THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

Techniques for Estimation of the Area of Integrated Digital Circuits

Wen-Tai Liu

CRL-TR-2-83

JANUARY 1983

Computing Research Laboratory
Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

---

[1]All correspondence should be sent to Professor Daniel E. Atkins. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the funding agencies.

TECHNIQUES FOR ESTIMATION OF THE AREA OF INTEGRATED DIGITAL CIRCUITS

by
Wen-tai Liu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer, Information, and Control Engineering)
in The University of Michigan
1983

Doctoral Committee:

Professor Daniel Atkins, Chairman
Professor Keki Irani
Professor Kensall Wise
Assistant Professor Trevor Mudge

ABSTRACT

TECHNIQUES FOR ESTIMATION OF THE AREA
OF INTEGRATED DIGITAL CIRCUITS

by
Wen-tai Liu

Chairman: Daniel E. Atkins

This dissertation describes a sub-system of an Arithmetic Design System (ADS) which is intended to estimate figures of merit for a particular arithmetic design at the VLSI technology realization level. We emphasize the gate array, the programmable logic array (PLA) and the general cell approach. The fact that the design processes are so time consuming motivates us to develop estimation methods for figures of merit (*i.e.* area) resulting from design characteristics. These estimates offer advice which avoids wasting design time to achieve only a small amount of improvement. We develop the estimation methods for the gate array and the PLA approaches. In both cases, Rent's relationship represents the design characteristics.

In the gate array approach, we propose a point model for the layout study and show both the final area and the average interconnection length increase as Rent's exponent $\mathbf{r}$ increases. In the PLA approach, based on a partition model for the folding process, the saved area ratio δ is shown to be bounded by $\frac{1}{2^{\frac{1}{r}}} \leq \delta \leq \frac{1}{2}$.

We propose a constructive layout approach for the general cell model, which is needed to map the hierarchical design description into physical structures. The routability problem and channel router are particularly discussed in detail. The minimum condition set for the routability test and channel routing order generation algorithm are derived by using a graph model to represent all the legal routing orders. The routing process can be entirely completed by sequentially applying a channel router to each channel according to the routing order. The new channel router consists of two graphs, namely the interval graph G_I and the constraint graph G_C, which represent respectively the overlap and constraint relations among the nets. The reduced graph G_r is formed by removing the constraint edges from the complement of G_I. The router chooses the best candidate from the dynamically updates G_r.

DEDICATION

To My Family

ACKNOWLEDGMENTS

I wish to express my greatest gratitude to Professor Daniel Atkins for his understanding, encouragement, help, and guidance throughout my graduate study at Michigan. I am also grateful to Professors K. Irani, T. Mudge, and K. Wise for serving on my committee. Special thanks are due to Professor Mudge who gave me great support to ease the culture shock in the early years of my graduate study.

I also wish to thank my wife and my boys who share the hardship during the period of great uncertainty in our life. I bear in mind for their understandings and sacrifices. Meanwhile, I owe my parents and my family very much for their sacrifices and spiritual support. The hardship and persistent struggle to make a living in an extremely difficult environment deeply affects me. Without that kind of persistence, it is impossible for the son of an illiterate farmer in a remote country to have an education at Michigan.

Finally, I would like to thank my friends in the Computing Research Laboratory. The financial support through the NSF grant MCS-800-7298 is greatly appreciated.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

LIST OF APPENDICES

APPENDIX

CHAPTER 1

INTRODUCTION

This thesis describes a sub-system of the Arithmetic Design System (ADS). The main purpose of the sub-system is to provide tools for estimating the figures of merit for a particular arithmetic design at the realization level. The Arithmetic Design System can be described at three levels , namely *the application level*, *the arithmetic design level*, and *the realization level*. At the realization level, we will focus on VLSI technology. Area and time complexity are the important figures of merit to be considered in VLSI design.

The research is concerned with obtaining estimates of the silicon area required for the layout of a given logic circuit. The goal is to account for both device and the interconnection areas and to produce the estimate with respect to a reasonable layout (placement and routing) procedure.

In addition, we are concerned with finding estimates of propagation delay. A typical assumption in the current literature is that minimization of the average interconnection length tends to minimize the total layout area. We will derive quantitative relationships between layout area and interconnection length. For the "point model" in this thesis, our analysis shows that the assumption is true.

How do we devise a method for estimating the area or the interconnection length of a design ? In fact, there are numerous design spaces as well as design methodologies. It is impossible to devise an estimation method spanning all the design spaces and design methodologies. For example, the logic description at the realization level can be a computation graph of the logic devices and/or a hierarchical partition description. On the other hand, gate array, polycell, and

general cell approaches are viable schemes for the design of complex VLSI and have the potential to be automated. Since we can map a computation graph of the logic devices into a gate array scheme, and a hierarchical description into general cell scheme, we restrict ourselves in this study to developing the necessary estimation *tools* for the gate array and general cell approaches. Keeping this in mind, we develop two models namely a *point model* and a *rectangle model*. The point model represents the gate array approach, while the rectangle model represents the general cell approach.

The two approaches require different sophisticated layout processes. In general, layout is a process of mapping the logical structure into the physical structure of the design. Placement and routing are two aspects of the layout. They interact strongly. The way in which the modules are placed may dramatically affect the quality and the routability of the routing process. The global nature of even a single aspect of the layout can make the problem difficult. For example, the way a single net is routed affects the potential solutions for the others. Ideally, for each design approach, there are two extremes to choose from:

(1) enumerating the routing process associated with each given placement and choosing the best or

(2) accurately estimating the area required for routing in a placement without doing the complete routing.

However, in the current literature, the interaction between placement and routing processes is poorly understood. The two extremes seem to be unrealistic. We offer a constructive approach ,namely, to develop efficient algorithms to give figures of merit for the layout process.

In regard to the general cell approach, a hierarchical description is partitioned and then translated into a network of Programmable Logic Arrays (PLAs)

because PLAs have been used frequently to implement both control logic and data path. There are, of course, problems of partitioning and placing the PLAs. The partition problem is very difficult and has been studied by other researchers. Hopefully, we can get insights into partitions through techniques developed in the thesis. At present we assume that a partition into PLAs is given, and we concentrate on the related placement and routing problems.

Chapter 2 gives a literature review in which we emphasize the development of the gate array approach and the general cell approach. The fact that the layout processes are time consuming motivates a *thesis philosophy*. Keeping the thesis philosophy in mind, we discuss the results associated with gate array, PLA, and general cell layout methods in the following chapters.

Chapter 3 describes a gate array model and a rectangle model. Our gate array model consists of an array of lattice points and has only one unit width for both vertical and horizontal channels. The circuits are placed at the set of lattice points while the connections are mapped to the channels by a placement and routing procedure. We also define the area to be the smallest rectangle enclosing the lattice points. The analysis shows that both the area and average length are functions of Rent's exponent of the circuit. The results indicate that the interconnections contribute a significant part to the layout area. We also define the circuit elements and channels associated with the rectangle (general cell) model. The problems associated with the *general cell* model are then formally defined. We also survey the related literature for the general cell model. Finally, a layout approach for a general cell model is proposed. In this approach, we discuss the subproblems and propose a strategy for them.

Chapter 4 rigorously treats the routability problem in the rectangle model in terms of graph theory. The goal of the placement procedure is to avoid creating unroutable channels and to minimize the circuit area. In this chapter, a suf-

ficient and necessary condition for testing the routability of a placement is given. Under this condition, we compare with and criticize previous published methods. Our method is the most complete and efficient method as shown in section 4.4. In addition, an algorithm for testing routability and generating the legal routing orders is proposed. At the end, we propose a method for the generation of the routing orders in the unroutable layout by breaking some channels in the layout.

Chapter 5 describes the problems which affect the final area. Among them are the *displacement problem*, *topology net assignments*, the **T-shape router**, the *cross channel router*, and the *channel routing problem*. In particular, we discuss and formulate the channel routing problem in detail. The problems are either with cyclic constraints or without cyclic constraints. By defining an *eligible set*, we are able to discuss a class of channel routing algorithms. Graph theoretic techniques are used to develop a new heuristic channel routing algorithm. For the case without the cyclic constraints, our algorithm takes $O(n^3)$ steps, where n is the number of connections in the channel. Our algorithm is compared with the previous algorithm [1], and gets much better results. Possible extensions of our algorithm are mentioned too. Some theoretical limitations for the channel routing problems with cyclic constraints are also discussed. At the end of the chapter, we study the algorithms for the **T** shape router and *cross channel* router.

Since PLA is a basic module at the realization level, techniques to reduce the area should be addressed. Chapter 6 discusses those techniques. Most of them are very complicated and time consuming. In particular, we develop a model for predicting the saved area ratio without executing the complex folding algorithm. Our results are validated by folding the PLA implementation of some arithmetic functions. In the end, we pose an open problem related to logic minimization and the folding process.

Chapter 7 summaries the major contributions of the thesis. Further improvements related to this work are also mentioned.

CHAPTER 2

LITERATURE REVIEW

2.1. Introduction

In recent years, the figure of merit for a digital design has been shifted from traditional *gate counts* [2] toward the *final area* for the implementation in *very large scale integrated circuits* [3]. The major difference between these two figures of merit is that in VLSI implementations, the interconnections contribute a significant portion of the *final area*. It is not uncommon for the area for the interconnections to contribute more than half of the final circuit area [4]. In this thesis, the figure of merit is defined as the *final area* of a digital design. In other words, the design is discussed in the context of very large scale integrated circuits.

Advances in VLSI technology has made it possible to incorporate a large digital system on a single chip. Both circuit complexity (*i.e.* the number of gates and their connections) and technology constraints (*i.e.* design rules as well as available layers) make the design of VLSI a complicated process, frequently beyond the management capacity of humans. In order to reduce design complexity, researchers have proposed different methodologies for VLSI design. These are the manual layout approach, grid symbolic layout, non-grid symbolic layout, Programmable Logic Array (PLA) approach, gate array approach, standard cell approach, general cell approach, silicon compiler, and hierarchical structured design, etc. In the next section, we will briefly survey these methodologies.

2.2. Layout Approach

A layout process is a process that maps the logical devices and their connections to the physical structures and their corresponding connections. This generally includes *placement* and *routing* processes. In the following section, we distinguish the methodologies according to whether they have the ability for automatic placement and routing.

2.2.1. Non-automatic Approach

2.2.1.1. Manual Approach

This is the most straightforward approach. In this approach, the designer manually finishes the entire design by taking care of logic partitions, chip planing, shape management of the circuit components, and routing. In the process, the designer usually uses the interactive graphic system to carry out the layout in detail. A post-processor is used to check design consistency such as design rule check and performance evaluation. The natural limitation of this approach is the extent to which it is humanly manageable. This depends not only the complexity of the design but also on the designer's experience. While this method usually produces a compact layout, it is very time consuming.

Since the designer needs to specify every component and interconnection in the manual approach, the designer manages the shapes of the components and then puts them together to get a compact layout. One way to reduce the burden of the designer is to use the *symbolic layout* method. In this method, the designer only inputs a shorthand notation for each component, interconnection, via etc., according to some layout configuration. Two kinds of *symbolic layout* methods are discussed in the following:

2.2.1.2. Grid Symbolic Layout

The method divides the plan into equally spaced grids. Then the symbols are placed into the gridded plan. A program transforms the symbols into their real geometric structure [5] , [6]. In this method, the space between the grids depends on the design rule governing the technology. Usually, there are many layers with different minimum legal spaces. A biggest minimum layer space is chosen for the common grid space such that the design does not violate the design rules. Under this restriction, some portion of the layout may be sparse. To removed this disadvantage, a *non-grid* method is proposed. The quality of the final layout in this method still depends on the designer's ability to input a dense symbolic layout.

2.2.1.3. Non-grid Symbolic Layout

In this method, the designer only specifies the relative placement and interconnections of the symbols in the entire circuits. For example, in the stick diagram system [7], the line represents an interconnection, while the intersections of lines represent devices. In general, this method allows the designers to do topology planning. It then requires a set of compaction and expansion algorithms to transform the symbols into a final layout. Different non-grid based symbolic layout systems have been implemented [8] , [9] , [10] , [11].

2.2.2. Automatic Approach

In the next section, we discuss the methods that can be automated in both placement and routing process. In general, the methods have regular structures which are good for automatic processing.

2.2.2.1. Programmable Logic Array

Figure 2.1 shows a PLA floor plan in which input and their complement lines run vertically through the AND-plan, while the product terms are represented by

horizontal lines run through both AND and OR plans. The output lines run through OR plan vertically by connecting each output line to a set of selective product terms. Figure 2.2 shows the electric circuit model.

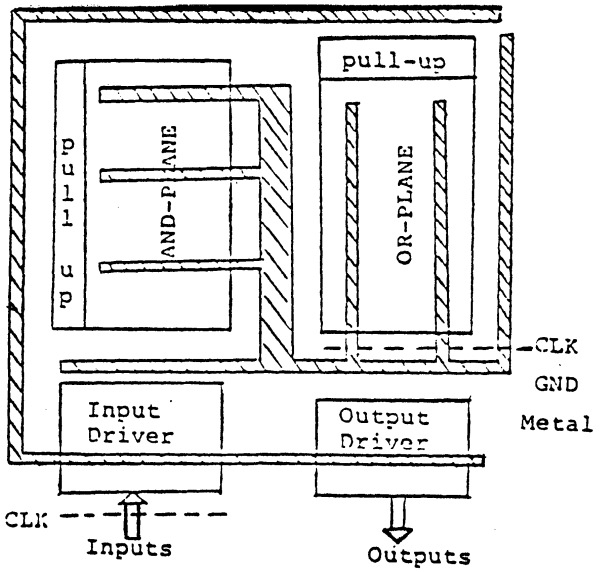


Figure 2.1 A PLA Floor Plan

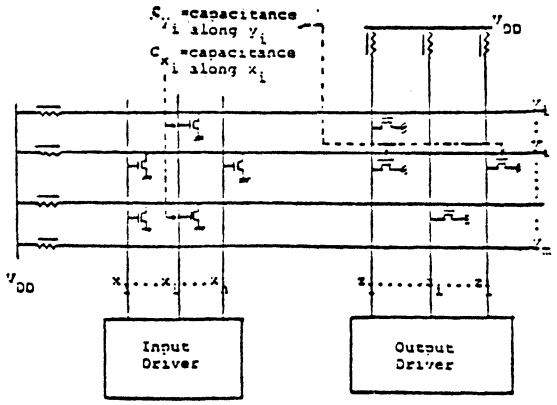


Figure 2.2 The Electric Circuit Model for PLA

Programmable Logic Arrays (PLAs) are used frequently to implement control logic, and are increasingly being considered as an implementation mode for the datapath [12] and advocated as a viable methodology for VLSI designers [13]. Although PLAs are generally considered to be wasteful of silicon area in comparison to random logic, they offer a conceptual and structural regularity which is quite attractive for VLSI design. Optimization to reduce area and delay

can be automatically applied to the structure in a manner which is transparent to the logic designer.

Conventional PLA designs are characterized by three factors: number of inputs, number of outputs, and number of product terms. Statistical results [14] show that typical personalization densities for the conventional AND and OR arrays are 10 percent and 4 percent respectively. Such sparse densities together with long delay time are two major disadvantages of PLA-based design styles but numerous schemes have been proposed to improve the area efficiency and performance of a PLA. These include the two bit decoder [15], phase selection [15], logic minimization [15], and PLA folding [14]. We will discuss techniques in the later sections. Some of the techniques such as logic minimization and phase selection have been developed over a long time. On the other hand techniques such as the two bit decoder and folding were developed following progress in VLSI technology.

2.2.2.2. Gate Array (Masterslice) Approach

In this method, some basic circuit components are placed and prefabricated on pre-defined slots arranged in rectangular arrays, and are separated from each other by areas (vertical and horizontal channels) through which interconnection wires may be run. A classical random logic design can be implemented automatically in a gate array chip by using placement and routing procedures. Automatic algorithms for both placement and routing processes are available [16]-[19]. One paper [20] reports that the placement time is almost linear with the number of circuits to be placed. A major problem associated with this approach is how much space should be reserved for the wiring in both vertical and horizontal channels. Several researchers have worked on this problem [21], [22]. Figure 2.3 shows a schematic diagram for a gate array.

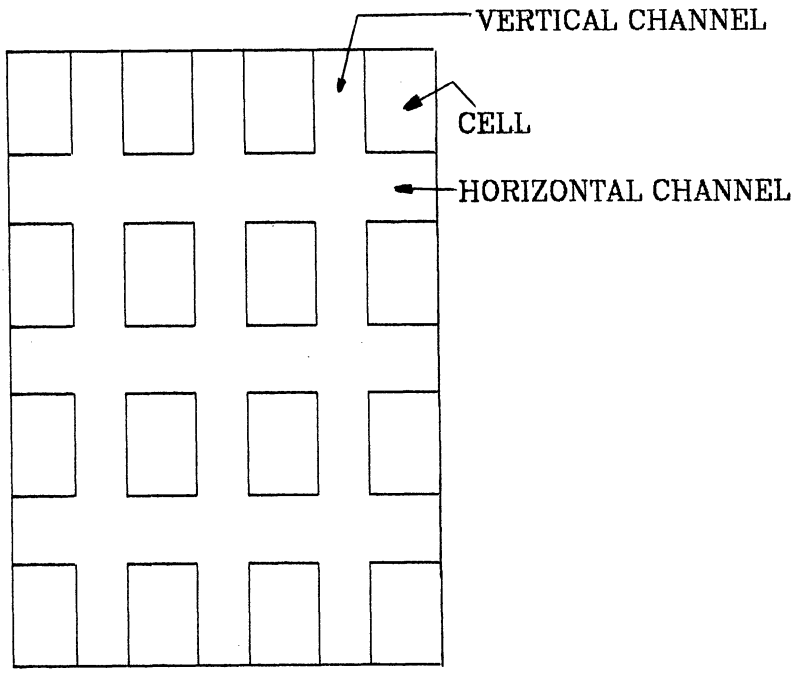


Figure 2.3 A Schematic Diagram for a Gate Array

2.2.2.3. Standard Cell (Polycell) Approach

A schematic diagram of the standard cell approach is shown in Figure 2.4. The chip plan has been divided into several rows separated by a set of horizontal channels. Each row contains several standard cells. Each cell in a row has the same height but variable width. The cell represents a group of logic gates or a function unit. In this method, a placement procedure assigns the logic gates to the standard cells. A routing procedure then performs the interconnections by connecting them through the horizontal channels and some feed-through cells. Reserving enough space for routing in the horizontal channels is also an important problem. Both placement and routing procedures have been intensively

studied and can be fully automatic in a successful system [23].

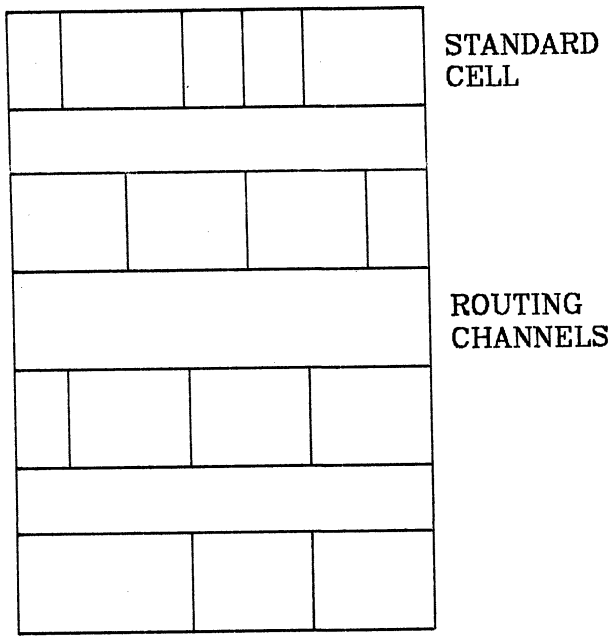


Figure 2.4 A Schematic Diagram for the Standard Cell Approach

2.2.2.4. General Cell Approach

A *general cell* is a cell with no restriction on its height and width, and no restriction on the location of its input-output pins. Building blocks like the **PLA** and **ROM** are general cells. Since the complexity of VLSI circuit is increasing rapidly, the general cell approach is viable for designing complicated circuits [24 , 25] , [26] , [27]-[29] , [30] , [31], [32] , [33]. The general cell approach to layout consists of a placement procedure and a routing procedure. The placement procedure gives the relative positions among the building blocks and produces a rough layout. Before doing a final routing procedure we ask the question "*Is the rough layout routable*?" The problems associated with this approach are numerous, and suffer from being neither rigorously treated nor intensively studied. Since the *general cell* approach is an important model for developing esti-

mate techniques, in section 3.3.3, we will in section 3.3.3. intensively survey the problems associated with the general cell approach.

2.2.2.5. Silicon Compiler

The silicon compiler is a process to translate a design description into a layout. *Bristle blocks* [34] is a typical silicon compiler. It is a system for constructing a microprocessor data path. The data path consists of functional elements strung along two horizontal busses. Functional elements are the shifter, register, I/O ports, ALU etc. A functional unit consists of a group of basic cells stacked vertically. On the bottom, a decoder decodes the externally supplied microcodes and generates the necessary symbols to control functional elements. Since the cells are parameterized, the *bristle blocks* can generate the layout of the desired data path by specifying the width of a data path, the kinds of cells, and the microcodes. One special feature associated with *bristle blocks* is the *stretching* ability, and hence connecting by *abutment*, of the functional elements. Small elements are stretched to make connections, lengthen the vertical wires inside the elements as necessary to reach the horizontal bus. Since the floor plan has been fixed, this approach has limitations for general digital design.

2.2.2.6. Hierarchical Structured Design

This design approach is advocated by Mead [3]. The design style emphasizes the consideration of overall context. The design is carried out in several hierarchical levels by choosing the most suitable layout scheme on each level. In this method, small modules are fitted into large modules. Placement is predetermined by overall consideration and routing is done through the abutment of modules.

2.3. Layout Approach Used in This Thesis

The common goal of the various layout methods is to map the logical devices and their connections into physical structures such that the final area is as small as possible. A survey of the method reveals two facts: 1) each method has its own limitations; 2) each layout method is very time consuming. Each method has been decomposed into several subprocesses. Again, each subprocess is also time consuming. This motivates the following thesis philosophy:

Thesis Philosophy:

Given a logic design, can we estimate the figures of merit (average or the worst case) resulting from the process or the subprocess by extracting some characteristics of the design ? If the answer is positive, we propose some model and do the estimations. If the answer is negative, we use a constructive approach by developing a set of efficient algorithms to estimate or even exactly compute the desired figures of merit.

Since the processes or subprocesses are time consuming, not every design instance deserves going through the processes. The first part of the philosophy has the merit of advising the designer to avoid wasting time by just getting a small amount of improvement. In this thesis, the results relating to the *gate array* and *PLA* follow this philosophy. The nature of the hierarchical descriptions in the ADS cannot fit the gate array and PLA layout approach. We need a general model to map the design descriptions into physical structures. This motivates us to develop the *rectangle model* which is the same as the *general cell* approach. Since the general cell approach is a very complicated process, this motivates us to use the *constructive approach* which follows the second part of the philosophy. In summary, the thesis investigates the techniques for reaching the goal outlined in Chapter 1 under the thesis philosophy in this section. We discuss the problems associated with the gate array, PLA, and general cell lay-

out methods. Detailed discussion of our work is given in each chapters.

CHAPTER 3

MODELS AND CONSTRAINTS

3.1. Introduction

This research is concerned with obtaining estimates of the silicon area required for the layout of a given logic circuit. The goal is to account for both the device and the interconnection areas and to produce the estimate with respect to a reasonable layout (placement and routing) procedure.

Moreover, we are concerned with finding estimates of propagation delay. A typical assumption in the current literature is that minimizing the average interconnection length tends to minimize the total layout area. For the point model in this thesis, our analysis shows that the assumption is true.

In this thesis we study the problem under two particular models, namely a Point Model and a Rectangular Model. They represent two viable schemes for designing complex VLSI. The point model represents the gate array approach, while the rectangle model represents the general cell approach.

3.2. Point Model and Computation Graph

A computation graph is an indirect graph which represents the logical relations among the circuit modules. In the graph, the nodes represent the circuit modules, while the edges represent the logical relations. Figure 3.1 is a computation graph, where we think of both AND and OR gates as nodes. The point model represents the physical layout of the computation graph. The model shown in Figure 3.2 consists of an array of lattice points. Each lattice point is the intersection of vertical and horizontal grids. The circuit modules are mapped into the set of the lattice points. The interspace between two

consecutive grids is for the interconnection. In order to make the analysis of the layout area and the routing length tractable, we allow only one unit width in each interspace in our model. The design problem becomes that of finding an efficient method to map the computation graph into the point model and then to predict the figures of merit associated with the proposed mapping method. Both Donath [35] and Feuer [36] derived the average routing length of a circuit layout as a function of Rent's exponent of the circuit without the unit width restriction. In other words, their results are normalized by the channel width, and hence are not exactly the real length. Moreover there are deficiencies in their results, which will be discussed in section 3.2.5.

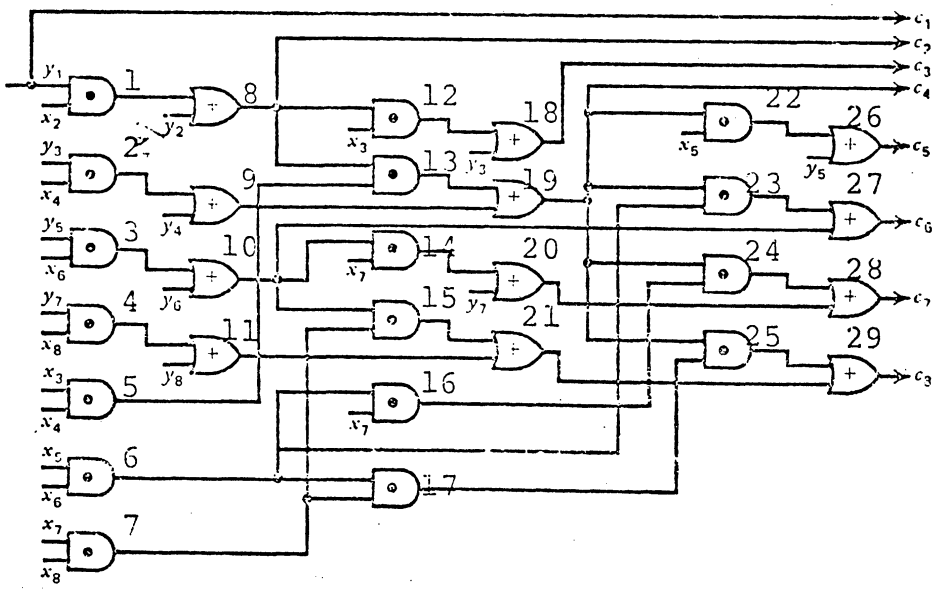


Figure 3.1 Eight Bit Carry Look-ahead Generator.

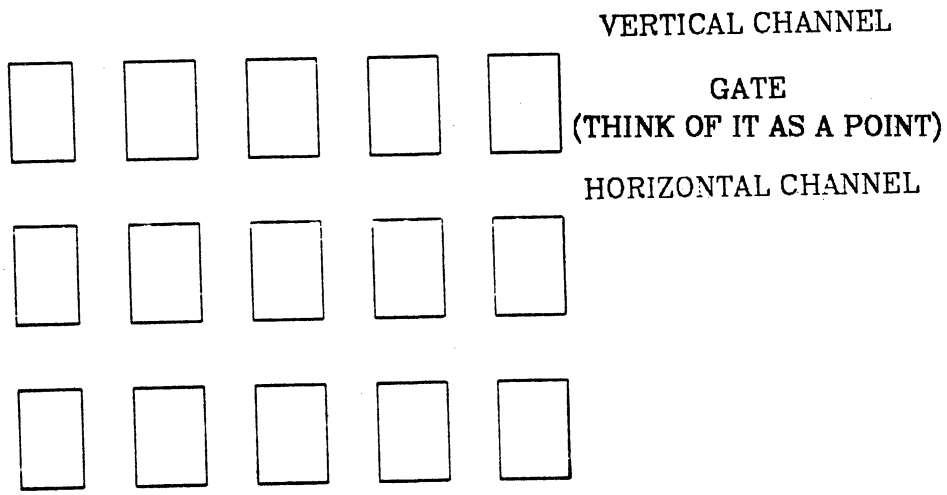


Figure 3.2 A Point Model

In accomplishing the mapping of the computation graph into the point model, we propose to apply the "divide-and-conquer" technique to the layout strategy. Based on this technique and its implied Rent's relationship, we can derive the final area and average routing length for a circuit. Divide-and-conquer techniques, widely used in tackling complex problems are being studied for their application to the IC layout problem. The use of divide and conquer techniques implies the need for an algorithm (for example, a Min-Cut algorithm) to partition the computation graph and a procedure to reconnect the parts under some specified criteria.

Rent's rule [37] provides a relationship between the number of terminals and number of gates in a computation graph (logic network). It appears to be useful in deriving estimates of the number of interconnections required between subnets of a partition logic net (computation graph). This information together with a layout strategy provides estimates of the circuit area and the average routing length. The analysis of the point model are based on the following assumptions:

(1) Rent's relationship holds in every level of the partitions.

(2) The features of the point model are useful.

(3) The layout strategy is efficient.

3.2.1. Area and Routing Length

We define the area to be the smallest rectangle which encloses the set of the mapped lattice points.

The routing length for two connected modules is the manhattan distance between two objects connected together.

3.2.2. Layout Strategy

Since we are interested in the approach to partition the computation graph and to reconnect the parts under the partitions as shown in Figure 3.3, we are looking for some layout procedure that uses "divide-and-conquer" techniques.

