1 0 ? ? ? ? ?	Byte 4
High X	0 1 ? ? ? ? ?	Byte 3
Low Y	1 1 ? ? ? ? ?	Byte 2
High Y	0 1 ? ? ? ? ?	Byte 1

---

^

Identification Bits

Bit patterns transmitted to terminal to draw line

Fig 6.4.1

It will be noticed that the identification bits for the high x and high y coordinates are the same. The order of transmission of these characters must be high y, low y, high x and finally low x coordinates and it is the receipt of the low x coordinate that initiates the actual vector drawing process.

The setting of the identification bits allows the characters sent to switch the terminal into another mode to be detected and also allows one or more characters to be omitted under certain circumstances when specifying vectors. For example, as the low y coordinate can be uniquely identified then if the high y coordinate is not changed for the vector then it may be omitted. Similarly the high x coordinate may be omitted if it has not changed

as the low x coordinate can be uniquely identified. However, the low y coordinate cannot be omitted if either a high x or high y coordinate has to be sent as they both have the same identification bit settings. The minimum number of coordinates that need to be transmitted is one and that is the low x coordinate. This is the character that initiates the drawing process and so must always be sent even if none of the coordinates have been changed. This situation does occur when plotting points on the screen. To enable the terminal to draw vectors as fast as possible at low data transmission speeds, the basic software must send the minimum number of characters for each vector.

At high data rates i.e. in excess of 4.8K baud, the minimum number of characters will cause trouble when the terminal is connected via a teletype interface. This is caused by the time required to draw a vector. On receipt of the low x coordinate, the terminal takes 2.6 mSec to set up the D to A convertors and to draw the vector, most of the time being spent setting up the convertors. at 9.6K Baud, a character is received approximately every 1.5 mSec i.e. at least two characters can be received while the previous vector is being drawn. Therefore any vectors requiring two characters or less i.e. vectors requiring low y and low x, high x and low x or just low x will be initiated before the previous vector has been completed. The result of this is to change the D to A convertors as the vector is being drawn which will give curved vectors

on the screen at apparent random spacing. If the software is to be used at varying data transmission speeds, it is essential to either send at least 3 coordinates or else to send null characters when the data rate is 9.6K Baud, and to send the minimum number of characters at lower speeds.

If the terminal is connected to the computer by means of a link involving a busy signal, the software need only send the minimum number of characters.

The conversion of normal coordinate data into the required characters is common to all applications that require graphic output and conversion routines must form part of the basic software. However, very rarely does the coordinate range used in the application program map exactly with the coordinate range of the terminal i.e. 1-1024 in x and 1-780 in y. This creates two additional requirements for the basic software: firstly routines are required that will scale the application programs coordinates so that they will appear on the terminal screen. Secondly 'clipping' routines are required that will only display the part of the drawn or design that lies within a specified 'window'. This routines will take each vector in turn and clip it so only the part of the vector that appears on the screen is drawn.

Because of the fundamental modus operandi of the storage tube, it is not possible to vary the intensity of a vector nor is it usually possible to blink or flash vectors. The only way of distinguishing vectors,

therefore, is to have dotted or dashed vectors with different mark space ratios. This can be done by hardware on the Tektronix 4014 and 4015 and in this case all the basic software need do is ensure that the appropriate characters are sent whenever the vector characteristics are to be changed. The smaller Tektronix, the 4010 and 4012, however, do not have the hardware facility and so it must be done by software and so in this case the basic software must contain routines to break a long vector down into alternate light and dark vectors of appropriate length.

The terminal is switched into graphics input mode by sending it two characters known as 'esc' and 'sub'. This causes the cross hair cursor to be displayed on the screen and the cursor position can be controlled by the two thumb wheel potentiometers. Pressing any key will cause a series of characters to be transmitted to the computer. The characters include that of the key pressed and four characters to define the position of the cross hair cursor. The characters transmitted are shown in fig 6.4.2.

Bits	8 7 6 5 4 3 2 1	
EOT	1 0 0 0 0 1 0 0	Byte 7
CR	1 0 0 0 1 1 0 1	Byte 6
Low Y	1 0 1 ? ? ? ? ?	Byte 5
High Y	1 0 1 ? ? ? ? ?	Byte 4
Low X	1 0 1 ? ? ? ? ?	Byte 3
High X	1 0 1 ? ? ? ? ?	Byte 2
Char	1 0 ? ? ? ? ? ?	Byte 1

Bit patterns transmitted from terminal
in Graphic Input Mode

Fig 6.4.2

The last two characters sent i.e. bytes 6 and 7, are strappable options, the terminal can be set to transmit neither character, carriage return only or carriage return and end of tape. Any basic software system should therefore contain routines to set up the cross hair cursor and read the characters transmitted and convert them into coordinates either in Tektronix increments or scaled into the user coordinates.

It is essential to send certain non printing characters to the terminal to perform such functions as switching from one mode to another. Often these characters cannot be transmitted directly from a Fortran program because of the computer used. However, most computers can output these required characters using

routines written in machine code or assembler and callable from Fortran. This is the approach used in GAELIC on the Decsystem 10 where the output to the terminal in graphics output mode and input form it in graphics input mode are controlled by MACRO 10 routines written by Dr W.D. Hay.

The original version of the GAELIC interactive program that uses the Tektronix 4010 terminal used the author's routines for drawing vectors and for handling the cross hair cursor and used a set of routines written by Dr. P.F.A. Reilly to do the vector clipping. As the original version was to work at low data transmission i.e. 110 or 300 Baud dotted and dashed vectors were not practical as they took so long to draw.

However, when the program was implemented on the Decsystem 10 using a data transmission speed of 1200 Baud, dotted and dashed lines became ergonomically possible and therefore desirable. A package of Fortran routines written by Tektronix was available on the Decsystem 10 and this package, known as the Terminal Control System or TCS, not only contained routines to plot vectors and to handle the cross hair cursor but also had its own clipping routines and routines to produce dotted and dashed lines.

The TCS routines use the same technique as the original GAELIC program i.e. routines to do the characters handling written in MACRO10 but callable from Fortran. All other routines are written in a subset of Fortran IV to enable them to be installed on as many computers as

possible. It was therefore decided to use these routines as a) there was no point in reinventing the wheel and b) the TCS software was already implemented on many computers and so using it would ease the transportability of the program.

The TCS system does have some minor disadvantages as it is a general purpose package. To give the user a large range of increments on a computer with a small word size e.g. a PDP8 with its 12 bit word, the user coordinates are stored as real values and each value therefore requires two words of storage. The scaling to the Tektronix increments is therefore done using real arithmetic i.e. requires the floating point arithmetic package to be in core. This system is therefore slower than it need be and requires more store. This is not noticeable on the Decsystem 10 as: a) it stores real values in one 36 bit word and b) it uses a hardware floating point unit. However, if the GAELIC programs are mounted on smaller machines some modification to the TCS software will be necessary to obtain maximum efficiency.

Only one major software modification was required to the TCS software and that was to allow for use at a data transmission rate of 9.6K baud. The original software minimised the number of characters transmitted to the terminal to draw a vector. On an asynchronous transmission system, this causes apparently random curved vectors due to the arrival of the one or two characters to specify the next vector before the previous vector has

been drawn. This was cured by transmitting the necessary null characters at 9.6K baud, to ensure the previous vector was drawn before the next vector was specified.

The 'clipping' routines written by Dr P.F.A. Reilley have been used in other programs in the GAELIC suite, notably the program that plots all or part of a layout on the Calcomp incremental plotter.

Chapter 7 : Program Descriptions

This chapter is mainly devoted to a description of the various programs comprising the GAELIC suite, concentrating on their general requirements and how these are met. The detailed descriptions of the subroutines appear in the GAELIC Systems Manual. The chapter starts, however, with a discussion of the languages available for programming and why FORTRAN was chosen as this choice did affect the program requirements.

7.1 Choice of Programming Language

The objectives of the GAELIC programs were discussed in Chapter 2 and resulted in the requirement for a minimum capital cost system that was, as far as possible, hardware independent so that it could be easily transferred from one computer to another.

The most efficient CAD system for designing integrated circuit layouts, can theoretically be obtained by selecting or building the best hardware for each part of the system and programming at the lowest possible level. to get the fastest operation. This approach, however, has a lot of disadvantages.

1) The best hardware may come from a series of different manufactuers and may well require special interfaces to interconnect them.

2) The problems encountered when servicing this mixed

hardware are quite formidable (the fault is always in the other manufacturers product).

3) Any special purpose hardware is extremely expensive to design, build and test.

4) Programming and debugging in a low level language is slower and more difficult than in a high level one.

5) It is not possible to write extremely long low level programs and maintain maximum efficiency. However, the higher level language requires much less code to be written and so can be written efficiently.

6) Programs written in the low level language for one computer cannot be transfered to another. Instead the programs must be rewritten in the low level language for the new machine.

These disadvantages, therefore, preclude the use of special purpose hardware. They also discourage the use of low level languages for a system that is to be as portable as possible.

The use of a high level language has the following advantages:

1) The amount of code that has to be written is much less than when using a low level language.

2) The widespread use of high level languages has justified the writing of extremely efficient compilers.

3) The high level languages are to quite a large extent self documenting and so only a small amount of extra documentation is required.

4) Although it would be foolish to claim that there

are no problems in transferring a high level language program from one computer to another, the original programmer's intention is always clear and so the only problems are those of obtaining equivalent facilities on the new computer.

5) Once a system is working in a high level language, it is possible to speed it up by rewriting the critical parts in a low level language.

There are many high level languages that are used nowadays. Unfortunately most of them are not available on a wide range of computers and this precludes the use of some very good languages such as Algol-68 even though it contains facilities for handling the complex data structures There are other languages that have facilities for handling data structures but these are comparatively new and do not have all the other required features such as floating point arithmetic.

This leaves four main contenders for the programming language BASIC, IMP, ALGOL and FORTRAN. These languages will be now considered in more detail.

BASIC

This is probably the simplest of the 'high level languages' in general use. It was originally written by Dartmouth College and is implemented on most commercial time-sharing computers. Because of its simplicity, it is

very easy to use and so is an ideal language for beginners. However, it does have certain disadvantages. Firstly the array and variable names are restricted to either a single letter or a letter followed by a digit. This means that it is extremely difficult to have mnemonic variable names i.e. names that convey the function of the variable to the reader. For example it is extremely useful to store the base emitter voltage of transistor 1 in a variable called VBE1 whereas in BASIC it would have to be called V1 or B1. Another disadvantage that is common to all the other languages is that the computer manufacturers have extended the language to provide extra facilities but each manufacturer has done so in a different way. In general BASIC compilers are extremely fast but do not produce optimum code.

IMP

This is a high level language based on ATLAS AUTOCODE and is in extensive use on various computers at Edinburgh University. There was considerable pressure therefore to use this language. It has a lot of attractive features in the language such as the ability to read in a character at a time from the input channel and the ability to read the next number from the input channel regardless of how many digits it contains. It also contains facilities for bit manipulation which are required in GAELIC. However, it is not a generally available program outside the Edinburgh

University community and as it was hoped that the GAELIC programs would be used outside, the language was not used. Since the decision was taken to use FORTRAN IMP has been extended to contain RECORDS which facilitate the setting up of complex data structures.

ALGOL-60

This is a high level language which is used extensively in Europe especially by educational establishments. It is the international language which most conforms to a standard. Unfortunately the standard does not cover all the facilities available: the notable exception is the input and output routines (There is now a version of ALGOL available which was invented by Worth which does define the input and output and is known as ALGOL-W). In general, however, these input-output facilities are implemented in differing ways by the different computer manufacturers and so again the program is not completely transportable. Algol is not used as much in the United States and consequently most American computer manufactuers have not put as much effort into their Algol as they have into their FORTRAN compilers.

FORTRAN

This is the most widely know and widely used high level language for scientific applications. It was originally invented by I.B.M. for use with the data and program on cards and the output on a line printer. It has been considerably extended since then but still shows signs of its humble beginning by having an extremely rigid input-output system. It now exists in a standard form known as the ANSI Standard FORTRAN which is fairly rigidly defined but almost inevitably has been extended beyond this standard by the individual computer manufacturers. Again programs are therefore not directly transportable between computers. However virtually every computer of a reasonable size has its own FORTRAN compiler and the compilers are often extremely efficient because of the interest in them. Therefore, although parts of programs are not always directly transportable, a programmer can always understand what was intended and what modifications are required to get the program to work on a new machine.

FORTRAN was chosen as the programming language despite certain short-comings because of its universal availability.

7.2 GAEL1A Digitiser Program

This program takes the output data generated by a specially modified Metrograph digitiser and converts it into the GAELIC manual input language and the GAELIC dump code. The output is in these two forms to allow for the correction of any errors made when digitising. If no syntax errors are detected then the dump code file is processed to give the ring data structure of the layout: this saves considerable computer time by not having to run GAEL2A. However, if errors are detected when running GAEL1A they must be corrected. Correcting the digitiser output or the dump code file is extremely difficult whereas editing the manual input language is relatively easy. If the manual input language is corrected then it is then converted into dump code by means of GAEL2A.

The output from the digitiser consists of records of the form shown in Fig. 7.2.1.

```
12340%2X%+00100%Y%+00200        where % indicates a
12350%X%+00150%Y%+00300         space.
12360%X%+00600%Y%+00800
```

Fig 7.2.1 Typical output from Metrograph Digitiser

which can be generalised as a record of the form shown in Fig. 7.2.2.

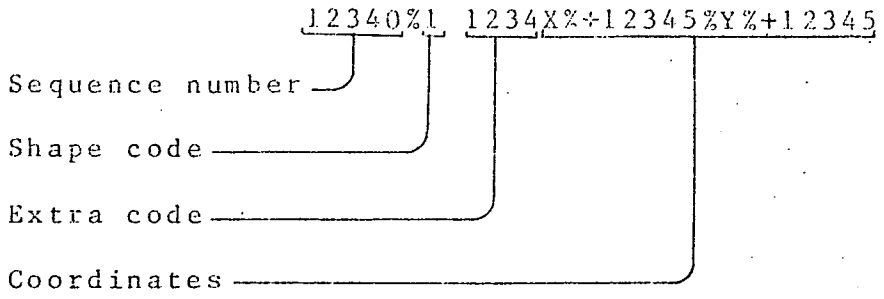


Fig 7.2.2. General form of input record

Fig 7.2.2 also shows how the general record can be subdivided into four sections, two of which are not necessarily present in any particular record.

The sequence number is always present and is always a five digit number which specifies each record. It can be followed by a single digit shape code which is used to indicate the start of a new shape and its type. Consequently the shape code is only present in the record under these circumstances. The four digit extra code is only present when it has information to convey and is used for the name of a group or the radius of a fillet. The final section is always present and contains the coordinates of the digitiser cursor when the READ button was pressed. These coordinates are always in fixed format i.e. the letters, spaces, digits etc. are always in the same position relative to the end of the record though the actual digits present will vary.

The coordinates and sequence number are automatically produced when the READ button is pressed and will always give the same number of digits in the same position. The

shape code or extra code, on the other hand, are entered by the operator, a digit at a time, via the digitiser keyboard and can be a source of errors. Such input cannot be handled by the standard Fortran input/output routines and so another method had to be found.

There are many versions of Fortran that have a non-standard facility which allows the next number to be read regardless of how many digits or their position in the record. This 'free format read' as it is usually called, appears at first sight to be the answer to reading the input records. Unfortunately this facility cannot be used for three reasons:

1) The facility is implemented in different ways on different computers.

2) Invariably the numbers read in under this free format have to be delimited by a standard terminator e.g. a comma or a space. It should consequently be noted that is not the case in an input record.

3) This free format facility will not cope with the correction of digitising errors by means of the ERROR key. The ERROR key on the digitiser inserts the character '#' in the record: this non-numeric character can occur in several places in the record.

The program therefore reads the whole record into the array, each character, be it a letter, digit or punctuation mark is put into one word of the array. A simple arithmetic or logical operation on each word converts it into a number that uniquely specifies the

character. In most computers including the Decsystem-10, this is the ASCII number for the character. For example, the ASCII number for the letter 'A' is 65 and for the digit '1' is 49. By doing various checks and various arithmetic operations on these ASCII numbers, the actual integer numbers in the record can be calculated or the fact that a '#' is present can be detected. There are a series of routines in the program that read in the records, convert the characters to their corresponding ASCII numbers and calculate the integer numbers entered and evaluate their terminating characters.

Two algorithms have been used for the transformation from digitiser coordinates to GAELIC increments. The first was written by R. Newton of R.R.E. for use with the CAMP programs [ref 7.1] and the second by the author. The first algorithm is designed to cope with linear paper distortion in all directions as shown in Fig. 7.2.3. The derivation of the algorithm is given in Appendix 3 where it is shown that it involves the solution of quadratic equations. The roots chosen are those which cause the transformed point to lie on the paper. There is a considerable amount of computation required for each pair of coordinates and there is always a slight doubt as to whether the other roots would also produce a point that lies on the paper. In practice the transformation appeared extremely sensitive to the 'initialisation' digitising and this had to be done with great care.

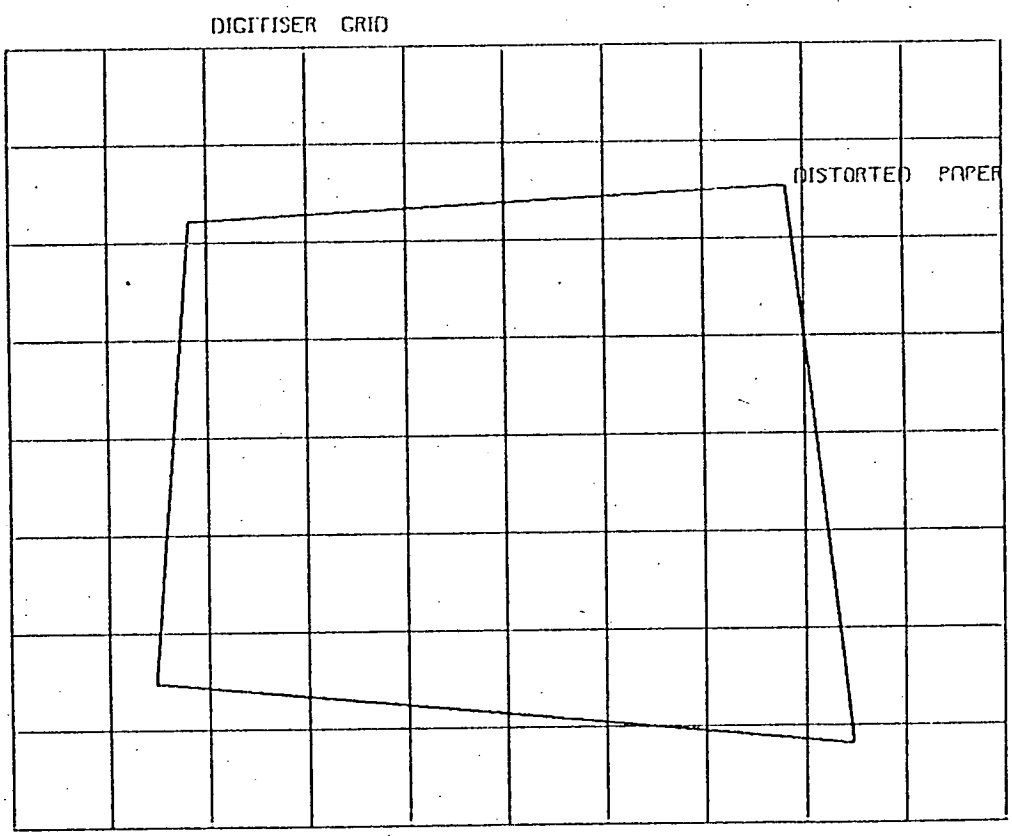


FIG 7 2 3 PAPER DISTORTION ALLOWED IN NEWTONS ALGORITHM

The paper distortion that is allowed by the algorithm is generally more than is required. The grid is usually printed on the paper by means of a roller which ensures that the grid lines are perpendicular but does have a high probability of the scaling being slightly different in the x and y directions. It is also always possible to put the paper on the digitiser at a slight angle to the digitiser axes. This results in a paper distortion shown in Fig. 7.2.4.

By restricting the paper distortion allowed to that shown in Fig. 7.2.4 the simpler algorithm derived in Appendix 4 can be used. This involves the solution of simple linear equations requiring only a small amount of computation and only giving one pair of coordinates. In practice this simpler algorithm appears to be less sensitive to the accuracy of the 'initialisation' digitising. This simpler algorithm is therefore used in GAELIA to transform all the digitiser coordinates into their corresponding paper coordinates which in turn are specified in terms of GAELIC increments. The output from the program i.e. the manual input language and dump code files are therefore in terms of these increments.

The production of the dump code file is straightforward as it is a standard binary file consisting of numbers only; this can be written directly from a Fortran program on almost all computers. To reduce the number of disc writes on the Decsystem-10, a subroutine is used to add the individual numbers to a buffer and then

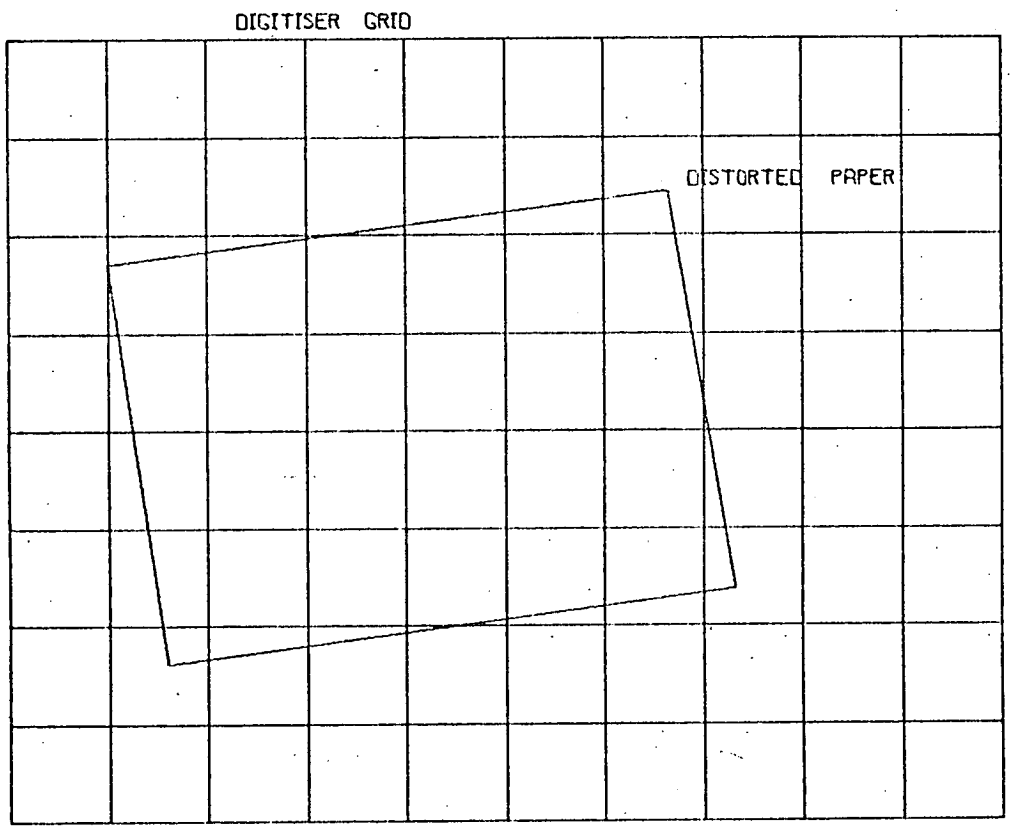


FIG 7 2 4 PAPER DISTORTION ALLOWED BY SECOND ALGORITHM

write the contents of the buffer to disc when necessary.

The production of the manual input language, however, is far more complex as it cannot be done using standard Fortran. Standard Fortran output is similar to the standard input in that the number of data items to be transfered and the number of characters in each item must be known. The manual input language produced by GAEL1A is slightly more rigidly defined than that described in the GAELIC Users Manual in that the order words are always a fixed length, e.g. "RECT" and "POLY" are always used instead of "REC" or "RECTANGLE", "P" or "POLYGON" etc. and the group names are always 5 characters long, e.g. G1234 etc. However the number of digits in a coordinate will vary with its value. It could be argued that as leading zeros or spaces are acceptable in GAEL2A, the program that processes the input language, and so they should be allowed in the output from GAEL1A. However the main purpose of producing the manual input file is in order to modify it to correct errors. This can only be done efficiently if the input language itself is easy to understand: once the user is accustomed to seeing the description of a polygon as:-

```
"POLY" (1) S,5,20:20,2,20,2,-22,-2,-18,-2;
```

then it is very difficult to recognise:-

```
"POLY"
```

```
(01)
S,
 00005, 00020:
 00020, 00002,
 00020, 00002,
-00022,-00002,
-00018,-00002;
```

as the same polygon. The later description is typical of that obtained using standard Fortran output. For this reason the conventional form of the manual input language was chosen for the output from GAELIA. To obtain this output, characters are loaded into a buffer and the buffer is written out to disc when necessary. One subroutine adds the characters that make up an order word or group name to the output buffer, another adds the characters that make up the significant digits of integer numbers and a third adds the punctuation marks.