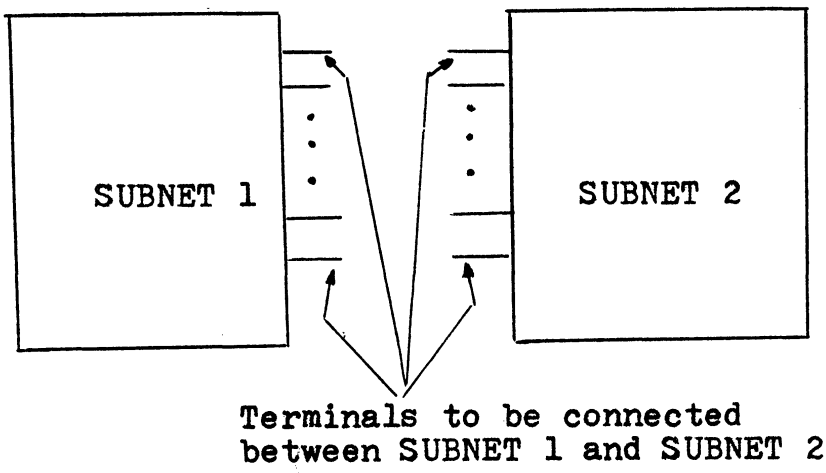


Figure 3.3 A Partition Module.

Ullman [38] and Leiserson [39] propose a routing method to connect two modules shown as Figure 3.4. The method is to add at most two channels in either directions such that the required interconnections can be embedded into the added channels. In this method, we have two routing layers for the connect-

ing channels. One of them is for the horizontal channels, the other is for the vertical channels.

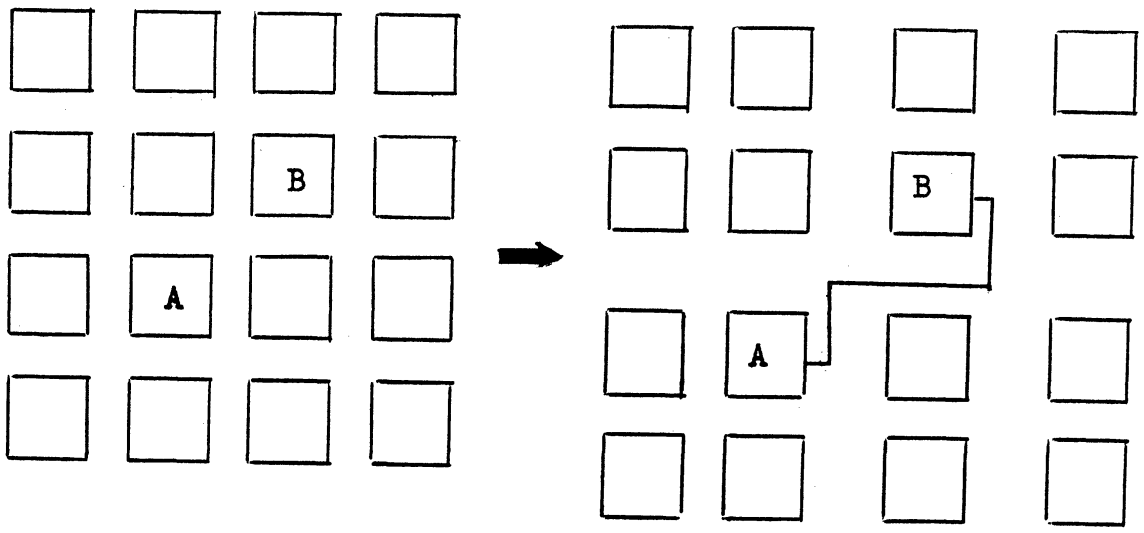


Figure 3.4 The Connecting Method.

Suppose we put the left module at the lower left corner and right module at the upper right corner as arranged in Figure 3.4. Table 3.1 shows the required number of channels to be added in order to connect two modules. Here the north, south, east, west denote the side of the connecting points in a module.

Left Module	Right Module	Horizontal Channels	Vertical Channels
east	east	1	2
east	west	1	2
east	north	1	1
east	south	1	1
west	east	1	2
west	west	1	2
west	north	1	1
west	south	1	1
north	east	1	1
north	west	1	1
north	north	2	1
north	south	2	1
south	east	1	1
south	west	1	1
south	north	2	1
south	south	2	1

Table 3.1 The Required Minimal Added Channels

3.2.3. Rent's Relationship

The partition of a logic network produces a collection of subnets. Is there any relationship between the number of terminals and the number of gates (in general, they can be a block of gates, a chip, etc.) contained in each subnet? Rent's rule [37] (Dr. Rent of IBM first discovered the existence of the relationship) provides a relationship between them. The relationship had been observed experimentally by several authors before a serious study was conducted by Landman and Russo [37]. It has been also derived theoretically by Donath [40] using a stochastic model.

Rent's rule is

$$T=KC^{r}$$

where

(1) $0 \le r \le 1$

(2) T is the average number of terminals of a subnet containing an average of C gates,

(3) K is the average number of terminals per gate in the subnet, and

(4) r is a constant related to the structure and partition schemes of a given logic network.

Example 3.1:

Given: An 8 bit carry look-ahead generator shown in Figure 3.1 taken from [2] , and a heuristic Min-Cut partition algorithm [41] (the Min-Cut algorithm minimizes the connected terminals in the partition. The algorithm has been implemented in APL as shown in *APPENDIX I*).

Find: Rent's relationship for the logic network shown in Figure 3.1.

T	C
33	29
21	16
17	13
10	8
13	8
11	8
11	5
7	4
8	4
8	4
10	4
8	4
8	4
7	3
4	2
4	2
4	2
4	2
5	2
5	2
6	2
4	2
5	2
5	2
5	2
4	2
4	2

Table 3.2 Terminals and Gates Count in the Min-Cut Algorithm.

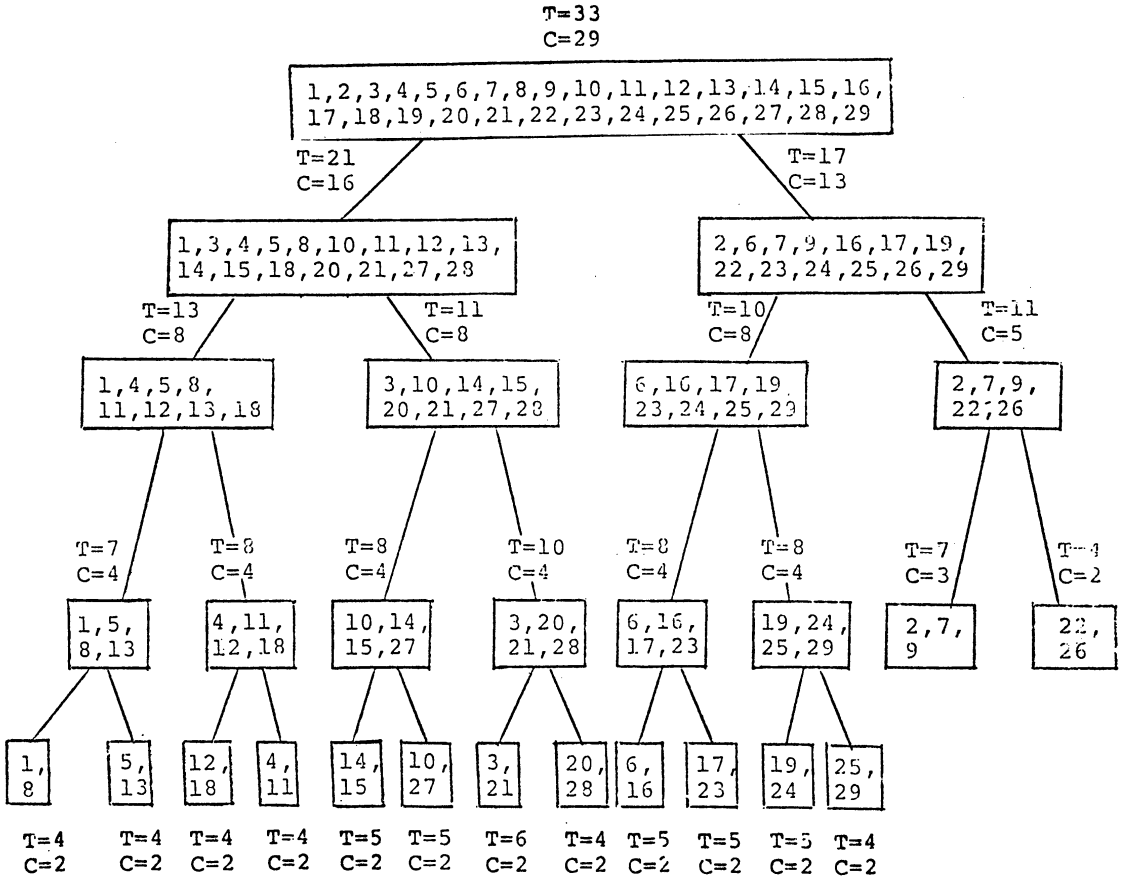


Figure 3.5 The Partitions of a Carry Look-ahead Generator by the Min-Cut Algorithm

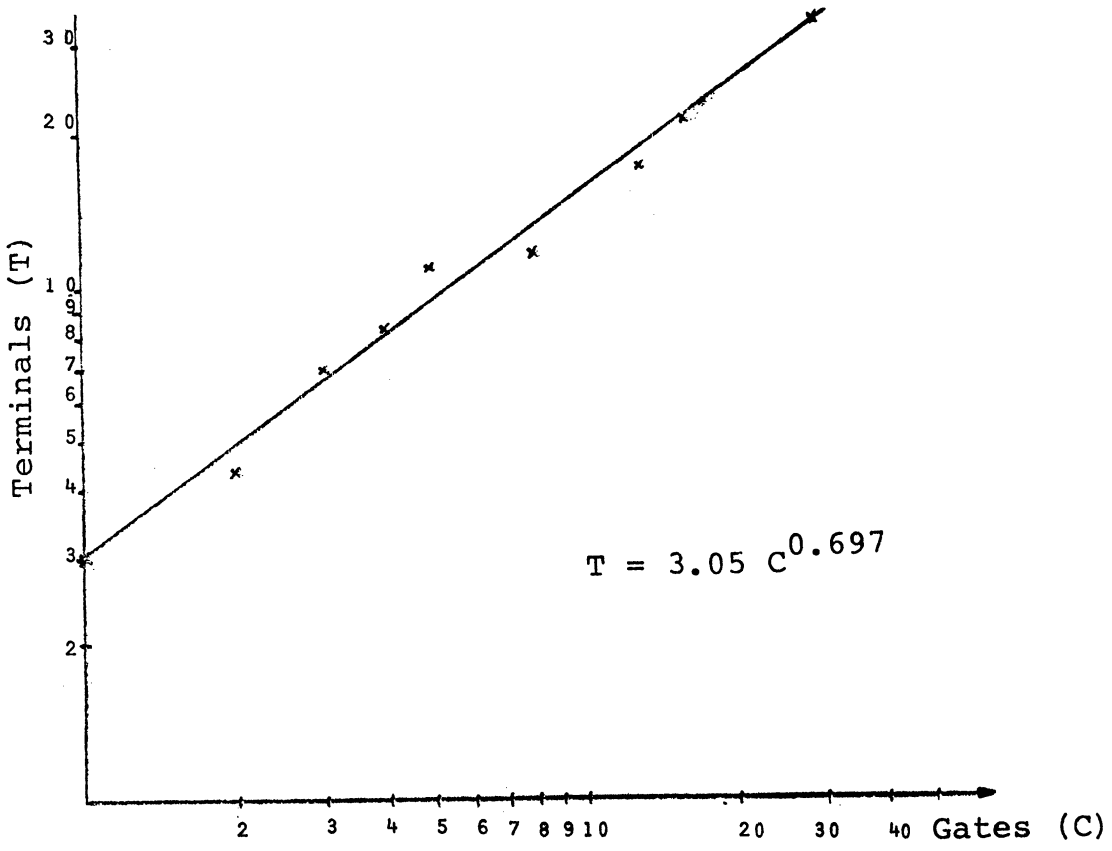


Figure 3.6 The Least-square-error Line for Table 3.2.

The procedure is as follows:

(1) Obtain the set of subnets shown in Figure 3.5 by recursively applying the Min-Cut algorithm.

(2) Find the average terminals among the rows which have same number of gates C in Table 3.2.

(3) Plot the terminals vs gates counts shown in Figure 3.6.

(4) Draw a line with the least-square-error.

Finally, we get the Rent's relationship

$T=3.053C^{0.6966}$

Rent's exponent, r, depends on both the partition algorithm and the structure of the logic network. If we fix the partition algorithm, the variations in **r** can be attributed to the structure of the logic network. Russo [42] shows that high performance networks (those which reduce the number of gate delays) have large Rent's exponents **r**. On the other hand, if we fix the logic network, **r** can be attributed to variations in the partition algorithms, and serves as an index of the relative "figure of merit" of various partition algorithms. The next example shows that a different Rent's relationship is obtained if the Min-Cut algorithm is replaced by another partition algorithm. In this case the partitioning is done by human inspection with an attempt to minimize the number of terminals of subnets.

Example 3.2:

Given: An 8 bit carry look-ahead generator (Figure 3.1), and the partition shown in Figure 3.7.

Find: Rent's relationship of the generator.

Table 3.3 shows the relationship between the average terminals and the gates in subnets. Figure 3.8 is a plot of Table 3.3. From Figure 3.7 and Figure 3.8, we get a Rent's relationship

$T=3.54C^{0.7147}$

T	C
33	29
21	16
28	13
16	8
22	8
14	7
13	6
10	4
12	4
9	4
12	4
9	4
9	3
9	3
6	3

Table 3.3. Terminals and Gates Count in the Intuitive Partition.

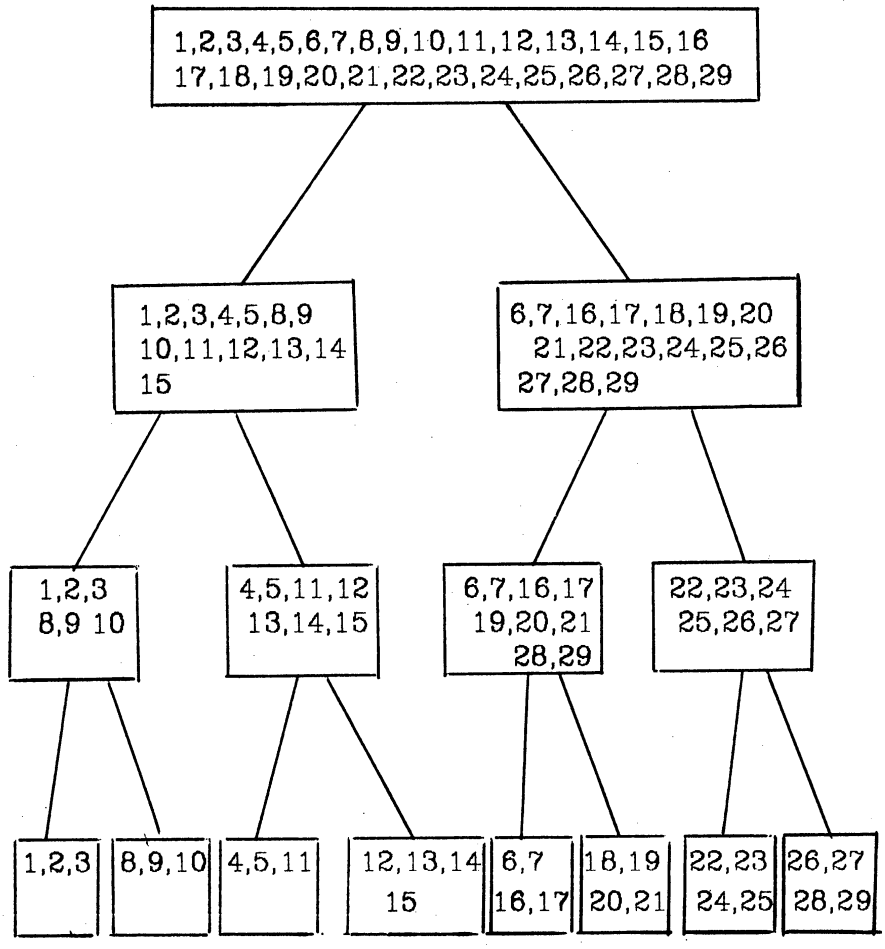


Figure 3.7 Intuitive Partition of the Generator.

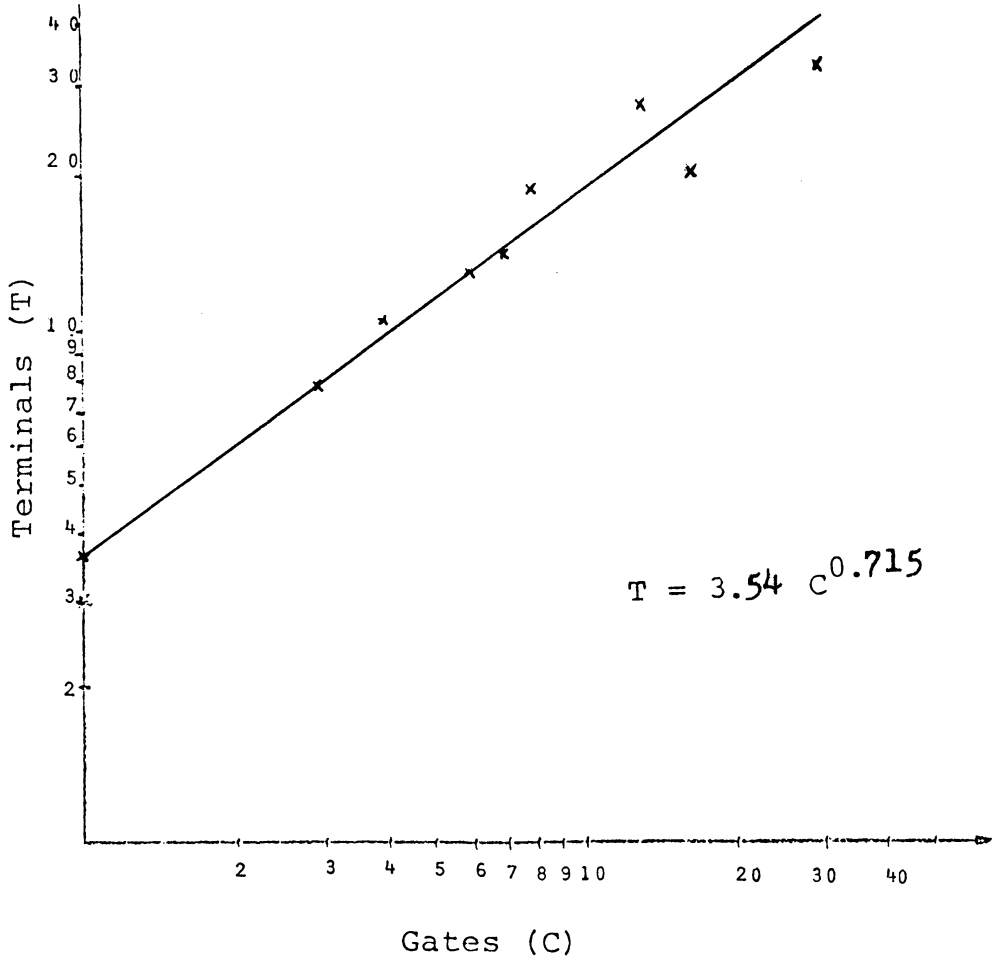


Figure 3.8 The Least-square-error Line for Table 3.3.

Both partitions produce big Rent's exponents since the circuit is a high performance carry look ahead generator which, as predicted, should have a big Rent's exponent. Moreover, the Min-Cut algorithm produces a smaller Rent's exponent than the intuitive partition. One might infer that the Min-Cut algorithm always produces a smaller Rent's exponent than any other partition algorithms. Unfortunately the following theorem shows that the inference is not true.

Let

$t=\log T$

$k=\log K$

$c=\log C$

Since $T=KC^r$ then $t=k+rc$. And since the least-square-error fit is used, the linear regression model defines the parameter:

$$r_i=\frac{\sum_{j=1}^{N}(t_i^j-\bar{t}_i)(c_i^j-\bar{c}_i)}{\sum_{j=1}^{N}(c_i^j-\bar{c}_i)^2}$$

$$k_i=\bar{t}_i-r_i\bar{c}_i$$

Where N is the number of the terminal-gate pair in the plot, i means ith partition algorithm which generates N data sets, $\bar{c}_i(\bar{t}_i)$ means the average number of gates (terminals) in ith partition algorithm.

In particular, we are interested in the case that the algorithm divides the whole logic network into two halves and the effect of shuffling the gates between two halves. Therefore, we make the restriction that $C_1^j=C_2^j=C^j$ for $1\le j\le N$ in order to prove the following theorem.

Theorem 3.1: For a given logic network, partition algorithm 1 produces a smaller Rent's exponent than partition algorithm 2 does if and only if $E[c(t_1-t_2)]\le E(c)E(t_1-t_2)$, where E(x) denotes the expect value of the random variable x.

Proof:

Necessary Condition: We want to show $r_1 \le r_2$ implies that $E[c(t_1-t_2)]\le E(c)E(t_1-t_2)$.

By definition of r

$$r_1=\frac{\sum_{j=1}^{N}(t_1^j-\bar{t}_1)(c_1^j-\bar{c}_1)}{\sum_{j=1}^{N}(c_1^j-\bar{c}_1^2)}$$

$$r_2 = \frac{\sum_{j=1}^{N} (t_2^j - \bar{t}_2)(c_2^j - \bar{c}_2)}{\sum_{j=1}^{N} (c_2^j - \bar{c}_2^2)}$$

We are interested in the case:

$$\sum_{j=1}^{N} (c_1^j - \bar{c}_1^2) = \sum_{j=1}^{1} (c_2^j - \bar{c}_2^2)$$

Then $r_1 \leq r_2$ implies that

$$\sum_{j=1}^{N} (t_1^j - \bar{t}_1)(c_1^j - \bar{c}_1) \leq \sum_{j=1}^{N} (t_2^j - \bar{t}_2)(c_2^j - \bar{c}_2)$$

Therefore

$$\sum_{j=1}^{N} (t_1^j - \bar{t}_1)c_1^j - \sum_{j=1}^{N} (t_1^j - \bar{t}_1)\bar{c}_1 \leq \sum_{j=1}^{N} (t_2^j - \bar{t}_2)c_2^j - \sum_{j=1}^{N} (t_2^j - \bar{t}_2)\bar{c}_2$$

$$\sum_{j=1}^{N} (t_1^j - t_2^j)c^j \leq \sum_{j=1}^{N} (t_1^j - t_2^j)\bar{c}$$

We get $E[c(t_1\text{-}t_2)] \leq E(c)E(t_1\text{-}t_2)$.

Sufficient Condition: It is straightforward by reversing the process for the necessary condition.

Let algorithm 1 be the Min-Cut algorithm and algorithm 2 be any partition algorithm. Obviously, $t_2^j \geq t_1^j$ holds for every j. However, the condition $E[c(t_1\text{-}t_2)] \leq E(c)E(t_1\text{-}t_2)$ is not always true. Unfortunately, we get the conclusion that the Min-Cut algorithm doesn't always produce the smallest Rent's exponent. However, it can be mechanized and be adapted to our study later.

3.2.4. Layout and Rent's Relationship

In this section, we study the effects of the Rent's exponent on the area and the average interconnection length. We study two partition alternatives. We find that the dependence of the area on the Rent's exponent holds in both cases:

(1) Partition approach 1: divide the computation graph into two equal halves.

(2) Partition approach 2: divide the computation graph into four quarters.

3.2.4.1. Partition Approach 1

Suppose that we have found a partition procedure to partition the given computation graph G(n) with n gates into two halves $G_1(\frac{n}{2})$ and $G_2(\frac{n}{2})$ with $\frac{n}{2}$ gates respectively. Then we can use the "divide-and-conquer" technique to lay-out the whole graph recursively. Here is the procedure:

(1) Layout $G_1(\frac{n}{2})$ with area $A_1(\frac{n}{2})$.

(2) Layout $G_2(\frac{n}{2})$ with area $A_2(\frac{n}{2})$.

(3) Layout the interconnection with the number of interconnections I(n) which is to be derived in the following.

(4) Merge (1),(2), and (3) to produce a whole layout with area A(n).

Now we are going to use Rent's rule to derive I(n). Since $T=KC^r$, then a circuit with size n, if divided into a set of equal sized subcircuits each of size m, has total terminals

$$T_{total}(m)=Km^r(\frac{n}{m})=knm^{r-1}$$

The number of the interconnections is proportional to T_{total} (*i.e.* $I_{total}(m)=cT_{total}(m)=cknm^{r-1}$, c is a constant. Every interconnection consists of a pair of terminals if $c=\frac{1}{2}$). Therefore the number of interconnections connecting two subcircuits is

$$I(n)=I_{total}(\frac{n}{2})-I_{total}(n)$$

$$=ckn[(\frac{n}{2})^{r-1}-n^{r-1}]$$

$$=cknn^{r-1}[2^{r-1}-1]$$

$$=\kappa n^r \quad where\ \kappa=ck[2^{1-r}-1]$$

3.2.4.1.1. Area to Embed Computation Graph

Before we derived the closed form formula for A(n), we state a geometric lemma. Theorem 2 follows [39] but with differences which are specially tailored to fit our cases.

Lemma 3.1: For every rectangle R with aspect ratio σ_R satisfying $\sigma_{min} \leq \sigma_R \leq 1$ and $\sigma_{min} \leq \frac{1}{2}$, the rectangle R' as shown in Figure 3.9, generated by putting two Rs together in the manner that two long sides are combined together, has aspect ratio bounded by σ_{min}.

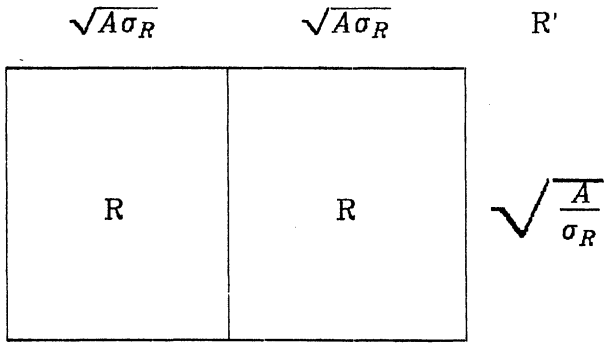


Figure 3.9 Geometric Construction of Lemma 3.1.

Proof:

Suppose that A is the area of the rectangle R, then the length of R is $l(R)=\sqrt{\frac{A}{\sigma_R}}$ and the width of R is $w(R)=\sqrt{A\sigma_R}$. On the other hand, the length of R' is $l(R')=2\sqrt{A\sigma_R}$ and the width of R' is $w(R')=\sqrt{\frac{A}{\sigma_R}}$.

This implies that $\sigma_{R'}=\frac{w(R')}{l(R')}=\frac{1}{2\sigma_R}$

Since $\sigma_{min} \le \frac{1}{2} \le \frac{1}{2\sigma_R}$ (*since* $\frac{1}{\sigma_R} \ge 1$)

therefore $\sigma_{R'} = \frac{1}{2\sigma_R} \ge \sigma_{min}$

Theorem 3.2: Let

$$a(n)=2a(\frac{n}{2})+2\sqrt{2a(\frac{n}{2})}I(n)+I^2(n)=\left[\sqrt{2a(\frac{n}{2})}+I(n)\right]^2,$$

then $A(n)=\frac{4}{\sigma_{min}}a(n)$ is big enough to embed the graph G(n).

Proof:

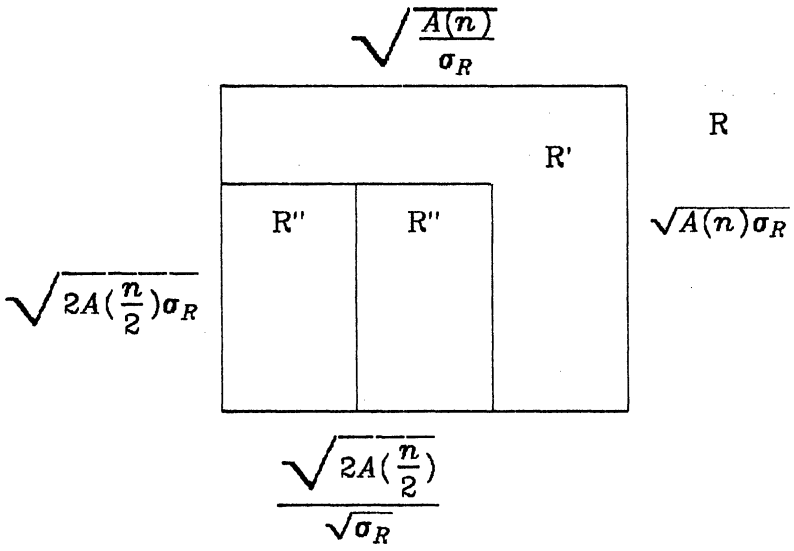


Figure 3.10 Geometric Construction of Theorem 2

(1) Partition recursively by using "divide-and-conquer" to divide G(n) into $G_1(\frac{n}{2})$ and $G_2(\frac{n}{2})$. For simplicity, we assume that G_1 and G_2 can be laid out in R'' with the same size $A(\frac{n}{2})$.

(2) Construct a rectangle R' with area=$2A(\frac{n}{2})$ such that the width of R' is $\sqrt{2A(\frac{n}{2})\sigma_{R'}} = \sqrt{\frac{A(\frac{n}{2})}{\sigma_{R'}}}$ and the length of R' is $\frac{\sqrt{2A(\frac{n}{2})}}{\sqrt{\sigma_{R'}}} =$

$2\sqrt{A(\frac{n}{2})\sigma_{R'}}$ respectively. By *Lemma 3.1*, R' is still bounded by aspect ratio σ_{min}.

(3) Construct a rectangle R as shown in Figure 3.10 which is similar to R' and has area A(n). In this case, $\sigma_R=\sigma_{R'}$.

(4) Perform routing for I(n) interconnections.

In order to use the layout techniques in section 3.2.2, we must show that

$l(R)\geq l(R')+2I(n)$

$w(R)\geq w(R')+2I(n)$

Remember that

$$l(R)=\sqrt{\frac{A(n)}{\sigma_R}}=\frac{\sqrt{4}}{\sigma_R\sigma_{min}}\sqrt{a(n)}$$

$$=\frac{\sqrt{4}}{\sigma_R\sigma_{min}}(\sqrt{2a(\frac{n}{2})}+I(n))$$

$$=\frac{\sqrt{2A(\frac{n}{2})}}{\sigma_R}+\frac{\sqrt{4}}{\sigma_R\sigma_{min}}I(n)$$

$$=l(R')+\frac{\sqrt{4}}{\sigma_R\sigma_{min}}I(n)$$

$$\geq l(R')+2I(n) \text{ since } \frac{1}{\sigma_R\sigma_{min}}\geq 1$$

Similarly

$$w(R)=\sqrt{A(n)\sigma_R}=\sqrt{4\frac{\sigma_R}{\sigma_{min}}\sqrt{a(n)}}$$

$$=\sqrt{4\frac{\sigma_R}{\sigma_{min}}}(\sqrt{2a(\frac{n}{2})}+I(n))$$

$$=\sqrt{2A(\frac{n}{2})\sigma_R}+2\sqrt{\frac{\sigma_R}{\sigma_{min}}}I(n)$$

$$\geq w(R')+2I(n) \quad \text{since } \sigma_R \geq \sigma_{\min}$$

This shows that A(n) is big enough.

Note that A(n) is just a constant factor bigger than a(n). We only refer to a(n) in the later analysis.

Theorem 3.3: For I(n)=κn^r, then a(n)=$[\sqrt{2a(\frac{n}{2})}+I(n)]^2$ has closed form solutions:

(1) $0 \leq r < \frac{1}{2}$, $a(n)=O(n)$.

(2) $r=\frac{1}{2}$, $a(n)=O(n)$.

(3) $\frac{1}{2} < r \leq 1$, $a(n)=O(n^{2r})$.

Proof:

Take a square root of

$$a(n)=[\sqrt{2a(\frac{n}{2})}+I(n)]^2$$

We get a nonlinear recurrence relation:

$$\sqrt{a(n)}=\sqrt{2a(\frac{n}{2})}+I(n)$$

Let $X(n)=\sqrt{a(n)}$ then $X(n)=\sqrt{2}X(\frac{n}{2})+I(n)$.

Suppose $n=2^N$, (i.e. $N=log_2 n$)

By substitution:

$$X(n)=\sqrt{2}[\sqrt{2}X(\frac{n}{4})+I(\frac{n}{2})]+I(n)$$

$$=I(n)+\sqrt{2}I(\frac{n}{2})+(\sqrt{2})^2 I(\frac{n}{4})+....+(\sqrt{2})^N I(1)$$

$$=\kappa n^r \sum_{i=0}^{N} 2^{\frac{i}{2}-ri}$$

$$=\kappa n^r \sum_{i=0}^{N} 2^{(\frac{1}{2}-r)i}$$

(1) $0 \le r < \frac{1}{2}$

$$X(n) = \kappa n^r \left[\frac{2^{(\frac{1}{2}-r)(N+1)}-1}{2^{\frac{1}{2}-r}-1} \right] \quad (since \quad 2^{\frac{1}{2}-r} > 1)$$

$$X(n) \approx \kappa n^r 2^{(\frac{1}{2}-r)N} = \kappa\sqrt{n}$$

$$X(n) = O(\sqrt{n}) \rightarrow a(n) = O(n).$$

(2) $r = \frac{1}{2}$

$$X(n) = O(\sqrt{n}) \rightarrow a(n) = O(n)$$

(3) $\frac{1}{2} < r \le 1$

$$X(n) = \kappa n^r \left[\frac{1-2^{(\frac{1}{2}-r)(N+1)}}{1-2^{\frac{1}{2}-r}} \right]$$

$$X(n) \approx \kappa n^r$$

$$X(n) = O(n^r)$$

$$a(n) = O(n^{2r})$$

3.2.4.1.2. Average Routing Length

Theorem 3.4: By using "divide-and-conquer" technique in the layout, the average routing length $\bar{L}(n)$ is:

(1) $0 \le r < \frac{1}{2}$, $\bar{L}(n) = f(r)$.

(2) $r = \frac{1}{2}$, $\bar{L}(n) = O(\log n)$.

(3) $\frac{1}{2} < r \le 1$, $\bar{L}(n) = O(n^{2r-1})$.

Proof:

Refer to the following figure, the distance between point (x_1,y_1) and point (x_2,y_2) is $L+x_2-x_1+|y_1-y_2|$. Let L_{2^i} and W_{2^i} denote the length and the width required for a circuit with 2^i components. We derive the average interconnection length as follows:

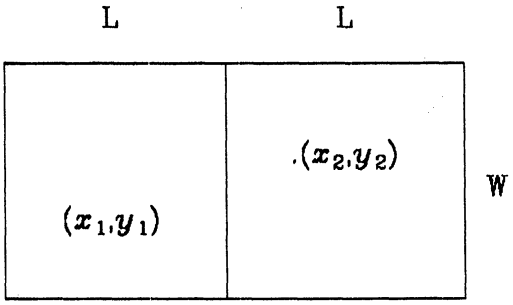
The average interconnection length is

$$\bar{L}_{2^i}=\frac{1}{W_{2^i}{}^2L_{2^i}{}^2}\sum_{x_1=1}^{L_{2^i}}\sum_{x_2=1}^{L_{2^i}}\sum_{y_1=1}^{W_{2^i}}\sum_{y_2=1}^{L_{2^i}}L+x_2-x_1+|y_1-y_2|$$

$$=\frac{1}{W_{2^i}{}^2L_{2^i}{}^2}\sum_{x_1=1}^{L_{2^i}}\sum_{x_2=1}^{L_{2^i}}\sum_{y_1=1}^{W_{2^i}}\sum_{y_2=1}^{L_{2^i}}(L+x_2-x_1)+(|y_1-y_2|)$$

$$=\frac{1}{W_{2^i}{}^2L_{2^i}{}^2}(\sum_{x_1=1}^{L_{2^i}}\sum_{x_2=1}^{L_{2^i}}L+x_2-x_1+\sum_{y_1=1}^{W_{2^i}}\sum_{y_2=1}^{W_{2^i}}|y_1-y_2|)$$

$$=L_{2^i}+\frac{W_{2^i}}{3}-\frac{1}{3W_{2^i}}$$



By following the definition of section 3.2.4.1, $I(2^i)$ is the number of the total interconnection terminals in a circuit with size 2^i. Therefore the total terminals in the entire hierarchical decomposition are

$$I_{total}=\sum_{i=0}^{N}I(2^i)\frac{n}{2^i}$$

$$=\sum_{i=0}^{N}\kappa n(2^i)^{r-1}$$

$$=\kappa' n$$

The total sum of the interconnection length is

$$L_{total}=\sum_{i=0}^{N}\bar{L}_{2^i}I(2^i)\frac{n}{2^i}$$

$$=\sum_{i=0}^{N}\kappa n(2^i)^{r-1}[L_{2^i}+\frac{W_{2^i}}{3}-\frac{1}{3W_{2^i}}]$$

(1). $0\le r<\frac{1}{2}$

At level i, $A_{2^i}=\alpha 2^i$, therefore $L_{2^i}=\alpha_L 2^{\frac{i}{2}}$ and $W_{2^i}=\alpha_W 2^{\frac{i}{2}}$.

$$L_{total}=\sum_{i=0}^{N}\kappa n(2^i)^{r-1}[L_{2^i}+\frac{W_{2^i}}{3}-\frac{1}{3W_{2^i}}]$$

$$=\kappa n\sum_{i=0}^{N}[\alpha'(2^i)^{r-\frac{1}{2}}-\frac{1}{3\alpha_W}2^{i^{r-\frac{3}{2}}}]$$

$$\approx\alpha'\kappa n\frac{1-(2^{r-\frac{1}{2}})^{N+1}}{1-2^{r-\frac{1}{2}}}$$

$$\approx f(r)\kappa n \quad with \quad f(r)=\alpha'\frac{1}{1-2^{r-\frac{1}{2}}}$$

Therefore the average interconnection length is

$$\bar{L}_{total}=\frac{L_{total}}{I_{total}}$$

$$=f(r)$$

(2). $r=\frac{1}{2}$

At level i, $A_{2^i}=\alpha 2^i$, therefore $L_{2^i}=\alpha_L 2^{\frac{i}{2}}$ and $W_{2^i}=\alpha_W 2^{\frac{i}{2}}$.

$$L_{total}=\sum_{i=0}^{N}\kappa n(2^i)^{r-1}[L_{2^i}+\frac{W_{2^i}}{3}-\frac{1}{3W_{2^i}}]$$

$$=\kappa n\sum_{i=0}^{N}[\alpha'(2^i)^{r-\frac{1}{2}}-\frac{1}{3\alpha_W}(2^i)^{r-\frac{3}{2}}]$$

$$=\kappa n\sum_{i=0}^{N}[\alpha'-\frac{1}{3\alpha_W}2^{-i}]$$

$$\approx O(\kappa nN) \quad where \ N=logn$$

Therefore the average interconnection length is

$$\bar{L}_{total}=\frac{L_{total}}{I_{total}}$$

$$=O(N) = O(logn)$$

(3). $\frac{1}{2} < r \leq 1$

At level i, $A_{2^i}= \alpha 2^{i2r}$, therefore $L_{2^i}=\alpha_L 2^{ir}$ and $W_{2^i}=\alpha_W 2^{ir}$.

$$L_{total}=\sum_{i=0}^{N}\kappa n(2^i)^{r-1}[L_{2^i}+\frac{W_{2^i}}{3}-\frac{1}{3W_{2^i}}]$$

$$=\kappa n\sum_{i=0}^{N}[\alpha'(2^i)^{2r-1}-\frac{1}{3\alpha_W}2^{-i}]$$

$$\approx O(n^{2r})$$

Therefore the average interconnection length is

$$\bar{L}_{total}=\frac{L_{total}}{I_{total}}$$

$$=O(n^{2r-1})$$

3.2.4.2. Partition Approach 2

Another approach to investigate the relationship between the area and the Rent's exponent is as follows:

(1) Divide the circuit into four quarters as shown in Figure 3.11 (recursively).

(2) Perform routing by using layout techniques in section 3.2.2.

(3) Put them together.

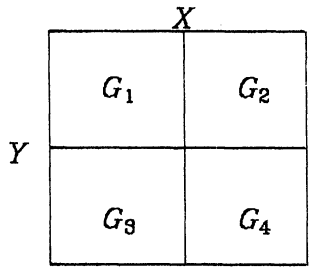


Figure 3.11 Partition Approach 2.

Now we are going to use Rent's relationship to derive the terminals among the partitions G_1, G_2, G_3, G_4.

Since $T=KC^r$, then a circuit with size n, if divided into a set of equal sized subcircuits each of size m, has total terminals

$$T_{total}(m)=Km^r(\frac{n}{m})=knm^{r-1}$$

The number of interconnections is proportional to T_{total} (*i.e.* $I_{total}(m)=cT_{total}(\mathrm{m})=cknm^{r-1}$, c is a constant. Every interconnection consists of a pair of terminals if $c=\frac{1}{2}$). Therefore the number of interconnections connecting two subcircuits are

$$\begin{aligned} I'(n) &= I_{total}(\frac{n}{4})-I_{total}(n) \\ &= ckn[(\frac{n}{4})^{r-1}-n^{r-1}] \\ &= cknn^{r-1}[4^{1-r}-1] \\ &= \kappa' n^r \quad where\ \kappa'=ck[4^{1-r}-1] \end{aligned}$$

Now we want to make further assumption that I'(n) are equally distributed among each pair of G_1, G_2, G_3, G_4. That is

$$\Gamma_{12}(n)=\Gamma_{13}(n)=\Gamma_{14}(n)=\Gamma_{23}(n)=\Gamma_{24}(n)=\Gamma_{34}(n)$$

$$=\frac{1}{6}\Gamma(n)=I(n)$$

We get I(n)=$\frac{\kappa'}{6}n_r$.

3.2.4.2.1. Area to Embed Computation Graph

Theorem 3.5: The area by Partition 2 is

$$A(n)=[2A^{\frac{1}{2}}(\frac{n}{4})]^2+4A^{\frac{1}{2}}(\frac{n}{4})\times5\times2I(n)+5\times2I^2(n)$$

$$=\left[2A^{\frac{1}{2}}(\frac{n}{4})+10I(n)\right]^2$$

Moreover, if I(n)=κn^r, then A(n) has closed form solution:

(1) $0\le r<\frac{1}{2}$, A(n)=O(n).

(2) $r=\frac{1}{2}$, A(n)=O(n).

(3) $\frac{1}{2}<r\le1$, A(n)=O(n^{2r}).

Proof: Consider the layout in Figure 3.11. There are six pairs of subcircuits to be routed together. First all, let's route pair (G_1,G_2). By our routing strategy, this will increase dimension X and Y by 2I(n) units respectively. Therefore, if we finish routing the pairs $(G_1, G_2),(G_3,G_4),(G_1,G_3),(G_2,G_4)$, the total dimension in X and Y are

$$L'_X=2A(\frac{n}{4})^{\frac{1}{2}}+8I(n)$$

$$L'_Y=2A(\frac{n}{4})^{\frac{1}{2}}+8I(n)$$

Now it is the time to route the pairs (G_1,G_4) and (G_2,G_3) as shown in Figure 3.12.

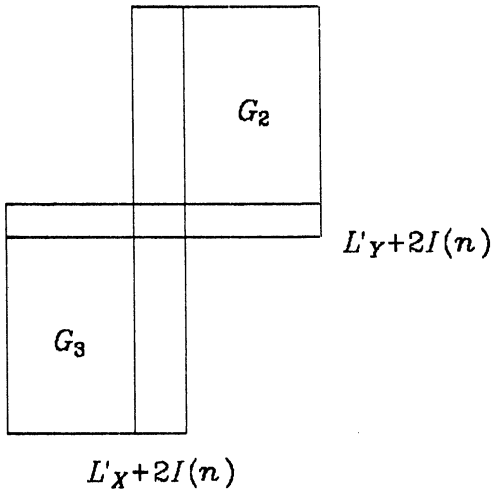


Figure 3.12 Routing Between Diagonal Pair.

Therefore the final layout would be

$$L_X=2A^{\frac{1}{2}}(\frac{n}{4})+10I(n)$$

And

$$A(n)=L_XL_Y=\left[2A^{\frac{1}{2}}(\frac{n}{4})+10I(n)\right]^2$$

Now we want to solve the non-linear recurrence relation. Suppose that $n=4^L$ (*i.e.* L=$\log_4 n$), for convienence.

By substitution and the same techniques used in the proof of Theorem 2.

Let X(n)=$\sqrt{A(n)}$, then

$$X(n)=I(n)+2I(\frac{n}{4})+(2)^2I(\frac{n}{16})+....+(2)^LI(1)$$

$$=10\kappa n^r\sum_{i=0}^{L}4^{\frac{i}{2}-ri}$$

$$=\kappa n^r\sum_{i=0}^{L}4^{(\frac{1}{2}-r)i}$$

(1) $0\le r<\frac{1}{2}$

$$X(n) = 10\kappa n^r \left[\frac{4^{(\frac{1}{2}-r)(L+1)} - 1}{4^{\frac{1}{2}-r} - 1} \right] \quad (since \quad 2^{\frac{1}{2}-r} > 1)$$

$$X(n) = O(\sqrt{n}) \rightarrow a(n) = O(n).$$

(2) $r = \frac{1}{2}$

$$X(n) = O(\sqrt{n}) \rightarrow a(n) = O(n)$$

(3) $\frac{1}{2} < r \leq 1$

$$X(n) = 10\kappa n^r \left[\frac{1 - 4^{(\frac{1}{2}-r)(L+1)}}{1 - 4^{\frac{1}{2}-r}} \right]$$

$$X(n) \approx \kappa n^r$$

$$X(n) = O(n^r)$$

$$a(n) = O(n^{2r})$$

3.2.4.2.2. Average Routing Length

By following the partition 2, we can prove the Theorem about the length of interconnection.

Theorem 3.6: By the partition 2, the average routing length $\bar{L}(n)$ is as follows:

(1) $0 \leq r < \frac{1}{2}$, $\bar{L}(n) = f(r)$.

(2) $r = \frac{1}{2}$, $\bar{L}(n) = O(\log n)$.

(3) $\frac{1}{2} < r \leq 1$, $\bar{L}(n) = O(n^{2r-1})$.

Proof: The proof is similar to those of Theorem 3.4.

3.2.5. Summary of Point Model

Our model consists of an array of lattice points and has only one unit width for both the vertical and the horizontal channels. The circuit are placed at the set of lattice points while the connection are mapped into the channels. Under our model, we define the area to be the smallest rectangle enclosing the set of mapped lattice points. The results show that

(1) both the area and the routing length are function of Rent's exponent of the circuit. By our analysis, both figures of merit grow fast when a circuit has big Rent's exponent which implies the circuit has high performance.

(2) good partition algorithm implies small r. In order to reduce the area and the average routing length, we need a good partition algorithm.

However, if we removed the restriction of unit width of channel, the problem to figure out the area becomes extremely difficult. Donath [35], [36] assume unspecified width on the channel and do not impose the unit width restriction in studying the relation between Rent's exponent and the average routing length. However, their solutions have two deficits:

(1) They assume that any circuit with size n always can be placed into an array with n lattice points. Some circuits have routability problem under this assumption.

(2) They get the results (the average routing length) by normalizing them with the channel width. However, it is another difficult problem to estimate the channel width [21], [22].

Although our model is restrictive , we feel our analysis help us to understand how the interconnection would contribute to the area of the final layout.

3.3. The Partition and Rectangle Model of a Circuit

As we mentioned in Chapter 1, the Programmable Logic Array (PLA) is a basic module in the realization level. Programmable Logic Arrays (PLAs) are used frequently to implement control logic, and are increasingly being considered as an implementation mode for the datapath [12] and advocated as a viable methodology for VLSI designers [13]. Although PLAs are generally considered to be wasteful of silicon area in comparison to random logic, they offer a conceptual and structural regularity which is quite attractive for VLSI design. Besides the conceptual simplicity, many related software tools have been developed for PLA-based design system. These include logic minimization [15], [43] , and the automatic geometric layout for the optimal performance of a single PLA [44], [45]. From the user's point of view, optimization to reduce area and delay can be automatically applied to the structure in a manner which is transparent to the logic designer. In ADS, a hierarchical description of a design is partitioned and then translated into a network of Programming Logic Arrays. Since the partition problem is very difficult and has been studied by other researchers [46] , we are making assumption that a partition into PLAs is given and concentrate on how to provide the estimates of its final layout. An example is shown in Figure 3.13. Suppose that the Terminal Control Circuit (TCC) has been partitioned into eight PLAs and has the interconnections as in Figure 3.13. The problem is to obtain estimates of the final layout are. We develop the following method to solve this complicated problem. The proposed Rectangle Model consists of a set of rectangles for the set of PLAs (eventually it can be any logic block) and a set of channels for the interconnections among the PLAs.

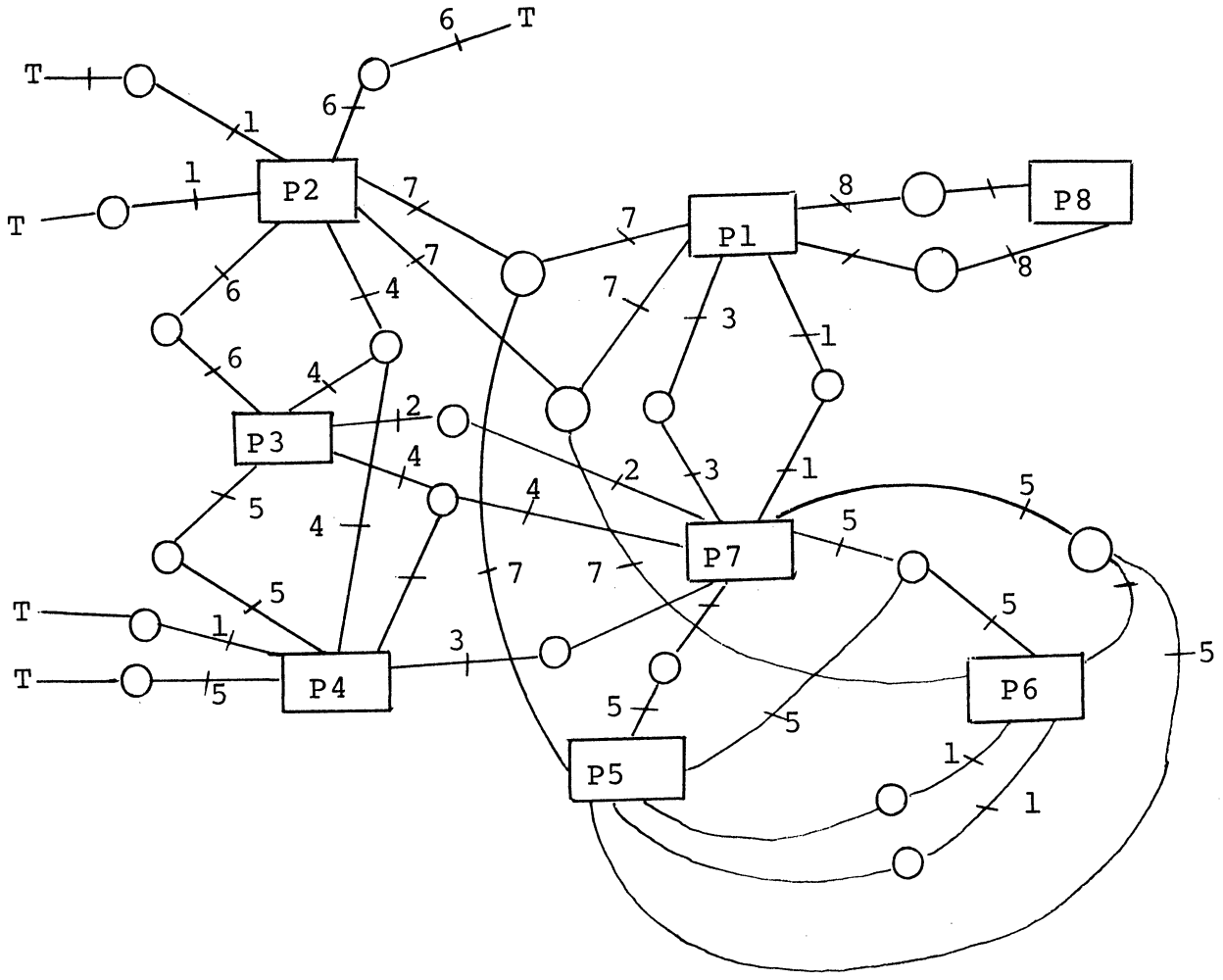


Figure 3.13 A PLA-based Terminal Control Circuit.

3.3.1. The Circuit Model

A single PLA can be modeled as an arbitrary-sized rectangle, as shown in Figure 3.14. In each rectangle, the terminals are distributed around the peripherals of the rectangle.

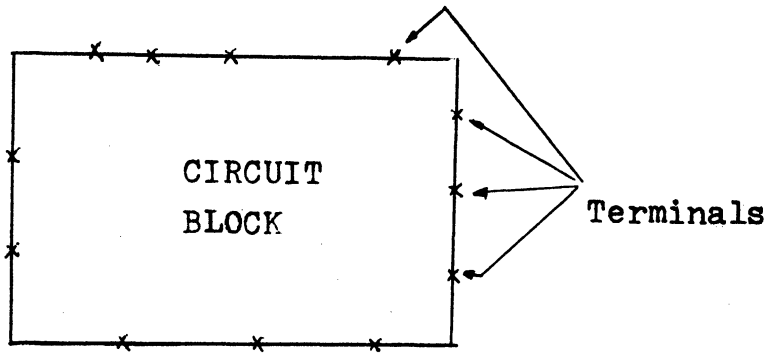


Figure 3.14 A Rectangle Represents for a PLA.

3.3.2. The Channel Definitions

As we mentioned earlier, a channel in the layout is used for an interconnection. A channel is a space between two rectangles as shown in Figure 3.15. In order to use the channel routing algorithm, we restrict ourselves to the case where there are two layers for the routing process. One of them is for the connections in the vertical direction, while the other is for those in the horizontal direction. Under this construction, the routing can not go through the rectangles.

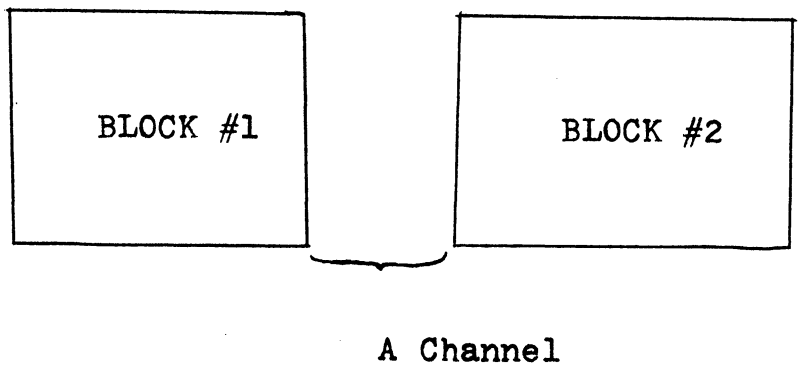


Figure 3.15 The Definition of a Channel.

3.3.3. Problems with the General Cell Approach

Now we study the problem of estimating the figure of merits for a particular arithmetic design under the rectangle model. For example, we will ask the question "Given Figure 3.13, how big is the final circuit ?" As we have said in Chapter 1, it is impossible to provide an estimate of the interesting figures of merit without referring to some particular design methodology. In relation to the PLA realization, the rectangle model corresponds to the general cell approach for VLSI design. We, therefore, offer a constructive solution which includes the placement and the routing processes. Later chapters are devoted to developing tools and algorithms to provide a quick estimates of the figures of merit.

3.3.3.1. Works Related to the General Cell Approach

The general cell approach to layout consists of placement procedure and routing procedure.

3.3.3.1.1. Placement Procedure

The *placement problem* of the general cell approach is defined as follows:

Given: A set of predefined but different sized rectangles.

Find: A placement with minimal area and interconnection length.

Two classes of algorithms, initial and constructive, have been intensively surveyed [47]. However, they are not useful for the placement of the general cell approach because the modules in [47] are dimensionless, which are in contrast to those of general cells. Since the general cell approach is a viable tool for custom IC design, much effort has been devoted to the general cell placement problem. Two kinds of algorithms have been proposed. The first is the *exhaustive search* approach [24], [26]. In this category, a module is added by exhaustively searching the best location under some estimated figure of merit. Possible figures of merit are the length of interconnections or space for a channel. The

disadvantage of this approach is that it is time consuming. The other category is the *min-cut* oriented algorithms suggested by [25], [48], [49]. The idea is to divide the rectangle region into two subregions and the cluster of modules into two subclusters, and then assign each subcluster a subregion such that the number of the connections crossing over two subregions is minimal. The development of this class of algorithms is based on the assumption that the area can be reduced by minimizing the crossover connections. However the assumption is never confirmed.

3.3.3.1.2. Routing Procedure

The *routing* problem is defined as follows:

Given: (1) A relative position of the set of placed rectangle blocks. (2) The signal nets with terminals on the boundaries of the blocks.

Find: Completed interconnections.

The procedure includes three subprocedures. An important component of the routing is to find a good router. Good early surveys of the routing problem can be found in [50]. Literatures reveal four categories of routers.

Lee type router - The original idea comes from [51]. However, some speed up version of routers have been proposed [52], [53]. One major disadvantage of the Lee type router is that we cannot prevent the congestion of the interconnections.

Hightower [54] proposed two other kinds of routers, namely cellar router and line router. Again they also have the disadvantage of congestion.

Channel router - This kind of router was originally proposed by [1]. Sophisticated extensions of the router can be found in Chapter 6. This router enforced by the following subprocedures can generate completed routing.

3.3.3.1.2.1. Channel Routing Order

The procedure tries to generate a legal routing order such that we can apply a router one by one according to the order. The problem has been discussed in [24], [55], [56], but lacks rigorous treatment. In Chapter 4, we investigate the necessary and sufficient conditions for the routability of the layout based on a routing order constraint graph.

3.3.3.1.2.2. Global Routing

The procedure tries to allocate a path or a tree structure for each signal net. It is a global optimization problem. Figure 3.16 shows a typical channel. The important task by global routing is to decide which channel absorb the *pass-through connections* and which side of the channel the *flow-in connections* come from.

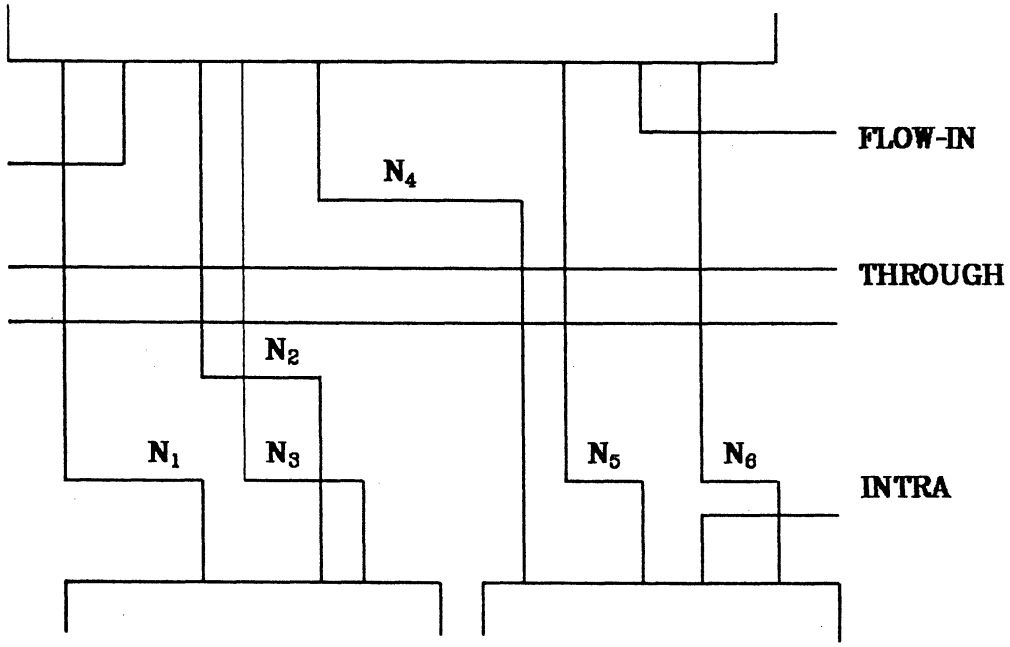


Figure 3.16 The Contribution in a Channel

The arguments in [24], [57], the author argues that finding the shortest path for signal nets is meaningless unless we know the exact width of each channel. Therefore, he proposes a stochastic model to estimate the channel width. The model is based on two unverified assumptions:

(1) Distribution of pins around the periphery is a *Poisson* distribution.