The simplified flow diagram of the GAELIA program is shown in Fig. 7.2.5. It does not show the sophisticated error correcting system associated with the '#' character nor does it show in detail the processing of any particular type of shape. After defining and opening the input file and the two output files, the input file is processed until the first sequence number is found. The shape code and possible extra code are then read in followed by the coordinates of the shape. The coordinates are stored in an array until the next shape code is

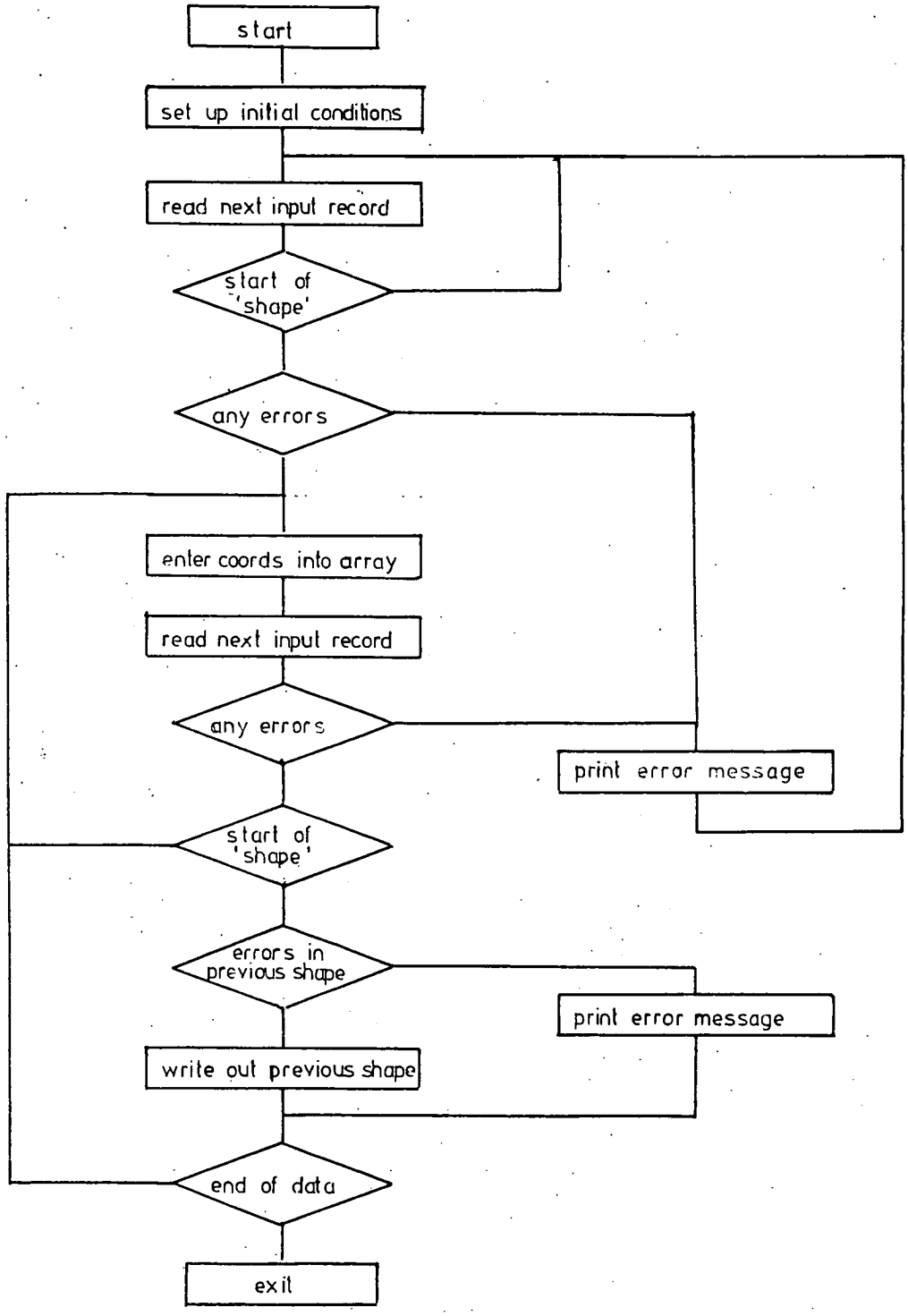


Fig. 7.2.5 Simplified flow diagram of GAEL1A

detected indicating the end of the present shape. This shape is then written out as both GAELIC manual input language and GAELIC dump code after suitable tranformation of the coordinates to allow for paper distortion.

Some interesting features affect the running of the program. When someone first starts using any program, he or she requires a great deal of help and so instructions given by the program must be clear and unambiguous and any errors detected must be explained in full. Both these requirements result in long verbose messages being written on the terminal. If a teletype or any other terminal running at 10 characters per second is being used, these messages will take an appreciable time to be written. However, once the user is familiar with the system then these messages are superfluous and time waisting. All the user requires are short criptic aide memoires. The program provides the user with both types of message and the user selects which is used by answering "YES" or "NO" to the question 'DO YOU WANT EXTENDED PRINTOUT'. The extended printout produces long explicit messages of the form 'ENTER NAME OF FILE TO CONTAIN GAELIC LANGUAGE' rather than the criptic 'GAELIC FILE'. The extended printout also controls the length of any error messages.

7.3 GAEL2A Manual Input Language Processor

This program takes the description of all or part of an integrated circuit layout coded in the GAELIC manual input language and after extensive syntactic checks converts it into the numeric form of the language known as the dump code.

The syntactic checking attempts to detect and produce a meaningful error message for every error in the input data. Unfortunately although the first error in every shape is detected, it is not possible to detect all subsequent errors in the same shape. This is because data containing other errors may have been processed before the first error was detected. As the dump code is stored in a sequential file, it is difficult to modify the data once it has been written. It is therefore easier to ignore data for shapes containing errors rather than write the incorrect shape descriptions to a file. The user is given facilities for adding the corrected shape description at a later stage when again the description will be checked for syntax. This means that if the user corrects the first mistake but repeats the second, it is detected and the user is consequently stopped from entering any illegal data into the dump code file.

The GAELIC manual input language is fully described in the GAELIC users manual [ref 7.2] and so it will be sufficient here to give only the two examples shown below.

```
"RECTANGLE" (1) 5,5: 2480,5860;

"REC" (1:4) 1050,2486: 10,5;
```

Here the respective order words, mask specifiers and coordinates contain a different number of characters and cannot therefore be read using the standard Fortran input routines. A similar technique to that used in GAELIA is employed to read this input data. A complete record is read into an array, one character to one word, a logical operation is performed on each word in turn to give the ASCII number for the character and these numbers are processed to give the order words, the group names, the mask specifiers and the coordinates. The actual routines used are detailed in the GAELIC system manual. Errors are also detected by these routines, suitable messages are then produced messages and, in general, the shape containing the error is ignored.

The simplified flow diagram of the program is shown in Fig. 7.3.1 where it can be seen that after setting up the input and output files the input data is scanned for the double quote forming the start of the first order work. The routine that identifies the order word is then called. This routine first of all checks that the order word is valid and if so returns an integer number which uniquely identifies the word found; otherwise it writes out an error message and sets a flag. The program is directed to various blocks of code depending on the integer number when the rest of the data for the order

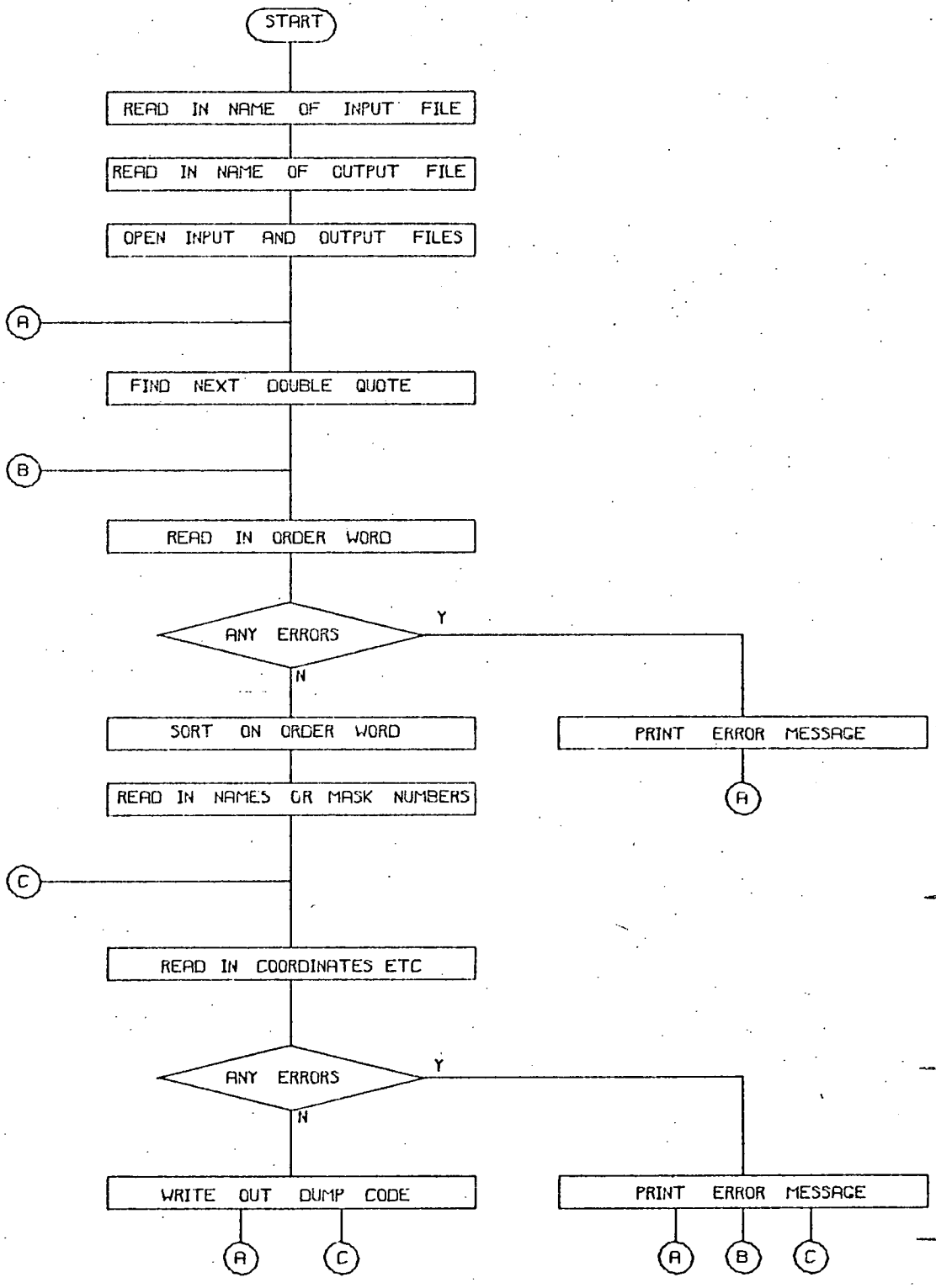


FIG 7 3 1 SIMPLIFIED FLOW DIAGRAM FOR GAEL2A

word is processed. The order words for the basic shapes i.e. rectangles, polygons and lines are processed in the same block of code, this is a relic from the original version of the program running on the Systemshare time-sharing service where core was at a premium and there was a heavy overhead on subroutine calls. Here the rest of the data describing the shape is checked and if correct, the appropriate dump code is written out. The program then either returns to look for the next order word or to look for more coordinates if the DITTO flag is set. The use of the "DITTO" order word is described in the GAELIC users manual and is a facility for reducing the amount of data required for a series of rectangles etc. If errors are detected then the program returns to one of three different places depending on where the error was found. This feature of the program can be best understood by reference to the flow diagram for the processing of a rectangle shown in Fig 7.3.2. Here the point A is the point in the main flow diagram (Fig 7.3.1) where the start of the next order word is being sorted : point B is where the starting quote has been found and point C is where the coordinates are read and is the point returned to after the completion of a shape if the DITTO flag is set.

The processing of a polygon is very similar to that of the rectangle and the flow diagram is shown in Fig 7.3.3. It will be seen that there are three main differences: there is an undefined number of coordinates in a polygon and the polygon must be checked for closure.

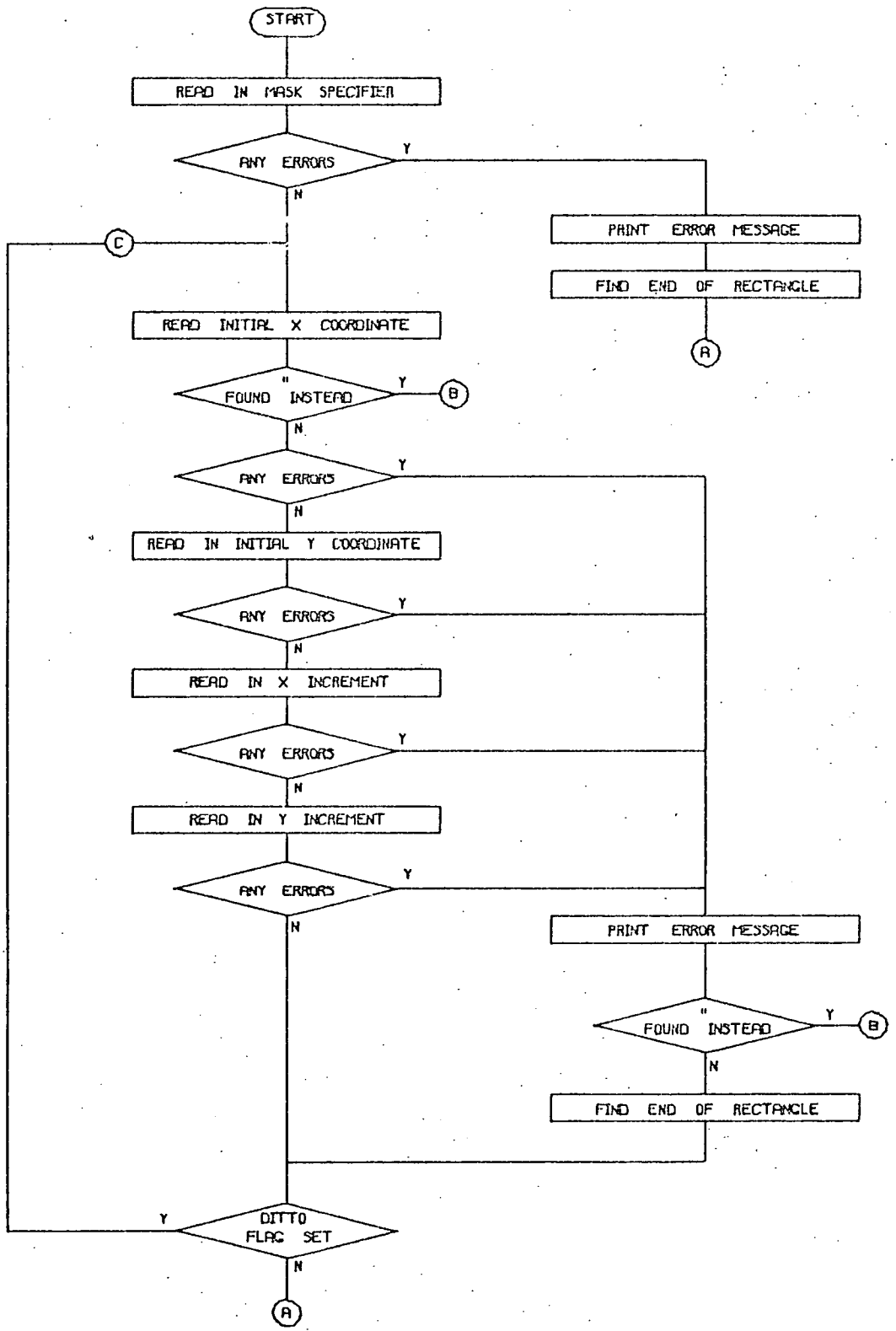


FIG 7 3 2 FLOW DIAGRAM FOR PROCESSING A RECTANGLE

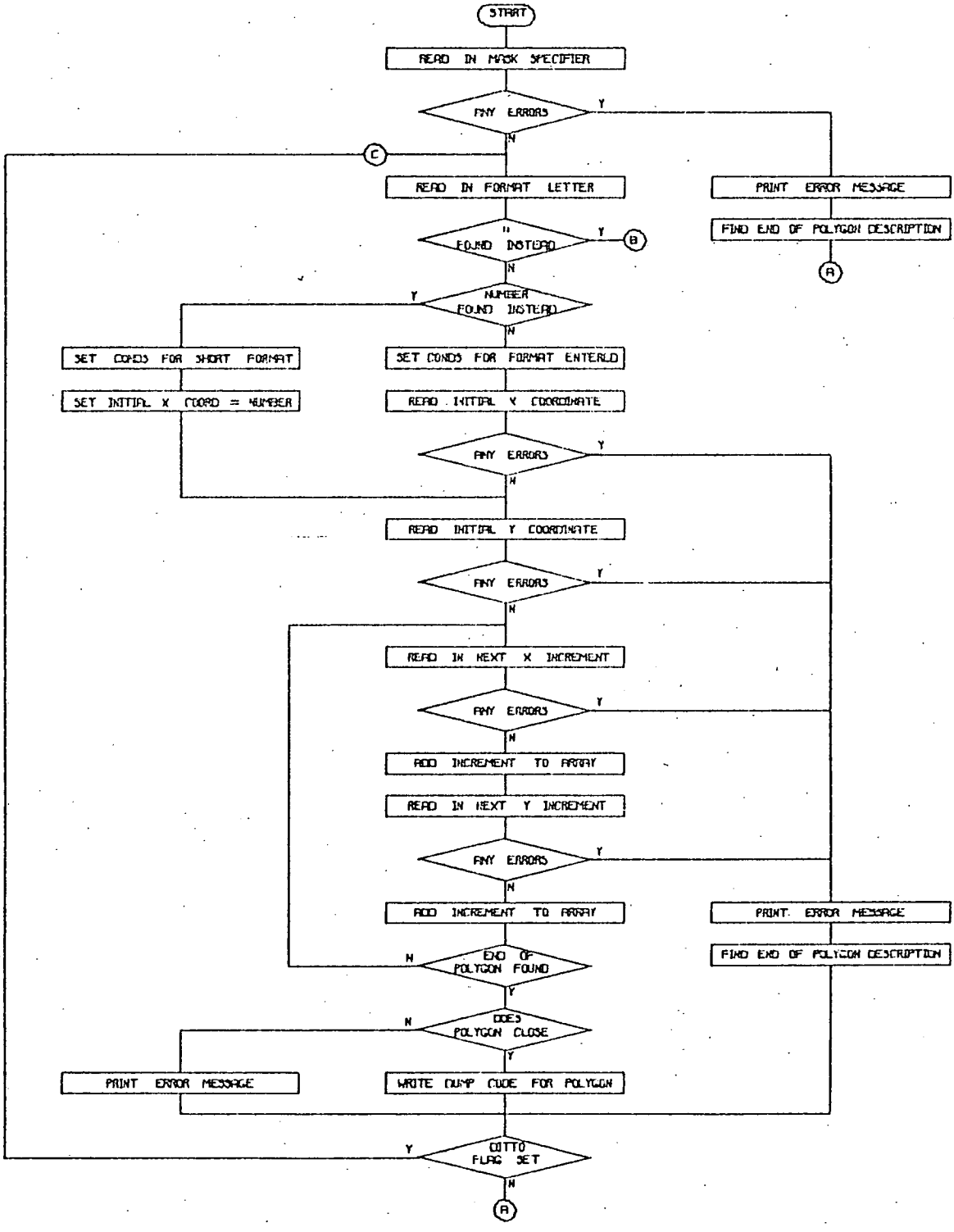


FIG 7 3 3 FLOW DIAGRAM FOR PROCESSING A POLYGON

The processing of the line is very similar to that of a polygon except that it has an optional width inserted before the format letter and there is no need for the closure check.

Most other words are simple to process involving the setting of flags or parameters and is easily understood from the listings.

The writing of the dump code is exactly the same as in GAEL1A where numbers are loaded into a buffer and written out when necessary.

7.4 GAEL3A Compiler into Ring Data Structure

This program takes the description of all or part of an integrated circuit layout coded in the dump code file and converts it into the ring data structure.

This is the first program of the GAELIC suite that handles the ring data structure. The ring data structure for a large integrated circuit is so big that it is held on disc with only copies of a few pages held in core. All interaction takes place with the data that is held in core. The data must be transferred from disc to core and vica-versa. This 'paging' as it is called is described in Chapter 5. The main data structure handling routines are built up using calls to two basic routines: one reads from and the other writes to the data structure. The main routines include one that allocates the required amount of

space for a bead and enters the contents of the head word into the data structure, one that sets up a null ring, i.e. enters the negative of the address at the address, another that adds a ring pointer of a bead onto a ring starting at a given address and also include more specialised routines like the one that searches the area or mask ring for a bead for a given area or mask number and if not found creates a new bead. These routines are described in detail in the GAELIC system manual [ref 7.3].

The flow diagram of the GAEL3A is shown in Fig. 7.4.1. It essentially consists of reading in data from a dump code file and adding it to the ring data structure file. The ring data structure file can either be a new file or an existing one and the selection of which structure illustrates an important point in the design of interactive programs.

One of the important features of any program operating in a time-sharing mode is its interaction with the user; the program must ask the right questions and must correctly interpret the users answers. The number of questions asked has to be carefully balanced against the information required. If there are a large number of questions asked that just require a short 'YES' or 'NO' answer then the user can find although his answer is unambiguous, that he spend a lot of time waiting while the program asks questions. The frustration can be heightened by knowing that his answer will lead to an obvious next question. This can be understood by considering the

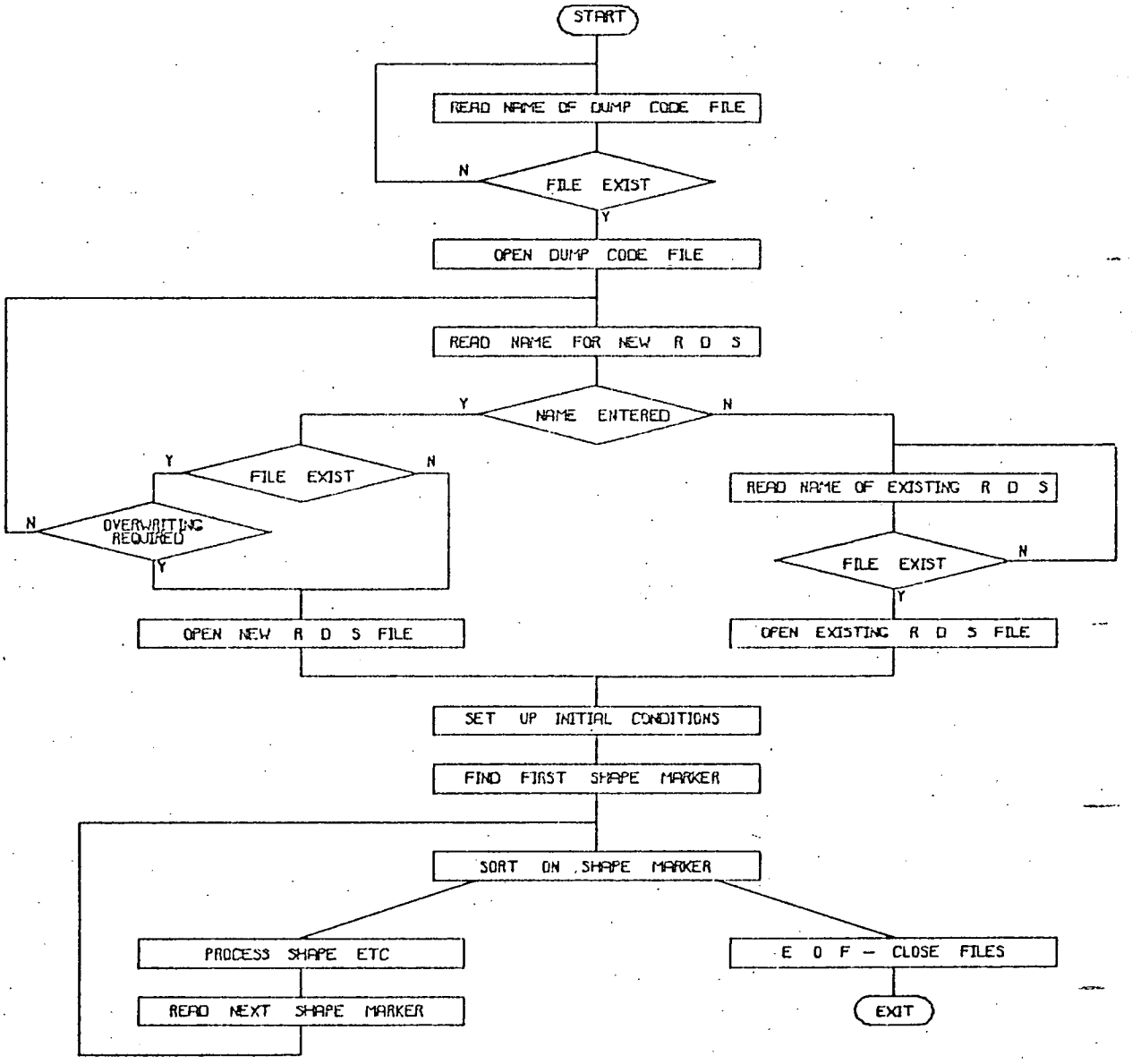


FIG 7 4 1 SIMPLIFIED FLOW DIAGRAM OF GREL3A

question:

DO YOU WANT TO USE A NEW OR EXISTING DATA STRUCTURE?