(2) Distribution of the net length is a *geometric* distribution.

Careful examination reveals that in his model *zero width* of channel is assumed in order to compute the parameters of the distribution of pins and the distribution of net length. This is a serious factor of the distribution of net length if the circuit has a big interconnection area. Ignoring the factor in computing the parameters would cause error in estimating the width of each channel. Therefore, we don't think the stochastic model would produce better results. Our strategy for this problem is to compute the width of each channel contributed by *intra-connections* within each channel. Based on this width, a shortest path or a *Steiner* tree procedure is applied to perform the path allocation [58].

3.3.3.1.2.3. Track Assignment

We develop a new efficient channel router. Besides that we develop the **T** shape router and **cross channel** router. We discuss these routers in Chapter 6.

3.3.3.2. A Proposed Layout Approach

Two specific goals of the layout, namely to minimize the chip area and to achieve 100% connections, make the automatic process for the general cell approach very difficult. In particular, to achieve 100% connections implies that the routability problem is the first issue to be considered. Methods must be developed to test the routability for the given placement. Chapter 4 addresses the routability problem. Since the other goal is to minimize the chip area, the arrangement of the relative locations of the rectangles affects the final layout. The interaction being so complicated, researchers generally break the process into

(1) Placement

The goal of the placement is to avoid an unroutable channel and to make the final area as small as possible. The necessary and sufficient condition for the routability test provides a guideline for avoiding unroutable channels. while the global allocation is a process to minimize the final area.

(2) Routing

The goal of the routing process is to have 100% complete interconnections and make the final area as small as possible. We break the routing process into the following processes:

(2-1) Finding a legal channel routing order.

(2-2) Performing topology net assignments.

(2-3) Solving the displacement problem for each channel.

(2-4) Performing channel routing by following the legal channel order.

(2-5) Performing the **T** shape router on each **T** shape intersection if any.

(2-6) Performing the **cross channel** router on each + shape intersection if any.

Both chapter 4 and chapter 5 are devoted to the entire step (2).

CHAPTER 4

THE ROUTABILITY AND CHANNEL ROUTING ORDERS

4.1. Introduction

The general cell approach is a way to automatically generate a VLSI layout. A *general cell* is a cell with no restriction on its height and width, and no restriction on the location of its input-output pins. Building blocks like the **PLA** and **ROM** are general cells. Since the complexity of VLSI circuit is increasing rapidly, the general cell approach is viable for designing complicated circuits [24], [25], [33]. The problems associated with this approach are numerous, and suffer from being neither rigorously treated nor intensively studied. One of them considered in this paper is the *routability* problem. Though previously discussed by [24], [55], [56], it still needs complete investigation.

The general cell approach to layout consists of a placement procedure and a routing procedure. The placement procedure gives the relative positions among the building blocks and produces a rough layout. Before doing a final routing procedure we ask the following question: *Is the rough layout routable*?

We begin at section 4.2 by defining the channel routing order among the channel set $C=\{c_1,\ldots,c_n\}$ and the routability of a channel $c_i \in C$. Then we define the *routability* of the entire rough layout. The nature of the channel routing algorithm imposes some inherent routing order constraints on the channels [1], [59], and hence reduces the number of legal routing orders.

In section 4.3 we define the legal routing order by constructing a routing order graph. The routing order graph consists of **n** channels as its nodes and has a directed edge when two channels have an inherent routing order constraint.

This allows us to discuss the routability in term of the routing order graph. Based on the graph, we investigate the necessary and sufficient conditions for the routability of the rough layout. Our condition set is the minimum one and hence a subset of the condition set given by [55]. This set will speed up the test of routability. By using a different approach from [55], our results are derived directly from considering all routing orders.

In section 4.4 we make some comparisons and show the advantages of our method over previously published methods [24], [55], [56] for testing routability. Finally we propose an algorithm for testing routability and generating a legal routing order for the rough layout.

4.2. Definitions

Parallel routing is generally possible but poorly understood at present. Here we restrict our interest to routing the channels sequentially. We give the following definitions, assuming sequential routing:

Definition 4.1: A routing order is a permutation of the channel set $C=\{c_1, c_2,\ldots,c_n\}$

Definition 4.2: A specification is a set of order pairs describing the relative positions of the building blocks.

Definition 4.3: A final specification is the one obtained from the initial specification under a routing order.

A trivial initial specification is the one which leaves the relative position of the blocks totally arbitrary.

Definition 4.4: A channel is routable if there exists a final specification such that it does not over specify the width of the channel.

Definition 4.5: A routing order is legal for a channel c_i if we apply it to some initial specifications, and the channel c_i is routable. Figure 4.1 is an example of

this case.

Example 1:

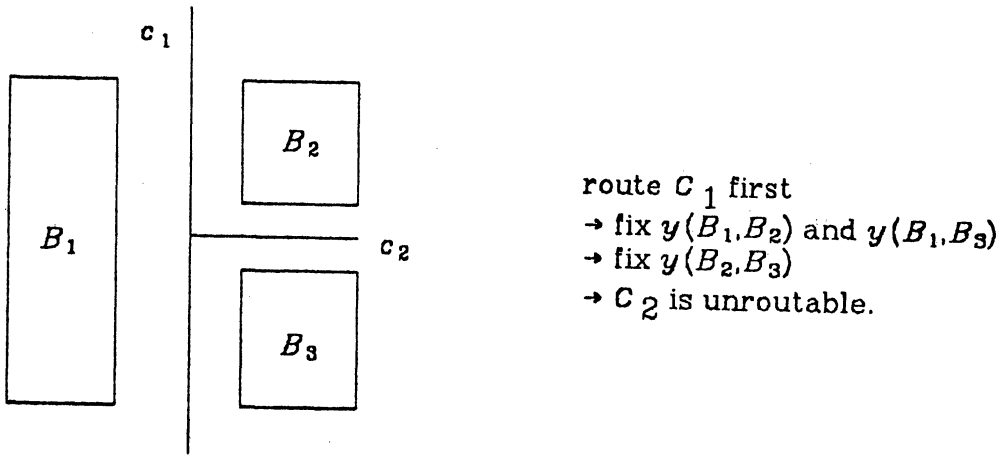


Figure 4.1 **T** Shape Channel Intersection.

Definition 4.6: An initial specification is complete if and only if there exists a final specification such that all the channels in the rough layout are routable.

The existence of a complete initial specification implies there exists a routing order which is legal for all the channels in C.

Definition 4.7: A rough layout is inherently unroutable, if and only if there does not exist any complete initial specification.

Definition 4.7 provides a method for testing the routability of a rough layout by checking all the routing orders on the trivial initial specification, and then to see if there exists any unroutable channel. Obviously the number of routing order is **n**!. An exhaustive method to test whether the given layout is routable is to check all the routing orders and see if there exists any complete specification. The following section describes a method for cutting down the number of routing orders checked. By applying the graph representations, the method is reduced to testing whether it has a directed cycle in the routing order graph.

4.3. Inherent Channel Routing Order

Theoretically, we need to study all of the $\binom{n}{2}$ pairs of order relationships. However, careful analysis shows that only some special kinds of channel intersections cause order constraints. In this section, we identify all the inherent channel routing order constraints. There are three types of channel intersection, namely "**T**" shape intersection, "+" shape intersection and "**L**" shape intersection.

The order constraints resulting from these three types of channel intersections are called direct order constraints in contrast to indirect order constraints resulting from indirect channel interactions under some special channel configurations.

4.3.1. Direct Order Constraints

4.3.2. T Shape Intersection

As shown in the Figure 4.1., **T** shape intersection can have only one legal routing order, namely that c_2 must precede c_1. No matter what the specifications, the sequence c_1 precedes c_2 is always an illegal order. Therefore we always put a directed edge in the channel routing order graph from c_2 to c_1 (*i.e.* $c_2 \rightarrow c_1$). Note that the relation c_2 **T** c_1 implies the relation c_1 **T'** c_2.

4.3.2.1. + Shape Intersection

Let (x_i,y_i) denote the coordinates of the lower left corner of the building block B_i. Every building block B_i has width w_i and length l_i. Let $x(B_i,B_j) = x_i - x_j$ denote the relative position of B_i and B_j in the X-coordinate. Similarly $y(B_i,B_j)$ is for the Y-coordinate.

Figure 4.2 shows the " + " shape intersection. Let us consider the routing order of the channels, namely c_1 and c_2 which form a "+" shape intersection.

For routing c_1 first, we need to specify $x(B_1, B_2)$, $x(B_3,B_4)$, $y(B_1,B_3)$, and $y(B_2,B_4)$. The solution of the *displacement problem* would give the optimal value of $x(B_1,B_2)$ and $x(B_3, B_4)$ such that the width of channel c_1 is minimal. This would fix $y(B_1,B_2)$ and $y(B_3,B_4)$ but not fix either $x(B_1,B_3)$ or $x(B_2,B_4)$. The width of the channel c_2 is fixed after routing c_1. Channel c_2 is still routable by *Definition 4.4*. However, in order to route the channel c_2 we need to specify $y(B_1,B_3)$, $y(B_2,B_4)$, $x(B_1,B_2)$, and $x(B_3,B_4)$. All these required specifications have been fixed after routing c_1. Similar analysis applies to the case for routing c_2 before c_1. The analysis shows that either sequence c_1,c_2 or c_2,c_1 is a legal routing order, therefore we do not have any directed edge between nodes c_1 and c_2.

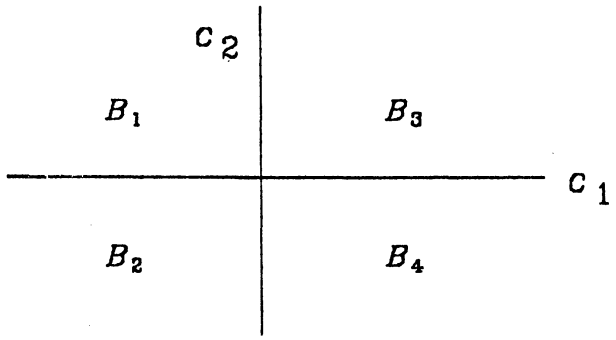


Figure 4.2 + Shape Channel Intersection.

4.3.2.2. L Shape Intersection

The possible locations of **L** shape intersection are only in the four corners of the rough layout. Figure 4.3 shows the **L** shape intersection. Obviously two channels c_1 and c_2 are symmetric, so there are no order constraints between them.

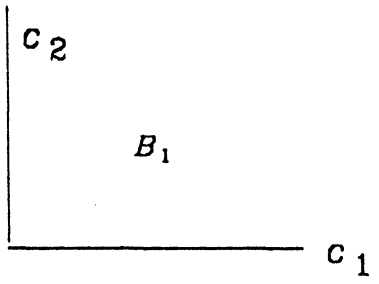


Figure 4.3 L Shape Intersection.

4.3.2. Indirect Order Constraints

In this section, we study the constraints generated from some special geometric configurations.

4.3.2.1. Constraints Due To Propagations

Let us define the following relations between the channels:

1. c_2 **T** c_1 and c_1 **T'** c_2 if the channels c_1 and c_2 form a **T** shape intersection.
2. $c_1 + c_2$ if the channels c_1 and c_2 form a + shape intersection.

Let $r_i \in R=\{$ **T,T',+** $\}$ denote a relation between two channels. Since the graph representing the rough layout is strongly connected, the following Lemma is true.

Lemma 4.1: For any two channels c_i and c_f in C, there exist channels $c_1, \ldots, c_k$ such that $S=c_i r_1 c_1 \ldots c_k r_k c_f$, where $r_1, \ldots, r_k \in R$.

Proof:

Let's consider the point p_i in the channel c_i and the point p_f in the channel c_f. We prove the lemma by answering the question "Is p_i connected with p_f?". We prove the statement by contradiction. Assuming that there doesn't exist any path that connects p_i with p_f. This implies that the layout can be divided into two parts. By definition, the space that separates these two parts is a channel.

Obviously, p_i can be connected to p_f. This is a contradiction.

Lemma 4.2: The constraints between c_i and c_f is the union of all the constraints generated by all the sequences S in *Lemma 4.1.*

Proof:

Lemma 1 guarantees that there exists at least one S. Every relation sequence $r_1, \ldots, r_k$ define a routing order constraint between c_i and c_f. Therefore to see whether c_i precedes c_f or the reverse is to explore all the sequences Ss between c_i and c_f. This is equivalent to say the constraints between c_i and c_f is the union of all the constraints generated by all the sequences S in *Lemma 4.1*

The relation sequence $r_1, \ldots, r_k$ can be classified into two cases:

Case

All the relations in $r_1,\ldots,r_k$ are either **T** or **T'**.

Subcase

All the r_i's are **T**, then $c_i \rightarrow c_f$. Since the constraint $c_i \rightarrow c_f$ is automatically implied by the relation sequence, we do not have the directed edge in the routing order graph.

Subcase

Otherwise, there exists a channel c_p, c_q, c_r such that S=$c_i \ldots c_p$ **T** c_q **T'** $c_r \ldots c_f$. In this case we have three geometric configurations shown in Figure 4.4. All the configurations have the routing order graph as shown in Figure 4.5. Obviously there are no order constraints between c_i and c_f.

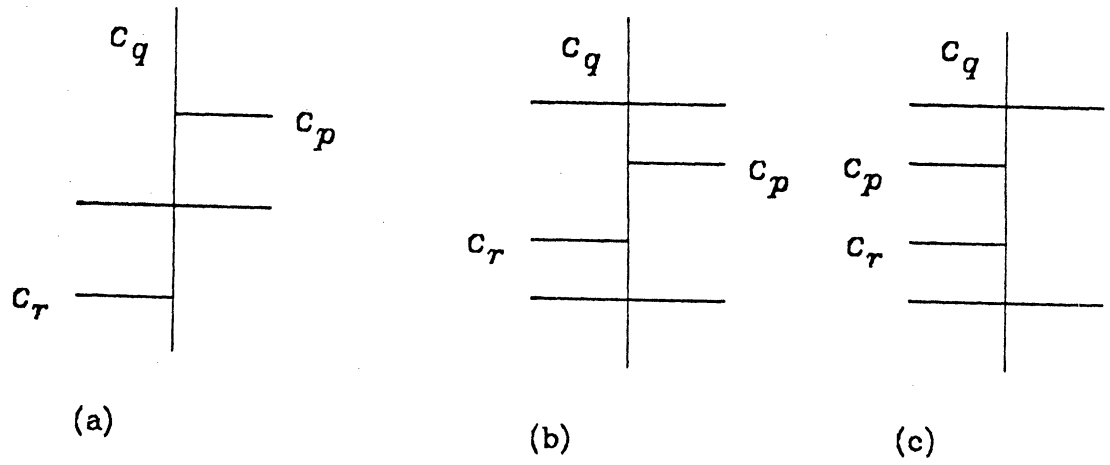


Figure 4.4 Geometric Configurations

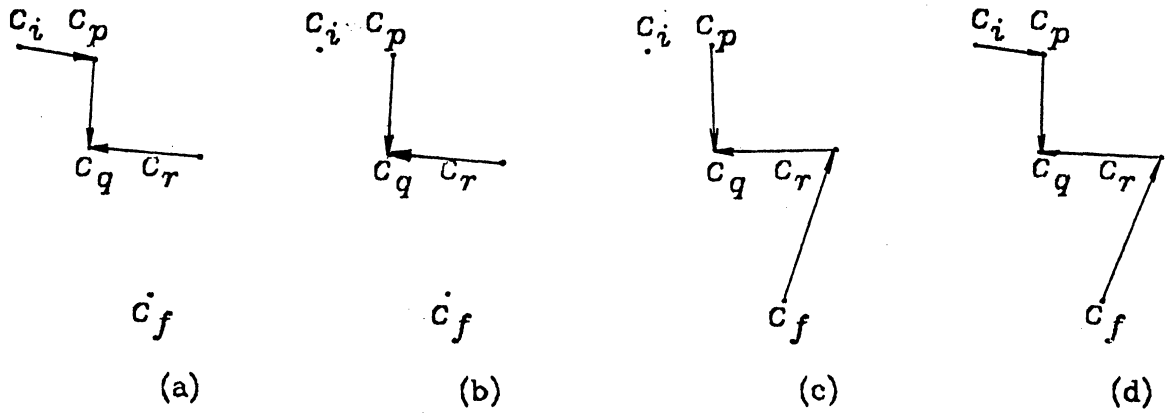


Figure 4.5 Order Constraints Corresponding to Figure 4.4.

Case

At least one relation r_m in $r_1, \ldots, r_k$ is +, We divide S into $S_1 = c_i r_1 c_1 \cdots c_m$ and $S_2 = c_l \cdots c_f$ such that S=S_1 + S_2 and c_m + c_l. The analysis of subcase 1.1 in section 4.3.2 shows that there are at most four cases as shown in Figure 4.6. We exclude the case that no constraints exist between c_i and c_m as well as between c_l and c_f. Since c_m and c_l form a + shape intersection, there are no constraints between c_m and c_l in section 4.3.1.2. Hence we have no constraints between c_i and c_f.

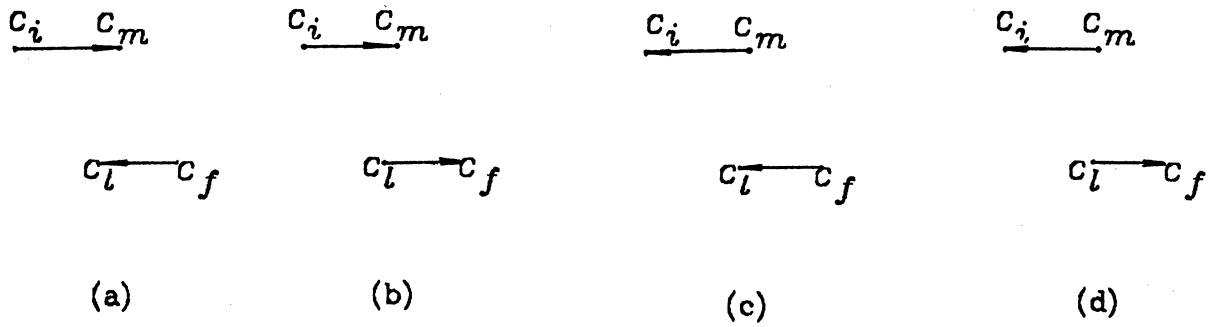


Figure 4.6 Order Constraints.

The analysis shown above suggests that, if we are concerned only with the relationship between any pair of channels, the order constraint graph can be constructed from only the direct constraints. However, since we don't impose any constraint between any pair of channels except that they form a **T** shape intersection, the following special case deserves our attention.

4.3.4.1. Constraints due to Parallel Blocks

Let's consider the case shown in Figure 4.7. So far we don't have any constraint between c_1 and c_2 as well as c_1 and c_3. The routing sequence c_2 c_3 c_1 seems to be legitimate. But it is not true and we would show it as follows: Routing c_2 first, we need to specify $x(B_4,B_1)$, $x(B_2,B_5)$, $y(B_4,B_5)$ and $y(B_1,B_2)$. This would fix $y(B_1,B_2)$. Again routing c_3 next, we need to specify $x(B_1,B_6)$, $x(B_3,B_7)$, $y(B_1,B_3)$, $y(B_6,B_7)$. This again would fix $y(B_1,B_3)$. Then we have $y(B_2,B_3)$ fixed which would violate *Definition 4.5* (*i.e.* c_1 is unroutable!). Any routing sequence ending with c_2 or c_3 (that is, two-thirds of the permutation sequences) are legal. Since we represent the legal routing order in terms of a constraint graph, we are looking for the minimum cardinality of graphs such that the union of sequences generated by the graphs is equal to the set of all the legal sequences. The union of sequences generated by Figure 4.8.a and Figure 4.8.b is equal to all the legal sequences associated with Figure 4.7.

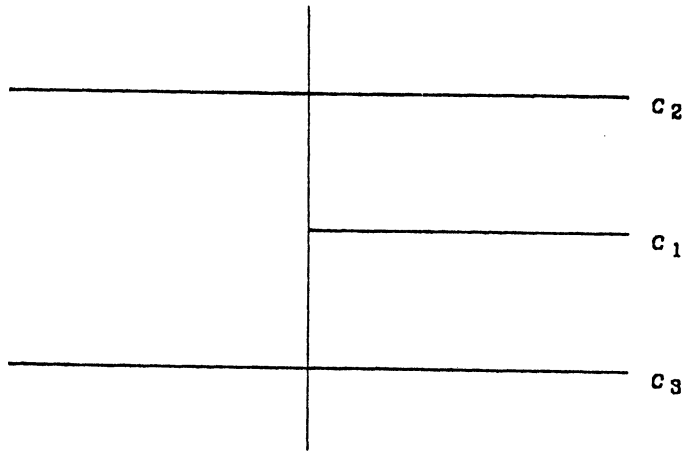


Figure 4.7 Parallel Blocks P_{23}.

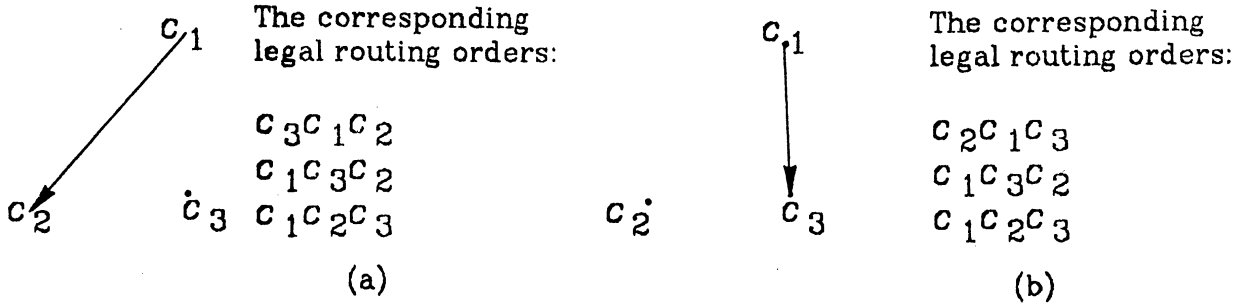


Figure 4.8 The Spanning Graphs of Figure 4.7.

The case in Figure 4.7 can be generalized to Figure 4.9 and Figure 4.10. It is clear that Figure 4.9 is a special case of Figure 4.10. We only consider the case of Figure 4.10 later in this chapter. Let C_{ij} denote the set of the channels between two parallel channels c_i and c_j.

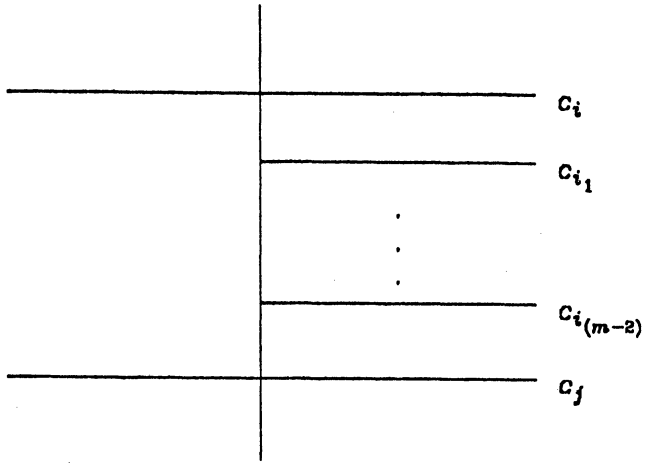


Figure 4.9 Generalized Parallel Blocks P_{ij}

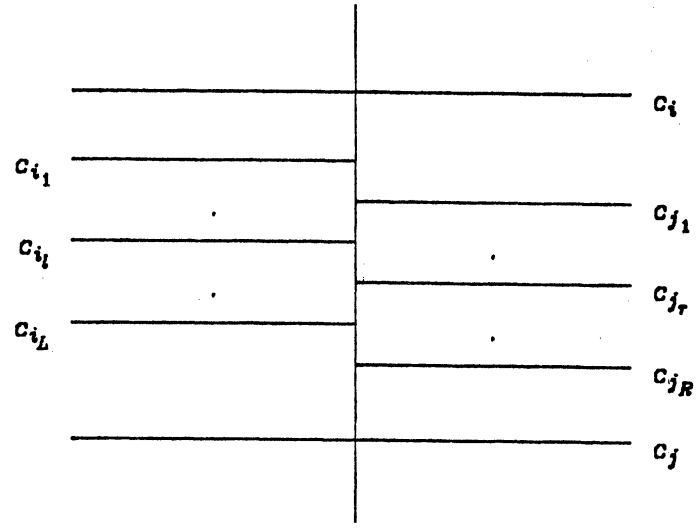


Figure 4.10 Generalized Parallel Blocks P_{ij}.

Let S(G) denote all the sequences generated from directed graph G by applying topological sort [60]. Also let $L_s(P_{ij})$ be the set of all the legal sequences for the parallel block P_{ij}.

Definition 4.8: The spanning graphs of a configuration are the set of graphs by which the union of sequences generated is equal to all the legal sequences.

Lemma 4.3: All the legal sequences $L_s(P_{ij})$ of Figure 4.10 are the sequences ending with either c_i or c_j. Moreover $|L_s(P_{ij})|=2(m-1)!$, where m is the number of channels within P_{ij}.

Proof:

We prove the Lemma by contradiction. Suppose that there exists s $\in L_s(P_{ij})$ such that s ends with neither c_i nor c_j. This implies that there exists a $c_l \in C_{ij}$-$\{c_i, c_j\}$ such that s=....c_i c_j c_l ends with c_l. Then c_i and c_j would fix the width of c_l before routing it. Therefore, the sequence s is not a legal routing order. This contradicts the assumption.

Let $G_u(P_{ij})$ and $G_b(P_{ij})$ denote the graphs in Figure 4.11 and Figure 4.12, respectively.

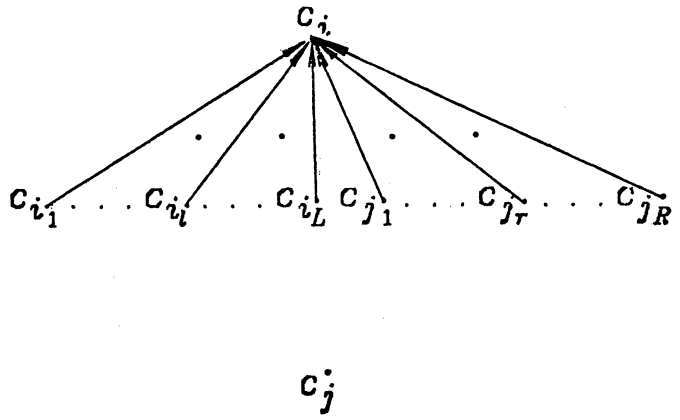


Figure 4.11 Graph $G_u(P_{ij})$.

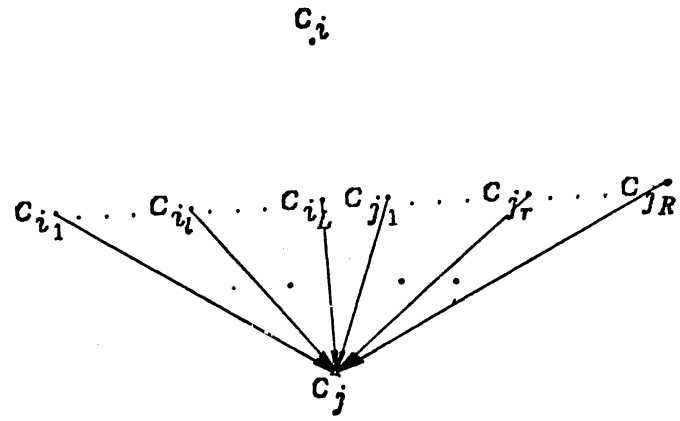


Figure 4.12 Graph $G_b(P_{ij})$.

Lemma 4.4:

$S(G_u(P_{ij})) \cup S(G_b(P_{ij})) = L_s(P_{ij})$.

Proof:

(1). We show $S(G_u(P_{ij})) \cup S(G_b(P_{ij})) \subset L_s(P_{ij})$:

For every $s \in S(G_u(P_{ij})) \cup S(G_b(P_{ij}))$ implies that s ends with either c_i or c_j. This shows $s \in L_s(P_{ij})$.

(2). We now show $L_s(P_{ij}) \subset S(G_u(P_{ij})) \cup S(G_b(P_{ij}))$:

For every $s \in L_s(P_{ij})$ implies that s end with either c_i or c_j. Let $s=c_{i_1}\ldots c_{i_{(m-1)}}c_i$ where $c_{i_l} \in C_{ij}-\{c_i\}$. We construct the graph by having $c_{i_1} \rightarrow c_{i_2},\ldots,c_{i_1} \rightarrow c_i$, $c_{i_2} \rightarrow c_{i_3},\ldots,c_{i_2} \rightarrow c_i,\ldots,c_{i_{(m-1)}} \rightarrow c_i$. Then we know G_u is a subgraph of the constructed graph, and $s \in S(G_u(P_{ij}))$. Similarly for every $s=c_{i_1}\ldots c_{i_{(m-1)}}c_j$, $s \in S(G_b(P_{ij}))$.

(1) and (2) show that $S(G_u(P_{ij})) \cup S(G_b(P_{ij})) = L_s(P_{ij})$.

By *Lemma 4.4*, it can be shown that $\{G_u(P_{ij}), G_b(P_{ij})\}$ are the minimum spanning graphs of the *parallel blocks* P_{ij}.

Definition 4.9: Let $\mathbf{A}=(a_1,\ldots,a_m)$ be a vector with $a_i \in \{u,b\}$. Then $G_A(V,E)$ is an *order graph* constructed in the following way:

(1) If c_p **T** c_q, put a directed arc from c_p to c_q.