This: question is ambiguous does the user answer 'YES' or 'NO' or should he answer 'NEW' or 'OLD'? Obviously this type of question should not be asked. However if the question askcd:

DO YOU WANT TO CREATE A NEW DATA STRUCTURE?

Here the answer is obviously 'YES' or 'NO' but if the answer is 'YES' the equally obvious next instruction will be of the form:

ENTER NAME FOR NEW RING DATA STRUCTURE FILE

It is better therefore to obtain both pieces of information with the same question or instruction e.g.

ENTER NAME FOR NEW DATA STRUCTURE OR PRESS RETURN

This approach is used in GAEL3A and other programs in the GAELIC suite. The exact choice of question is governed by the usual requirement. In the case under discussion, the user usually requires a new data structure and so on the odd occasion when an existing structure is to be updated, he will press RETURN and will then be asked for the name of the existing file.

When the appropriate ring data structure file has been opened, certain initial conditions are set up including the main definition bead, as 'shapes' are read in from the dump code file they are converted to the appropriate beads and these are added to the main definition bead. When the marker integer for "FINISH" is found i.e. the end of the dump code file, the complete data structure is written back to disc.

The processing of a basic shape is shown in more detail in Fig. 7.4.2. After reading the numbers of the masks containing the shape, the actual coordinates are read into an array and their minimum and maximum values found. This 'bounding rectangle' is then used to calculate the number of the area associated with the shape. The area ring starting in the appropriate definition is then searched to find the area bead for that number. If it is not present, a new area bead is created and added in the appropriate position on the ring. The mask ring on the area bead is then searched in turn for the appropriate mask bead for each mask containing the shape in turn and a new bead is created if necessary. A shape bead is created for each mask in turn and is added to the shape ring on the appropriate mask bead.

When a group call is found in the input data, a similar process to that just described takes place and is shown in Fig. 7.4.3. The bounding rectangle of the group instance is calculated from the coordinates of the origin of the group call, the movement code and the bounding

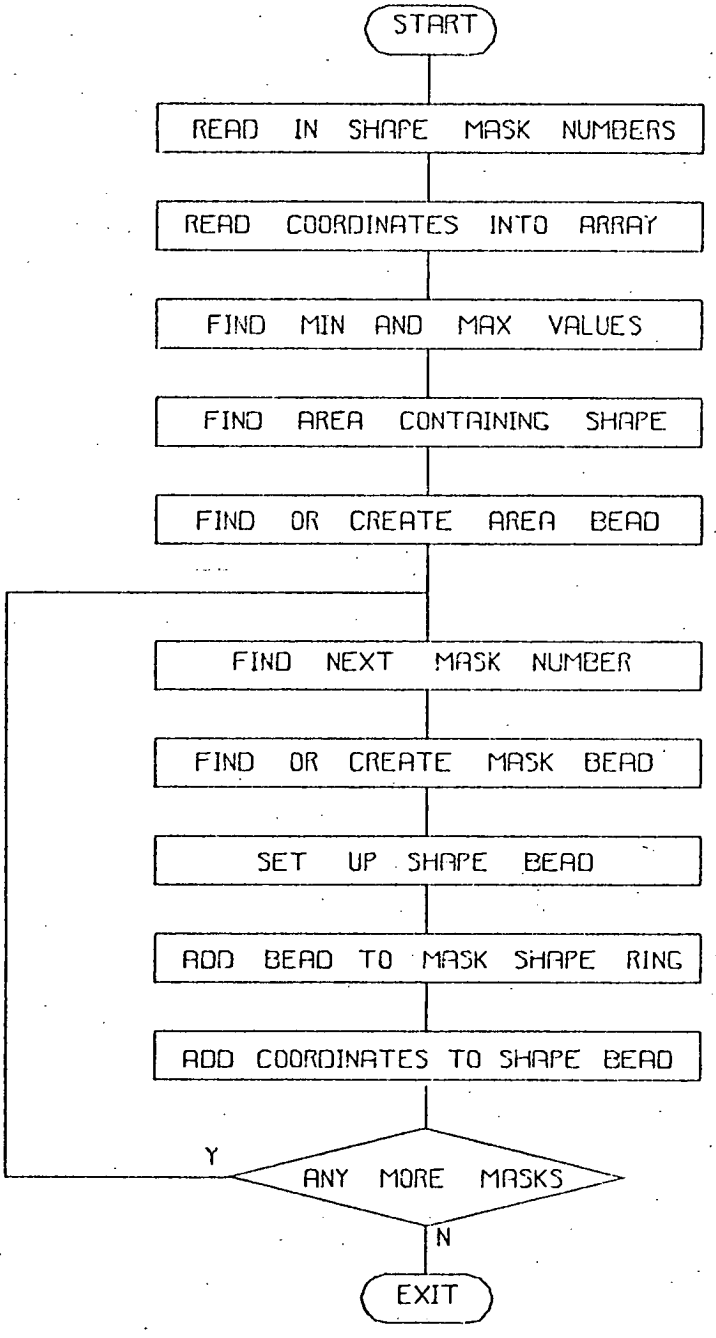


FIG 7 4 2 FLOW DIAGRAM OF RECTANGLE PROCESS

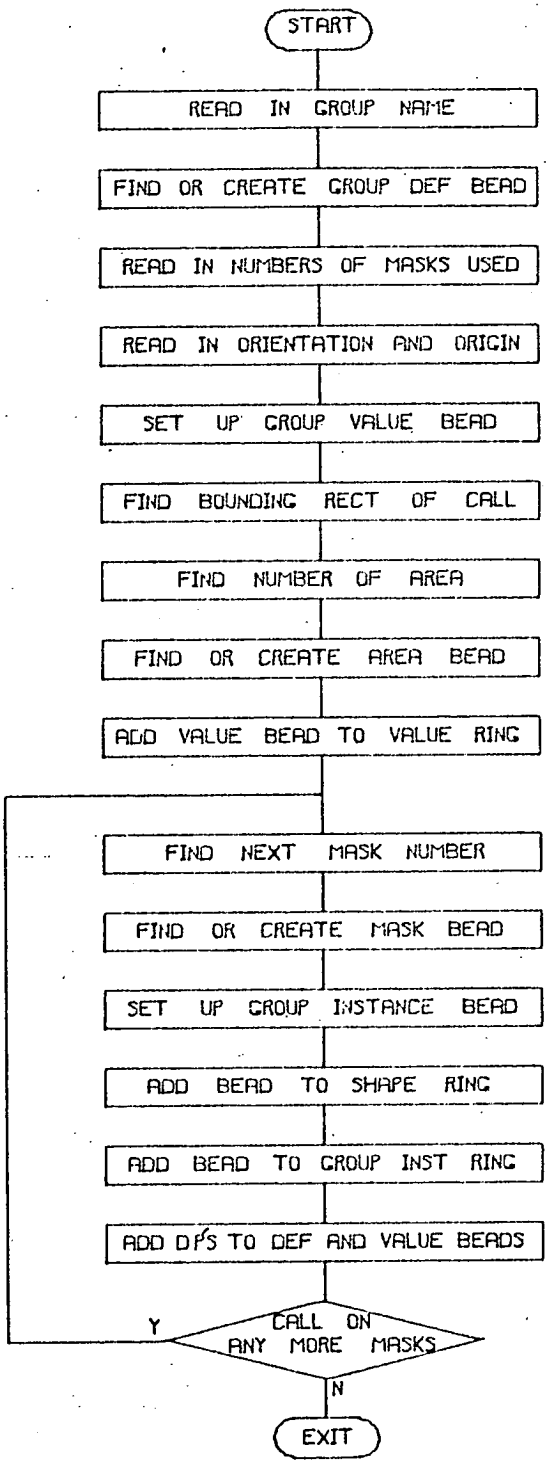


FIG 7 4 3 FLOW DIAGRAM FOR PROCESSING GROUP CALL

rectangle of the group definition. A problem that arises if the group definition has not been specified when the group call is processed, as the bounding rectangle of the definition is not defined. It is assumed to be zero and the area allocated to the instance is therefore governed by the coordinates of the group origin alone. Another problem that arises if the group definition has not been specified is concerned with the pointers. Chapter 5 showed how the group instance bead is not only connected to the shape ring but is also connected to the group instance ring and also has a direct pointer to the group definition bead. The head pointer of the instance ring is in the group definition bead, consequently without the group definition bead neither the group instance ring pointer nor the direct pointer can be set up. Therefore, if not already present, the definition bead must be created and added to the group definition ring when the call is processed. This is done automatically by a special routine. The routine is extremely complex as it not only has to check if the definition bead exists and create one if not, but also has to ensure that the definition is in the correct position on the group definition ring. The operation of the routine is described in detail in Appendix 2 but it will suffice here to explain the problem that it has to solve.

The bounding rectangles of shapes and group instances are used in many of the GAELIC programs to minimise the amount of data that has to be processed when plotting out

a window or when identifying the nearest point to the cross hair cursor. The bounding rectangle of a group instance cannot be correctly calculated unless the complete group definition is already in the data structure and its bounding rectangle correctly computed. Unless the group definitions are entered before the group calls, the bounding rectangles will not be correct. This situation is normally corrected by running GAEL8A immediately after GAEL3A. GAEL8A, however, will still not give the correct bounding rectangles unless the group definitions are in the correct order. If an instance of group A is called in the definition of group B then it is essential that the definition of A preceeds that of B on the group definition ring. The bounding rectangle of A will therefore have been correctly computed before the bounding rectangle of B is evaluated. As there is no restriction in the input language on the ordering of the data, the ordering of the definitions on the group definition ring must be done dynamically in GAEL3A by the routine. The same routine is used when the definition of a group appears in the dump code where again it finds or creates the group definition bead.

While the shapes in the definition are being entered from the dump code file, the appropriate area beads are added to the area ring starting in the group definition bead instead of to the ring starting in the main definition. The rest of the process of finding or creating the mask beads and adding the actual shape beads

is identical. At the end of the definition area beads and hence shapes are once more added to the main definition. A similar process is used for repeat definitions when it is arranged that subsequent area beads are added to the ring starting in the repeat definition bead.

7.5 GAEL4A Interactive Program

This program plots out all or part of an integrated circuit layout on a Tektronix storage tube terminal. The user can interact with the plot and modify existing shapes or add new ones. This is the main program of the GAELIC suite and is the most demanding from a programming point of view. It is absolutely essential that the program performs its various functions quickly to avoid user frustration and the design of the data structure has been done mainly with this requirement in mind.

One of the problems with using interactive graphics is that it is always compared with pencil and paper. A designer has been using a pencil and paper for years and so is completely familiar with the techniques and forgets that it took him the first five years of his life to become reasonably proficient with them. He sits in front of an interactive graphics terminal with its input device and becomes extremely dissolusioned if he cannot master the techniques required to use it within an hour. Consequently the ergonomics of any interactive graphics system have to be extremely good or the user will become

dissatisfied. One of the most critical features is response time i.e. how long it will take to draw a window or how long it will take to identify a point. Using GAELIC in a time-sharing environment makes this problem even more severe as there are time-sharing delays in addition to the other delays. The program can do nothing about the time-sharing delays and therefore concentrates on minimising the other delays. These other delays are due to two factors: the first is the time taken to write to and read from disc and the second is the amount of CPU time required to process the data. The data structure has been designed to minimise the number of disc transfers required to plot and modify a layout and the CPU time has been minimised by working as far as possible using integer arithmetic and doing preliminary sorts to avoid processing every shape.

The simplified flow diagram for GAEL4A is shown in Fig 7.5.1 where it can be seen that after setting up the initial conditions, the program allows the user to select one of a series of options, these are known as 'program command level options'. The selected option is then processed and can either automatically call in another option or return to the part of the program where the user selects the next option. The options allow the user to do such things as select which group definition is to be modified, select the masks to be plotted, select whether grid axes are produced or close the ring data structure file and exit from the program. The initial conditions

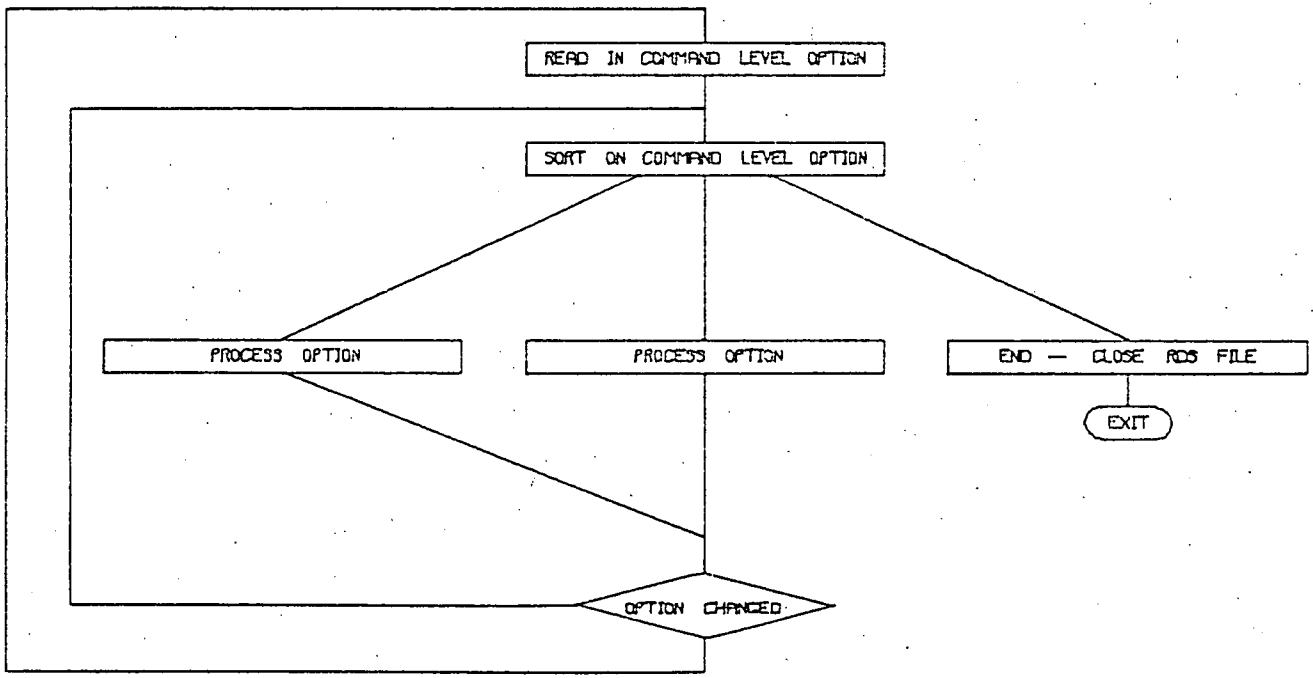


FIG 7 5 1 SIMPLIFIED FLOW DIAGRAM OF CAEL4A

that are set up include the selection of an existing ring data structure file or the creation of a new one. The flow diagram for this is shown in Fig. 7.5.2 where it can be seen that the philosophy of minimising the number of questions that have to be answered is used again. Instead of asking if an existing or new file is required the user is asked to name the existing file which is his normal requirement. When he wants to create a new file then he presses RETURN in answer to the first question, and then the program asks for the name of the new file.

The various command level options are shown below with a brief description of their functions.

AXES - Plot grid axes on the screen

DASH - Select line specification

DEPTH - Change depth of grouping to be plotted

DRAW - Draw additional shapes on screen

END - Close files and exit from the program

GROUP - Plot or modify a specific group definition

HELP - Clear the screen and print this list

LIST - List the names of all the group definitions

MODIFY - Modify shapes within window

ORIGIN - Plot triangles at group origins

PLOT - Set up mask list and plot window

REPLOT - Replot window for previous mask list

ROUND - Round cursor coordinates to nearest grid point

SAVE - Take backup copy of data structure

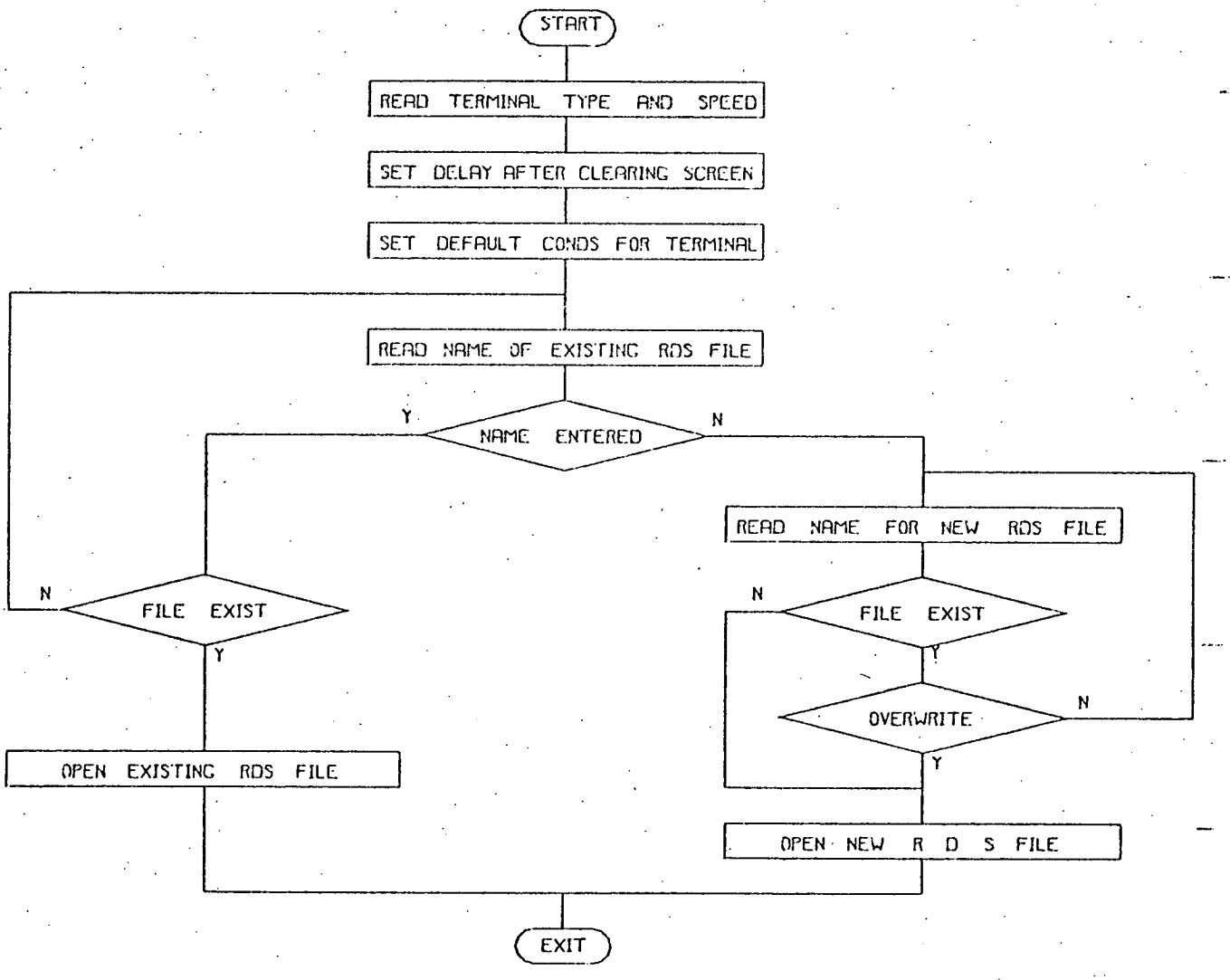


FIG 7 5 2 FLOW DIAGRAM OF INITIAL CONDITIONS

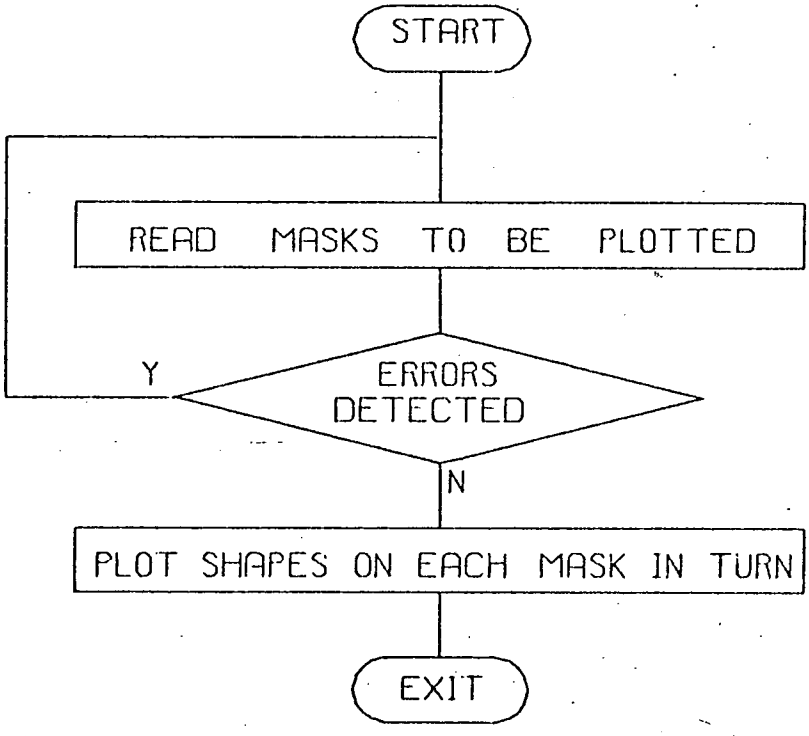


FIG 7 5 3 FLOW DIAGRAM OF OPTION PLOT

TEAR - Tear layout along defined line

TRACK - Modify track width

WINDOW - Change dimensions of window

Many of the options are just concerned with setting values or flags for example WINDOW allows the user to type in the bottom left hand and top right hand coordinates of the window to be plotted and ORIGIN allows the user to set a flag that governs whether the origins of the groups are marked with a triangle when the layout is plotted. Other options are more powerful and versatile. PLOT for example allows the user to specify a list of masks and then plots out all the shapes on each of the masks specified in turn while REPLOT plots out the shapes on the previous mask list. Thus the REPLOT option is a subset of the PLOT option. The flow diagram for these options are shown in Figs. 7.5.4 and 7.5.5. When running the REPLOT option, the area beads are examined in turn to see if any shapes in the area could lie within the window to be plotted. If so, the mask ring is searched for the appropriate mask number. As mask beads are arranged on the ring in numeric order then the beads need only be examined until the required mask number or higher number is found. If the required mask bead is found, then shapes on the shape ring are examined. The bounding rectangle of each shape is compared with the window to be plotted and if completely outside the window, the shape is ignored. If any of the bounding rectangle overlaps the window, then all the line segments that lie within the window are plotted. When all

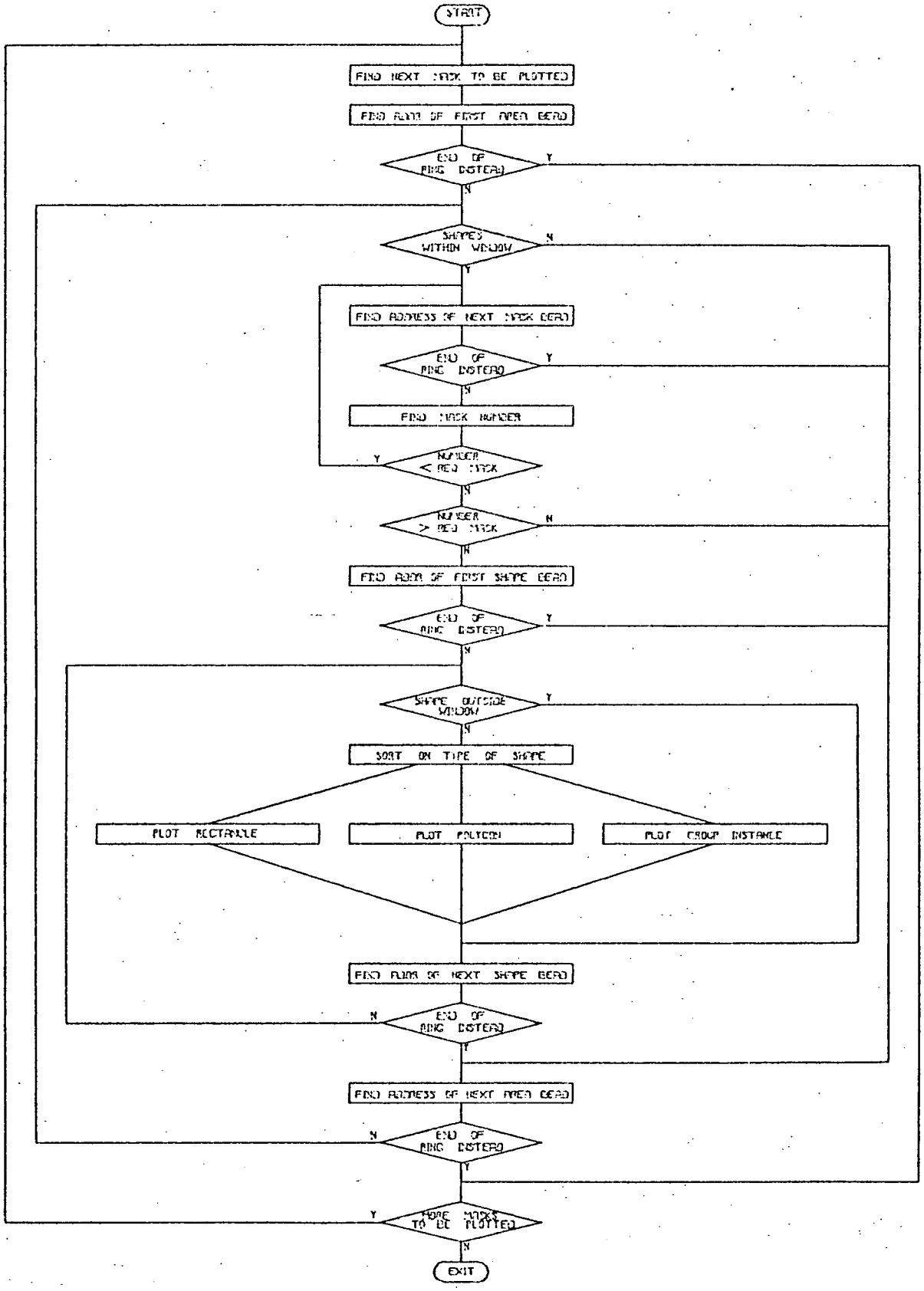


FIG 7 5 4 SIMPLIFIED FLOW DIAGRAM FOR REPLOT

the shapes on a mask have been plotted then the next area bead is examined and the process continued until the end of the area ring is reached. All area beads are examined but if the area cannot contain shapes that lie within the window, the mask and shape beads are ignored. By arranging that all the area beads are on the same page, while the mask and shape beads for each area are on other pages, then the area beads can be examined with a minimum number of disc transfers and it is only necessary to bring in new pages when the area can possibly contain shapes within the window. Thus the number of disc reads required is considerably reduced as is shown by the results in Chapter 8.

MODIFY, DRAW and TRACK are program command level options that allow the user to interact with the layout. By suitable positioning of the cross hair cursor and by pressing suitable character keys the user can identify points on shapes, define lines etc. The choice of character key to press gives the user another level of options which are known as 'cursor command level options'.

These different levels of option therefore form a tree structure. The tree starts with the main option which is to select the ring data structure to be modified, at the second level are the program command level options, the third level contains the main cursor command level options and it will be seen later that there is a fourth level consisting of secondary cursor command level options.

The simplified flow diagram of MODIFY is shown in Fig. 7.5.5 where it can be seen that by choosing the character to press when the cross hair cursor is in the screen the user can select the cursor command level option required. The option is then processed and the progream returns to the cursor command level i.e. the cross hair cursor is set up.

Certain of the cursor command level options initiate processes involving other cursor command level options: these are known as main cursor command level and secondary cursor command level options respectively. For example in MODIFY the letter I is used at main cursor command level option to identify the nearest point in the data structure to the cross hair cursor, the secondary cursor command options A, H, O, Y and [then dictate whether the shape is moved, modified or deleted. The flow diagram for this is shown in Fig. 7.5.6 where it has been seen that as well as modifying the basic shapes it can also modify repeat parameters.

TEAR is the option that allows the user to define a line through his layout or then move all the shapes to the one side of the line by a distance. It was written by J. Phillips of the CAD Project and is called as a subroutine in the program.

DRAW was at one time a completely separate option to MODIFY in the first version of the program running on the Systemshare time-sharing service when it formed a

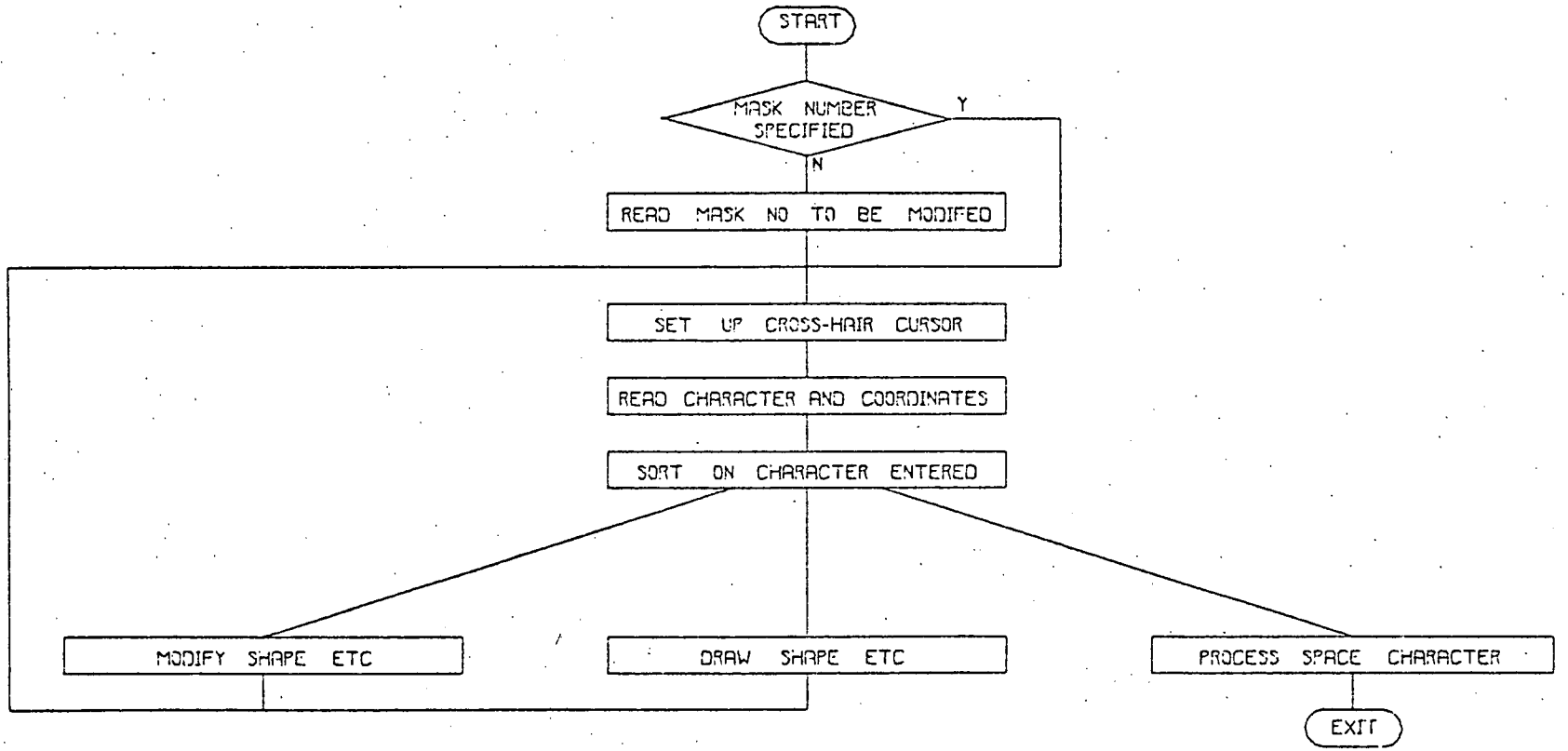


FIG 7 5 5 SIMPLIFIED FLOW DIAGRAM OF MODMSK

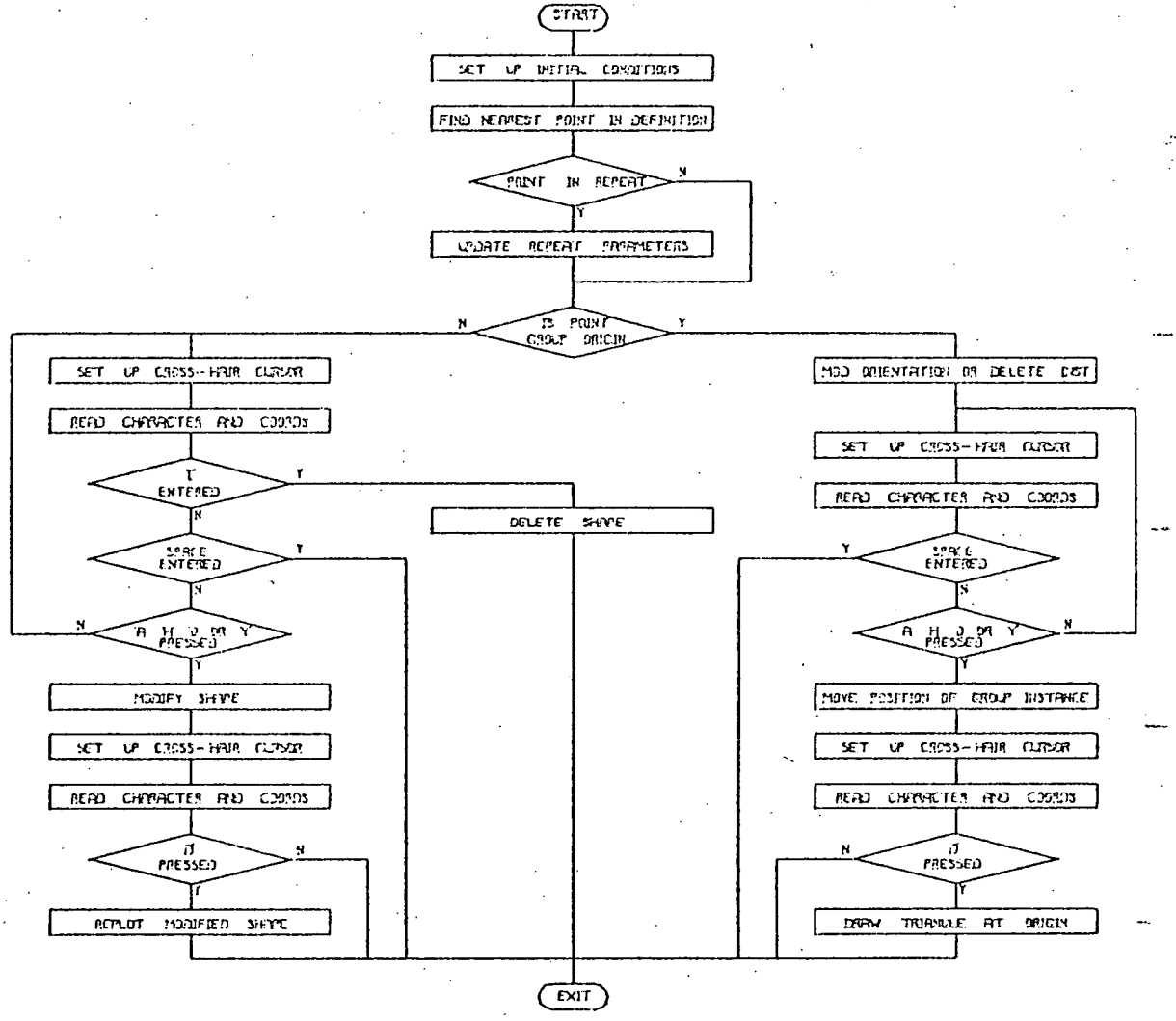


FIG 7 5 6 SIMPLIFIED FLOW DIAGRAM OF IDENTIFY

convenient overlay segment. However since that time it has now been amalgamated with MODIFY and so requests to DRAW are treated in exactly the same way as requests to MODIFY.

The data structure handling techniques are the same as those used in GAEL3A apart from the paging routines described earlier.

Another of the interesting parts of GAEL4A is concerned with processing group and repeat instances where there are two problems to be solved. The first is due to the fact that the group and repeat instances can be nested and is in the determination of which areas to process in the group or repeat definition. The nesting of groups means that the absolute movement code applied to the shapes in a definition must be computed from the individual movement codes of the group instances. The previous absolute movement codes must also be kept to avoid recalculating them then when a group instance has been processed. It is also essential to keep the addresses of the previous area and shape ring pointers that were being processed when the group call was detected in order to process the remainder of the shapes.

The second problem is concerned with the area beads. The main savings in computer time come from the fact that only the shapes in areas that can overlap the window are processed, masks and shapes in all other areas are ignored. The group definitions are built up in exactly

the same way as the main definition i.e. with area beads, mask beads and shape beads, the areas being allocated in exactly the same way. However, if a group instance is rotated about the x axis, then the areas that are required to be plotted in the group definition are different from those that are required to be plotted in the main definition but are related to them by the movement code. The transformation involves mapping the window onto the group definition which is a different operation to that performed on the shape coordinates in a definition when they are mapped onto the main layout.

The flow diagram for handling the group call is shown in Fig. 7.5.6 where the coordinates of the origin and the movement code are read from the value bead. The coordinates of the origin are then transposed to take account of the case where the group instance is called from another group definition. They are transposed according to the previous movement code. If the transposed origin lies within the window and ORIGINS are requested, a triangle is plotted at the origin or if modifying, the coordinates are compared with cursor coordinates. When normal modification takes place the program does not search the contents of the group definitions and so can exit at this point. However, if plotting or using the special routines that find the nearest point in the complete layout, then conditions are set up to process the contents of the actual definition. The depth of grouping is incremented and the new

displacements for the coordinates are calculated from the old displacement and the transposed coordinates of the origin. The new absolute movement code is also evaluated from the previous absolute code and the movement code of the instance. This in turn enables the area coordinates for the part of the definition within the window to be found. The bounding rectangle of the instance can then be calculated from the bounding rectangle of the definition, the new absolute movement code and the displacements. If it is outside the required window the shapes are ignored. If the bounding rectangle is inside the window then the necessary initial conditions are set up which keep details of the previous definition and set up to examine shapes in the present definition.

At the end of the area ring, ie. when all the shapes in the definition have been processed then the reverse process takes place where conditions are reset for the previous definition and the depth of grouping is decremented.

A similar process has to be followed for repeat instances with the exception of the movement code which does not have to be updated.

7.6 GAEL5A/B Plotting programs

The GAEL5A program takes a GAELIC ring data structure and plots all or part of it out on a Calcomp 563 incremental plotter. GAEL5B is a similar program that plots the layout on a Calcomp 563 plotter via a Calcomp DP212 controller.

The plotting program is very similar to that used in GAEL4A to plot on the Tektronix screen i.e. has a flow diagram similar to Fig. 7.5.5 and handles groups in a similar way to Fig. 7.5.6. The main difference is that as the program is usually used to plot the complete layout: in this case all areas are processed and there is no point in checking to see if each individual area is required. The checks for the individual shape however are left in just in case only a part of the layout is required. In which case the processing is not quite as efficient as it is in GAEL4A. The plotting however is very much limited by the speed of the plotter rather than the time taken for disc reads so the inefficiency has negligible effect as far as the user is concerned. The program uses standard Calcomp driver routines which are part of the Decsystem 10 basic software library. These basic routines take the real number coordinates created by the program and convert these into the necessary increments for the plotter. The actual plotter requires 3 bits of a character for each increment. The increment can be in one of 8 different directions. The increment for the 563 plotter is either 5 or 10 thou. and can handle up to 300 increments per

second i.e. 1.5 or 3 inches/per second.

The driver routines written for GAEL5B to drive the 563 plotter via the DP212 controller send characters that specify the number of increments as well as the direction. This allows the plotter to work at near full speed despite using a 110 Baud line as the DP212 converts each character into the required number of characters that specify one increment in a given direction. Angled lines other than 45 degrees are slow because they do not take many consecutive steps in any one direction and so if the pen is up, it is moved as far as possible at 45 degrees and then horizontally or vertically to the final destination as shown in Fig. 7.6.1. On the other hand if the pen is down then the best possible straight line is drawn as shown in Fig. 7.6.2.

The algorithm used for the line with the pen down is a modification of an algorithm written by Dr. J.V. Oldfield for driving the 563 directly. It transposes the line so that it has a major axis AC as shown and a minor axes AD as shown in Fig. 7.6.3.

A calculation is made at each major axis increment to see where straight line AB cuts the vertical line, if it is less than half a minor axis increment then the pen is moved one increment along the major axis, if it is greater than half an increment then the pen is moved at 45 degrees to the major axis. The process continues at each major axis increment until the end of the line is reached. The

Fig 7. 6.1 Pen movement when raised

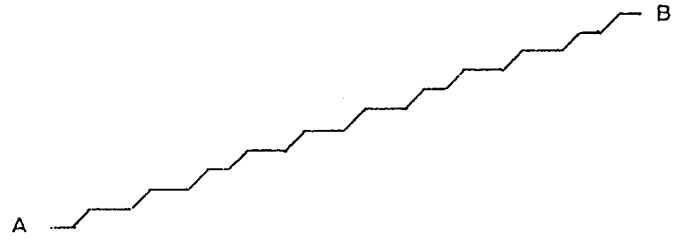


Fig 7.6.2 Pen movement when lowered

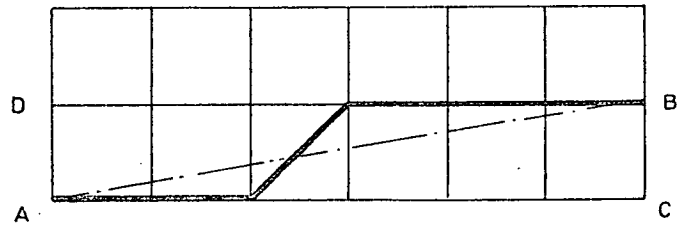


Fig. 7. 6.3 Simple angled line with pen lowered

characters are only sent to the plotter when the direction of the increment changes or the end of the line is reached. For example, for the line shown in Fig. 7.6.3, one character is sent to the plotter for the two increments along the major axis, one character for the one character at 45 degrees and then one increment for the 3 increments along the major axis.

7.7 GAEL6A Joins lines together to form closed polygons

The line with zero thickness is an extremely useful shape for defining metallisation tracks in groups as shown in Chapter 3. However, when photo-plotters are used to produce the masks, it is impossible to tell which side of a line has to be exposed. The lines must, therefore, be joined together to form polygons. This was the main reason for writing GAEL6A although using it gives two other advantages that are not quite so obvious.

Firstly when using a knife on cut and peel material for a series of short lines, the coordinatograph will no sooner have accelerated to full speed than it will have to decelerate again to enable it to stop at the end of the line. However if all the short line segments are joined together to give a large polygon then the coordinatograph can cut for a comparatively long period at full speed. The movement with the pen up is also reduced giving a substantial saving in time on the tape controlled coordinatograph.

Secondly when using the lines in the group definitions, it is possible to forget to add the lines at the end of the group instances to connect, for example, a 24 bit shift register stage to the rest of the circuitry. Errors of this nature are not easy to detect visually but are automatically detected as part of the process to join up lines to form polygons. The problem of detecting the missing lines may appear to be a good reason for using closed shapes instead of lines. However if the closed shape that connected the shift register to the outside world were omitted, it is even more difficult to detect visually and cannot be detected by present computer programs. (it will be possible to detect such errors with 'Mask Function Checking' when this program is finished).

The flow diagram for the process is shown in Fig. 7.7.1 where it can be seen that after setting the initial conditions, copies are taken of all the lines in the layout. This is not as easy as it first appears, as a copy of a line in a group definition has to be taken for each group instance. Each copy must naturally have the appropriate coordinate transformations. The program therefore behaves as if it were plotting i.e. follows the flow diagrams shown in Fig. 7.5.5 and Fig 7.5.6 but instead of plotting each shape, the rectangles and polygons are ignored and the lines are copied into a temporary ring data structure. This temporary data structure has area beads but no mask beads as the program only operates on one mask at a time.

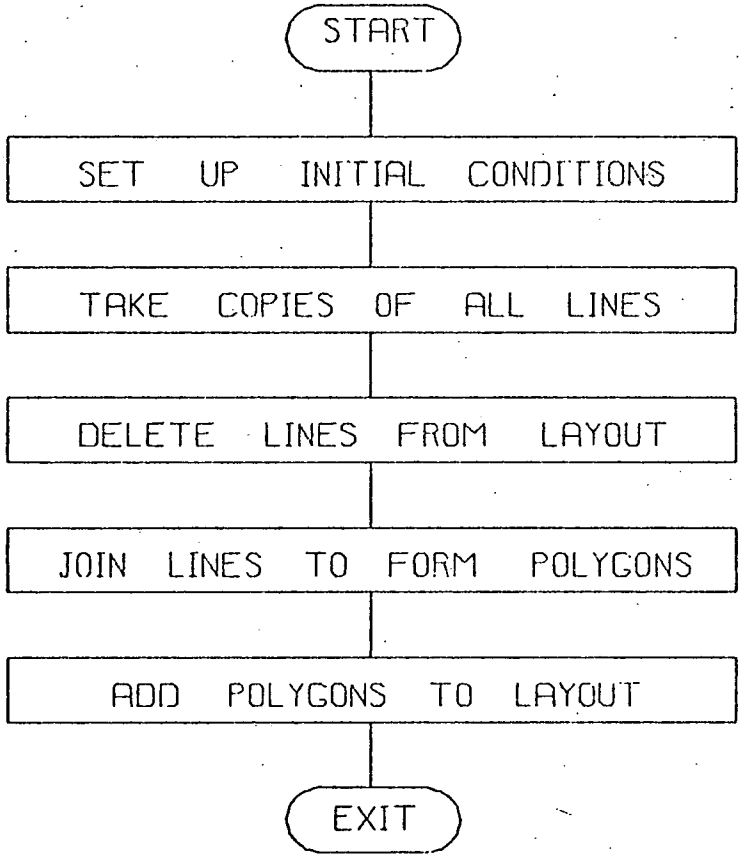


FIG 7 7 1 SIMPLIFIED FLOW DIAGRAM OF GAEL6A

When copies of all the lines have been added to this temporary data structure the original structure is then processed to remove all the lines. This time, however, it is only necessary to detect the line in a group definition once. Each shape ring for the required mask is examined in turn and all the line beads deleted: all other beads including group instances are ignored. The flow diagram for this process is similar to Fig. 7.8.1 which describes the operation of GAEL7A. GAEL7A is the program that produces the manual input language from the ring data structure.

The lines on the temporary ring data structure are then joined together to form polygons which are added to the original layout. The data for the first line in the first area is entered into an array and the line deleted from the temporary data structure: all subsequent lines in that area and all lines in other possible areas are then examined to find the line with either the same starting or finishing coordinates. The data for this second line is then added to the array and its bead deleted from the data structure. The process continues until the composite line in the array forms a closed polygon which is then added to the original data structure in the appropriate area. The program then returns to what is now the first line in the data structure and repeats the process until all the lines have been joined up to form polygons and all the polygons have been added to the original data structure.

If the lines do not join up two possible courses of action are taken. If another line has the same end coordinates within a given limit then the coordinates of the second line are changed and a message printed to that effect. If no lines have the same end coordinates then the line in the array is written back to the original layout and an error message printed. The user will then run GAEL4A to correct the errors.

7.8 GAEL7A Ring data structure to manual input language.

This program takes the definition of a layout in the ring data structure and writes it to a file as the manual input language. It is used to obtain a copy of the layout description that can be read by a user and which is independent of the computer running the GAELIC programs. It is also useful for taking the description of part of a layout and redefining it as a group definition.

The simplified flow diagram of the program is shown in Fig. 7.8.1 where the general aim is to first write out the descriptions of the group definitions followed by the description of the main layout definition. Unfortunately both group and main definitions may contain repeat definitions and to make things even worse the repeat definitions themselves may contain further repeat definitions.

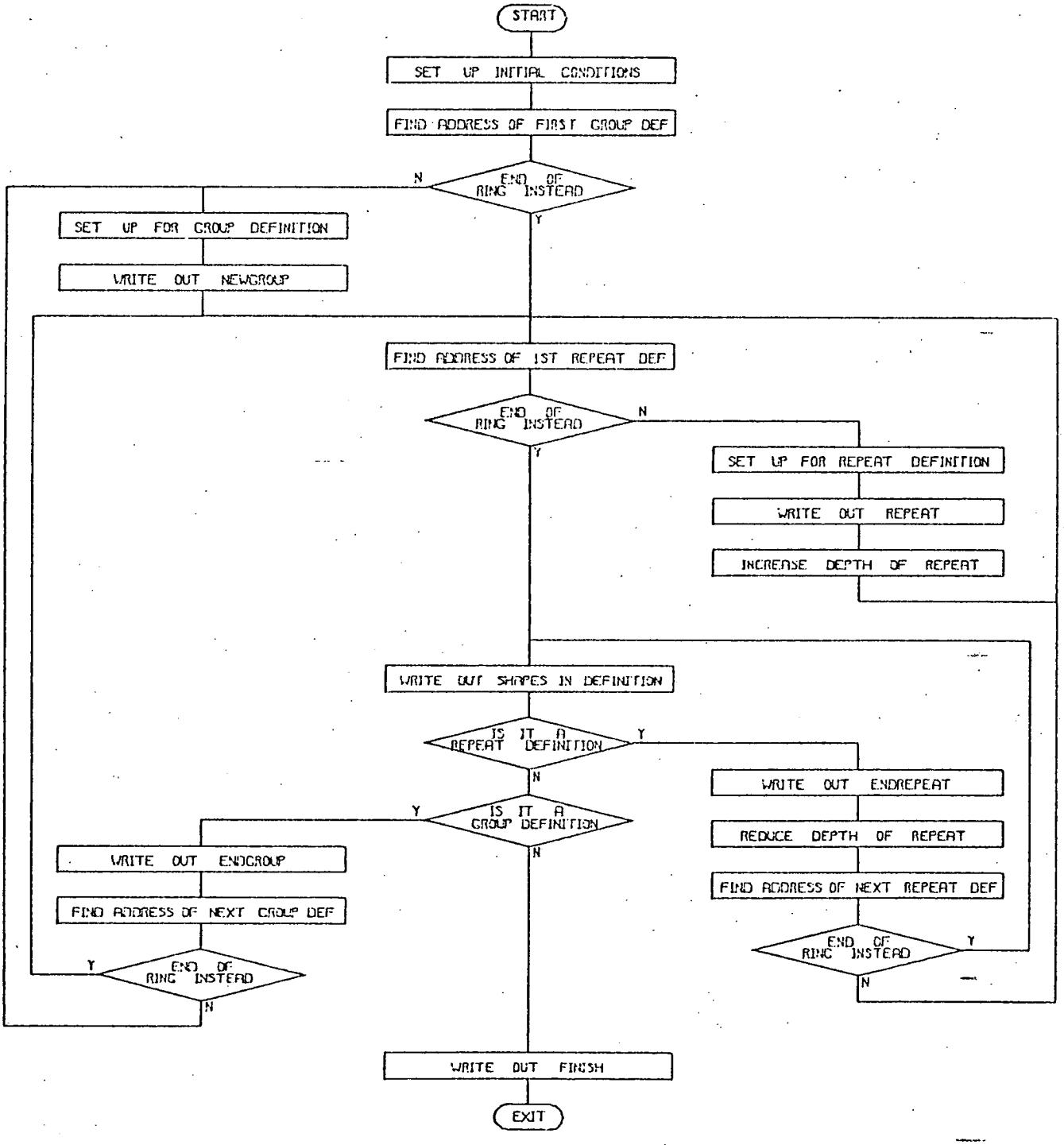


FIG 7 8 : SIMPLIFIED FLOW DIAGRAM OF CAELZA

After setting up the initial conditions i.e. reading in the names and opening the necessary files, the program finds the address of the first group definition. If the address is that of the ring head pointer then there are no group definitions present and the program starts to process the main definition. If a definition is present then the group name is found and is written out to the output file. The group definition bead is then examined to find if any repeat definitions are present. If one is present then the repeat parameters are read from the value bead and the repeat specifier is written to the output file. The repeat definition is then checked to see if it contains any repeat definitions and if so the depth of repeat is incremented and the process repeated. When the present definition does not contain any further repeat definitions then the shapes in the definition are written out. A sort is then made on the type of definition processed. If it was a repeat definition then the end repeat order word is written out, the depth of repeat is reduced and the address of the next repeat definition found. The process is repeated until all the repeat definitions have been processed, the program then writes out the shapes in the group definition and after writing out the endgroup order word finds the address of the next group definition. This processing of the group definitions continues until all the group definitions have been written out. The main definition is then processed but again must first go through the process of writing out all the nested repeat definitions before actually writing

out the shapes in the main definition. The finish order word is then written out and the files closed.