(2) For each parallel block P_{ij}, choose either $G_u(P_{ij})$ or $G_b(P_{ij})$ for the constraints on channels with this parallel block according to the vector **A**.

Theorem 4.1: The rough layout L is inherently unroutable if and only if every *order graph* $G_A(V,E)$ is cyclic.

Proof:

The core portion to prove the theorem is to show that the union of the sequences generated by the set of order graphs are the set of all legal routing sequences with respect to the given layout. In other words, $\bigcup_{\textit{all order graphs}} S(G_A) =$ { all the legal routing sequences with respect to the given layout }.

As we show in section 4.3.1 and 4.3.2, the constraints between two channels can be either direct or indirect constraint. The indirect constraint is due to the structure of parallel block. For each parallel block, in Lemma 4.4, we have shown that two spanning graphs generate all the legal sequences associated with a parallel block. By our construction, each order graph contains all the direct constraints and a combination of spanning graphs representing for the set of the constraints due to the corresponding parallel blocks. Therefore the set of order graphs with all the combinations of vector A will generate all the legal routing sequences. If a layout is unroutable, by *Definition 4.7*, there does not exist any initial complete specification. This implies that there dosen't exist any legal routing order, therefore, every order graph must be cyclic. The theorem is proved.

Figures 4.13 and 4.14 show an inherently unroutable layout and the associated routing order graph, respectively.

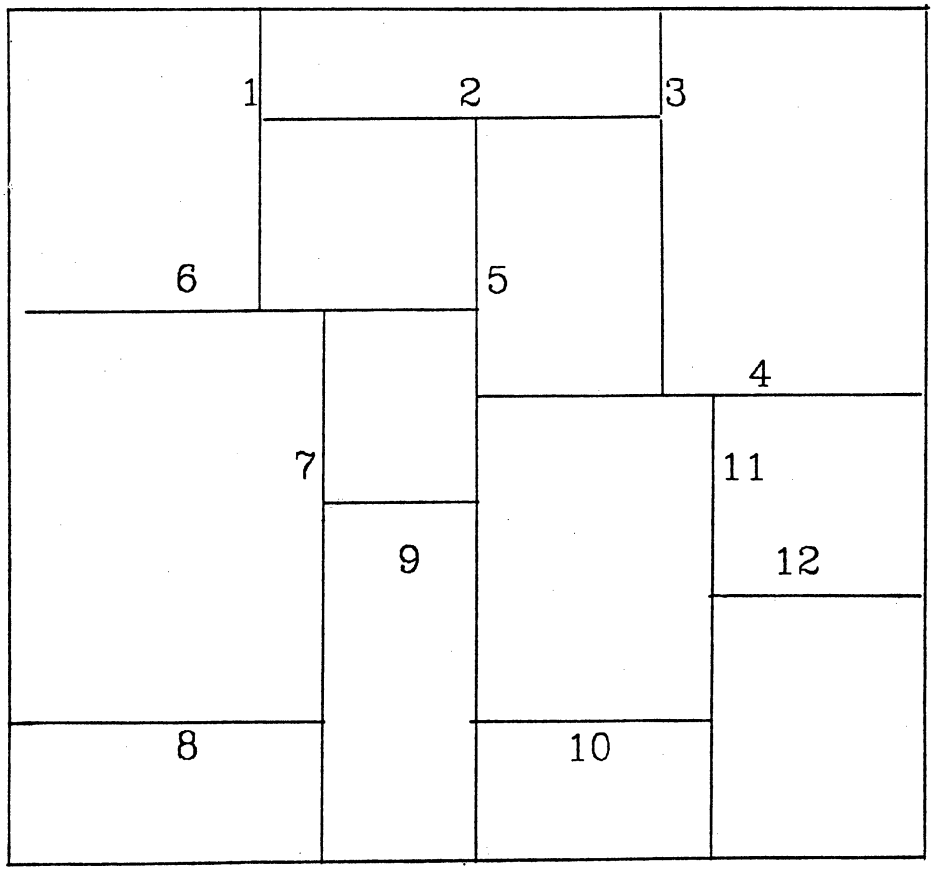


Figure 4.13 An Inherently Unroutable Layout.

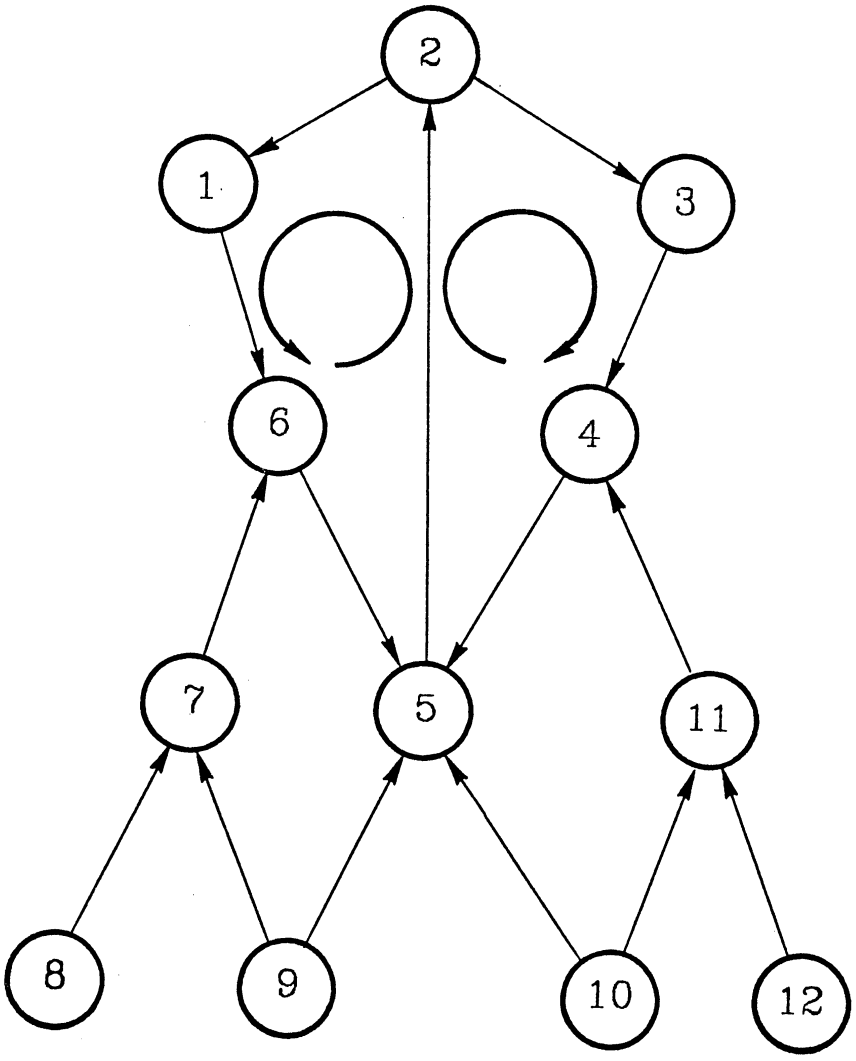


Figure 4.14 The Routing Order Graph of Figure 4.13.

4.4. Comparisons And Critiques

In this section, we will discuss the various proposed methods published in the literature to solve the problem we are addressing.

4.4.1. Classical Constraints

Kawanishi [56] proposes the following method to generate the order constraint graph in which routability can be tested.

Method I:

(1) If c_p **T** c_q, put a directed arc from c_p to c_q.

(2) For each parallel block P_{ij}, put a directed arc from every channel in C_{ij}-$\{c_i,c_j\}$ to both of $\{c_i,c_j\}$.

Theorem 4.2: If the constraint graph generated by the *Method I* is acyclic, then the layout is routable. However, the reverse is not true.

Proof:

We prove the Theorem by proving the negative statement to be true. The negative statement is that " if the layout is unroutable, then the constraint graph generated by *Method I* is cyclic." First that the layout is inherently unroutable implies there exists at least a directed loop in the *order graph* by Theorem 4.1. There are two kinds of loops. One is the loop consisting of only **T** constraints. This implies that the classical constraint graph is cyclic. The other is the one consisting of both constraints by **T** shape intersections as well as parallel blocks. This implies there exists a parallel block P_{ij} such that an arc of either G_u or G_b is in the directed loop of the *order graph* by *Theorem 4.1*. This case is shown in Figure 4.15. Since $S(G(P_{ij}))= S(G_u(P_{ij})) \cup S(G_b(P_{ij}))$, the classical constraint graph is cyclic.

The proof of the reverse part is shown by a counter example in Figure 4.16. By *Method II*, c_1,c_3, c_4,c_1 form a directed loop in the classical constraint graph. However, either $c_3\,c_4\,c_1c_2c_5$ or $c_3c_4\,c_1c_5c_2$ is a legal routing order.

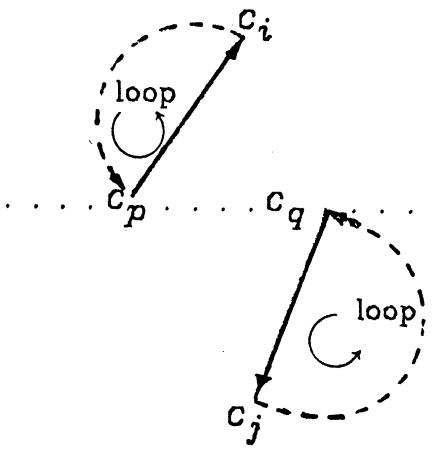
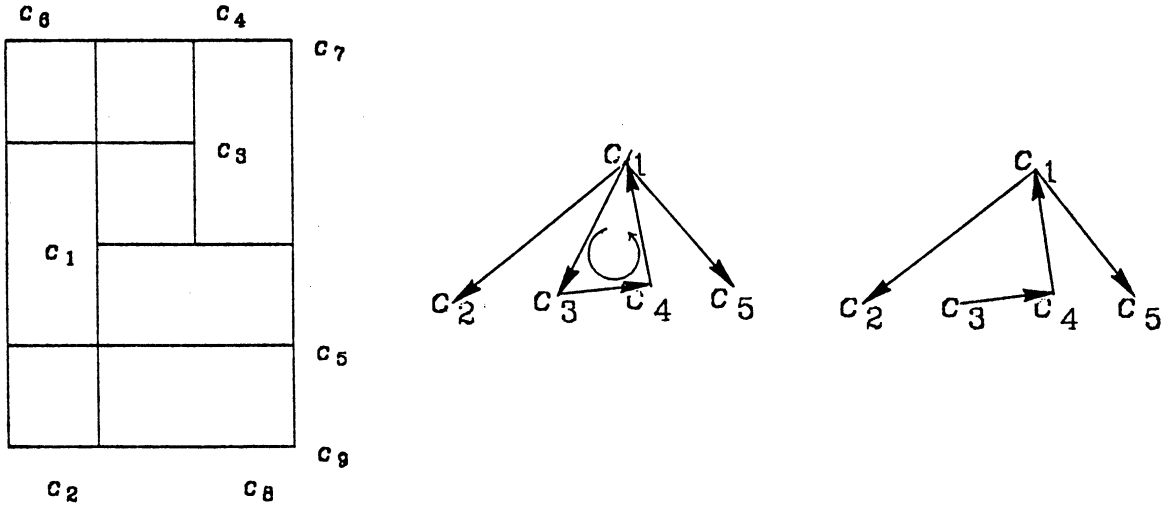


Figure 4.15 A Direct Loop in an Order Graph.



(a) (b). method I (c). Our method

Figure 4.16 A Counter Example for the Reverse Part.

Corollary 1: If the constraint graph generated by the *Method I* is acyclic, then every G_A(V,E) is acyclic.

Proof:

By the construction of Method I, the constraint graph contains the union of $G_b(P_{ij})$ and $G_u(P_{ij})$ which representing for the parallel block P_{ij}. Every graph $G_A(V,E)$ is a subgraph of the constraint graph constructed by *Method I*. If the constraint graph in Method I dosen't have any cycle, obviously $G_A(V,E)$ doesn't have any cycle.

4.4.2. Canonical Specifications

We do the following analysis by referring to the generalized parallel block in Figure 4.10, and we refer readers to the paper [55] for detail. In the following, we briefly describe the canonical specification and its corresponding constraint graph $G_{i_l j_r}(P_{ij})$.

The graph $G_{i_l j_r}(P_{ij})$ is constructed by the following steps.

(1) Specify $y(B_{i_l}, B_{j_r})$.

(2) Put an arc to c_i from every channel which is between c_{i_l} and c_i or between c_{j_r} and c_i.

(3) Put an arc to c_j from every channel which is between c_{i_l} and c_j or between c_{j_r} and c_j.

We construct the graph $G_B(V,E)$ as follows:

Method II:

(1) B is the vector to specify the canonical specification pair in every parallel block P_{ij}.

(2) If c_p **T** c_q, put a directed arc from c_p to c_q.

(3) For each P_{ij}, choose $G_{i_l j_r}(P_{ij})$ according to specification vector B.

There are $L_{ij}R_{ij}$ graphs $G_{i_l j_r}(P_{ij})$ associated with each P_{ij}. By our scheme in *Theorem 4.1*, we need to check only G_u and G_b. That is $\frac{L_{ij}R_{ij}}{2}$ reduction.

Lemma 4.5: $S(G_u(P_{ij})) \cup S(G_b(P_{ij})) = \bigcup_{all\ i_l j_r} S(G_{i_l j_r}(P_{ij}))$.

Proof:

(1) It is clear that $G_u(P_{ij}) = G_{i_l j_r}(P_{ij})$ if we choose both c_{i_l} and c_{j_r} to be c_j. Similarly for the case of $G_b(P_{ij}) = G_{i_l j_r}(P_{ij})$ if we choose both c_{i_l} and c_{j_r} to be c_i. Therefore $S(G_u(P_{ij})) \cup S(G_b(P_{ij})) \subset \bigcup_{all\ i_l j_r} S(G_{i_l j_r}(P_{ij}))$.

(2) Lemma 4.4 shows that both $G_u(P_{ij})$ and $G_b(P_{ij})$ generate all the legal routing orders. It is straightforward that $\bigcup_{all\ i_t j_r} S(G_{i_t j_r}(P_{ij})) \subset S(G_u(P_{ij})) \cup S(G_b(P_{ij}))$.

From (1) and (2), the equality holds.

■

Theorem 4.3: Every order graph G_A(V,E) is cyclic if and only if every G_B(V,E) is cyclic.

Proof:

(1) *Necessary condition:* It is obvious by following *Lemma 4.5.*

(2) *Sufficient condition:* We prove the negative statement of the Theorem by considering the only case in which G_B possibly produces an acyclic graph but not G_A. Figure 4.17 shows the case that the arcs $c_p \rightarrow c_j$ and $c_q \rightarrow c_i$ always form directed loops l_1 and l_2 under G_A. However, suitable G_B can get rid of both arcs and avoid both loops l_1 and l_2. In this case, G_B requires both arcs $c_p \rightarrow c_i$ and $c_q \rightarrow c_j$. Then a loop c_i, c_q, c_j, c_p, c_i is formed. Therefore there doesn't exist any acyclic graph under scheme G_B. This proves the necessary condition.

■

Theorem 4.3 suggests that our scheme is as powerful as that of Sato and Nagai [55] in the test of *routability*. However, our scheme is much simpler. Sato and Nagai [55] propose to test the routability of the layout by checking whether all G_B(V,E) are cyclic. The disadvantage of this method is that the number of G_B(V,E) is much bigger than that of our G_A(V,E). An example is for k parallel blocks and in each block there are L_{ij} channels in the left half and R_{ij} in the right half. Our scheme saves $\left\lfloor \frac{L_{ij} R_{ij}}{2} \right\rfloor^k$ cyclic tests of the graph. In general the cyclic test takes $\mathbf{O}(n^2)$ steps for the layout having n channels.

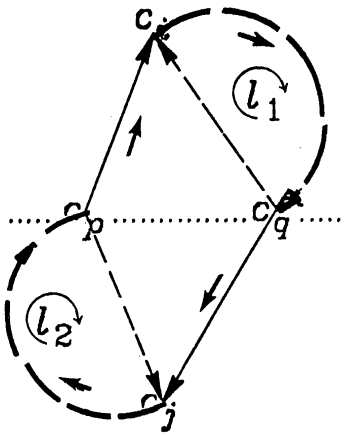


Figure 4.17 A Proof of Theorem 4.3.

4.4.3. Generalized T Constraints

Preas [24] proposes the following method for generating the order constraint graph in which routability can be tested.

Method III:

(1) If c_p **T** c_q, put a directed arc from c_p to c_q.

(2) Put an arc if two channels form a generalized **T** constraint as shown in Figure 4.18.

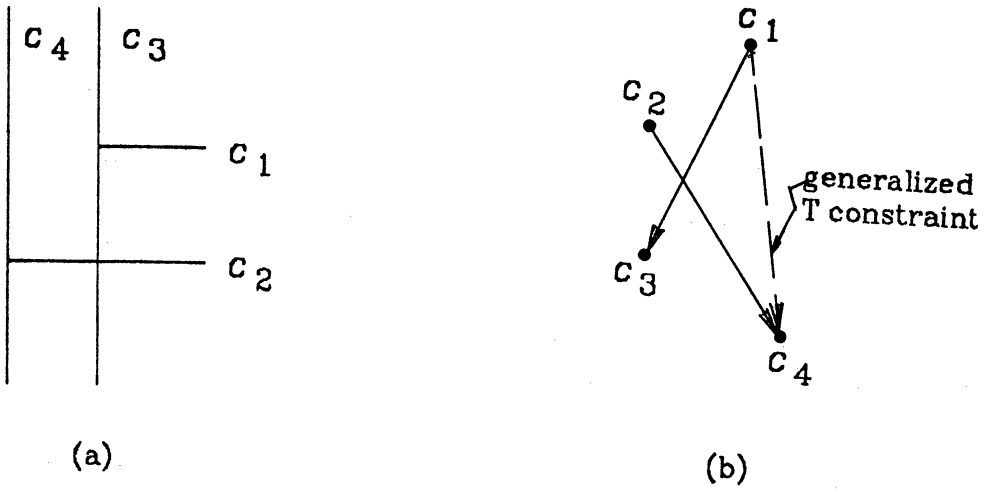


Figure 4.18 A Generalized T Constraint.

Theorem 4.4: If the constraint graph generated by *Method III* is acyclic,then the layout is routable. However, the reverse is not true.

Proof:

(1) The forward part is obvious.

(2) The reverse part is shown by the counter example in Figure 4.19. By *Method III*, $C_1c_4c_5c_6c_1$ form a directed loop in the constraint graph. However, the sequence c_2, c_4,c_5,c_6,c_1,c_7 is a legal routing order. Therefore, the layout is still routable.

Theorem 4.4 shows that the generalized **T** constraints are not absolutely necessary for testing *routability*.

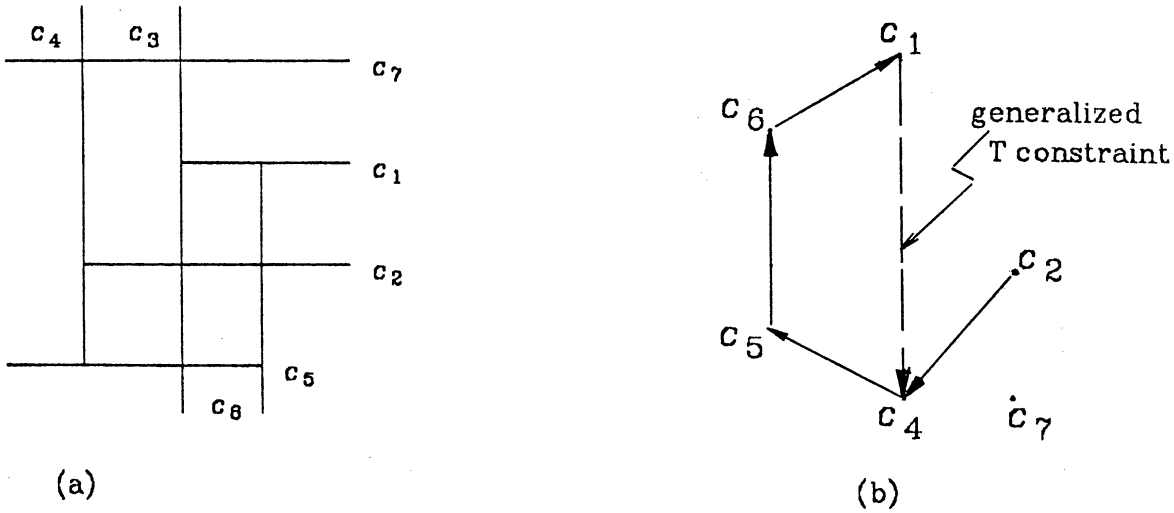


Figure 4.19 An Example for the Reverse Part of Theorem 4.4.

4.4.4. Algorithms For Generating Legal Routing Order

The comparison shows that previously published methods can be improved. We suggest the following algorithm for both testing *routability* and generating the *legal routing order*.

Algorithm Order:

(1) Construct any *order graph* $G_A(V,E)$ by randomly choosing a specification vector **A**.

(2) Perform the *acyclic test*.

(3) If it is acyclic, then perform a *topological sort* on $G_A(V,E)$ and stop.

(4) Otherwise, if there is any new specification, change to

a new **A** and goto step (1).

(5) Else if there has no new **A**, stop.

It is clear that the *Algorithm Order* always gives a legal routing order if the rough layout is routable. In the next section, we discuss the generation of the routing order for an unroutable layout.

4.5. Routing Order Generation for an Unroutable Layout

If the layout is unroutable, there are two possible cases in the directed cycle. The cycle is formed either by **T** constraints entirely or by the mixture of both **T** and the constraints due to the parallel blocks. For the first case, the rough layout is unroutable. The only way to find a routing order is to pick up a channel in the directed cycle and reserve a very wide space for that channel and then route that channel at last. While in the case of directed cycle consisted of mixed constraints, we can pick up and break one of the up or the bottom channel in the cross channel such that the directed cycle is broken. For example, we can choose channel c_5 in Figure 4.20 and break it into two channels c_5^l and c_5^r as shown in Figure 4.21. The constraint graph of Figure 4.21 does not have any directed cycle.

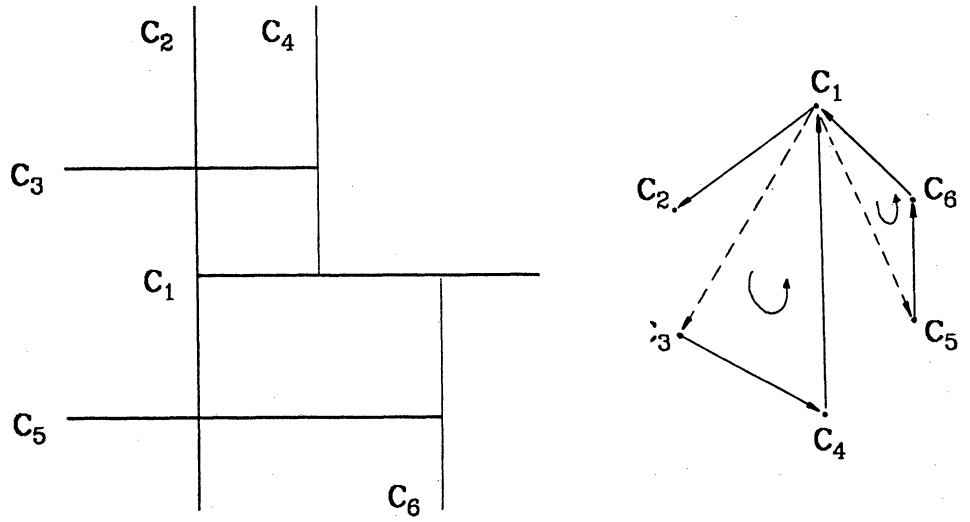


Figure 4.20 An Unroutable Layout

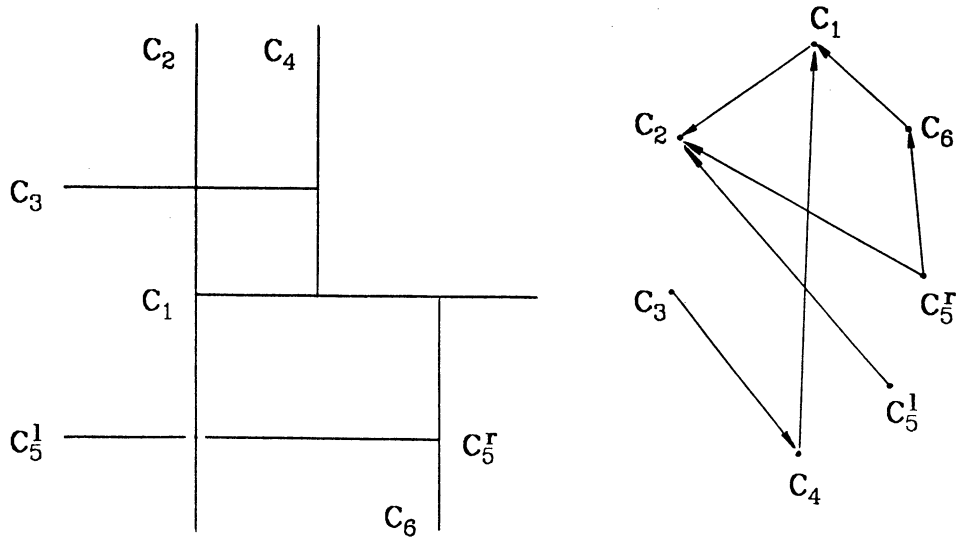


Figure 4.21 To Break the Directed Cycles by Breaking a Channel

4.6. Summary

In summary, the *routability problem* in the general cell approach for IC layout is treated rigorously. We study the *routability problem* of the layout from the point of view of the legal routing orders. Since the nature of the channel routing algorithm imposes some inherent routing order constraints on the channels, the number of legal routing order is reduced. We represent the legal routing orders in terms of a routing order constraint graph. The constraints among the channels are either due to **T** intersections or parallel blocks. For the case of a single block P_{ij}, the set of the legal routing orders cannot be represented by a single constraint graph, and must be represented by a union of constraint graphs. We, therefore, derive the minimum spanning graph $\{G_b(P_{ij}), G_u(P_{ij})\}$ which together generate all the legal routing orders for P_{ij}. This is a significant reduction, $\frac{L_{ij}R_{ij}}{2}$, in the number of the graphs over paper [55]. The global constraint graph is the *order graph* G_A(V,E) constructed by Definition 4.9. In terms of G_A(V,E), *Theorem 4.1* describes the necessary and sufficient conditions for the *routability* of the layout. Again, the number of the *order graph* G_A(V,E) is much

smaller than that of the so-called canonical specification graph $G_B(V,E)$ in [55]. Based on the conditions we derive, we show that previous methods for the *routability* problem [24], [56] are unnecessarily complex and and overly restrictive. In addition, an algorithm for testing routability and generating the legal routing orders is proposed.

CHAPTER 5

CHANNEL ROUTING PROBLEM

5.1. Introduction

As we state in Chapter 3, a channel is a space reserved for the inter-block wire routing. Our primary goal is to minimize the width of the channel. Figure 5.1 shows that a channel consists of three contributions, namely *flow-in-out connections*, *intra connections*, and *pass through connections*. The contribution due to *pass through connections* is a global optimization problem. It requires a complicated method to allocate the wire connections and even a sophisticated estimation of the global wire routing. However, the contribution due to both *flow-in-out connections* and *intra connections* can be computed exactly by the channel routing algorithm, which is the topic to be pursued in section 5.2.

In addition, there are several factors affecting the width of a channel. For example, the relative position between **cell #1** and **cell #2** affects the routability of a channel and, of course, the width of the given channel. This is called the *displacement problem*, which is studied in section 5.3.

5.2. Channel Routing Algorithms

In this section, we study the algorithms for different situations in a channel. The goal of the algorithms is to minimize the width of the given channel provided that the constraints are satisfied.

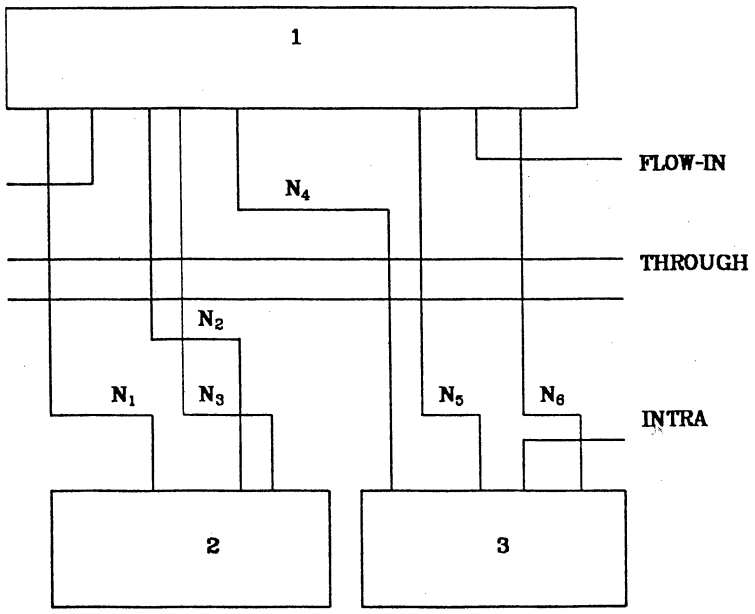


Figure 5.1 An Example of a Channel.

5.2.1. Problem Specifications

The set $\mathbf{N} = \{\mathbf{N}_1,\ldots,\mathbf{N}_n\}$ denotes the nets in a channel. Let $\mathbf{U_c} = \{u_1,u_2,\ldots,u_u\}$ be the coordinates of the grid points in the upper row of the channel, while $\mathbf{L_c} = \{l_1,l_2,\ldots,l_l\}$ be the coordinates of the grid points in the lower row of the channel. Also let $\mathbf{U}(\mathbf{N}_i) = \{u_i^1,u_i^2, \ldots, u_i^{u(i)}\}$ be the coordinates of the grid points in the upper row of the channel, which net $\mathbf{N}_i$ is connected. Similarly, $\mathbf{L}(\mathbf{N}_i) = \{l_i^1,l_i^2,\ldots,l_i^{l(i)}\}$ are the connecting points in the lower row.