The actual writing out of the manual input language requires an identical technique to that described in section 7.2 for the digitiser program. The reading of the ring data structure requires the data structure handling routines used in most of the other programs.

7.9 GAEL8A Data structure reorganiser.

This program reorganises the data structure on disc so that it is in an optimum order for interaction. The program also removes all unused area and mask beads and recalculates all the bounding rectangles.

The data is sorted so that the definition beads and area beads are on the same page or on consecutive pages so that the area beads can be checked to see if they can contain shapes within the window using the minimum number page changes. However once an area bead has been found to be capable of containing shapes within the window then all the shapes must be processed, hence all the mask and shape beads for a particular area bead must be on either the same page or on consecutive pages so that again the number of page transfers required to process the shapes is minimised. The simplified flow diagram to perform the reordering of the data is shown in Fig. 7.9.1. The initial conditions consist of reading in the names and

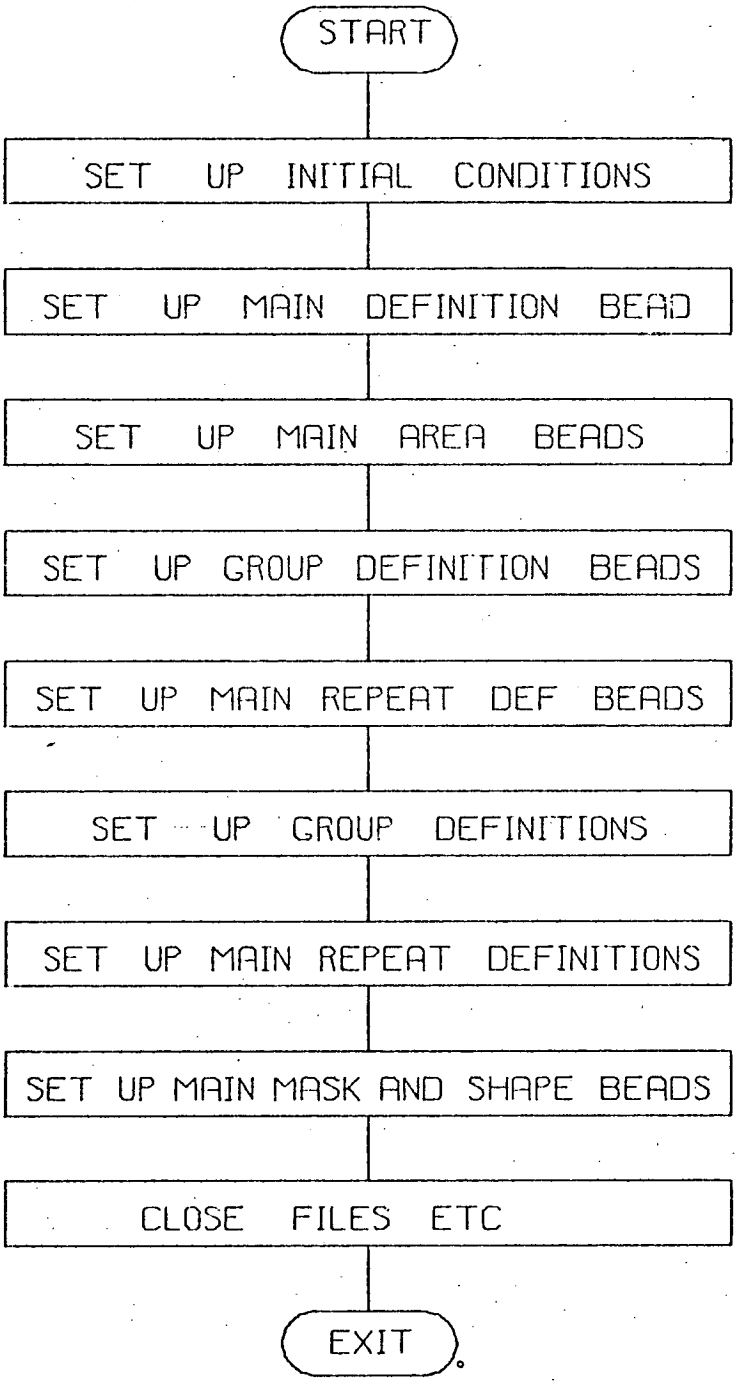


FIG 7 9 1 SIMPLIFIED FLOW DIAGRAM OF GAEL8A

opening the necessary files. The main layout head bead is then set up in the new data structure which is followed by beads for every area that is used in the old layout. A group definition bead is then set up in the new data structure for each group definition in the original structure. These are followed by repeat definition beads for all the main repeat definitions present in the original structure. The actual group definitions i.e. any repeat definitions, the area beads, mask beads and shape beads, are then copied onto the new structure. The repeat definitions may well be nested and if so all the necessary definition contents i.e. area, mask, value and shape beads, must be set up. The same process applies to the main repeat definitions which again can contain nested repeats. Finally the main definition mask and shape beads are set up.

The techniques used to handle the data structures are virtually identical to those used in other programs except that provision is made for the two data structures that must be handled. The task is simplified as there is a requirement to read from the original data structure but no requirement to write to it. The new data structure however has to be both written to and read from.

7.10 GAEL9A etc. Post Processors.

These programs convert the ring data structure into the drive tapes for a range of mask making machines ranging from the Coradi tape controlled coordinatograph to the Gyrex Pattern Generator.

All these programs work the same way as GAEL5A i.e. the flow diagrams are those shown in Fig. 7.5.5 and Fig. 7.5.6.. The actual output varies from program to program but usually uses the buffered output techniques described in section 7.2.

CHAPTER 8: Performance

This chapter is devoted to the subjective assessments and objective measurements that were carried out on the programs in order to evaluate their performance at various stages during their development. The chapter starts with a subjective assessment of the PAELLA suite of programs running on the Systemshare time-sharing service and is followed by a comparison between the performance of the sequential block data structure used in PAELLA with the initial ring data structure used in the first version of GAELIC. The results of the various improvements made to the ring data structure culminating in the final data structure using area beads are then described and the chapter closes with a subjective assessment of the interactive facilities available in the GAELIC suite.

8.1 PAELLA Performance

The PAELLA programs were written, mainly by the author, for General Instruments Microelectronics Ltd (GIM) for use on the Systemshare time-sharing service. Most of the programs belong to GIM and are not generally available. However, the Wolfson Unit retained rights to the two input programs and these are used in the GAELIC suite. Most of the integrated circuit masks produced by GIM were of a proprietary nature and no objective measurements were made. It is nevertheless worth discussing their subjective assessment of the PAELLA programs.

The programs are designed to take the output from a Metrograph digitiser or a manual input language description of the layout, and convert it initially into check drawings on an on-line incremental plotter and finally into drive tapes for tape controlled coordinatographs to produce the mask masters. It was realised that without any interactive correction facilities, the programs would have limited facilities but it was necessary to produce a working system in the minimum time. GIM found the correction of errors by either modifying the digitiser output or the manual input language was extremely tedious and most of the time was spent looking for and correcting errors rather than digitising the layout. The method of providing the input data was found to be successful provided that either an accurately gridded mylar was used for the drawing or only small parts of a layout were drawn on normal gridded paper. The major problem that appeared if these requirements were not met, was that many polygons had one or more sides at a small angle rather than all the sides paraxial. Finding which point is in error and correcting it by examining the input language is far too slow and cumbersome.

8.2 PAELLA and GAELIC Comparison

As most of the designs processed by GIM were of a proprietary nature, the completed designs were not available and so no timing or costing data was possible. However, permission was obtained from GIM to use the PAELLA programs to do some comparative tests on three circuits that were available for publication. The circuits are shown in figs 8.1, 8.2 and 8,3 and the results of the measurements are shown in table 8.1.

		Layout 1	Layout 2	Layout 3
PAELLA Sequential Block D. S.	CPU Time Secs	20.4	29.6	724.8
	Connect Mins	3	3	24
GAELIC Original Ring D. S.	CPU Time Secs	17.6	14.2	584.0
	Connect Mins	3	3	9

Comparison of Sequential Block and Ring Data Structures

Table 8.1

The object of the test was to compare the performance of the PAELLA programs and the original version of GAELIC, both of which ran on the Systemshare time-sharing service. The digitiser input and manual input language programs both of which can produce dump code files are common to

both systems as are the post-processors converting the coordinate file into drive tapes for tape controlled coordinatographs. However, the method of producing the coordinate file from the dump code file is different in each system using different data structures as discussed in Chapter 4. The tests are therefore a comparison of the sequential Block Data Structure used in PAELLA with the Ring Data Structure used in GAELIC. Errors are detected in PAELLA either by the syntax checkers or by plotting all or part of the drawing on an incremental plotter. They are corrected, however, by modifying the input language file. GAELIC, in addition can detect errors by plotting part of the layout on the Tektronix and the errors can be corrected interactively. This means that the only valid comparison between the two systems is the time to convert the Dump Code file into the Coordinate file.

The first layout shown in fig 8.1 is a small MOS test circuit designed by the Wolfson Unit to evaluate an MOS process. It does not contain any group definitions or repeats and because it is small is entirely core resident in GAELIC. The second circuit is a small test example used during the development of the CAMP system [ref] and contains both group and repeat facilities But again is core resident in GAELIC. The third circuit is part ofthe correlator layout designed by the Wolfson Unit and uses both group and repeat facilities and is large enough to use the paging facilities in GAELIC.

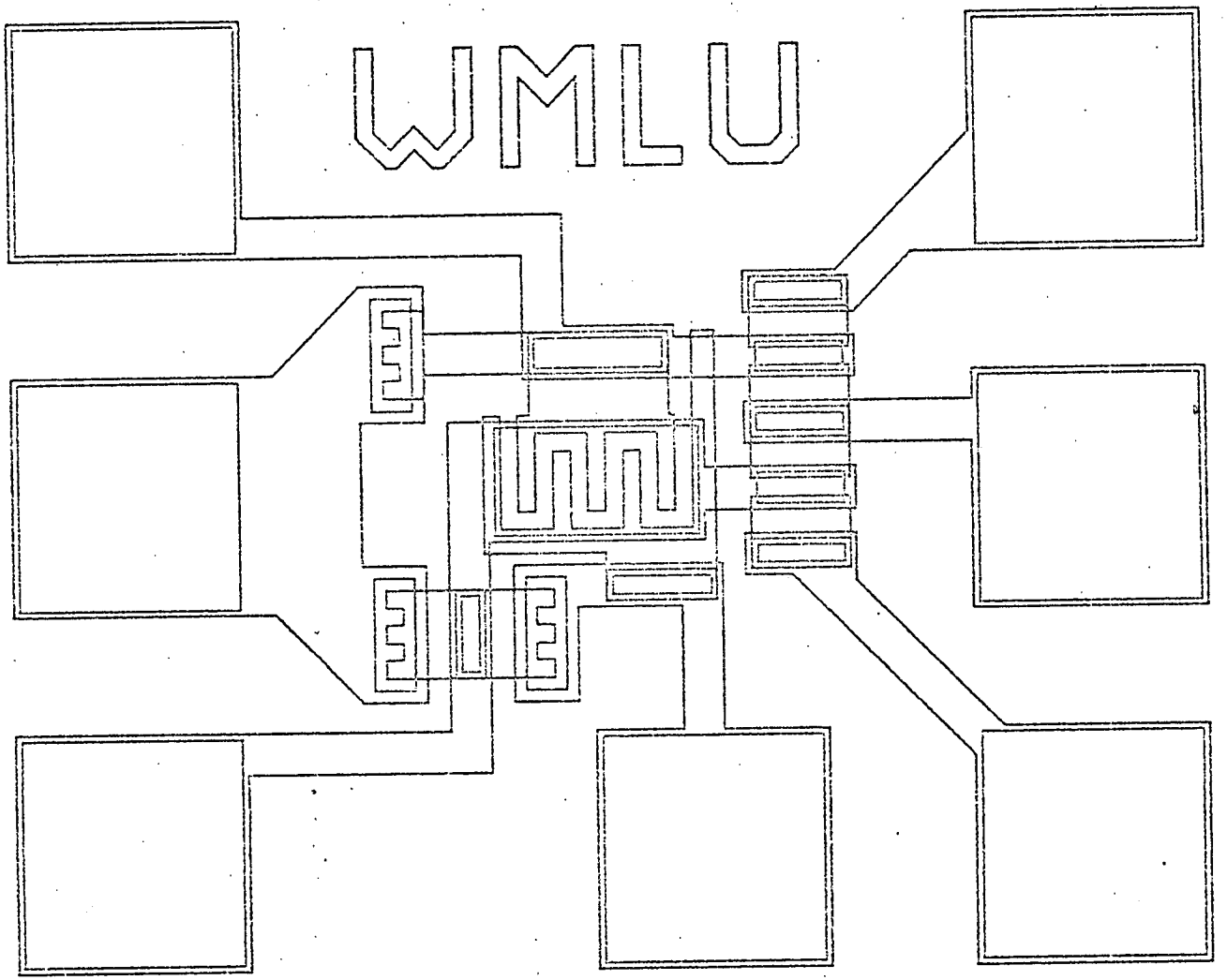


Fig 8, 1 Layout of small circuit with no grouped and repeated shapes

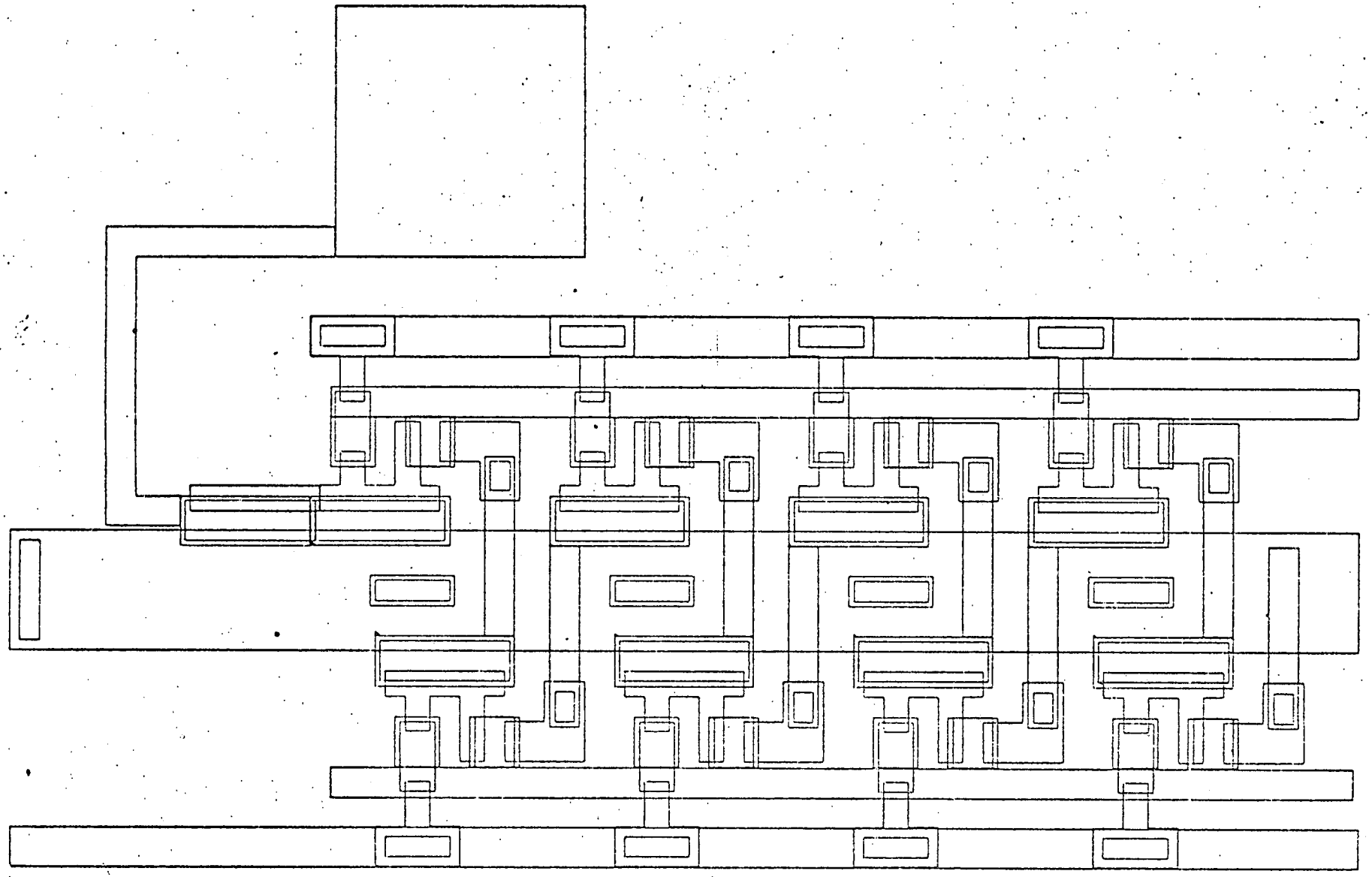


Fig 8.2 Layout of small circuit with grouped and repeated shapes

The results show that for a small circuit without group definitions etc there is a slight but significant saving in CPU time when using the ring data structure, though there is no variation in the connect time used. This slight variation is probably due to the fact that using the ring data structure, the data is always in core while using the sequental block data structure requires the files to be rewound for each mask. The small layout using group and repeat facilities shows a considerable reduction in the CPU requirements to process the data using the ring data structure because of the way the group and repeat facilities are handled. The data for the ring data structure is again held entirely in core.

The CPU time to process the large section of layout [fig 8.3] shows a 20% saving using the ring data structure but there is an even more dramatic saving in connect time. Normally the connect time required to run a program on a timesharing service varies with the number of users on the system and so in general is not a meaningful measurement. However, these particular measurements were made late in the evening when the system was very lightly loaded and so there was a significant variation in the connect time. This variation is almost certainly greater than the variation in CPU even after allowing for the effects of loading and an explanation must be found for it. A significant point to note is that 584 seconds of CPU time will require 9 mins 44 seconds of connect time. The connect time is only measured to the nearest minute so the

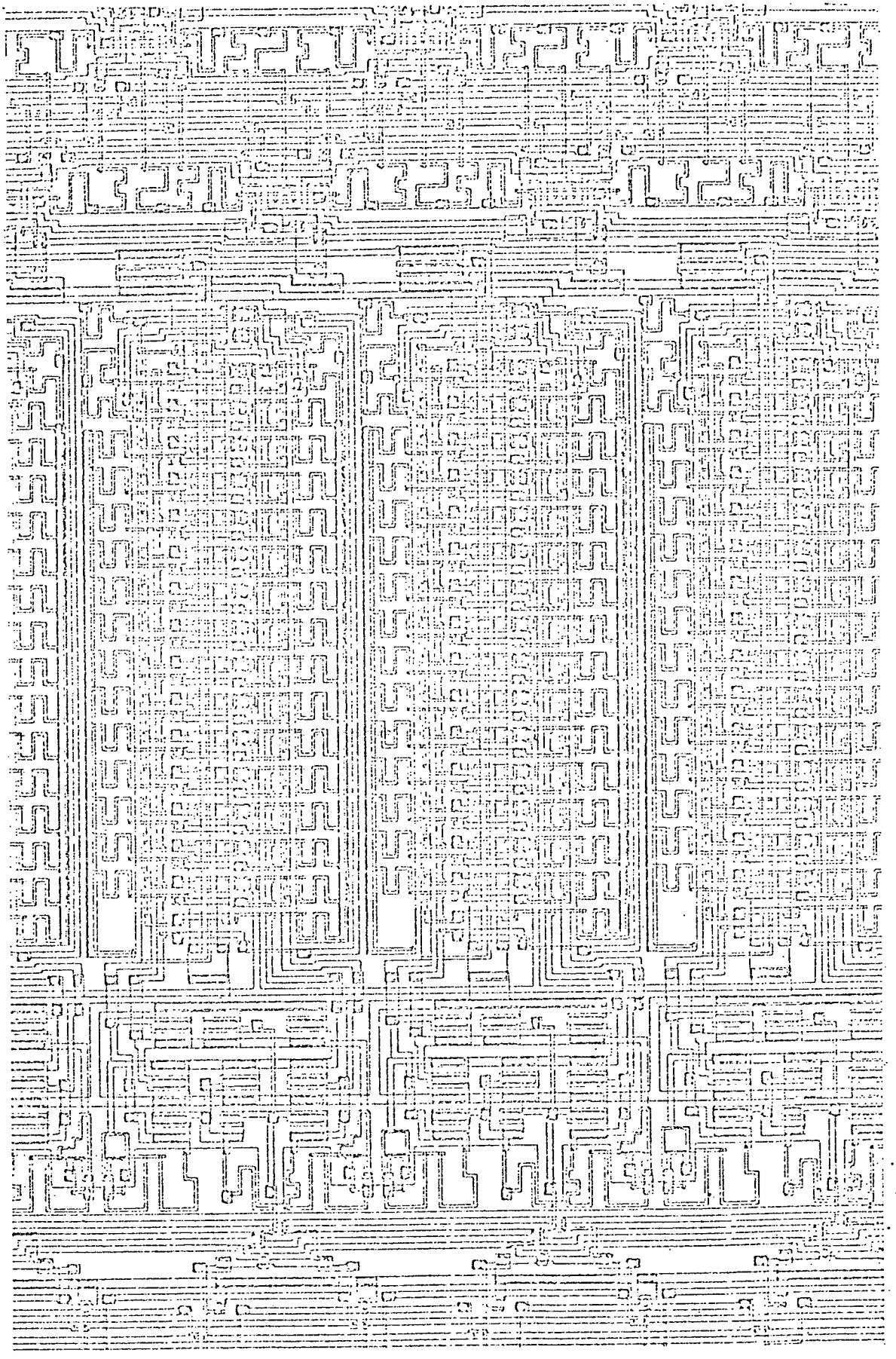


Fig 8. 3 Part of large layout with grouped and repeated shapes

9 min measurement indicates that there was only one user on the system and that the time taken for random access disc transfers was included in the CPU time mesurements. If the time for the sequential file handling was charged to the connect time and not to CPU time then this would explain the fact that the connect time was correspondingly longer for the sequential block data structure.

8.3 Minor Improvements to GAELIC

The GAELIC programs were transferred from the GE430 computer used for the Systemshare time-sharing service to the Decsystem 10 of the CAD project. The programs were then modified to calculate and store the bounding rectangle of each definition in the data structure and to make use of these bounding rectangles when plotting or modifying the layout. Certain other minor improvements were made to the code at the same time to speed up the operation. The original programs required each shape to be checked against the window to be plotted regardless of the fact that it is in the instance of a group definition and the whole instance is outside the window. The new programs check the bounding rectangle of an instance of a group against the window by making the necessary transformations of the bounding rectangle of the definition and the instance is ignore if completely outside the window. These two versions of the program were compared for processing the same layouts as before i.e. figs 8.1, 8.2 and 8.3 and the results are shown in

table 8.2.

These results show that the version with the bounding rectangles and other minor improvements requires less CPU time to create the ring data structure than the original version. This was due to certain of the minor modifications. The time to plot the complete layouts showed no appreciable difference between the two versions; the bounding rectangle version would of course have taken longer without the minor modifications. The CPU time required to plot a window of the large circuit was appreciably reduced from 63 sec to 18 sec.