Let t_{u_i} be a function from $\{u_1, \ldots, u_u\}$ to $\{\Phi, \mathbf{N}_1,\ldots,\mathbf{N}_n\}$. Similarly, t_{l_i} be a function from $\{l_1, \ldots, l_l\}$ to $\{\Phi, \mathbf{N}_1,\ldots,\mathbf{N}_n\}$. Both functions assign a net to each grid point in the lower and upper rows. Then both $\mathbf{U(C)} = \{ t_u^1, \ldots, t_u^i, \ldots, t_u^u \}$ and $\mathbf{L(C)} = \{ t_u^1, \ldots, t_u^i, \ldots, t_u^u \}$ are the channel specification.

5.2.2. No Vertical Constraints Case

A net **N** is represented by an interval I=[x,y] where x is the leftmost point and y is the rightmost point of the net **N**. The routing of the nets $\mathbf{N} = \{\mathbf{N}_1,\ldots,\mathbf{N}_n\}$ is to assign each net $\mathbf{N}_i \in \mathbf{N}$ a track such that there is no overlap between two

representative intervals $\mathbf{I}_i$ and $\mathbf{I}_j$.

The problem of the optimum channel routing without constraints* is:

Given: The set of the nets $\mathbf{N} = \{\mathbf{N}_1,\ldots,\mathbf{N}_n\}$ in a channel.

Find: Assign each net in $\mathbf{N}$ a track in which no two nets overlap each other and the number of the tracks in a channel is the minimum.

We first define the overlap relation **R** and construct an interval graph based on the relation **R**. Then we show that the optimum routing problem can be transformed into a graph theoretic problem.

Definition 5.1: $\mathbf{N}_1\mathbf{R}\mathbf{N}_2$ if and only if $\mathbf{I}_1$ and $\mathbf{I}_2$ overlap (*i.e.* $[x_1,y_1] \cap [x_2,y_2] \neq \Phi$).

Now let the set $\mathbf{N} = \{\mathbf{N}_1,\mathbf{N}_2,\ldots,\mathbf{N}_n\}$ be the set of nets and $\mathbf{E} = \{(\mathbf{N}_1,\mathbf{N}_2) \mid$ for *every* $\mathbf{I}_1,\mathbf{I}_2 \in \mathbf{V}$ and $\mathbf{I}_1\mathbf{R}\mathbf{I}_2\}$. Then we can construct a graph $\mathbf{G}=(\mathbf{N},\mathbf{E})$ which is an ***interval graph.*** Example 5.1 shows the construction of the interval graph corresponding to the channel in Figure 5.2.

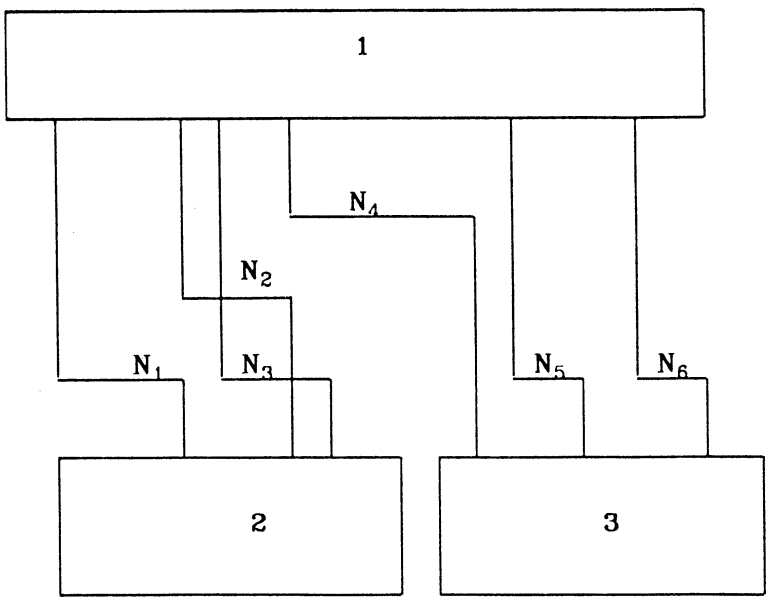


Figure 5.2 An Example for Channel Routing

* Note that there is strict difference between ***optimum*** and ***optimal*** in complexity theory. The set with property **P** is the ***optimum*** means the set is absolutely the best with respect to the property **P**. On the other hand, the set with the property **P** is ***optimal*** means the set is not contained in any set satisfying the property **P**. Similarly the definitions apply for ***maximum*** and ***maximal*** as well as ***minimum*** and ***minimal***.

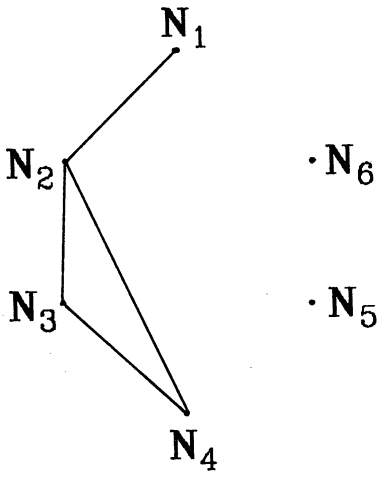
Example 5.1:

$$\mathbf{N} = \{\mathbf{N_1}, \mathbf{N_2}, \mathbf{N_3}, \mathbf{N_4}, \mathbf{N_5}, \mathbf{N_6}\}$$

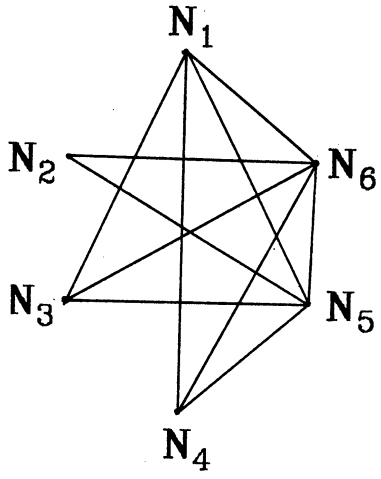
$$\mathbf{I} = \{[1,3],[3,5],[4,7],[5,9],[10,11],[12,13]\}$$

$$\mathbf{E} = \{(\mathbf{N_1}, \mathbf{N_2}), (\mathbf{N_2}, \mathbf{N_3}), (\mathbf{N_3}, \mathbf{N_4}), (\mathbf{N_2}, \mathbf{N_4})\}$$

THE INTERVAL GRAPH $\mathbf{G_I(V,E)}$:



THE COMPLEMENT GRAPH $\overline{\mathbf{G_I}}\mathbf{(V,E)}$:



Definition 5.2: A clique of a graph $\mathbf{G=(N,E)}$ is a set $\mathbf{S} \subseteq \mathbf{N}$ such that the induced subgraph of $\mathbf{G}$ on $\mathbf{S}$ is a complete graph.

Let $P = \{C_1, C_2, \ldots, C_m\}$ be a partition of N into cliques.

Also let $\Theta(G) = \min_{P}\{|P|\}$ be the smallest possible number of cliques into which N can be partitioned.

Definition 5.3: The minimum cliques partition problem of G is to find $P = \{C_1, C_2, \ldots, C_m\}$ such that $m=\Theta(G)$.

Theorem 5.1: The optimum channel routing problem is equivalent to the minimum cliques partition problem of the complement graph $\overline{G}$.

Proof:

Since $G=(N,E)$ represents the overlap relationships among the nets. It is clear that $\overline{G}$ represents the nonoverlap relationships. To get the minimum number of the tracks in a channel means to have the minimum number of partition cliques of $\overline{G}$.

By *Theorem 5.1*, we know the key point in solving the optimum routing problem is to solve the minimum cliques partition problem. Here we are interested in

(1) finding an efficient algorithm to compute the number $\Theta(\overline{G})$.

(2) finding an efficient algorithm for the minimum cliques partition problem.

The solution of (1) provides the exact number of the required tracks, hence we solve (1) first.

Definition 5.4: An independent set of a graph $G=(N,E)$ is a set $S \subseteq N$ such that no two distinct nodes of S are adjacent.

Let $\alpha(G) = \max_{\textit{all independent sets}}\{|S|\}$

It can be shown that $\alpha(\overline{G}) = \Theta(G)$.

Theorem 5.2 (Berge [61]): If $\mathbf{G}$ is an interval graph, then its complement graph $\overline{\mathbf{G}}$ is a comparability graph.

Theorem 5.3 (Berge [61]): Every comparability graph $\mathbf{G}$ is α-perfect (*i.e.* $\alpha(\mathbf{G})=\Theta(\mathbf{G})$).

From *Theorem 5.2*, $\overline{\mathbf{G}}$ is a comparability graph. Therefore $\alpha(\overline{\mathbf{G}})= \Theta(\overline{\mathbf{G}})$ follows **Theorem 5.3**. This say that if we can find the maximum independent set of $\overline{\mathbf{G}}$, and we know the minimum number of tracks needed. However, to find the maximum independent set of $\overline{\mathbf{G}}$ is equivalent to finding the cardinality of the maximum clique of $\mathbf{G}$.

The most efficient algorithm for finding the maximum clique in the literature is in Gavril's [62] which runs for $\mathbf{O(|N|+|E|)}$ by the application of a lexicographic breadth first search developed by Rose et al. [63].

Example 5.1: (continued)

The maximum clique of $\mathbf{G}$ for Figure 5.1 is the set $\{\mathbf{N_2},\mathbf{N_3},\mathbf{N_4}\}$, therefore the optimum channel routing takes 3 tracks as we show in Figure 5.1.

Algorithm TRACKS:

Input: A set of nets represented by a set of intervals $\mathbf{I} = \{ \mathbf{I}_1,\mathbf{I}_2,\ldots,\mathbf{I}_n \}$ in a given channel.

Output: The number of tracks required for the given channel (*i.e.* $\alpha(\mathbf{G})$).

Method:

(1) Construct the interval graph $\mathbf{G=(N,E)}$

(2) Generate the perfect ordering of $\mathbf{G}$ (by Rose's algorithm [63]).

(3) Find the maximum clique of $\mathbf{G}$ (by Gavril's algorithm [62]).

Step (1) takes $\mathbf{O}(|\mathbf{N}|^2)$, while both step (2) and (3) take $\mathbf{O}(|\mathbf{N}|+|\mathbf{E}|)$. In total, the algorithm takes $\mathbf{O}(|\mathbf{N}|^2)$.

We then solve (2). Since $\overline{G}$ is a comparability graph by definition, a transitive orientation can be assigned to $\overline{G}$. Hence, we can define a partial order set on $\mathbf{N}$ (where $\overline{G}=(\mathbf{N},\overline{E})$).

Theorem 5.4: (Dilworth [64]) Let $(\mathbf{N},\leq)$ be a partially ordered set. The minimum number of linearly ordered subsets (called *chains*) needed to partition $\mathbf{N}$ is equal to the maximum cardinality of a subset of $\mathbf{N}$ having no two members comparable (called *antichains*).

Theorem 5.4 assures that the complement graph $\overline{G}$ can be partitioned. The minimum partition is the solution of the optimum channel routing.

Algorithm PARTITION:

Input: A set of nets represented by the set of intervals $I = \{ I_1, I_2, \ldots, I_n \}$ in a given channel.

Output: The minimum width of channel routing for the given channel.

Method:

(1) Construct the interval graph $G=(N,E)$

(2) Construct the complement graph $\overline{G}$.

(3) Find the perfect ordering of $\overline{G}$.

(4) Find the minimum number of cliques for G (by Gavril's algorithm [62]).

In both the algorithms **TRACK** and **PARTITION**, we apply the algorithm devised by Gavril [62]. Gavril develops an algorithm to find the maximum cliques of the so called *chordal* graph to run in $O(|N|+|E|)$ steps provided that a perfect ordering has been established. He also developed a perfect ordering algorithm to run in $O(t(|N|+|E|)+|N|+E|)$ steps, where $t(|N|+|E|)$ is the time required to square the adjacent matrix of the graph. However, Rose et al. [63] develop an algorithm using a lexicographic search to generate a perfect ordering in $O(|N|+|E|)$ steps. Therefore both algorithms of **TRACKS** and **PARTITION** are used to solve *the*

problem of the optimum channel routing without constraints by combining Rose and Gavril's work.

Note that several people [1], [65] have proposed algorithms solving *the problem of the optimum channel routing without constraints*. In the paper [1], Hashimoto and Stevens devise the so-called **Left Edge** algorithm.

Left Edge Algorithm: (without constraints)

Fill each track successively by choosing the net with the minimum left endpoint among the *eligible set* of those which can be fitted to the current track until the track is full.

Note that in the paper [1], the *eligible set*

S_e^1 = { the nets with the left endpoint greater than the right endpoint of the the last placed net in the current track }.

Although it is a kind of heuristic to select the next candidate by finding the net with the minimum left endpoint in the *eligible set*, the **Left Edge** algorithm does provide the optimum solution in the non-constraint case. There are several methods to define the so-called *eligible set*. The definition of the *eligible set* provides a broad basis for devising the heuristic algorithms. In the later section, we define the *eligible sets* and devise the new algorithms based on the *eligible sets* for the problem under constraints.

5.2.3. Constraint's Case

Section 5.2.2 deals with the channel routing without any vertical constraints. The overlap relations fully characterize the routing problem. In some cases, however, in order to fully characterize the routing problem we need not only the overlap relations but also the vertical constraints. Figure 5.3 shows the case where net $\mathbf{N_2}$ must be laid upper net $\mathbf{N_1}$, net $\mathbf{N_3}$ be upper net $\mathbf{N_2}$ etc. This kind of geometric restrictions is called *vertical constraints*. In the next section,

a directed *constraint graph* G_c is used to model the vertical constraints among the nets in the channel. The interval graph G_I together wit G_c fully characterize the channel routing problem. By following this kind of formulation, there are two cases to be considered. The cases depend on whether there exists directed cycles in the graph G_c. An example with directed cycles in G_c is shown in Figure 5.4.

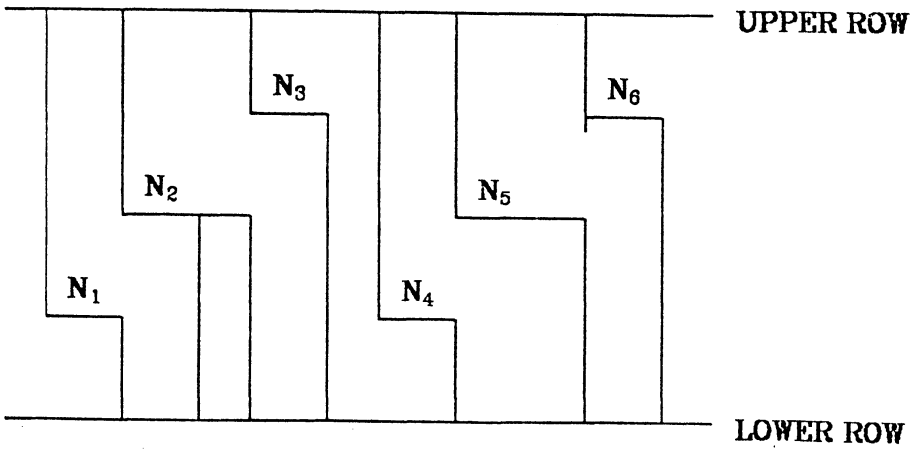


Figure 5.3 A Channel with Vertical Constraints.

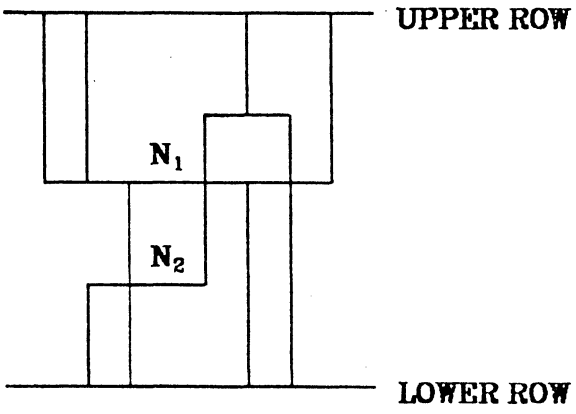
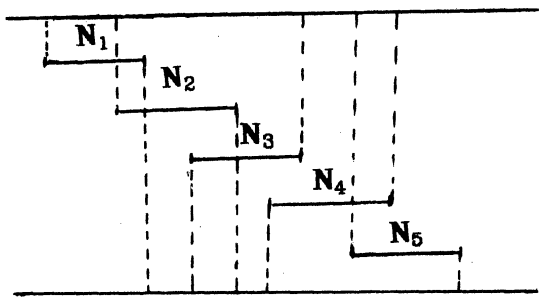
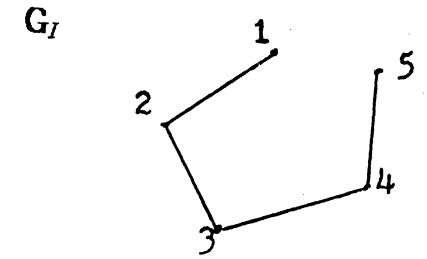
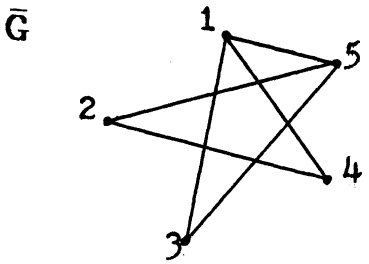


Figure 5.4 An Example for a Channel with Directed Cycle in G_c

Example 5.2:







The minimum number partition of cliques for $\bar{G}$ is { {N_1, N_3,N_5}, {N_2,N_4} } , therefore only two tracks are required in the case.

5.2.3.1. Acyclic Constraints

In this section, we first define the constraint graph G_c. Then *the problem of the optimum channel routing with vertical constraints* is introduced. A *reduced graph* G_r is formed by combining both G_c and G_I. We develop a heuristic channel routing algorithm based on G_r.

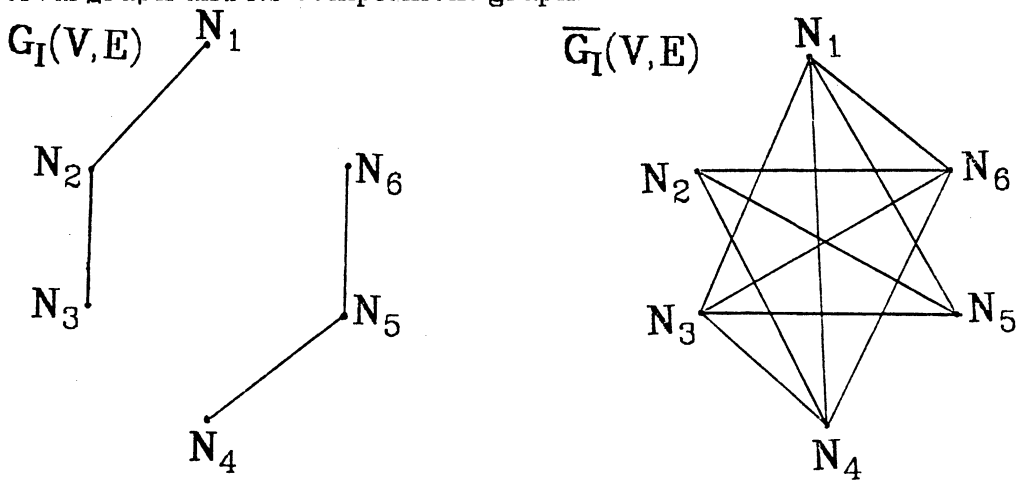
Definition 5.5: N_i R_c N_j if and only if $U(N_i) \cap L(N_j) \neq \Phi$.

Definition 5.6: The constraint graph is $G_c(N,E)$. Where the set $N = \{N_1, N_2, \ldots, N_n\}$ be the set of nets and $E = \{(N_1, N_2) \mid$ for *every* $I_1, I_2 \in V$ and $I_1 R_c I_2\}$.

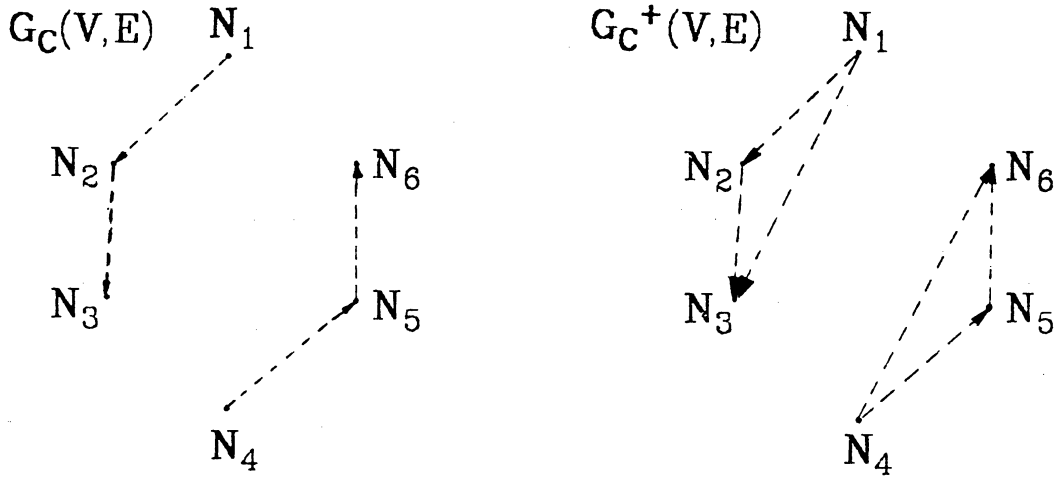
Example 5.3:

This example shows the interval graph and its complement graph as well as the constraint graph and its transitive closure of the Figure 5.3.

The interval graph and its complement graph:



The constraint graph and its transitive closure:



The problem of the optimum channel routing with vertical constraints is:

Given: $N = \{N_1, \ldots, N_n\}$.

Find: Assign each net in N a track such that

(1) No two nets overlap each other in the same track.

(2) Net $\mathbf{N}_i$ should be laid above net $\mathbf{N}_j$ if $\mathbf{N}_i$ precedes net $\mathbf{N}_j$ in the transitive closure $\mathbf{G}_c^+$.

(3) The number of the tracks in the channel is the minimum.

Definition 5.7: A reduced graph $\mathbf{G}_r$ is a graph constructed by deleting the edges from $\overline{\mathbf{G}}_I$ when there exists a directed arc between the pair of nodes in the graph $\mathbf{G}_c^+$.

The reduced graph $\mathbf{G}_r$ and the closure of the constraint graph $\mathbf{G}_c^+$ fully characterizes the channel routing problem with vertical constraints. Figure 5.5 shows the reduced graph for Figure 5.3.

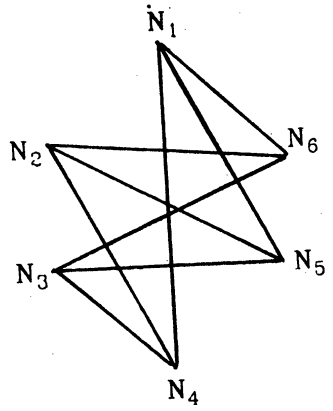


Figure 5.5 The Reduced Graph of Figure 5.3.

Theorem 5.5: The problem of the optimum channel routing with acyclic vertical constraints is equivalent to finding the minimum cliques partition of the graph $\mathbf{G}_r$ under the constraints $\mathbf{G}_c^+$.

Proof:

It is clear that the reduced graph $\mathbf{G}_r$ is no longer a comparability graph. There is no efficient algorithm for solving the minimum cliques partition problem for $\mathbf{G}_r$ even for the non-constraint case. It belongs to the class of the most difficult problems in complexity theory. It is an **NP**-complete problem [60]. It is very natural to infer that the minimum cliques partition problem for $\mathbf{G}_r$ under

the restriction G_c^+ is also an **NP**-complete problem. By our formulation, the problem of the optimum channel routing under vertical constraints is an **NP**-complete problem. In the past decade, people simply assume that the optimum channel routing problem under vertical constraints is an **NP**-complete problem and hence search the alternative solutions -*heuristic algorithms* [1], [66]-[68] formally showed that the problem is actually an **NP**-complete problem by reducing *the circular arcs coloring problem* to *the problem of the optimum channel routing under constraints*.

Now the fact that the problem of the optimum channel routing with vertical constraints is indeed an **NP**-complete problem demands the development of the heuristic algorithms. There are two classes of algorithms among heuristic algorithms. One is the *dogleg* version; that is, the algorithm allows the nets to be divided and assigned into several tracks by inserting *jogs* [23], [67], [69]-[72]. However, the inserted *jogs* increase the capacitance of the entire net, and hence decrease the performance of the circuit. One way to avoid this is to minimize the inserted *jogs*. The other way is by using another version of the heuristic algorithms, namely, the algorithms do not allow any inserted *jogs* in the channel [1], [59]. The *Left Edge* algorithm for vertical constraints cases is as follows:

Left Edge Algorithm: (with vertical constraints)

Fill each track successively by choosing the net with the minimum left endpoint among the *eligible set* of those that can be fitted to the current track until the track is full.

Here we show how to generate the *eligible set*. First, we assign a level to each net by the following algorithm:

Level Algorithm

Assign a level to each net such that

(1) no two nets are in the same level if there exists an arc from $\mathbf{N}_i$ to $\mathbf{N}_j$ in $\mathbf{G}_c^+$ and

(2) level ($\mathbf{N}_i$) < level ($\mathbf{N}_j$) if there exists an arc from $\mathbf{N}_i$ to $\mathbf{N}_j$ in $\mathbf{G}_c^+$.

Then, the *eligible set*

$\mathbf{S}_e^2$ = { the nets have the properties: (1). They are, if any, in the same level of the last placed net, otherwise the nets with one level greater than the last placed one. (2). They are the nets with the left endpoint greater than the right endpoint of the last placed net in the current track }.

Example 5.3: (continued)

The *Left Edge* algorithm gives the solution { {$\mathbf{N}_1$, $\mathbf{N}_4$}, {$\mathbf{N}_2$, $\mathbf{N}_5$} , {$\mathbf{N}_3$, $\mathbf{N}_6$} } which is shown as Figure 5.3. In this case, the number of the required tracks is three, and hence is the optimum solution.

Although in some cases the *Left Edge* algorithm gives the optimum solution, yet, for n nets in a channel, LaPaugh [68] shows that $\frac{Solution_{Left\ Edge}}{Solution_{Optimum}} \leq \sqrt{n}$, where $Solution_{Left\ Edge}$ means the solution by applying the *Left Edge* algorithm, similarly for $Solution_{Optimum}$. Figure 5.6 shows the worst case with $\frac{Solution_{Left\ Edge}}{Solution_{Optimum}} = \sqrt{n}$, where $\mathbf{n} = \mathbf{m}^2$ is the number of nets to be routed. Figure 5.7 shows the union of the interval graph $\mathbf{G}_I$ and the constraint graph $\mathbf{G}_c$. The *Left Edge* algorithm give a solution with $\mathbf{m}^2$ tracks while the optimum solution only need $\mathbf{m}$ tracks. Figure 5.8 shows the optimum solution of Figure 5.6. The reason for the bad performance is that the *Left Edge* algorithm always looks for the candidate from the same level or one level higher than the last placed net. This puts a severe restriction on searching for the next candidate. To

remove the restriction, we define the *eligible set*

S_e^3 = { the nets are *compatible* with the last placed net, and with the left endpoint greater than the right endpoint of the last placed net in the current track }.

Note that the set S_e^3 differs from the set S_e^2 by discarding the level restriction in S_e^2, and hence enhances the degree of freedom in searching for the next candidate. We also like to define the *compatible* relation.

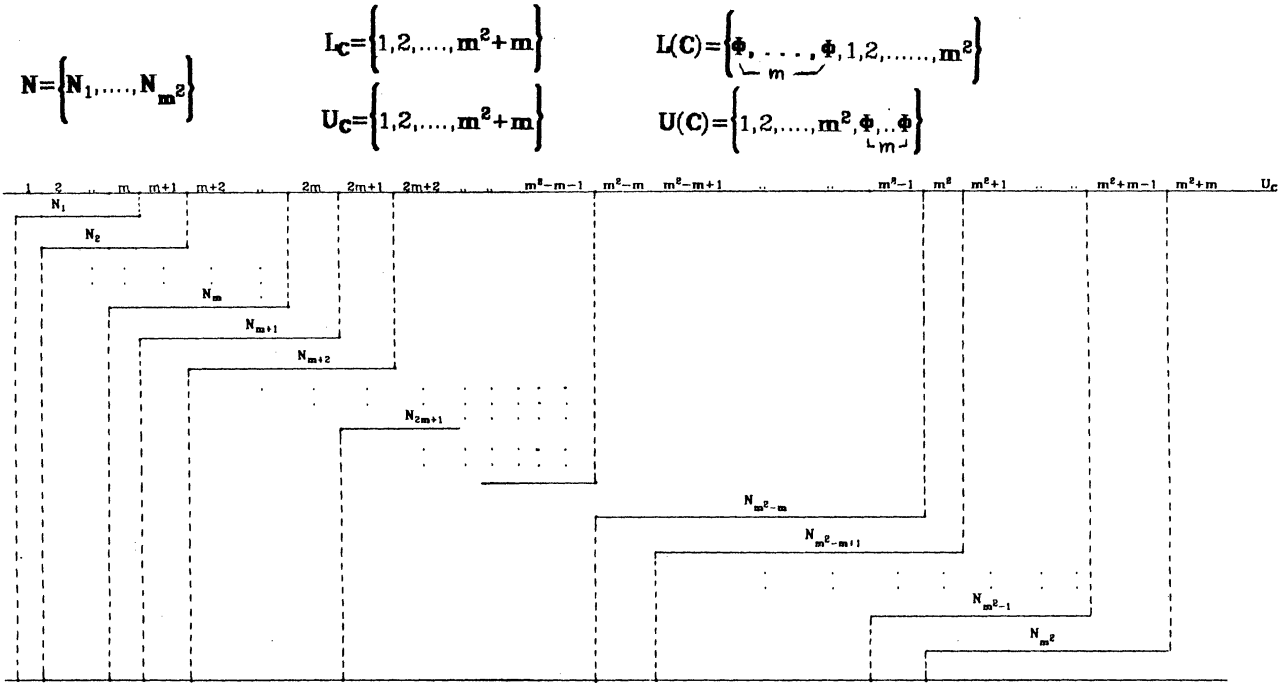


Figure 5.6 The Worst Case for the Left Edge Algorithm (taken from [68])

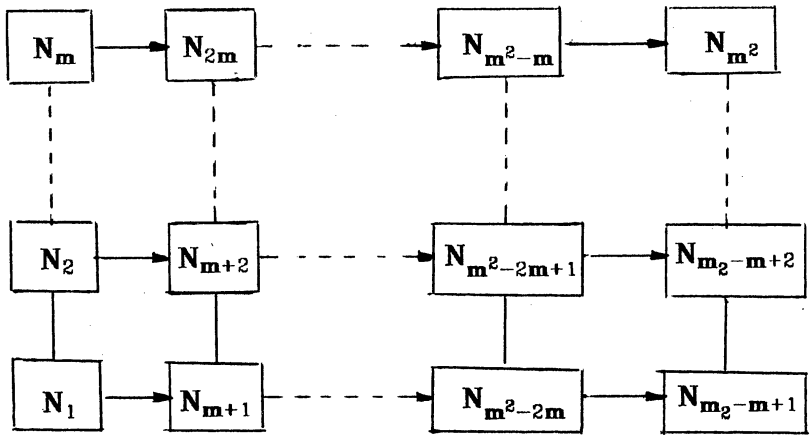


Figure 5.7 The Union of the Interval Graph G_I and the Constraint Graph G_c

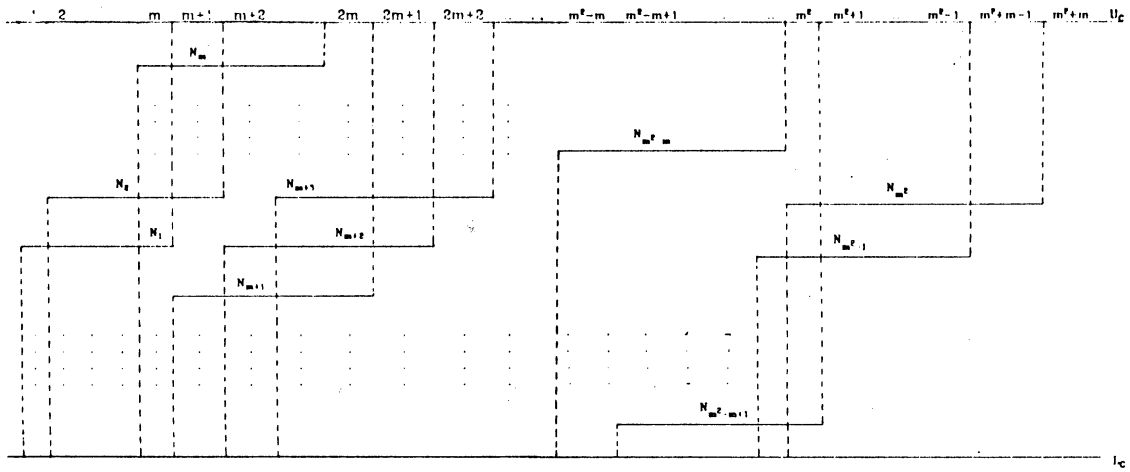


Figure 5.8 The Optimum Routing of Figure 5.6

Definition 5.8: Two nets N_i and N_j are compatible if there exists an edge between N_i and N_j in G_r.

In the algorithm, we dynamically update the reduced graph G_r by removing the edges when there are any new constraints due to the new placed net.

Modified Left Edge Algorithm

Begin

Generate the graph G_r;

Do until all nets are routed

Start at a new track;

Do until the current track is full

Search another net with the minimum left endpoint in *eligible set* S_e^3;

Update the reduced graph G_r.

End;

End;

End Modified Left Edge;

Example 5.4:

In this example, we use Figure 5.9 to demonstrate how the algorithm works. Figure 5.10 and Figure 5.11 show the interval graph and the vertical constraint graph respectively. Now the reduced graph G_r is shown in Figure 5.12. At first we choose N_2 and N_{11} for track 1. Then we need to update the reduced graph G_r as shown in Figure 5.13. Since track 1 is full, we start at track 2 by choosing N_3, N_5, and then choosing N_{12}. Note that when we choose N_3 and N_5, we remove the following edges: (N_1,N_7), (N_1,N_8), (N_1,N_9), and (N_1,N_{10}). Again we need to update the reduced graph G_r as shown in Figure 5.14. Now we choose N_1 and N_6 for track 3. We keep executing the **Modified Left Edge** algorithm, and finally get the following result:

Track	Nets
1	N_2, N_{11}
2	N_4
3	N_1, N_6
4	N_3, N_5, N_{12}
5	N_9
6	N_7
7	N_8
8	N_{10}

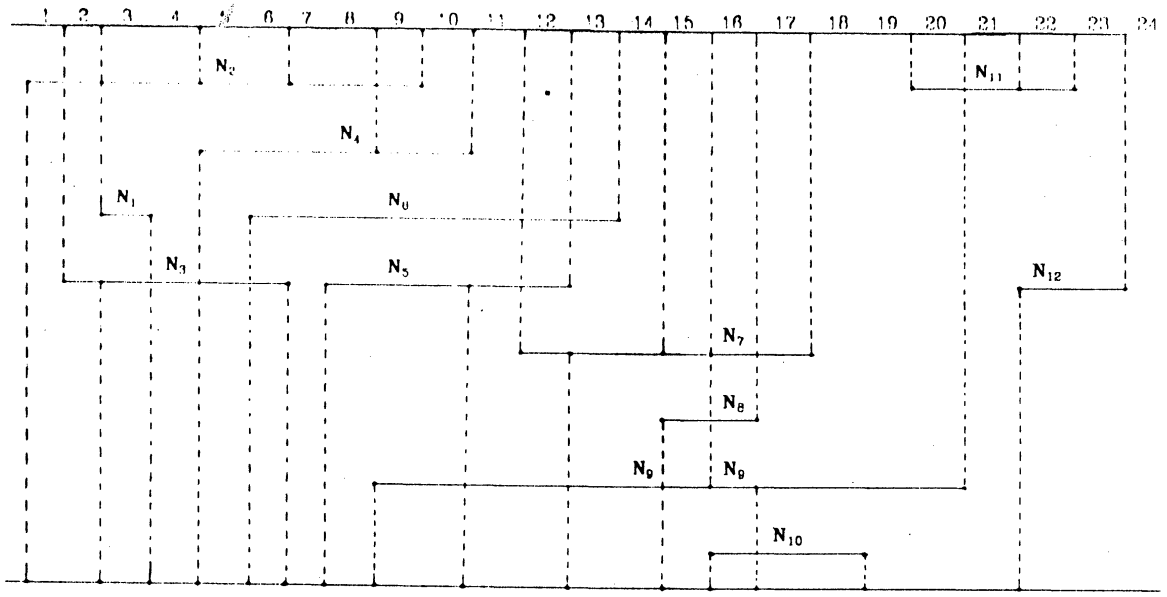


Figure 5.9 A Channel to Be Routed

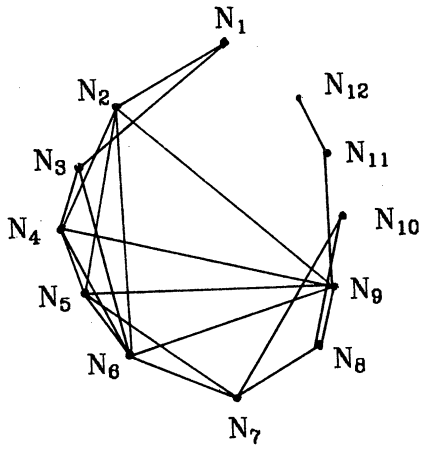


Figure 5.10 The Interval Graph G_I of Figure 5.9.

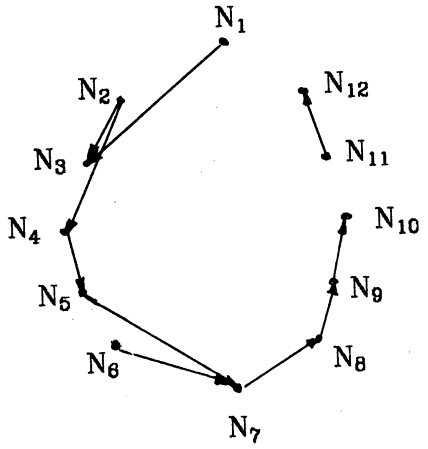


Figure 5.11 The Constraint Graph G_c of Figure 5.9.

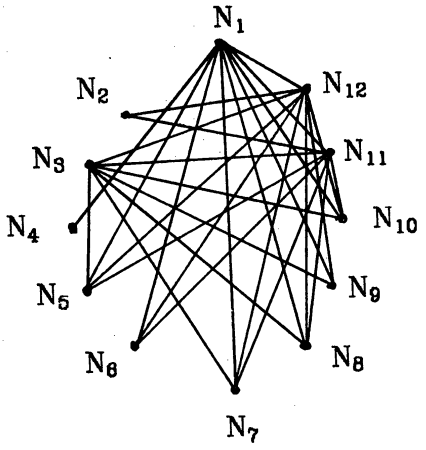


Figure 5.12 The Reduced Graph G_r.

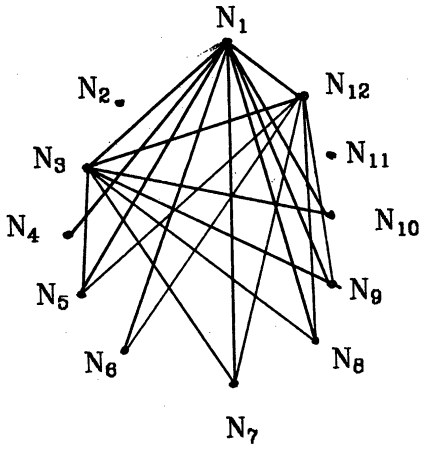


Figure 5.13 The Updated Reduced Graph G_r (1st Iteration)

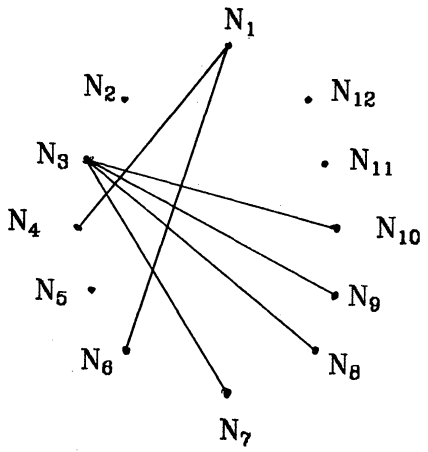


Figure 5.14 The Updated Reduced Graph G_r (2nd Iteration)

The result is the optimum. The major difference between the *Left Edge* and *Modified Left Edge* algorithms is the relaxation of the level restriction, and hence the dynamic updates of the reduced graph G_r. Now let us turn to the *Left Edge* algorithm. First of all, the *Level* algorithm gives the following result:

Level	Nets
1	$\mathbf{N_1}$, $\mathbf{N_2}$, $\mathbf{N_6}$, $\mathbf{N_{11}}$
2	$\mathbf{N_3}$, $\mathbf{N_4}$, $\mathbf{N_{12}}$
3	$\mathbf{N_5}$
4	$\mathbf{N_7}$
5	$\mathbf{N_8}$
6	$\mathbf{N_9}$
7	$\mathbf{N_{10}}$

Then we assign $\mathbf{N_2}$ and $\mathbf{N_{11}}$ to track 1. Next we assign $\mathbf{N_1}$ and $\mathbf{N_6}$ to track 2. Since the set at level 1 is empty at this point, we move to level 2. Now we can assign only $\mathbf{N_3}$ and $\mathbf{N_{12}}$ to track 3 because that $\mathbf{N_5}$ is at level 3 and still not available yet. The result is as follows:

Track	Nets
1	$\mathbf{N_2}$, $\mathbf{N_{11}}$
2	$\mathbf{N_1}$, $\mathbf{N_6}$
3	$\mathbf{N_3}$, $\mathbf{N_{12}}$
4	$\mathbf{N_4}$
5	$\mathbf{N_5}$
6	$\mathbf{N_9}$
7	$\mathbf{N_7}$
8	$\mathbf{N_8}$
9	$\mathbf{N_{10}}$

The *Left Edge* algorithm needs 9 tracks while the *Modified Left Edge* algorithm only needs 8 tracks, which is the *optimum*.

Channel	Nets	Terminals	Left Edge Algorithm	Modified Left Edge Algorithm	Optimum* Algorithm
**1	m^2	$2m^2$	m^2	m	m
2	6	8	3	3	3
3	12	25	9	8	8
4	5	10	2	2	2
5	21	40	14	12	12
6	63	187	22	20	20
7	45	90	18	15	15
8	55	128	23	22	17

** the worst case for the **Left Edge** algorithm

* Branch-and-bound method [66].

Table 5.1 The Experimental Results of the Channel Router

The *Modified Left Edge* algorithm needs $\mathbf{O(n^3)}$ steps. An *APL* version of the *Modified Left Edge* algorithm is in *APPENDIX II*. Table 5.1 shows some experimental results of the *Modified Left Edge* Algorithm. Seven out of eight cases are the optimum solutions. The results show significant improvement over the *Left Edge* algorithm.

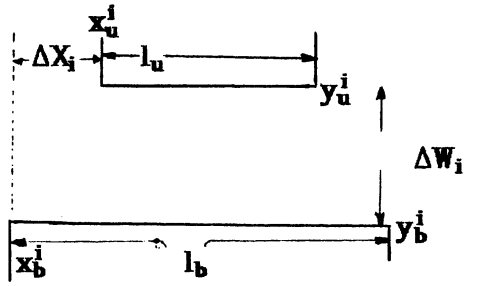
One extension of the algorithm is to allow *doglegs* to be inserted. This can be achieved by dividing each net into several subnets , and then treating the subnet as an individual net in the algorithm.

5.2.3.2. Cyclic Constraints

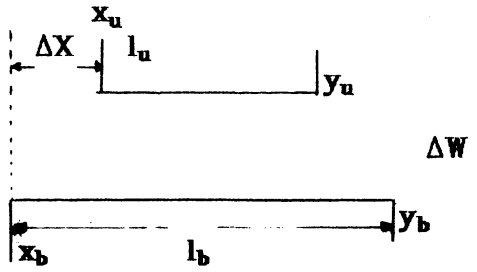
If the constraints graph has directed cycles, the channel is no longer routable. The only way to route the channel is to open the directed cycles in the constraint graph. There are two ways to break the cycles. One is to insert *doglegs* in the nets; while the other is to stretch a cell (*i.e.* to insert some space in a cell) and then insert *doglegs*. Both cases are very difficult processes. Neverthless, they are interesting problems, in terms of complexity theory and practical usefulness, for finding a good heuristic algorithm to solve them.

5.3. Displacement Problem

Besides good algorithms affecting the width of a channel, the following *Displacement Problem* also affects the channel width.



$\Delta X_i = x_b - x_u$
$\Delta W_i = y_u - y_b$



$\Delta X = x_b - x_u$
$\Delta W = y_u - y_b$

There exists a $\Delta X \neq \Delta X_i$ such that $\Delta W \leq \Delta W_i$

Figure 5.15 The Displacement Problem

Displacement Problem

Given: A set of nets in a channel

Find: The relative *displacement* ΔX such that the channel width ΔW is the minimum.

Figure 5.15 illustrates the *Displacement problem*. No quick solution is available except the exhaustive search. Since the density of a channel is the lowest bound of the channel width, we develop an exhaustive method to search for the best ΔX such that the density is the minimum. We make the assumption that every net must have connections in the upper and lower row of the channel. In this case, we only need to search the range $-l_b - 1 \leq \Delta X \leq l_u + 1$ since the density is always equal to the number of nets in the given channel when ΔX is outside the range. For the case when ΔX is within the range, we need to find the maximum density in the channel with length less than $l_u + l_b$. For each ΔX, we need to spend $(l_u + l_b)log(l_u + l_b)$ in time. Therefore the total time is $(l_u + l_b)^2 log(l_u + l_b)$. The method discussed above is good only if we want to minimized the channel density. However, the minimum density, in general, doesn't imply the minimum channel width. The general *Displacement problem* is still an open problem. For a special case, Leiserson and Pinter [73] have found the optimum algorithm for the river router.

5.4. T Shape Router

The **T** shape router performs the routing in the **T** intersection part of two channels. Figure 5.16 shows a **T** shape router. The algorithm is:

Algorithm T-Shape Router:

(1) Route the channel A and leave the connecting points around the boundary of channel B.

(2) Route channel B.

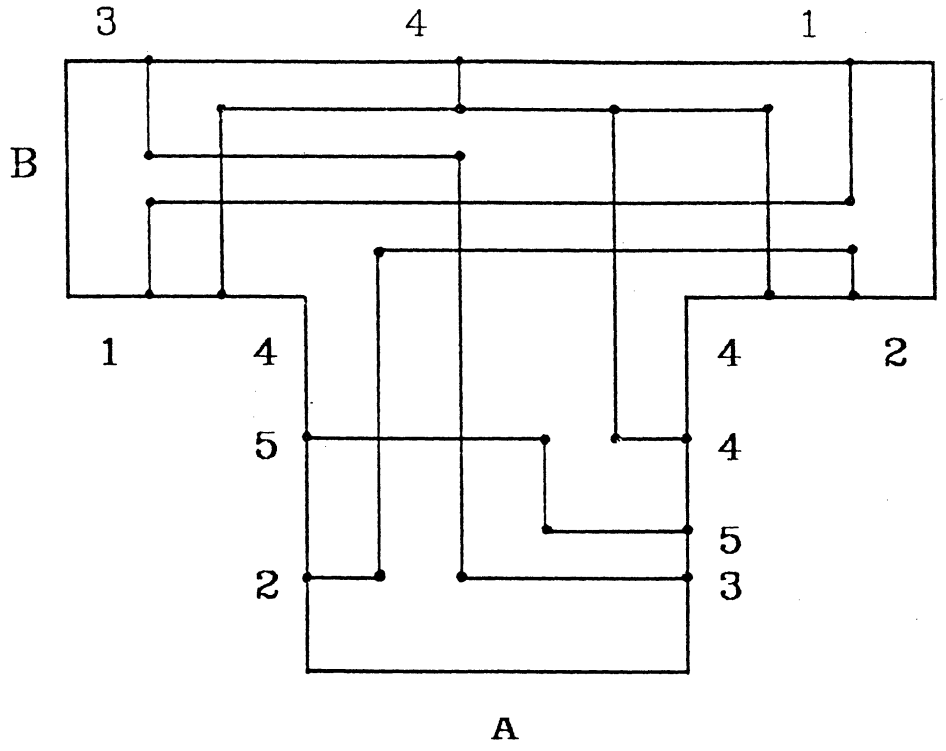


Figure 5.16 A **T** Shape Router

5.5. Cross Channel Router

The router performs the routing of the cross portion of intersection of two channels. Figure 5.17 shows an example.

Algorithm Cross Channel Router:

(1) Route any one channel first and leave the connecting points around the boundary of the cross portion.

(2) Route the remaining channel.

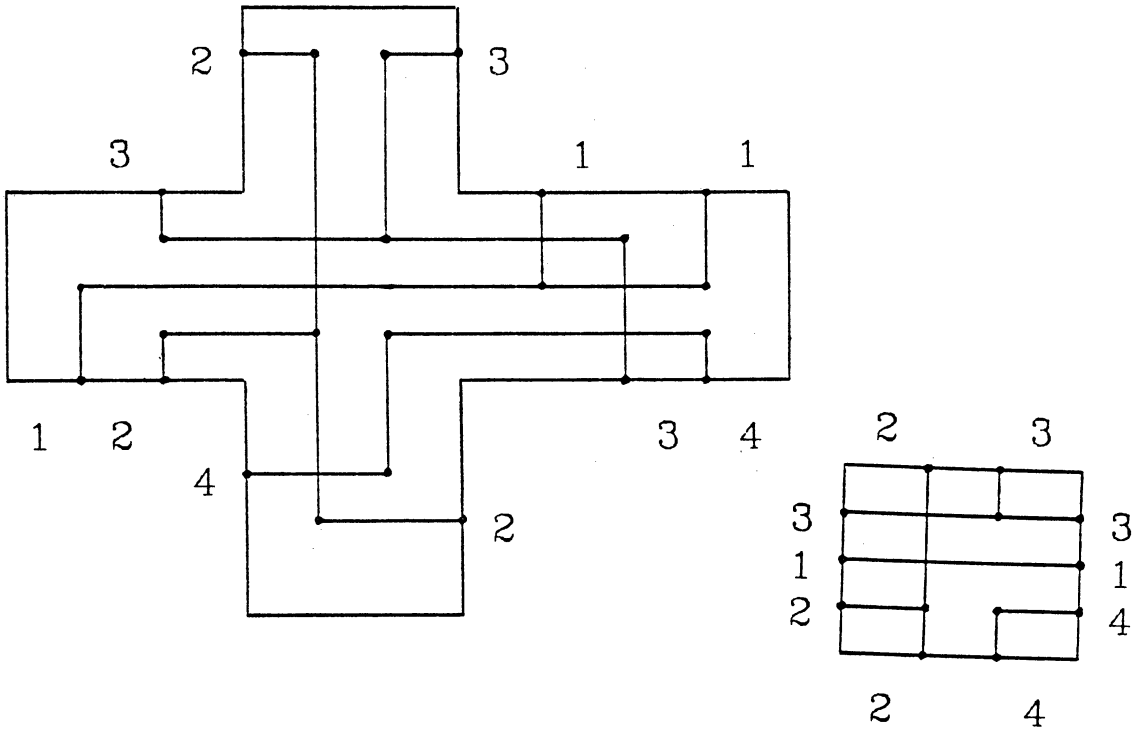


Figure 5.17 A Cross Channel Router

CHAPTER 6

ESTIMATES OF A SINGLE MODULE

6.1. Introduction

The Arithmetic Design System (ADS) supports the exploration of the use of many non-standard number systems in addition to standard two's complement. An overview of the ADS is available in [74]. The ADS traverses three levels of abstraction: a behavioral (application) level, an arithmetic (digit-vector) design level, and a realization level. At the realization level, time and space complexity are evaluated in terms of an MOS programmable logic array (PLA) model.

Figure 6.1 shows a PLA floor plan in which input and their complement lines running vertically through AND-plan, while the product terms are represented by horizontal lines run through both AND and OR plans. The output lines run through the OR plan vertically by connecting each output line to a set of selective product terms. Figure 6.2 shows the electric circuit model.

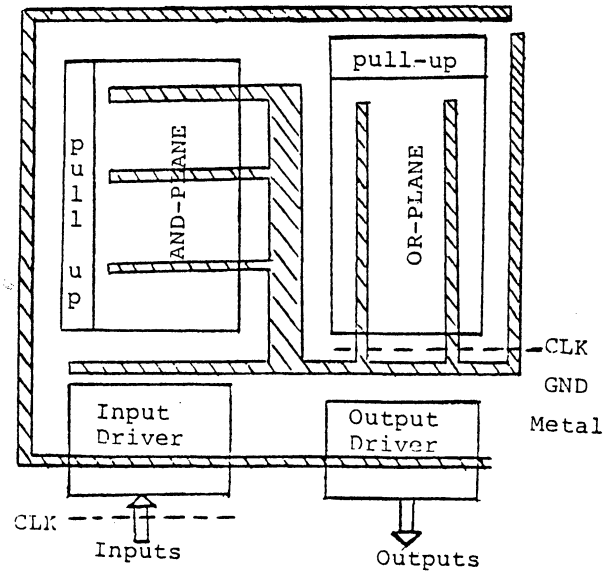


Figure 6.1 A PLA Floor Plan

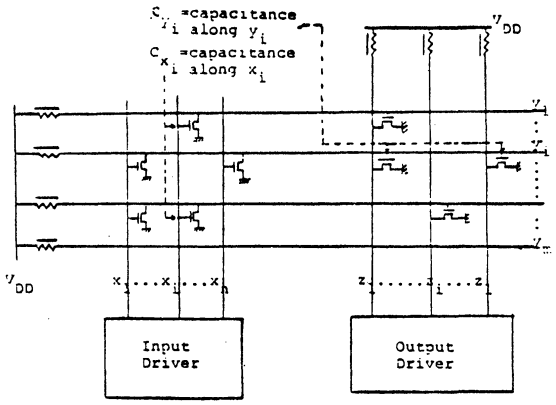


Figure 6.2 The Electric Circuit Model for PLA

We will discuss the area reduction techniques for a PLA in the later sections. Some of the techniques such as logic minimization and phase selection have been developed over a long time. On the other hand the techniques such as two bit decoder and folding were developed recently due to the progress of VLSI technology. Undoubtedly, most of the available techniques are very complicated, and therefore are time consuming. In other words, the problem sizes in the VLSI era are too big to get the optimum solution under those techniques in a reasonable amount of time. A major concern is whether it is worth applying the techniques to each problem instance. The goal for the techniques mentioned above is to reduce the area of a PLA. In general, we ask the question "how much area is saved by these techniques?". Besides the theoretical interest, the answer of the question has practical implications. It would be useful to find out some bounds for the amount of area saved by the application of those techniques before spending a large amount of time and ending up with a tiny amount of saving. Some research has been done along these lines. Mileto and Putzolu [75] find the average number of minterms in a two level logic minimization given the density of the logic function. In section 6.3, we develop a simple model to estimate the area saved by the folding process.

6.2. Area Reduction Methods for PLA

6.2.1. Logic Minimization

Many research efforts have been devoted to this classical research area. The problem is given a logic specification, find the minimum cost realization in terms of the given primitive logic units [76]. However, recently much effort has been devoted to the minimization problem of multiple outputs functions under two level restrictions. The result has direct application to PLA realization. The two level logic minimization of multiple outputs is an **NP**-complete problem. In general, there are two approaches to tackling the problem. One is to find out all the prime implicants and then choose a minimal cover set [77], [78]. The other is to begin at a small set of prime implicants and then either by using expansion or shrink operations to generate a minimal cover set [79], [80].

6.2.2. Two Bit Decoder

In conventional PLA, input and their complement lines are directly connected to the product terms. The two bit decoder is a way to reduce the number of the product terms. The two bit decoder first groups each pair of inputs together and then connects to the lines of the product terms. Suppose that a function **f**(A,B,C,D) is given. We can group A and B as well as C and D together such that **f**(A,B,C,D) = **F**(**g**(A,B),**h**(C,D)). Note that the two bit decoder doesn't change the number of input lines of a PLA.

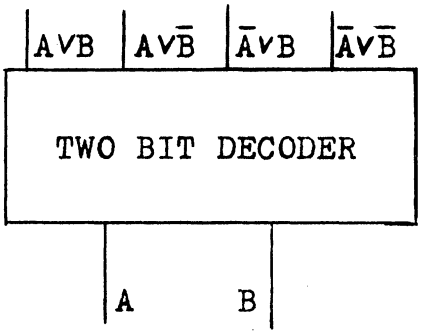


Figure 6.3 A Two Bit Decoder

Figure 6.3 shows a two bit decoder. Example 6.1 shows the application of a two bit decoder on a full adder. In this case, the two bit decoder reduces the number of product terms from 7 to 4. Paper [81] carefully studies the techniques of the two bit decoder. However, the optimum partition of a PLA by using two bit decoders is very difficult.

Example 6.1:

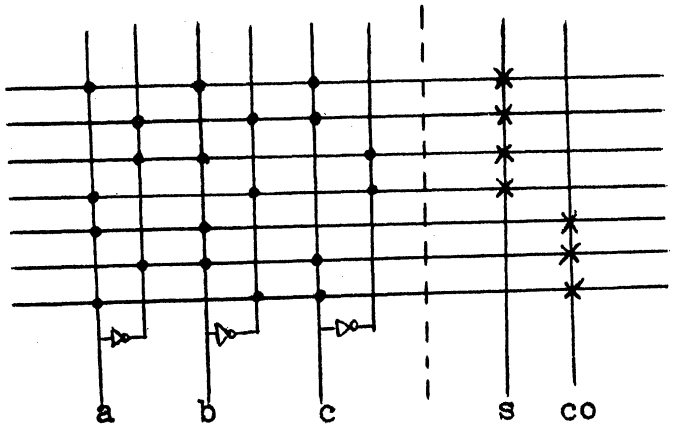
The truth table and their logic minimization of a full binary adder:

$$s = abc+\bar{a}\bar{b}c+\bar{a}b\bar{c}+a\bar{b}\bar{c}$$

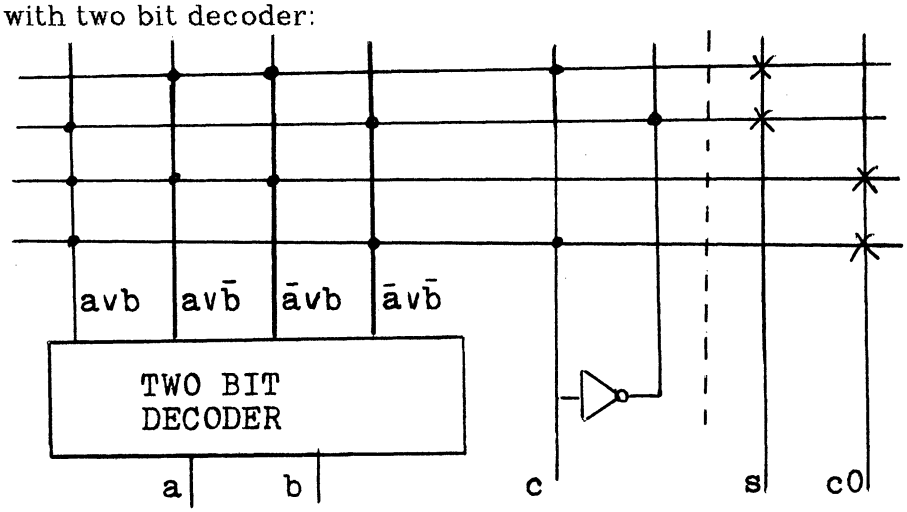
$$c_o = ab+\bar{a}bc+a\bar{b}c$$

a	b	c	s	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The original PLA:



The PLA with two bit decoder:



2.3. Phase Selection

Both the functions f and the complement $\bar{f}$ can be minimized and with a different set of product terms. If the set for the complement function $\bar{f}$ is smaller than that of function f, then *phase selection* chooses the complement function $\bar{f}$ to be minimized. For example, the minimization of $f(a,b,c) = \sum(1,4,5,7) = \bar{b}c+a\bar{b}+a\bar{c}$, while $\bar{f} = \sum(0,2,3,6) = \bar{a}c+bc$. In this case, we choose $\bar{f}$ to be minimized.