The bounding rectangle concept was therefore well worth implementing into the GAELIC system. This new version of the GAELIC programs known as the 'S' version was used by Mr R. Kelly of the Wolfson Unit to design the layout of an integrated circuit correlator. His experience of using the programs are summarised in a joint paper [ref 8.1] presented at the CAD Conference at Southampton April 1974. His comments on the ergonomics of the system were extremely useful and they enabled an order of priority to be obtained for the various modifications and improvements that were desired. His main criticism of the system was the time taken to plot out a different window of the layout as the coordinates of the window had to be typed in. Other ergonomic problems were those involving shapes within definitions. For reasons described in earlier chapters, it was only possible to identify and modify shapes in the particular definition

being processed at the time. One can identify and modify the origin of the instance of a group definition but one cannot modify shapes within the definition. However, it is always possible to discover an error in a definition when viewing an instance of it in another definition. A designer will often know the name of a definition containing a shape but on the occassions when he cannot remember, he has the difficult task of plotting each definition in turn. In general the facilities provided were performing the required functions but not necessarily in the most efficient way.

8.5 Effect of Area Beads.

Chapter 5 describes the reasons why the size of the areas into which the layout is divided should effect the performance of the data structure. Measurements were made of various aspects of the performance of the data structure. The main aspects of the performance are the time to create the data structure, the time to plot the whole layout, the time to plot a window and the time to identify a point in the definition. The effect of the area beads will not be noticed on small circuits and will be most noticeable on layouts with a minimum of group and repeat definitions. It is extremely difficult to obtain large integrated circuit designs that can be published and unless the compexity of the design can be shown with the results, the results loose much of their value. Part of a MOS shift register design was obtained from ICL with

permission to publish at a scale shown in fig 8.4. This can be seen as having a large number of group and repeat definitions, but the data was processed to remove all these definitions and just produce rectangles and polygons. It was the data for one layer, the metallisation, that was used for the first set of tests, the results of which are shown in tables 8.3 and 8.4 and figs 8.5 and 8.6. The tests are of the CPU time to compile the dump code file into the data structure, to plot the full layout on the Tektronix and to plot a window for a range of area sizes.

The first measurements are of the CPU time required to create the ring data structure file from a dump code file for various area size with the comparative time to produce the previous ring data structure without the area concept as comparison. The 'area' program also monitors the number of 'disc reads and writes' that were required to create the data structure and these measurements are shown with the others in table 8.3. The term 'disc reads and writes' refers to the number of times that the required page of the data structure was not in core and had to be read in, after writing out another page to disc if its contents had been changed, it does not refer to the actual number of disc transfers required. The results show a rapid increase in the number of 'disc reads and writes' as the size of the area is reduced with a proportionate increase in the CPU time. This is to be expected as the number of area beads must be increased as

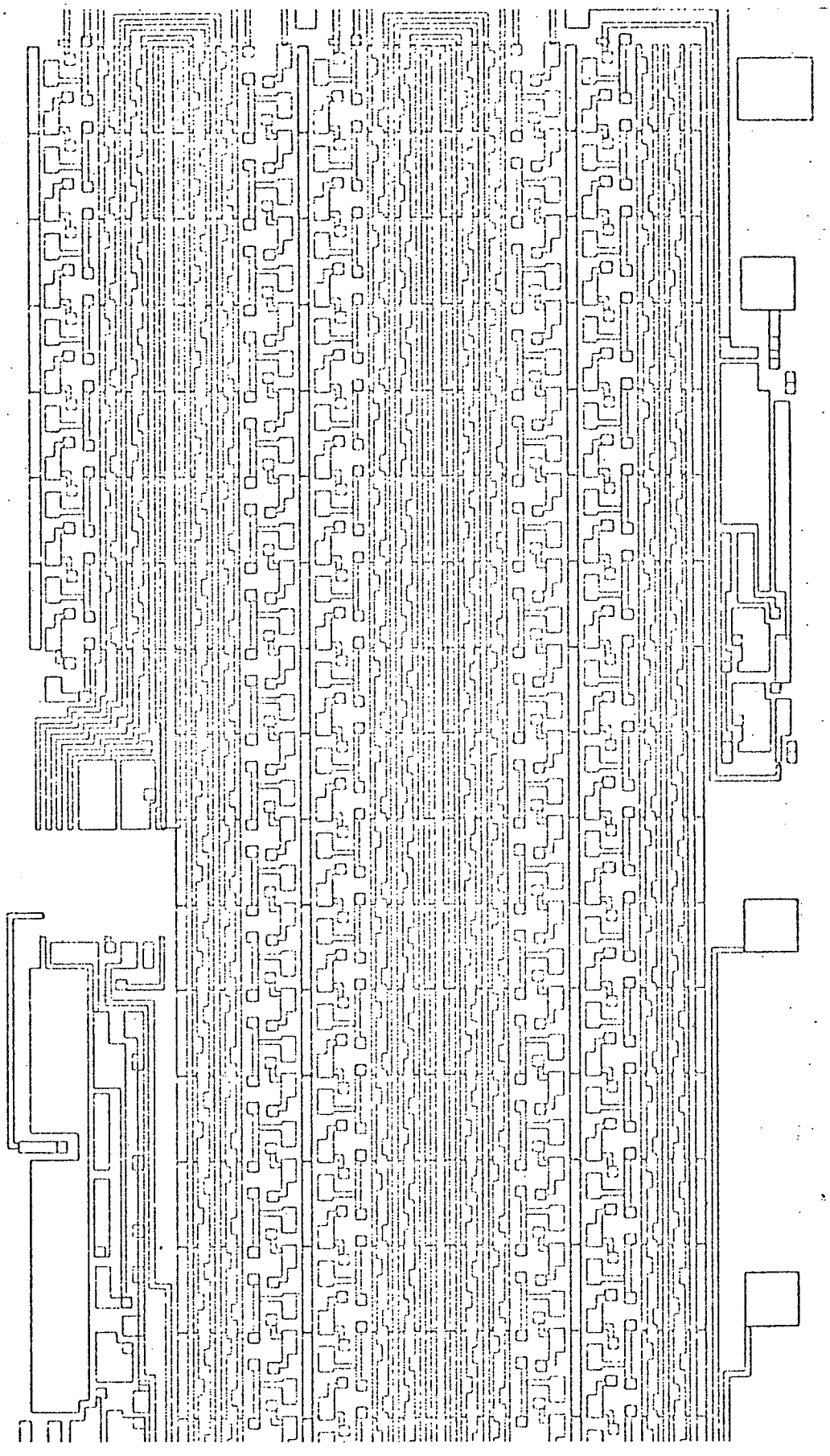


Fig 8. 4 ICL layout

the area size is reduced. As each new shape is entered its area bead is calculated, and each area bead already present must be checked to see the required bead is present and if missing must be added to the data structure. The first version of the program created the area bead only when required and this meant that the area beads were fragmented throughout the data structure and so many pages had to be read just to examine the area beads. The version actually used in the test required the bounding rectangle of the layout to be entered via the keyboard and the program then sets up all possible area beads before actually reading the data from the dump code file. This means that all the area beads are on the same page or on consecutive pages in the data structure and so the number of pages that need to be read in are minimised. This for areas of 1024 increments reduced the number of disc reads from 4650 to 20, the number of disc writes from 1008 to 52 and the CPU time from 23 mins 14 seconds to 55 seconds. The increased CPU time for the largest area size over the other large size areas is more difficult to explain - the reason is probably due to an optimising feature of the Fortran Operating System where the system detects that division by a power of 2 is taking place and does bit shifting on the number, division by any other number requires a normal division process which takes longer. The largest area size required a division by unity whereas all others requires a division by a factor of two. It therefore appears that unity is not processed as a power of 2.

The second set of measurements were of the CPU time and disc reads required to plot out all or part of the layout for various area sizes from the initial ring data structure created from the dump code file. The part of the layout plotted was a window whose bottom left hand corner was 1000,1000 increments and whose top right hand corner was 1200,1200 whereas the bounding rectangle of the complete layout was 348,287, and 6678,3270. The results are shown in a tabular form in table 8.4 and the CPU time and disc reads in graphical form in figs 8.5 and 8.6 respectively. For comparison, similar measurements for the original data structure without the use of areas are also shown in each figure.

When plotting a full layout, the CPU time required is always greater than that required for the original data structure. This is due to the time required to process the area beads, the time taken to check if each bead is within the plotting window. However, as the size of the area is decreased there is a substantial increase in the CPU time required. This increase is due to the fact that the data is fragmented, shapes within a given area are almost certainly not all on the same page and so the respective pages must be brought into core to enable the data to be extracted. The shapes for another area may be on the same pages but in a different order and the pages must be brought back in again when the next area is plotted. This would mean that a large number of page reads would be required and a corresponding increase in

CPU time to set up the necessary page transfers. The actual page reads were monitored at the same time as the CPU time was measured and the number of reads substantiate the hypothesis.

Plotting out a small window requires only certain areas to be processed. The actual areas required are the area 0 which contains all the large shapes and any areas that can contain shapes that could appear within the window, i.e. areas within the window and those immediately adjacent as described in Chapter 5. The results show that there is an optimum size of area for the window plotted that minimises both the CPU time and the disc reads to plot out the window. This is due to the varying number of shapes within each area: if the area is large, processing the areas within the window and all adjacent areas can result in processing most of the data structure. At the other extreme, however, if the area is too small, most shapes will be too big to be associated with the normal area beads and must therefore be classified as large shapes and associated with area number 0. When plotting a window, area number 0 must always be processed and so for any window most of the shapes must be processed and so again the CPU time and number of disc reads must again be high. The area size that requires the minimum of CPU time and disc reads must therefore lie between these two limits and for plotting a window of this circuit this optimum size is shown to be approx 512 increments square. At this size there is a saving of approximately 30% in the CPU

time and the number of disc reads compared with the original ring data structure.

If most of the interactive design time was spent plotting the full layout rather than a small window, then the area concept would not have been justified. Fortunately, most of the design time is spent plotting small windows and a reasonsable estimate is 90% of the plots are of small windows and only 10% of them are of the full layout. The area concept is therefore more than justified even on this initial area ring data structure.

This initial data structure is arranged on the disc in the order in which the data was entered. This is almost certainly not the best order for subsequent processing and so the program GAEL8A is used to rewrite the data structure onto disc in a more optimum order. The measurements made on the initial data structure were repeat on this 'clean' data structure and the results are shown in a tabular form in table 8.5 and in a graphical form in figs 8.7 and 8.8.

The CPU time and number of disc reads required to plot the full layout is considerably reduced and compares far more favourably with the original data structure. This is because the 'clean up' process ensures that all the shapes within a given area are written consecutively on the disc i.e. on the same page or consecutive pages and so once a page of data is brought into core the maximum amount of data is obtained from it.

		Layout 1	Layout 2	Layout 3
GAELIC Input Language	length blocks	4	2	75
GAEL2 Syntax Checker	CPU Time Secs	5	3	84
Dump Code File	length blocks	4	2	74
GAEL3 Compiler to RDS	CPU Time Secs	2	1	86
Ring Data Structure File	length blocks	8	4	100
GAEL4 Full Layout	CPU Time Secs	3	4	154
GAEL4 Small Window	CPU Time Secs	-	-	63
GAEL2A Syntax Checker	CPU Time Secs	6	3	65
Dump Code File	length blocks	4	2	74
GAEL3A Compiler to RDS	CPU Time Secs	1.5	1	74
Ring Data Structure File	length blocks	8	4	100
GAEL4A Full Layout	CPU Time Secs	3.5	5	151
GAEL4 Small Window	CPU Time Secs	-	-	18

Comparison of Original GAELIC with version using Bounding Rectangle

Table 8.2

Area Length increments	CPU Time min:secs	Number of Disc Reads	Number of Disc Writes	Length of D.S. Blocks
128	27:52	3574	1151	55
256	3:42	53	88	44
512	1:35	30	63	41
1024	:56	20	52	40
2048	:43	5	37	40
4096	:37	0	32	40
8192	:37	0	32	40
16384	:38	0	32	40
32768	:39	0	32	40
Original D. S.	:30	-	-	40

Results creating 'Initial' Data Structure

Table 8.3

Area Length increments	Plotting Full Layout		Plotting Small Window	
	CPU Time Secs	Number of Disc Reads	CPU Time Secs	Number of Disc Reads
128	118	195	19	54
256	100	198	14	41
512	71	125	9	27
1024	80	137	17	57
2048	64	91	16	51
4096	56	50	19	50
8192	51	40	15	40
16384	52	40	16	40
32768	52	40	16	40
Original D. S.	50	40	14	40

Results using 'Initial' Data Structure

Table 8.4

Area Length increments	Plotting Full Layout CPU Time Secs	Number of Disc Reads	Plotting Small Window CPU Time Secs	Number of Disc Reads
128	62	53	11	28
256	59	53	7	13
512	58	44	5	9
1024	55	51	8	18
2048	62	50	12	25
4096	52	43	17	43
8192	52	40	16	40
16384	51	40	16	40
32768	53	40	16	40
Original D. S.	50	40	14	40

Results using 'Clean' Data Structure

Table 8.5

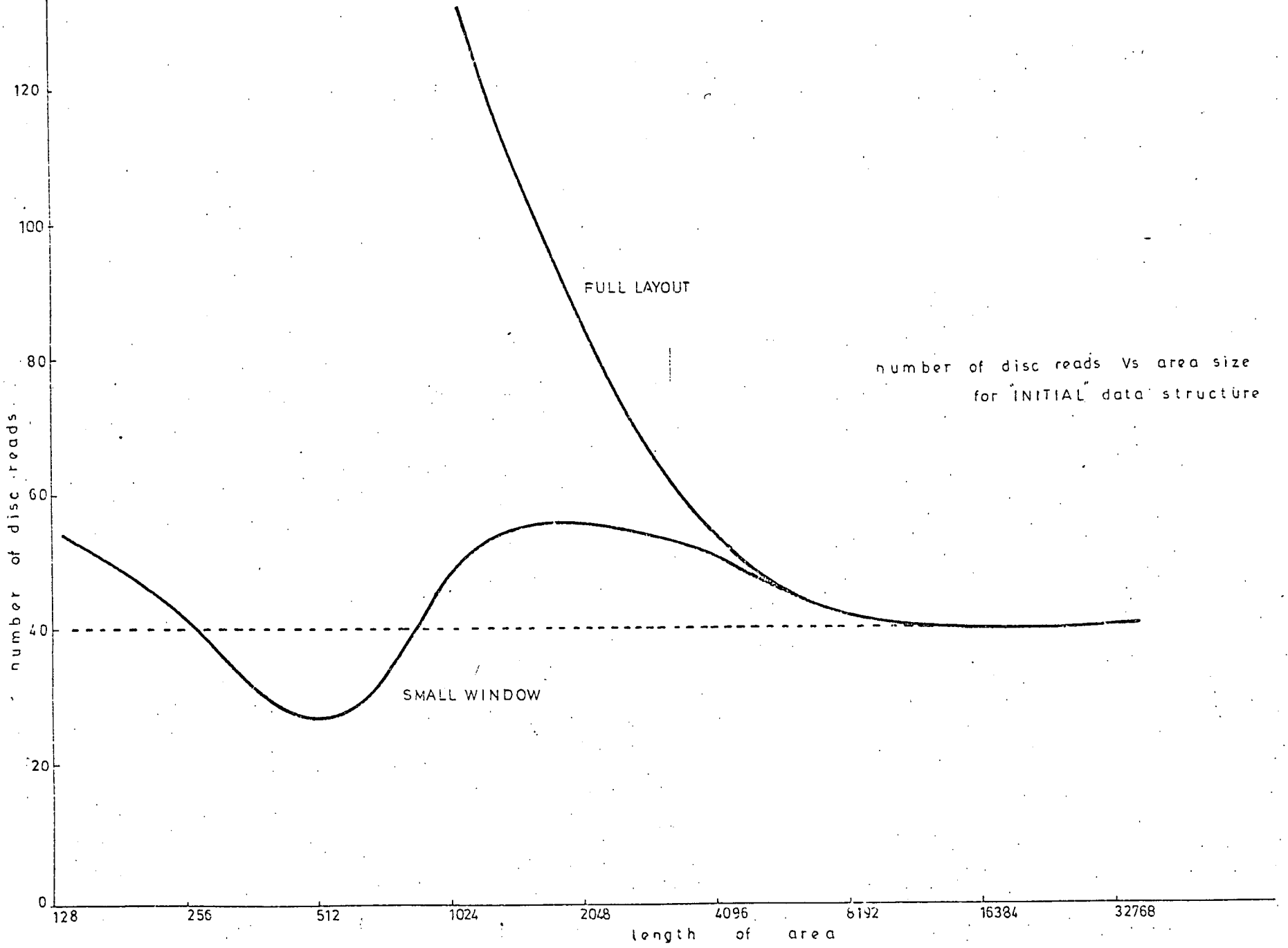

number of disc reads Vs area size
for "INITIAL" data structure
FULL LAYOUT
SMALL WINDOW
number of disc reads
length of area
0
20
40
60
80
100
120
128
256
512
1024
2048
4096
8192
16384
32768

Figure 8·5

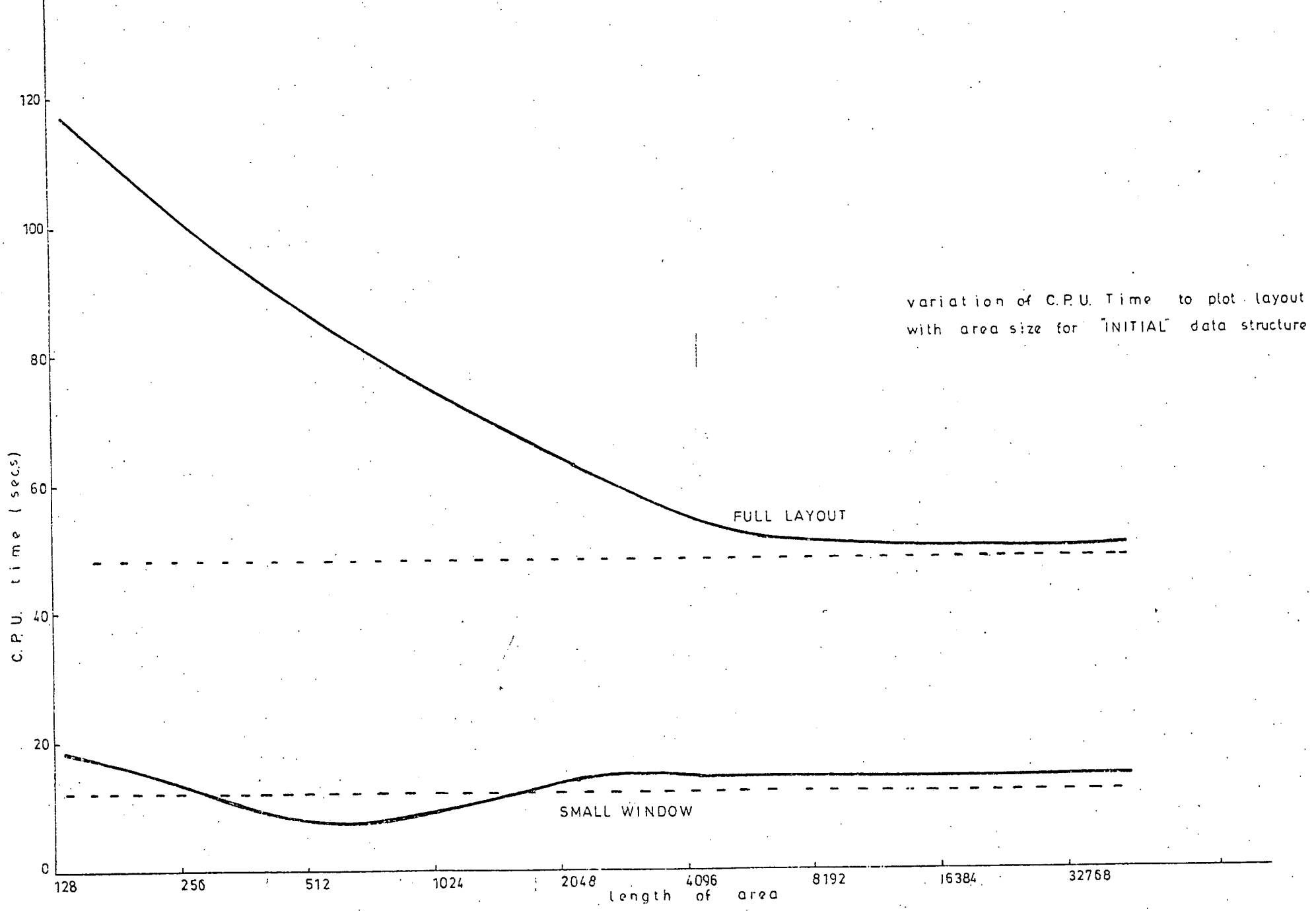

variation of C.P.U. Time to plot layout
with area size for "INITIAL" data structure
C.P.U. time (secs)
120
100
80
60
40
20
0
FULL LAYOUT
SMALL WINDOW
128
256
512
1024
2048
4096
8192
16384
32768
length of area

Figure 8·6

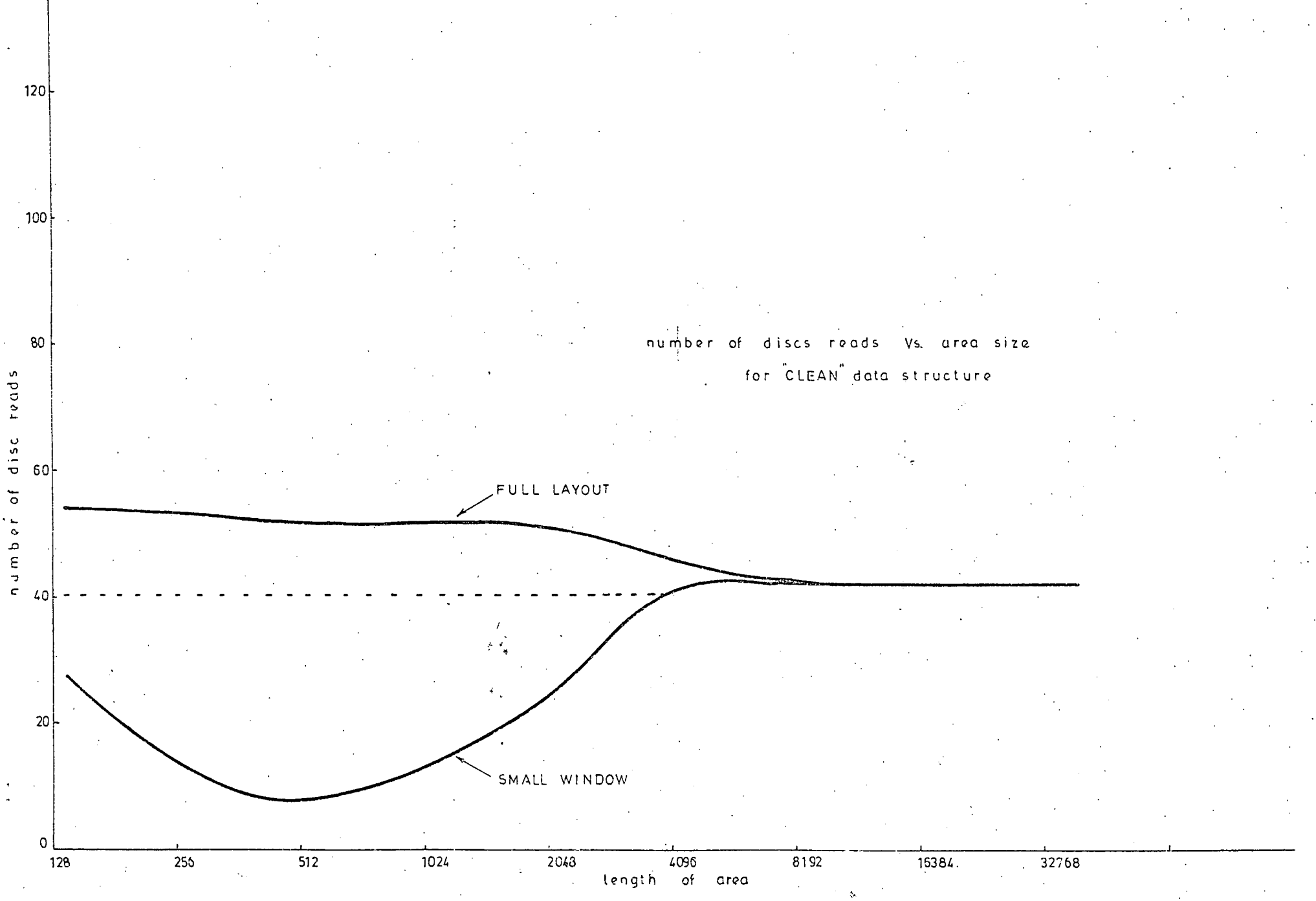

number of discs reads Vs. area size
for "CLEAN" data structure
number of disc reads
0
20
40
60
80
100
120
FULL LAYOUT
SMALL WINDOW
128
256
512
1024
2048
4096
8192
16384
32768
length of area

Figure 8·7

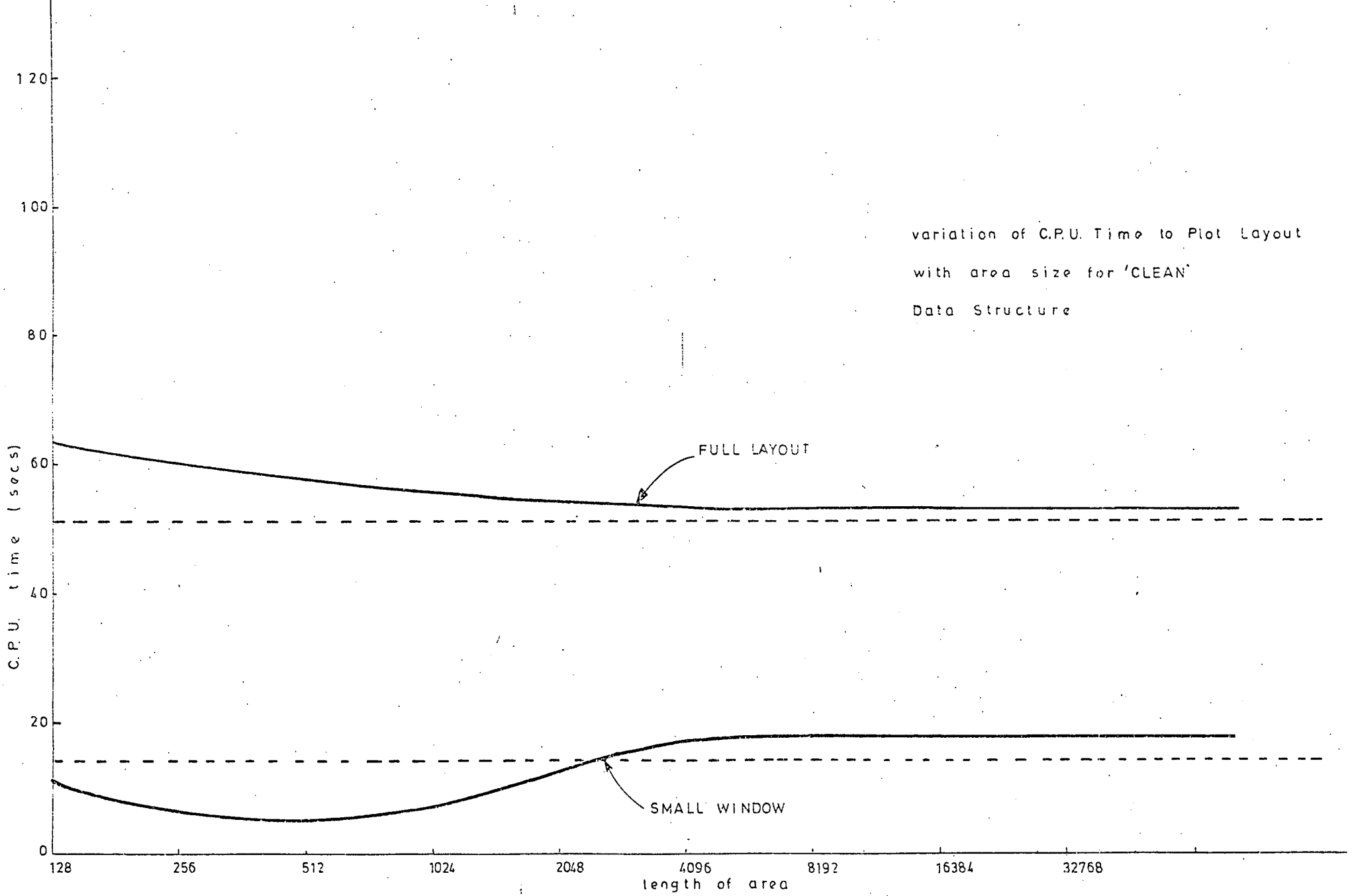

variation of C.P.U. Time to Plot Layout
with area size for 'CLEAN'
Data Structure
FULL LAYOUT
SMALL WINDOW
C.P.U. time (secs)
0
20
40
60
80
100
120
128
256
512
1024
2048
4096
8192
16384
32768
length of area

Figure 8·8

CHAPTER 9: Future Work

The GAELIC programs have been successfully applied to the design of integrated circuit layouts and the experience gained from this has indicated several possibilities for future work. This work can be divided into three main categories. Firstly there are direct extensions to the GAELIC programs which enhance its usefulness for integrated circuit design. Most of these extensions such as merging two shapes containing a common line segment are straightforward and are implemented when required. There are some that are far more difficult to implement such as the provision of constraints. Secondly there are additional programs that are required which extend the use of the system and layout rule checking and mask function checking fall into this category. Finally there are several applications in other disciplines that can be met by the GAELIC software such as thin film layout and timber framed house design.

9.1 Constraints

Often when designing an integrated circuit a situation occurs where one component or shape must be a fixed distance from another component. A typical example is the metallisation over a contact hole. Here if the contact hole is moved, the metallisation should be constrained to move by exactly the same amount. The contact hole and metallisation however do not share the

same coordinates as the metal must overlap the hole by a fixed distance. This type of constraint was a feature of the original Sketchpad work and was present in the Marconi Myriad graphics system. It was not, however, inserted into GAELIC as

1) it uses a large amount of computing power to check that the constraints are not being violated and

2) poses problems of how the constraints should be satisfied. Take for an example the contact hole and metallisation shown in fig. 9.1.

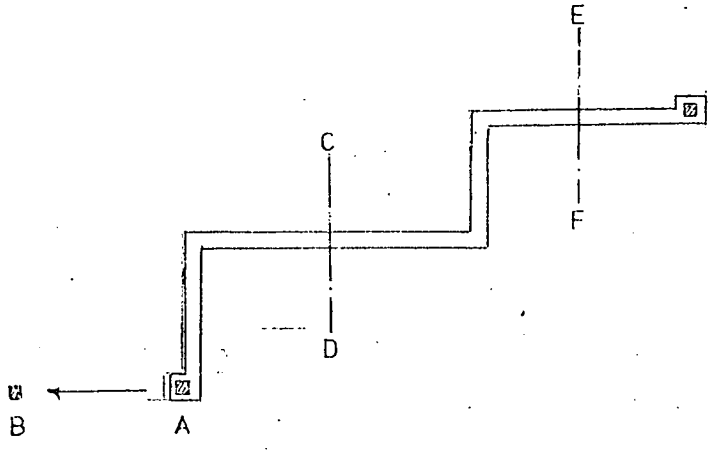


Fig. 9.1 Movement of Metallisation with Constraints.

If the contact hole is moved from 'A' to 'B' then the program is faced with the problem of whether the metallisation should be stretched along line CD or along line EF. The answer depends on the other components in the circuit and writing computer programs to solve this problem is a separate piece of research work.

9.2 Layout Rule checking

The semiconductor manufacturers always produce a set of 'layout rules' for each integrated circuit process. These specify the minimum and maximum dimensions that are allowed on the masks. For example, the rules will specify that minimum width for a particlar diffusion track is so many units or that one shape cannot be closer than a given amount to another shape on the same diffusion. Returning to our example with the contact hole and metallisation, the metallisation must always overlap each contact hole by a fixed quantity. These checks can obviously be done efficiently by the computer and the GAELIC ring data structure is an ideal way of holding the necessary layout description. Previous work on this problem has produced programs that require a great deal of computer time to run because all the data must be searched over and over again. The area association used in GAELIC means that only certain areas and consequently only certain parts of the data structure need be examined. One of my colleagues at the University of Edinburgh is at present working on a program using the GAELIC ring data structure and the results appear extremely promising.

9.3 Mask Function Checking

The layout rule checking just discussed will tell the designer that he has two shapes too close together or that he has a contact hole with no metallisation covering it. However,it will not tell him that it is the wrong metallisation over the contact hole and that the circuit cannot possibly work. This requires another type of program that is more concerned with the function of the circuit produced by the masks and is consequently called 'mask function checking'.

The problems associated with mask function checking are quite complex. Attempts have been made to feed in the electrical description of the shapes at the same time as their topological description. Unfortunately this does not necessarily mean that the electrical data is correct. Mistakes could be made when entering the description that are not detected and so although a program checking the layout will predict that it will work, the actual circuit may not. It is therefore imperative that the electrical description must be extracted automatically from the layout description. There are so many different integrated circuit components available that to search through all the shapes trying to decide if they form resistors, capacitors or transistors etc. would take far too much computer time. A method must be found where the user does not enter the electrical description per se, nor does the computer have to work everything out for itself.

A possible solution to this problem is to run an interactive program where the user tells the computer what component he thinks is formed by a series of shapes and the computer checks if it is true. The computer only checks if the shapes form one particular components ie. does not have to try all possible components nor does the users data have to be perfect. It also has the advantage that the user can restrict the range of data search by specifying that the component lies between certain topological limits. For example he could indicate that the emitter, base and collector terminals of a transistor are at coordinates x1,y1 x2,y2 etc. Again the area association of the GAELIC data structure can save considerable computer time.

Another problem that has to be solved is how does the computer check that it is the correct circuit. At first sight it would appear that the answer was to draw out a circuit diagram and let the user check it against his original diagram. However, there are two objections to this. There are stray components that would not have been specified in the original diagram which would make the circuit appear different. There is also the more fundamental problem of programming the computer to draw out the circuit diagram in the same style as the original circuit diagram. How many of us have looked at a circuit in a book for several minutes before realising that it is a common circuit just drawn to a different convention? A possible solution to this problem is to derive the input

data for a transient analysis or logic simulation program from the description of the layout in the ring data structure. The test sequence that will be used on the final circuit is then applied to the appropriate input terminals and if the response is identical with the required response from the finished circuit, then the layout is correct or an alternative layout has been designed. This does obviously rely on the test sequence being correct which is a problem in its own right.

Another colleague at the University of Edinburgh is now working on this problem of 'mask function checking'

9.4 Automatic Layout

For the reasons outlined in Chapters 1 and 2, it is extremely difficult to write fully automatic layout programs. For some time to come, therefore, it will be essential for the designer to manually interact with any attempted automatic layout design. This manual interaction can obviously be carried out using GAELIC.

One approach to the automatic design problem that merits further investigation is to use the approach of Radley of placing the component and then placing all the metallisation associated with that component. However, the components should not be as small as transistors or resistors rather gates and flip-flops. These can be predesigned (again using GAELIC) or just an estimate of

the required area of silicon given. The program can then allocate the required shape or the required area. The required area can be some of the space between existing components and the user can then use GAELIC to design the actual component to fit in the alloted space.

The other approach is to use entirely predefined components and to use a centre of gravity algorithm to get a placement. However, the routing is done first and so the component sizes are increased to take account of the space required by the interconnections.

Another feature that should be exploited in any automatic design program is that certain interconnections in the actual circuit must be extremely low impedance, while others can be quite high impedance without upsetting the performance. This information must be fed to the computer in order to decide on where crossovers must be placed.

Research work is being carried out using this approach by the C.A.D. Project at Edinburgh University.

9.5 Stand Alone Computers

The GAELIC programs can be run on smaller stand alone computers such as the PDP11, the Nova 1200 and the Modular 1. The computer must have a Fortran compiler, must have discs and must be capable of handling the Tektronix terminals. There should be at least 24k of core store,

preferably 32k to avoid problems with overlaying, and it is desirable to drive the Tektronix at 9.6KBaud or faster.

The advantages of using a standalone computer are

1) that the time-sharing delays while the computer services other users are removed completely,

2) the Tektronix can be driven at its maximum speed ie. 9.6KBaud or above,

3) the discs can have fixed heads giving a much faster transfer rate and

4) it may well be cheaper. If the program is to be used to design more than six large integrated circuits a year then the economics indicate that a standalone computer is cheaper than using a commercial time-sharing company.

There are disadvantages using a standalone computer. There has to be a certain amount of reprogramming to allow for the shorter word length. With the size of integrated circuits being produced nowadays a 16 bit word is not capable of holding the complete range of addresses required for the data structure. Often there are restrictions in the Fortran compilers where certain standard functions are not implemented and the standard disc handling routines supplied by the manufacturer are far from optimum for this type of application.

The first version of GAELIC has been fully implemented on a Modular 1 computer by Smith's Industries, Cheltenham and been used to design integrated circuits for

at least two years. The final version of GAELIC has been implemented on a P.D.P.11/40 but has not been extended to cater for double word addressing.

9.6 Refresh Graphics

There is no reason why the GAELIC programs cannot be used with a refresh graphics terminal and a stand-alone computer. There will be the problems of flicker described in chapter 6 and this will put a limit on the size of picture that can be displayed. Most of the work on integrated circuit design is done with a small window but occassionally the whole layout is required and this creates severe problems.

The area association in the GAELIC data structure will considerably reduce the time required to regenerate the display file although a lot of work will have to be done if a light pen is to be used to feed back information from the graphics terminal to the ring data structure when components are moved etc. If a tracking cross controlled by a tracker ball is used then the amount of new programming is minimised.

9.7 Layout Design with Automatic Rule Checking

The minimisation of the search time brought about by the area association of the data structure means that it should be possible to write a version of GAELIC that checks each component as it is entered or modified to ensure that it does not violate the layout rules. Using a data structure without area association would mean a search through the compete layout each time that a shape was moved or entered and could not be done in a reasonable time. The time taken to interact would be so slow that the system would not be usable.

The cost of using GAELIC on a commercial time-sharing computer is relatively high at the moment. Increasing the amount of computing done at each stage may make it too expensive too use. However, it is nevertheless a practical possibility on a standalone machine.

9.8 Thin Film Circuit Design

The ability to move components around and to add interconnections between them is obviously desirable when doing thin film circuit layout design and there is obviously a use for GAELIC here. There is also a problem of designing accurate thin film that can be considerably simplified by the use of the computer. Designing a resistor by hand to go into a given area involves a great deal of drawing a meandering resistor and the counting

squres to see if the value is correct. Realising this problem, a student was employed during a summer vacation to work on this problem and under the author's direction wrote a program to automatically design a given value resistor on been given its value and bounding rectangle. A typical design is shown in fig 9.2: the design is coded up in the GAELIC manual input language and can then be placed in the required position using the GAELIC programs.

The author is currently extending the work on thin film layout design on an S.R.C. grant.

9.9 Timber Framed House Design

This work resulted from the author's desire to design a house to be built on a plot of land he had purchased. Timber framed houses are constructed from a selection of mass produced timber frames. Certain standard designs are produced by the timber frame manufacturers but there are no substantial reasons why the frames cannot be used to construct individually designed houses. The designer need only know the types of frames available, the sizes of roof trusses that can be used and the positions of any load bearing walls and he is in a position to do his own designs. The GAELIC suite can be used to advantage here by setting up a library of standard frames and then interactively calling them up and positioning them on the Tektronix screen. The frames can be coded in the computer so that the actual frame dimensions are stored on one

Resistor value 150 KΩ

Resistivity 400Ω / sq

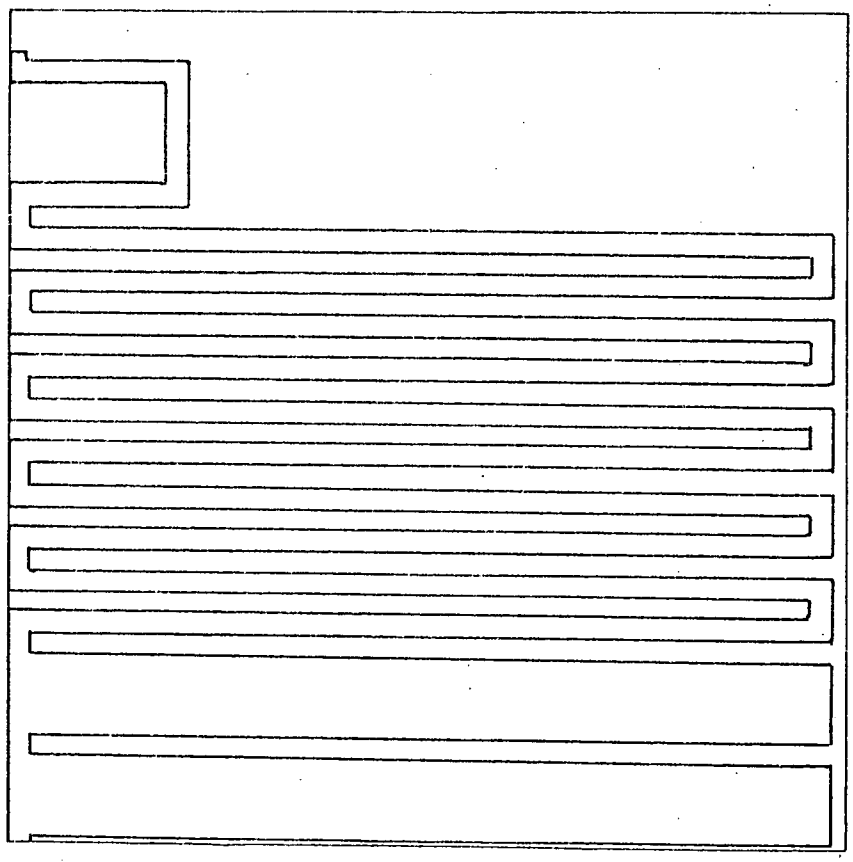


Fig. 9.2 Resistor design to fit in given area

layer and the overall dimensions stored on another layer. The timber frames are first drawn on the screen to ensure that they fit together correctly and then the other layer can be displayed to show the actual plan. The plan is then be plotted out on the Calcomp 563 plotter at the correct scale for submitting to the local planning authority. A similar process is employed to obtain the elevations. Examples of the plan and elevations of a bungalow are shown in figs. 9.3 and 9.4.

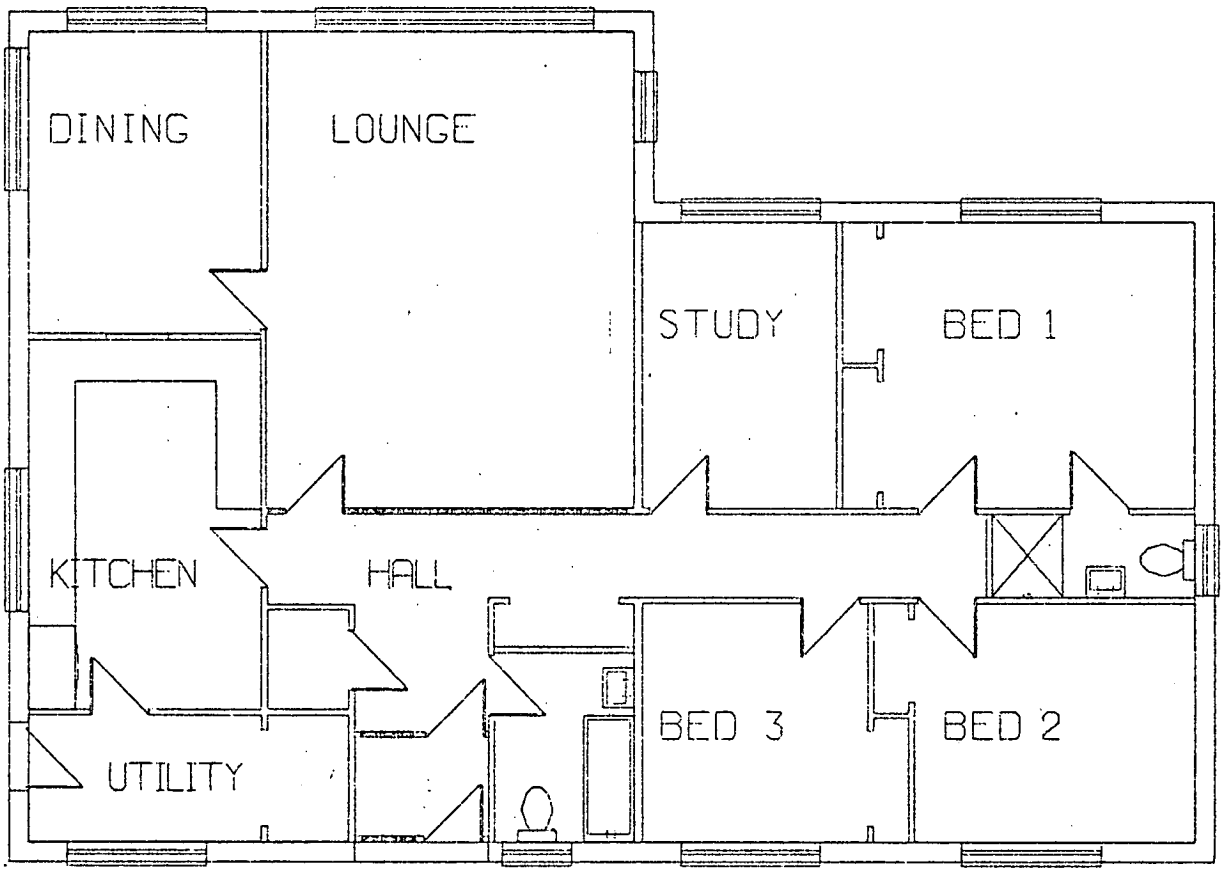


Fig. 9.3 Plan of timber framed house

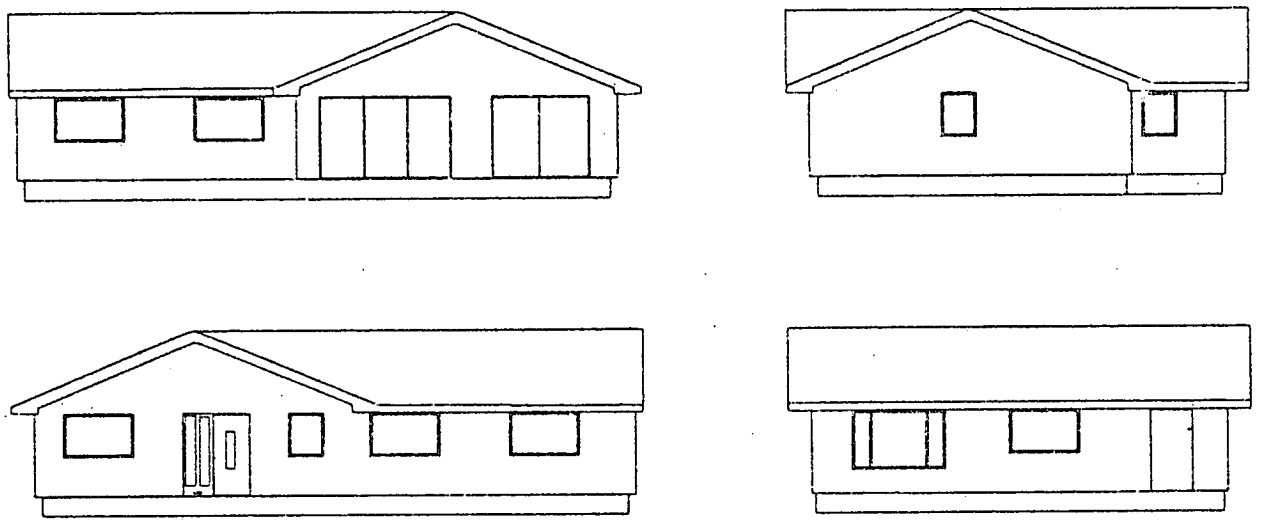


Fig. 9.4 Elevation of timber framed house

CHAPTER 10: Conclusions

This final chapter contains a brief discussion on whether the requirements for a computer aided mask design system were sound and whether they were met by the GAELIC programs.

10.1 Were the requirements sound?

The overall objective of the work was to produce a suite of computer programs that would assist in the production of integrated circuit masks. There were two possible methods of approach; the first was to write a suite of programs that would remove as much of the tedious repetitive work from the design cycle as possible, leaving the designer free to concentrate on the actual process of designing. The second was to write programs that automatically designed a layout on being given a schematic diagram of the circuit.

This second approach was rejected for four reasons:

1) The problems of writing such programs are severe and any proposed solution cannot be guaranteed to be succesful.

2) The programs would not allow for any variations in design technique as any new design ideas would take months to implement. The programs would therefore be continually out of date.

3) The designer is not going to be responsive to a

program that threatens to make him redundant.

4) The designer can always stop during his design process when he realises that a new situation has occured and think how to get round the problem. The computer, on the other hand, will continue to work the way it has been programmed, regardless of the consequences.

It was decided to adopt the first option where the designer is still in charge of the design. Since that decision all the British semiconductor manufacturers and most of the equipment manufacturers use this approach. None of them use fully automatic programs. The overall requirement was therefore extremely sound.

The decision to write a portable set of programs rather than a set of programs for a specific computer is open to slight doubt. At present GAELIC is the only suite of layout design programs that is running on a commercial time-sharing service and so is unique in this respect. It provides the opportunity for equipment manufacturers and educational establishments to try designing their own integrated circuits with a minimum capital cost. From a commercial point of view, however, it can be argued that this was not the correct decision as the integrated circuit manufacturers have all chosen 'turn-key' systems on mini-computers. However these systems were produced by very large teams of hardware and software engineers. It has been reported, for example, that Applicon have a team of 80 programmers working on their software. It would have been impossible to compete with that sort of backup.

GAELIC can be mounted on mini-computers as discussed in Chapter 9 and so can be used as a turn-key system.

The choice of the Tektronix storage tube terminal for the interactive part of the program was sound as it has enabled the software to remain extremely portable. It also allows the largest of integrated circuits to be displayed without flicker. Most of the successful 'turn-key' systems use the storage tube display, usually with special interface hardware.

10.2 Were the requirements met?

The original GAELIC software has been in use at Smith's Industries at Cheltenham for several years now and produced a large number of successful integrated circuit designs. The Wolfson Unit have used the original version to design the correlator discussed in Chapter 2 and have used the latest version to design another two large integrated circuits. The correlator design was so successful that working samples were obtained from the first batch of circuits produced and it has not been necessary to make any changes to the masks. This achievement is mainly due to the skill and patience of the designer but is nevertheless partly due to the ease with which the layout could be changed during the design phase. The two remaining designs are complete and are awaiting mask making.

The Post Office Research Establishment are currently using the latest version of GAELIC to design a variety of integrated circuits on a commercial time-sharing service and the Royal Radar and Signals Establishment are also using it for the design of charge coupled devices.

The overall requirements have without doubt been met and GAELIC has proved itself a commercially viable system.

APPENDIX 1: The variation of overall yield with die size

There are two main factors that effect the yield of an integrated circuit slice. The first is the number of complete die it is possible to get from the slice and the second is the number of the die that are perfect. Let us consider the effect of the yield on the number of perfect die that can be obtained from a given process.

Assume that for a given process the yield of die 100thou by 100thou is 50%. The probality of any one complete die being perfect is 0.5. If a 200thou by 200thou die was made using the same process then the probability of the top right hand quarter being perfect is 0.5, the probability of the top left hand quarter being perfect is 0.5 etc. The probability of the total die being perfect is the product of the probabilities of the quarters being perfect ie.

0.5 * 0.5 * 0.5 * 0.5 = 0.0675

This can be expressed more generally. If the probability of a die of area A being correct is Pa, then the probability of a die of area B being correct is Pb, where Pb is given by:

Pb = Pa ** (B/A)

where ** represents raised to the power of.

The total number of perfect die is therefore the maximum number of die possible times the probability that each die is perfect. It therefore is essential to calculate the maximum number of die possible.

Appendix 1

Calculating the number of complete die that can be obtained from a slice is relatively straightforward. However it must be remembered that the number of die possible not only depends on the size of the die and the size of the slice, but also on the positioning of the scribe lines between the die and the diameters of the slice. Consider the three slices shown in fig. 1.

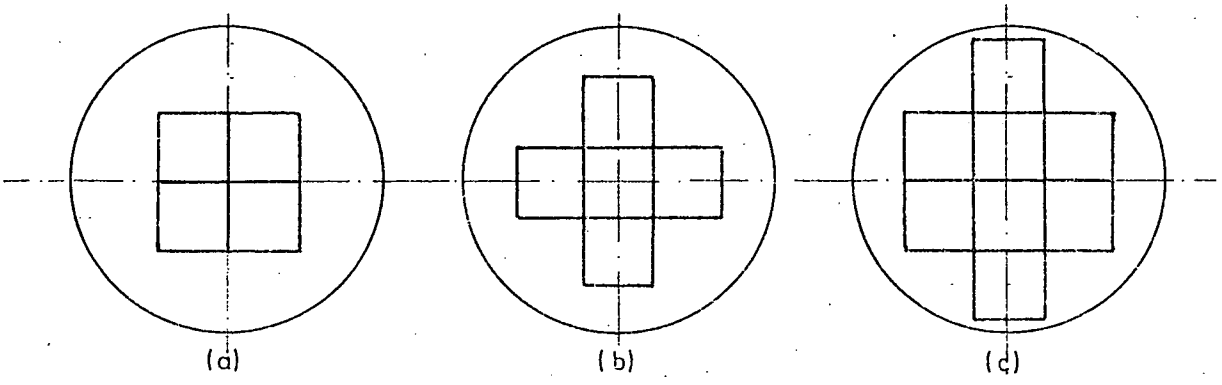


Figure 1 Possible complete die

The first slice (a) has the die positioned so that the scribe lines are coincident with the diagonal in both directions and results in four complete die. The second slice (b) has the diagonal exactly half way between the scribe lines in both directions and results in five die. The final slice has its diagonal coincident with the scribe line in one direction and half way between the scribe lines in the other direction and this results in eight die.

Fig 1 shows the three possible combinations on a square die, if the die is rectagular, there are four possibilities as slice (c) can be in two forms with either the long or the short side being coincident with the

diagonal.

A computer program was written by the author which calculates the cost of producing an integrated circuit of a given size compared with the cost of producing a circuit 100thou square for a given size of slice. The program calculates the number of complete 100thou square die from the given size of slice and from the given yield finds the number of good die per slice. The cost of producing a slice can then be found assuming that it costs one unit to produce one perfect 100thou square circuit. The number of complete die of the size entered can then be calculated and hence the number of good die per slice. The cost of producing a single die is then calculated from the cost of producing a slice.

This work was done in conjunction of B.R. Kirk of General Instruments, Glenrothes, Fife who checked the results against the actual yields obtained for various die sizes and found extremely good correlation.

APPENDIX 2: The insertion of beads into the group definition ring

In order that the bounding rectangles of the group definitions can be correctly computed, it essential that the beads on the group defintion ring are in a specific order. For example let us consider the definitions of two groups A and B where the definition of B contains an instance of A. It is necessary that the bounding rectangle of A is calculated before the bounding rectangle of B. This is simply done by ensuring that the definition bead for A preceeds the bead for B on the group definition ring and computing the bounding rectangles in order.

Unfortunately, one of the features of the GAELIC language is that it does not restrict the order in which the group definitions are entered. The defintion of B could therefore easily preceed that of A. The program that creates the ring data structure therefore must arrange the definitions in the correct order and this is done by an integer function called 'ISKGRP'.

Let us consider the problem in a little more detail by taking as an example the definitions of groups A - I which are structured as shown in fig. 1.

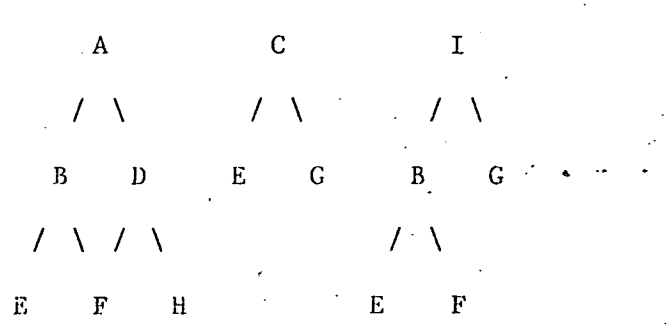


Fig. 1 Group structure for example 1.

Here the definition of group A contains calls to groups B and D, the definition of B contains calls to E and F etc. If we assume that these definitions are entered into the computer in alphabetic order, then the order of the group definition beads on the ring must be continually changed as shown in fig 2.

Appendix 2

Order of beads	Notes	Data entered
A		Definition of A
BA	[1]	Call to B in def of A
BDA or DBA	[2]	Call to D in def of A
BDA		Definition of B
EBDA		Call to E in def of B
EFBDA		Call to F in def of B
EFBDAC	[3]	Definition of C
EFBDAC		Call to C in def of C
EFBDAGC		Call to G in def of C
EFBDAGC		Definiton of D
EFBDAGC		Call to F in def of D
EFBHDAGC		Call to H in def of D
EFBHDAGC		Definition of E
EFBHDAGC		Definition of F
EFBHDAGC		Definition of G
EFBHDAGC		Definition of H
EFBHDAGCI		Definition of I
EFBHDAGCI		Call to B in def of I
EFBHDAGCI		Call to G in def of I

Fig 2 Ordering of Definition Beads

Notes

[1] If a group call is encountered before the actual definition, the definition bead must be inserted into the ring and it is sensible therefore to insert it before the bead of the calling definition.

[2] A bead created for the second group call in a

definition can be placed immediately before that of the calling definition or at the beginning of the ring. As the definition of the second group call can contain calls to other groups, the former position ie. immediately before the calling definition bead is prefered.

[3] A definition of a group that has not previously been called can contain calls to other groups and so it is better to insert it at the end of the ring rather than at the beginning.

From fig 2 and the notes, several rules can be derived for the insertion of definition beads into the group definition ring.

1) Actual definitions of groups must have the definition bead inserted at the end of the ring.

2) Group calls within another group definition must have their definition bead inserted before the bead for the definition containing the call.

3) Group calls within the main definition must have their definition bead inserted at the end of the ring.

The last rule is open to discussion but there is a greater probability of a definition called in the main definition containing calls to other definitions than there is of a definition called from another definition. It would, perhaps, be better to insert it in the middle of the ring between the definition beads set up because of group calls appearing in the input data and those set up because of the actual definitions appearing. This, however, is extremely difficult to do and so there only

remained the choice between the beginning and the end of the ring.

The above rules only apply when the definition bead is to be added to the group definition ring. If the actual definition or a call to the particular group has already been entered, then the definition bead will already be present on the ring. When this is the case it is sometimes necessary to move the definition beads on the ring to ensure that they remain in the correct order. For example consider the structure of groups shown in fig 3 which assuming that the definitions are again added in alphabetic order, gives the order of beads shown in fig 4.

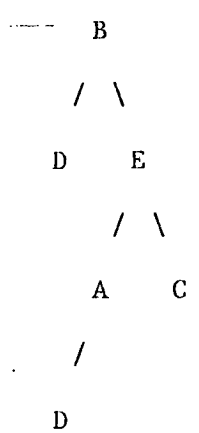


Fig. 3 Example of more complex group structure

Appendix 2

Order of beads	Notes	Data entered
A		Defintion of A
DA		Call to D in def of A
DFA		Call to F in def of A
DFAB		Definition of B
DFAB	[1]	Call to D in def of B
DFAEB		Call to E in def of B
DFAEBC		Definition of C
DFAEBC	[2]	Definition of D
DFAEBC		Definition of E
DFAEBC	[3]	Call to A in def of E
DFABCE	[4]	Call to C in def of E
DFACEB	[5]	Previous call to E in B

Fig. 4 Ordering of group definition beads

Notes

[1] Definition bead for D was already present and was positioned before the bead for B so no reordering was necessary.

[2] Definition bead for D was already present when actual definition entered so again no reoredering was necessary. Also the definition of D did not contain any calls to other definitions so again no reordering was required.

[3] The actual defintion of E contained a call to A but as the definition bead for A was already present on the ring before the bead for E no reordering was necessary.

[4] The call to C in the definition of E causes problems as the bead for A is present but is after the bead for E.

The decision to move bead E is discussed later.

[5] There is a call to E in the defintion of B and so the bead for B must be moved to immediately after the bead for E. The definition of B is not called from other definitions and so the order is now correct.

When a call to a group with an existing definition bead is entered in a group definition whose bead also exists, there is always the possibility of the beads being in the wrong order eg. note [4] in fig 4. There are two possible ways of correcting the order of definition beads on the ring. The first method is to move the calling definition ie. the bead for E and the second is to move the called definition bead ie. the bead for C. The latter move would have the solved the problem immediately in our case, as there would be no conflict in the positions of the beads for B and E. However, there are often circumstances when moving the called definition causes problems with definitions that it calls and so checks must be made. To check if a definition contains calls to other groups, all the area beads, all the mask beads and all the shape beads must be checked. However, to check if the calling definition is called from other definitions only the beads on the instance ring are examined, a much faster operation. It is therefore preferable to move the calling definition, in our case the bead for E.

It should be noted that it is only when a group call is entered as part of another group definition that the problems occur. If the group call is part of the main definition, it doesn't matter where the defintion bead is situated. When the definition itself is being entered then its position is not critical until the group calls arrive.

This situation with the beads already on the ring gives a fourth rule to be added to the list:

4) If both definition beads are present when a call to one definition is entered as part of the definition of a second definition, then the bead for the second defintion must be moved so that it is after the first definition. Any definitions calling the second definition must also be moved if necessary.

These rules are incorporated into 'ISKGRP'.

APPENDIX 3: Newton's digitiser coordinate transformation

Newton assumes that because of distortion the paper will appear as an unequal sided quadrilateral when measured with the digitiser. The result is shown in fig 1.

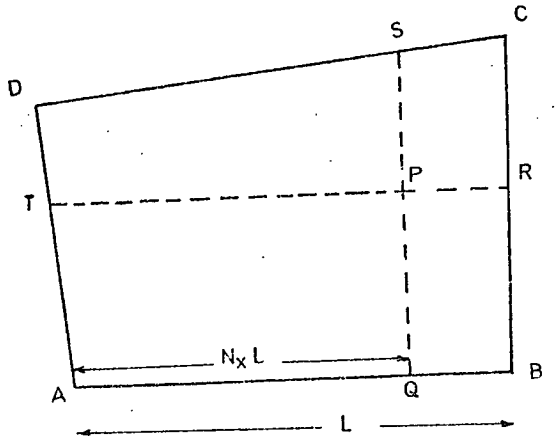


Fig. 1 Distorted paper on digitiser

The digitiser coordinates of the corners of the paper are xa, ya; xb, yb; xc, yc and xd, yd. The digitiser coordinates of the point P are xp, yp. The object of the algorithm is to calculate the paper coordinates of the point P ie. Xp, Yp.

Method

Straight lines QS and TR are drawn on the paper so that they pass through the point P and are parallel to the paper axes ie. QS is always Nx of the paper width away

from the left hand side and TR is always Ny of the paper height away from the bottom of the paper.

Therefore, for point P the equations for Nx and Ny can be shown to be of the form:

$$A.Nx^2 + B.Nx + C = 0$$

and

$$D.Ny^2 + E.Ny + F = 0$$

Where A, B, C, D, E and F are functions of the point P.

These give two roots for Nx and two for Ny; the correct roots are those which cause the point P to lie on the paper.

The digitiser coordinates of Q are xa + Nx.(xb-xa), ya + Ny.(yb-ya) and those for S are xd + Nx.(xc-xd), yd + Ny.(yc-yd)

Now the equation for a line through two general point x1, y1 and x2, y2 is:

$$y.(x2-x1) = x.(y2-y1) + (x2.y1-x1.y2)$$

Therefore the equation of QS is:

$$y.(xq-xs) = x.(yq-ys) + (xq.ys-xs.yq)$$

i.e.

$$y.[xa+Nx.(xb-xa)-xd-Nx.(xc-xd)] = x.[ya+Nx.(yb-ya)-yd-Nx.(yc-yd)] + [(xa+Nx.(xb-xa)).(yd+Nx.(yc-yd)) - (xd+Nx.(xc-xd)).(ya+Nx.(yb-ya))]$$

i.e.

$$y.[xa-xd+Nx.(xb-xa-xc+xd)]= x.[ya-yd+Nx.(-ya+yb-yc+yd)] +[(xa+Nx.(xb-xa)).(yd+Nx.(yc-yd))] -[(xd+Nx.(xc-xd))(ya+Nx.(yb-yd))]$$

This line passes through the point P, therefore

$$yp.[xa-xd+Nx.(xb-xa-xc+xd)]= xp.[ya-yd+Nx.(-ya+yb-yc+yd)] +[(xa+Nx.(xb-xa)).(yd+Nx.(yc-yd))] -[(xd+Nx.(xc-xd)).(ya+Nx.(yb-yd))]$$

Rearranging in terms of Nx^2, Nx, etc. we get

$$Nx^2.[(xb-xa).(yc-yd)-(xc-xd).(yb-ya)] +Nx.[-yp.(-xa+xb-xc+xd)+xp.(-ya+yb-yc+yd) +xa.(yc-yd)+yd.(xb-xa)-xd.(yb-ya)-ya.(xc-xd)] +[xp.(ya-yd)-yp.(xa-xd)+xa.yd-xd.ya] = 0$$

i.e. $A.Nx^2 + B.Nx + C = 0$

where:

$$A = (xb-xa).(yc-yd)-(xc-xd).(yb-ya)$$

$$B = xp.(-ya+yb-yc+yd) + yp.(-xa+xb-xc+xd) + xa.(yc-yd)+yd.(xb-xa)-xd.(yb-ya)-ya.(xc-xd)$$

$$C = xp.(ya-yd) + yp.(xa-xd) + xa.yd-xd.ya$$

This equation can be solved for given values of xp and yp to give values for Nx. These values are used to give values for Xp.

A similar treatment gives the equation for line TR which can be solved to give an equation for Ny of the form:

$$D.Ny^2 + E.Ny + F = 0$$

This can be solved to give values for Ny and hence values for Yp. The values of Xp and Yp chosen are those which cause point P to lie on the paper.

Note that the B, C, E and F have to be calculated for each point digitised as they depend on xp and yp.

APPENDIX 4: Simple digitiser coordinate transformation

The method assumes that the paper distortion is restricted to different scaling in the x and y directions and the paper being fixed to the digitiser at an angle as shown in fig 1.

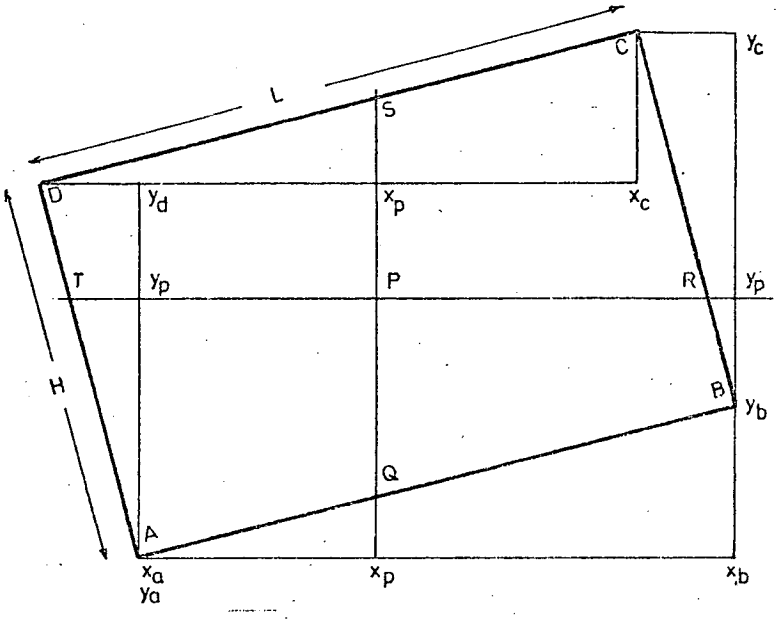


Fig 1 Distorted paper on digitiser

The problem to be solved is the same as before. Given the digitiser coordinates of the corners of the paper ie. xa, ya; xb, yb; xc, yc etc. and the length L and height H of the paper in paper coordinates, find the paper coordinates of a point P, ie. Xp, Yp, whose digitiser coordinates are xp, yp.

Appendix 4

Method

Draw horizontal and vertical lines through P to cut the edges of the paper at Q, R, S and T as shown in fig 1. The paper coordinates of Q, R, S and T, ie. Xq, Yq, Xr etc., are found in terms of the digitiser coordinates of the corners and the point P and the length and height of the paper. The paper coordinates of the point are then calculated from the intersection of the two lines QR and TR.

By similar triangles:

$$(xp-xa)/(xb-xa) = Xq/L$$

ie. $Xq = L.[(xp-xa)/(xb-xa)]$ and $Yq = 0$

and

$$(yp-yb)/(yc-yb) = Yr/H$$

ie. $Xr = L$ and $Yr = H.[(yp-yb)/(yc-yb)]$

and

$$(xp-xd)/(xc-xd) = Xs/L$$

ie. $Xs = L.[(xp-xd)/(xc-xd)]$ and $Ys = H$

and

$$(yp-ya)/(yd-ya) = Yt/H$$

ie. $Xt = 0$ and $Yt = H.[(yp-ya)/(yd-ya)]$

Line QS has the equation:

$$(X-Xq)/(Xs-Xq) = (Y-Yq)/(Ys-Yq)$$

but $Yq = 0$ and $Ys = H$, therefore

Appendix 4

$$(X-Xq)/(Xs-Xq) = Y/H$$

or

$$Yp = H.[(X-Xq)/(Xs-Xq)] \quad - - - - - - - - - \quad (1)$$

Line TR has the equation:

$$(X-Xt)/(Xr-Xt) = (Y-Yt)/(Yr-Yt)$$

but $Xt = 0$ and $Xr = L$, therefore

$$X/L = (Y-Yt)/(Yr-Yt)$$

or

$$X = L.[(Y-Yt)/(Yr-Yt)] \quad - - - - - - - - - \quad (2)$$

Lines QS and TR intersect at point P and therefore substituting (2) into (1) we get:

$$Yp.(Xs-Xq) = H.[L.(Yp-Yt)/(Yr-Yt)-Xq)$$

$$Yp.(Xs-Xq).(Yr-Yt) = H.[L.(Yp-Yt)-Xq.(Yr-Yt)]$$

$$Yp.[(Xs-Xq).(Yr-Yt)-H.L] = -H.[L.Yt-Xq.(Yr-Yt)]$$

$$Y_P = [H.(L.Yt+Xq.(Yr-Yt)]/[H.L-(Xs-Xq).(Yr-Yt)]$$

Similarly substituting (1) into (2) we get:

$$Xp.(Yr-Yt) = L.[H.(Xp-Xq)/(Xs-Xq)-Yt]$$

$$Xp.(Yr-Yt).(Xs-Xq) = L.[H.(Xp-Xq)-Yt.(Xs-Xq)]$$

$$Xp.[(Yr-Yt).(Xs-Xq)-L.H] = -L.[H.Xq+Yt.(Xs-Xq)]$$

$$Xp = [L.(H.Xq+Yt.(Xs-Xq)]/[H.L-(Xs-Xq).(Yr-Yt)]$$

REFERENCES:

1.1 Schoor, H. 'Computer-aided digital design and analysis using a register transfer language' IEEE Trans. Electronic Computers 1964 EC13 pp 730-737

1.2 Kerighan, B.W. and Lin, S. 'An efficient heuristic procedure for partitioning graphs' Bell Syst. Tech. J. 1970 49 pp 291-307

1.3 Hope, A.K. 'Application of interactive computer techniques and graph theory to printed circuit board design' PhD Thesis University of Edinburgh 1973

1.4 Stevenson, F. 'design and simulation of digital systems' Proc. Conf. on C.A.D. Sheffield March 1968

1.5 Kaposi, A. 'Logic testing by simulation' I.E.E. Conf. Publication 51 1969

1.6 Treble, D.P. 'Dimensional checking of MOS LSI layouts' I.E.E. Conf. Publication 86 April 1972

2.1 Bardsley,C.W. 'Computer aids for artwork generation' IEEE Spectrum Sept 1971 pp 64-79

2.2 Fletcher, A. 'The automatic layout of integrated circuit masks' I.E.E. Conf. Publication 51 1969

2.3 Radley, P. 'The automatic layout of electronic circuits by computer' I.E.E. Conf. Publication 86 April 1972

2.4 Rose, N,A. 'computer aided design of printed circuit boards' PhD Thesis University of Edinburgh 1970

2.5 Wood, J. et al 'Computer aided production of masks for silicon integrated circuits' I.E.E. Conf. Publication 51 1969

2.6 Atiya, J. 'The use of graphic display as an aid to integrated circuit mask generation' I.E.E. Conf. Publication 51 1969

2.7 Bird, S 'Myriad Graphics Software' The Marconi Co. Gt. Baddow Essex 1970

2.8 Richardson, F.K. et al 'An interactive graphical system for the design of photomasks' Proc N.E. Electron. Res. and Eng. Nov 1970 pp 182-183

3.1 Eades, J.D. 'GAELIC user's manual' Wolfson Microelectronics Liaison Unit, University of Edinburgh 1974

3.2 Wood, J. et al 'Computer aided production of masks for silicon integrated circuits' I.E.E. Conf. Publication 51 1969

4.1 Knuth, D.E. 'The art of computer programming in information systems' Voll Addison Wesley 1968

4.2 Dodd, G. 'Elements of data management systems' Comp. Surv. 1,2 June 1969 pp 115-135

4.3 Williams, R. 'A survey of data structures for computer graphic systems' Comp. Surv. 3 March 1971 pp 1-22

4.4 Morris, R. 'Scatter storage techniques' Comm. ACM 11 1 Jan 1968 pp 38-44

4.5 Feldman, J.A. and Rovner, P.D. 'An Algol based associative language' Comm. ACM 12 8 Aug 1969 pp 439-449

4.6 Sutherland, I.E. 'SKETCHPAD' Tech Report 296 Lincoln Labs M.I.T. Jan 1963

4.7 Evans, D.S. and Katzenelson, J. 'Data structure and man machine communication for network problems' Proc IEEE 55 7 Jul 1967 pp 1135-1144

4.8 McGuffin, R. et al 'Computer-aided placement and routing of high density chip interconnection systems' AGARD Conf. Proc. 130 May 1973

4.9 Childs, D.L. 'Description of a set theoretic data structure' Proc. AFIPS 1968 FJCC Vol 33 pp 557-564

4.10 Wood, J. et al 'Computer aided production of masks for silicon integrated circuits' I.E.E. Conf. Publication 51 1969

4.11 Bird, S 'Myriad Graphics Software' The Marconi Co. Gt. Baddow Essex 1970

4.12 Reilly, P.F,A. 'The automatic design of photomasks' PhD Thesis University of Edinburgh 1975

5.1 Hubbold, R.J. 'Software paging of list data structures for interactice engineering design' I.E.E. Conf. Publication 86 April 1972

7.1 Wood, J. et al 'Computer aided production of masks for silicon integrated circuits' I.E.E. Conf. Publication 51 1969

7.2 Eades, J.D. 'GAELIC user's manual' Wolfson Microelectronics Liaison Unit, University of Edinburgh 1974

7.3 Eades, J.D. 'GAELIC system manual' Wolfson Microelectronics Liaison Unit, University of Edinburgh 1976

8.1 Eades, J.D. 'Application of GAELIC to the design of a large scale integrated circuit' I.E.E. Conf. Publication 111 April 1974