6.2.4. Partition

Usually, a PLA is very sparse. Partition is a scheme to divide the big PLA into several small PLAs such that the final total area is less than that of a single PLA. Note that a portion of area must be set for the connections among the

small decomposed PLAs. Therefore, the final total area of the small PLAs is not necessarily less than that of the single PLA. In general, figuring out the connection area is a difficult problem and needs further study. Here we assume that the connection area is not the dominating factor. Under this assumption, the following shows why the partition gives a smaller area.

Suppose that in a given PLA, there are n_i inputs, n_o outputs, and p_o product terms. Then the area of a single PLA is

$$Area_{single\ PLA} = (2n_i+n_o)p_o = 2n_ip_o+n_op_o$$

Now suppose that the single PLA can be partitioned into k small PLAs. Let n_i^j, n_o^j, and p_j denote the numbers of inputs, outputs, and product terms respectively in the j's decomposed PLA.

$$p_o = p_1+p_2+....+p_k$$

$$n_o = n_1+n_2+....+n_o$$

$$n_i^j \le n_i$$

Now the final total area of the small decomposed PLAs is

$$\begin{aligned} Area_{small\ PLAs} &= \sum_{j=1}^{k}(2n_i^j+n_o^j)p_j \\ &\le 2n_ip_o+\sum_{j=1}^{k}n_o^jp_j \\ &\le 2n_ip_o+n_op_o \\ &= Area_{single\ PLA} \end{aligned}$$

The inequality follows:

$$n_op_o = \sum_{j=1}^{k}n_o^j\sum_{j=1}^{k}p_j \ge \sum_{j=1}^{k}n_o^jp_j$$

Now the PLA partition problem is defined as follows:

PLA Partition Problem

Given: A single PLA with n_i inputs, N_o outputs, and P_o product terms.

Find: A set of small PLAs which partition the given PLA.

Without introducing the redundancy of the product terms and outputs in the small decomposed PLA, the problem can be described as

$$Min \sum_{j=1}^{k} (2n_i^j + n_o^j) p_j$$

$$Subject\ to \begin{cases} \sum_{j=1}^{k} p_j = p_o \\ \sum_{j=1}^{k} n_o^j = n_o \\ \sum_{j=1}^{k} n_i^j \geq n_i \end{cases} \qquad \text{(Partition A)}$$

If we removed the redundancy constraints, then the problem can be described as

$$Min \sum_{j=1}^{k} (2n_i^j + n_o^j) p_j$$

$$Subject\ to \begin{cases} \sum_{j=1}^{k} p_j \geq p_o \\ \sum_{j=1}^{k} n_o^j = n_o \\ \sum_{j=1}^{k} n_i^j \geq n_i \end{cases} \qquad \text{(Partition B)}$$

Both problems are extremely difficult. In the following, we develop an algorithm that decomposes the big PLA into a set of small decomposed PLAs. Each decomposed PLA is a group of outputs when they share the same product terms. In this case, the constraints in *Partition A* is satisfied. Now let's form a bipartite graph $\mathbf{G}_B = (\mathbf{V}_o \cup \mathbf{V}_p, \mathbf{E})$, where $\mathbf{V}_o$ denotes the nodes set for the outputs, while $\mathbf{V}_p$ does so for the product terms. Whenever an output is a function of a product term, there is an edge between the nodes in the bipartite graph. Now we state the decomposition algorithm as follows:

Algorithm Decomposition

Input: A bipartite graph $\mathbf{G}_B = (\mathbf{V}_o \cup \mathbf{V}_p, \mathbf{E})$.

Output: A set of small decomposed PLAs.

Methods:

```
Procedure  CONNECTS(G_B)
begin
    for i from 1 to |V_o| do
        begin
            visited(i):=0;
            end;
    for i from 1 to |V_o| do
        begin
            if visited(i)=0 then CLUSTER(i);
            end;
    end CONNECTS;

Procedure  CLUSTER(v)
begin
    visited(v):=1;
    Queue(v);
    while Queue≠Φ do
        begin
            Dequeue(v, Queue);
            for all vertices w adjacent to v do
                begin
                    if visited(w)=0 then do
                                        begin
                                        Queue(w);
```

```
                    visited(w)= 1;
                    end;
                end;
            end;
        end CLUSTER;
```

The functions *Queue* and *Dequeue* are the operations on a queue either to put or to remove an item from it. The worst case of the decomposition algorithm is $\mathbf{O}(n_i n_o p_o)$. A version of the algorithm in *APL* is in *APPENDIX III.*

Example 6.2:

A set of equations describing an 8-bit multiplier with 40 outputs, 83 products, and 39 inputs are taken from [46]. We apply the decomposition algorithm to the set of equations and get 33 clusters which are merged together into two groups:

Group 1:

```
s1 = ¬MPY(1)*¬MPY(2)
s2 = ¬MPY(1)*MPY(2)
s3 = MPY(1)*¬MPY(2)
s4 = MPY(1)*MPY(2)
"1 = s1*start
"2 = s3*B(8)
"3 = s4*"10
"4 = s4*¬"10
"5 = p*"1
"6 = p*"1+p*s4
"7 = p*s3+p*"3+p*"1+p*s2+p*"4
"8 = p*s2+p*s4
"9 = p*s2+p*"2+p*s4
```

"10 = */MCOUNT

"11 = s3+"1

"12 = s3+s2+"4

"13 = s2*BUS(1)+s4*A(8)

"14 = "2*COUT(1)

CCOUT(1:2) = MCOUNT(1:2)*CCOUT(2:3)

CCOUT(3) = MCOUNT(3)

CSUM(1:2) = MCOUNT(1:2)@CCOUT(2:3)

CSUM(3) = MCOUNT(3)@1D1

Group 2:

COUT(1:7) = R(1:7)*A(1:7)+R(1:7)*COUT(2:8)+A(1:7)*COUT(2:8)

COUT(8) = R(8)*A(8)

SUM(1:7) = R(1:7)@A(1:7)@COUT(2:8)

SUM(8) = R(8)@A(8)@0D1

Note that there are 13 inputs, 24 outputs, and 27 product terms in *Group* 1, while there are 24 inputs, 16 outputs, and 54 product terms in *Group* 2. The total final area of two small PLAs is 5022 units *vs* 9784 units of the original PLA. A half of the original area has been saved through the decomposition algorithm.

6.2.5. Folding

Since the density of PLA is generally very sparse, we can put two rows or columns of PLA in the same physical line on the floor plan. The process of finding the set of row pairs or the set of the column pairs to share the same physical lines is called PLA *folding*. There are two kinds of folding processes. The optimum versions for both processes are **NP**-complete. Since it is impossible to get the optimum solution , some heuristic algorithm should be developed. A modified version of the algorithm in [45] has been coded in *APL* and shown in

APPENDIX IV. We show two folding processes based on Figure 6.4.

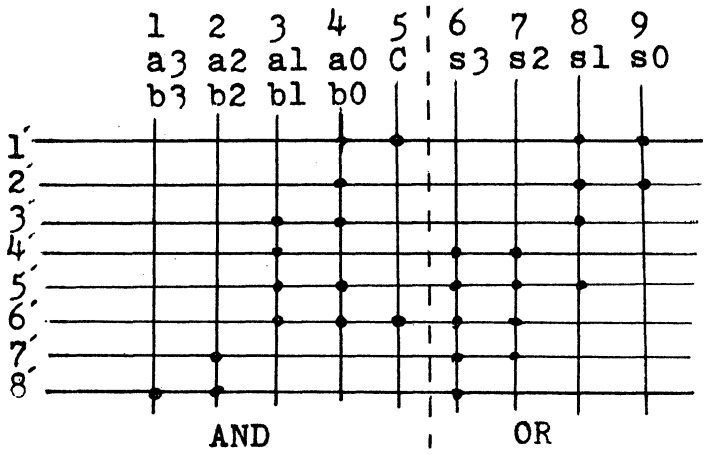


Figure 6.4 An Example for PLA Folding

6.2.5.1. Row Folding

The *optimum row folding problem* for a PLA is to find a *column permutation* which allows a maximum cardinality of row pairs to be implemented in an array in such a way that each row pair shares the same physical line.

An optimum row folding for Figure 6.4 is shown in Figure 6.5.

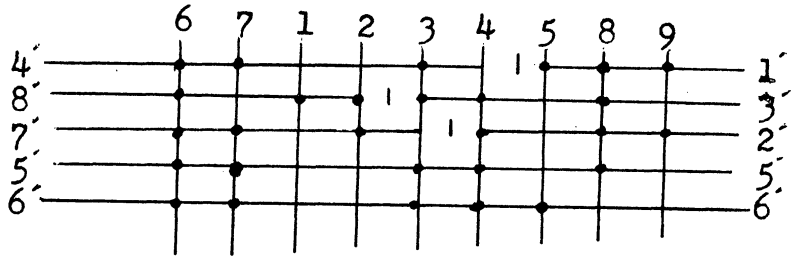


Figure 6.5 An Optimum Row Folding for Figure 6.4

6.2.5.2. Column Folding

The *optimum column folding problem* for a PLA is to find a *row permutation* which allows a maximum cardinality of row pairs to be implemented in an array in such a way that each column pair shares the same physical line.

An optimum column folding for Figure 6.4 is shown in Figure 6.6.

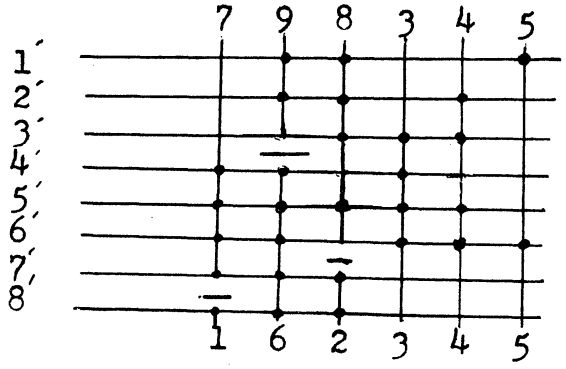


Figure 6.6 An Optimum Column Folding for Figure 6.4

6.3. A Model for Predicting Area Reduction due to Folding

6.3.1. Introduction

A theory of VLSI design requires the development of concrete figures of merit which are computable in a reasonable time even for systems with million of elements. Analysis of the analog behavior of VLSI circuits, for example, is emerging to be quite different from classical network theory and requires not only different analysis techniques but also the asking of fundamentally different questions [82], [83]. While exact calculations of time and/or area complexity may be practically impossible, useful upper and lower bounds may be computationally simple. These bounds may be quite adequate to guide the exploration of a design space in an automated design system. Furthermore in the specific case considered in this paper, the bounds can be made tighter, and thus serve as better estimates, by additional computation.

The work described in this section is in this spirit. The *optimum PLA folding* is an **NP**-complete problem. A heuristic procedure for PLA folding, with the worst case behavior taking $O(N^3)$ steps, has been proposed in [45]. In the Arithmetic Design System we are interested in estimating the area which can be saved

through PLA folding without carrying out the full folding procedure. In this section, we develop a model base on the well-known Rent's rule to predict the area being saved. Remember that Rent's Rule is

$$P = kC^r \quad , \quad 0 \leq r \leq 1$$

where P is the average number of terminals required by a group containing an average of C gates, k is the average number of terminals per gate, and r is a constant relating to the structure of a given logic network. In the next section, we use both k and r as parameters of the logic network.

6.3.2. Bound on the Saved Area Ratio for a Row Folded PLA

A Programmable Logic Array (PLA) can be described as a block shown in Figure 6.7. In the case of a PLA, the parameter C in Rent's rule can be replaced by W, the number of product terms (i.e. $P=kW^r$). The resultant area is defined as

$$A_{unfold} = PW.$$

Since a typical PLA is sparse, it is worth considering area reduction (and possible time delay reduction) through folding. This analysis is restricted to (optimal) row folding of the PLA. The *optimal row folding problem* for a PLA is to find a *column permutation* which allows a maximal cardinality of row pairs to be implemented in an array in such a way that each row pair shares the same physical line.

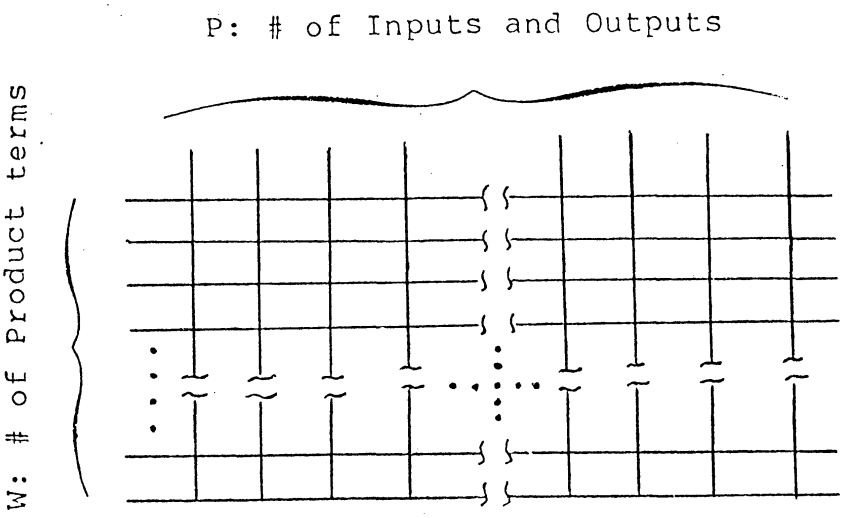


Figure 6.7 PLA Block Structure

The algorithm for the optimal row folding problem of a PLA will transform Figure 6.7 into Figure 6.8. The resultant area of the folded PLA is A_{fold}, where $S_i, 0 \leq i \leq P-1$ represents the number of product terms with a physical cut at column position i.

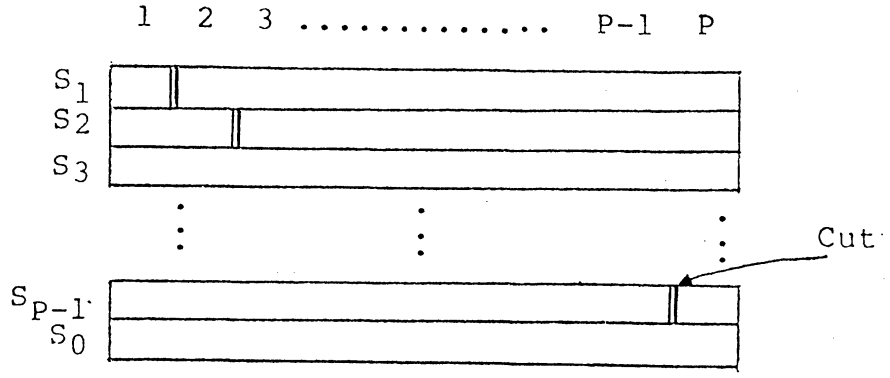


Figure 6.8 PLA after Row Folding.

The *saved area ratio* is defined to be

$$\delta = 1-\Gamma \quad \textit{where} \quad \Gamma = \frac{A_{fold}}{A_{unfold}}.$$

Figure 6.9 represents a transformation of the structure in Figure 6.8 in which cuts occurring within a window in the center of the PLA are used to define two sub arrays with P_{11} and P_{12} terminals, respectively, and W_1 terms each. It serves as an intermediate, conceptual structure in the derivation of the bounds on the saved area ratio. The rows without any cuts and those with cuts outside the center window are collected together in the area of Figure 6.9 with S_0 terminals. (An ALTO program developed by General Electric can perform this decomposition [84].) The area of the PLA in Figure 6.9 is

$$A_P = P_2(W_1 + S_0).$$

where

$$P_2 = P_{11} + P_{12}.$$

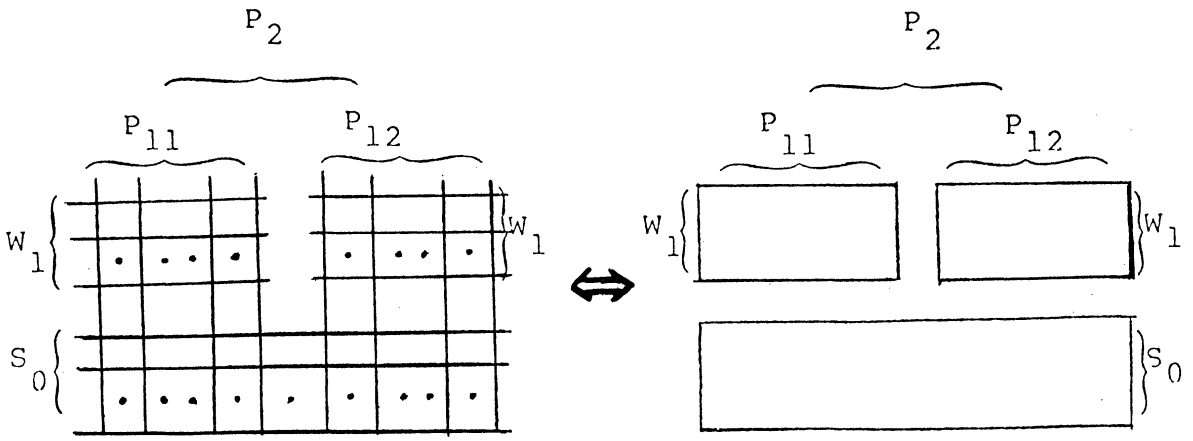


Figure 6.9 Partition Model of PLA.

The structure in Figure 6.9 is an intermediate model between the unfolded PLA and the totally (row) folded structure, and is used to find bounds on δ in terms of the Rent's rule exponent r. It is obvious that

$$\delta \leq \frac{1}{2}$$

since at best, half of the rows can be folded over the other half. Since the decomposition algorithm producing the Figure 6.9 structure never produces fewer rows than that of the optimal structure (Figure 6.8), and generally more, it also follows that

$$A_P \geq A_{fold}$$

and therefore that

$$\delta \geq 1 - \frac{A_P}{A_{unfold}}.$$

Assuming that the numbers of terminals for the two subarrays in Figure 6.9 are equal, it will now be shown that

$$\frac{A_P}{A_{unfold}} = 1 - \frac{1}{2^{\frac{1}{r}}}$$

The bounds on the saved area ratio follow immediately.

Given that a logic network can be characterized by k and r it follows that, $1 - \frac{A_p}{A_{unfold}}$ is a function of k and r, $f(k,r)$, for a given logic network. Figure 6.9 is a special case of the general model of a segmented matrix [84] and an application of Rent's rule on the model can yield a lower bound on $f(k,r)$, and therefore one for δ since $f(k,r) \leq \delta$. We assume that on the average $P_{11}=P_{12}$ in the partition and denote the number of terminals for either sub-array in Figure 6.9 by P_1. Note that this assumption implies that the PLA can be partitioned. Therefore the application of the results derived in this paper are only applicable for those cases satisfying the assumptions. Obviously, a PLA with a column fully personalized (sometimes the case for control logic implemented in PLAs) is unfoldable, in which case these results do not apply.

We have

$$P_1 = P_{11} = P_{12} \qquad (assumption) \tag{1}$$

$$P_2 = 2P_1$$

$$P_i = kW_i^r \qquad 1 \le i \le 2 \qquad (Rent's\ rule) \tag{2}$$

$$W_2 = S_0 + 2W_1 \qquad (by\ model) \tag{3}$$

From (2) and (3) we have

$$S_0 = W_2 - 2W_1$$
$$= \left(\frac{P_1}{k}\right)^{\frac{1}{r}}\left[2^{\frac{1}{r}} - 2\right]$$

(a) For the unfolded case:

$$A_{unfold} = PW_2$$

where

$$P = 2P_1$$

$$W_2 = \left(\frac{2P_1}{k}\right)^{\frac{1}{r}}$$

$$A_{unfold} = 2P_1\left(\frac{2P_1}{k}\right)^{\frac{1}{r}}$$

(b) For the model in Figure 6.9.

$$A_P = 2P_1\left[W_1 + S_0\right]$$
$$= 2P_1\left[\left(\frac{P_1}{k}\right)^{\frac{1}{r}} + \left\{\left(\frac{P}{k}\right)^{\frac{1}{r}}\left[2^{\frac{1}{r}} - 2\right]\right\}\right]$$
$$= P_1\left(\frac{P_1}{k}\right)^{\frac{1}{r}}\left[2\times 2^{\frac{1}{r}} - 2\right]$$

From (a) and (b):

$$\frac{A_P}{A_{unfold}} = 1 - \frac{1}{2^{\frac{1}{r}}}$$

$$f(k,r) = \frac{1}{2^{\frac{1}{r}}}$$

and therefore,

$$\frac{1}{2^{\frac{1}{r}}} \leq \delta \leq \frac{1}{2}$$

Figure 6.10 shows the upper and lower bounds for δ with respect to Rent's exponent.

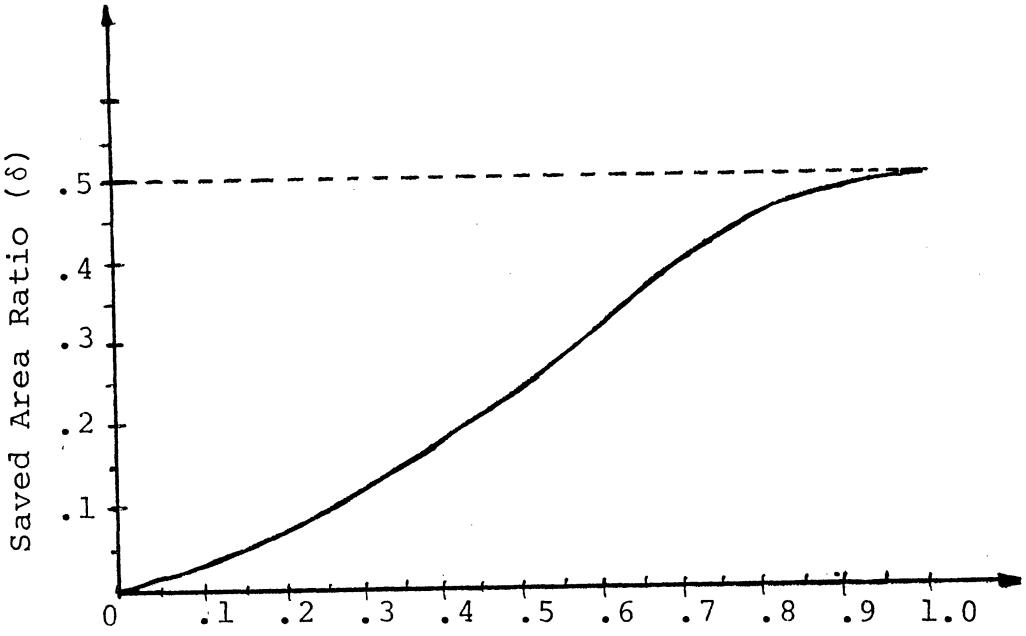


Figure 6.10 Saved Area Ratio *vs* Rent's Exponent.

6.3.3. Computation of Rent's Exponent r

Although for a Rent's rule analysis k and r are the parameters of a logic network, the inequality of saved area ratio is independent of k. The computation to compute r requires:

(1) forming the partition as shown in Figure 6.9.

(2) performing a least square fit on the three known points (P_2, W_2), (P_1, W_1), $(k,1)$ on the P-W plane as shown in Figure 6.11.

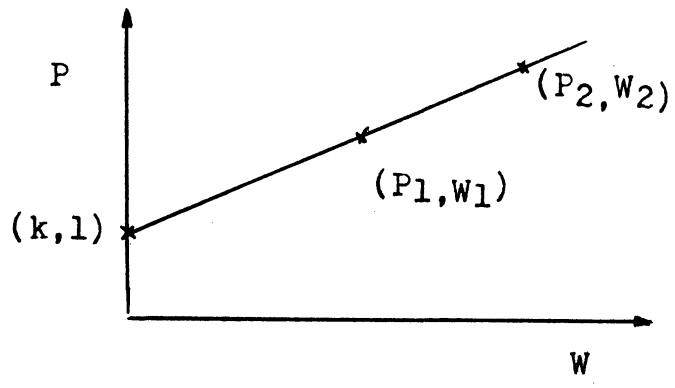


Figure 6.11 Log P *vs* Log W

Since the partitioning in step (1) may require a lot of computation effort, we propose an approximation of r which avoids step (1). Since the values of (P_2, W_2) are given, if we can compute k, we would have an approximation by Figure 6.11. By definition, k is the average number of terminals per gate, therefore, can be computed by the formula $k = \frac{1}{W_2} \times (\Sigma$ nonzero connections on the PLA). We know (P_2, W_2) and $(k, 1)$, and therefore, we can compute a value of r.

6.3.4. Example and Refinement

Table 6.1 shows that the actual saved area ratios fit well within the derived upper and lower bounds. The difference between the actual and the predicted saved area ratio is introduced since we approximately compute Rent's exponent r, as outlined in the previous section. To reduce the difference, we need to spend more computing time to get a third point (P_1, W_1) in Figure 6.11.

Design Description	Input Terminals	Output Terminals	Product Terms	Predicted Saved Area Ratio	Actual Saved Area Ratio
Radix-4 Adder (Figure 6.12)	5	3	20	.03	.05
Four Bit PLA Adder (Figure 6.13)	5	4	8	.22	.37
Eight Bit PLA Adder (Figure 6.14)	8	18	22	.28	.41
Eight Bit ALU (Figure 6.15)	10	23	54	.22	.39

Table 6.1 Comparison Between Actual and Predicted Saved Area Ratio.

y_1	y_0	x_1	x_0	c_{in}	c_o	s_1	s_0
x	0	0	0	0	1	x	x
0	x	x	1	0	1	x	x
0	x	1	0	1	1	x	x
0	1	x	0	1	1	x	x
0	0	0	0	0	1	x	x
1	x	x	0	0	1	x	x
1	0	x	1	1	1	x	x
1	x	1	1	1	1	x	x
0	x	x	0	1	x	1	x
0	x	x	1	0	x	1	x
1	x	x	0	0	x	1	x
1	0	x	1	1	x	1	x
1	x	1	1	1	x	1	x
x	x	1	1	1	x	x	1
x	1	x	1	1	x	x	1
x	1	1	x	x	x	x	1
1	x	1	x	1	x	x	1
1	x	1	x	1	x	x	1
1	x	1	x	1	x	x	1
1	x	1	1	x	x	x	1
1	1	x	x	1	x	x	1
1	1	x	1	x	x	x	1

Figure 6.12 Radix-4 Adder.

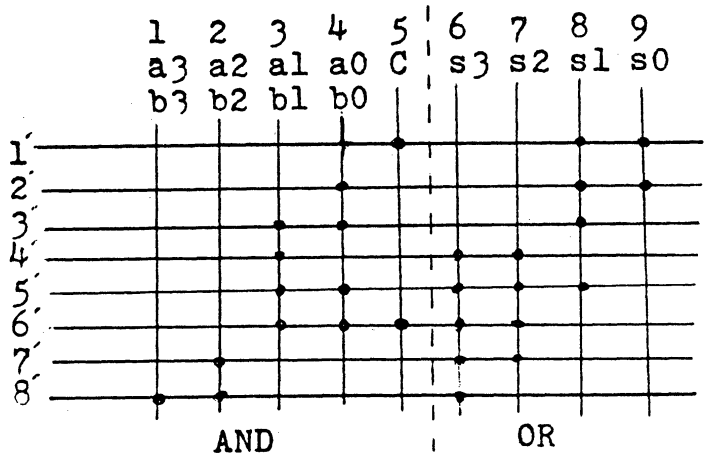


Figure 6.13 Four Bit PLA Adder.

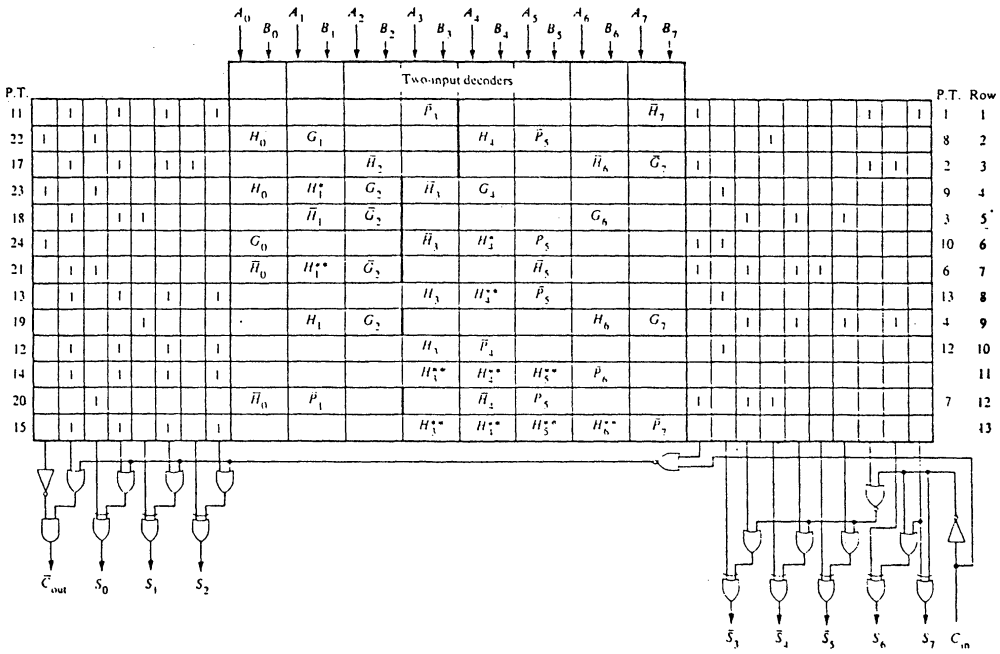


Figure 6.14 Eight Bit PLA Adder.

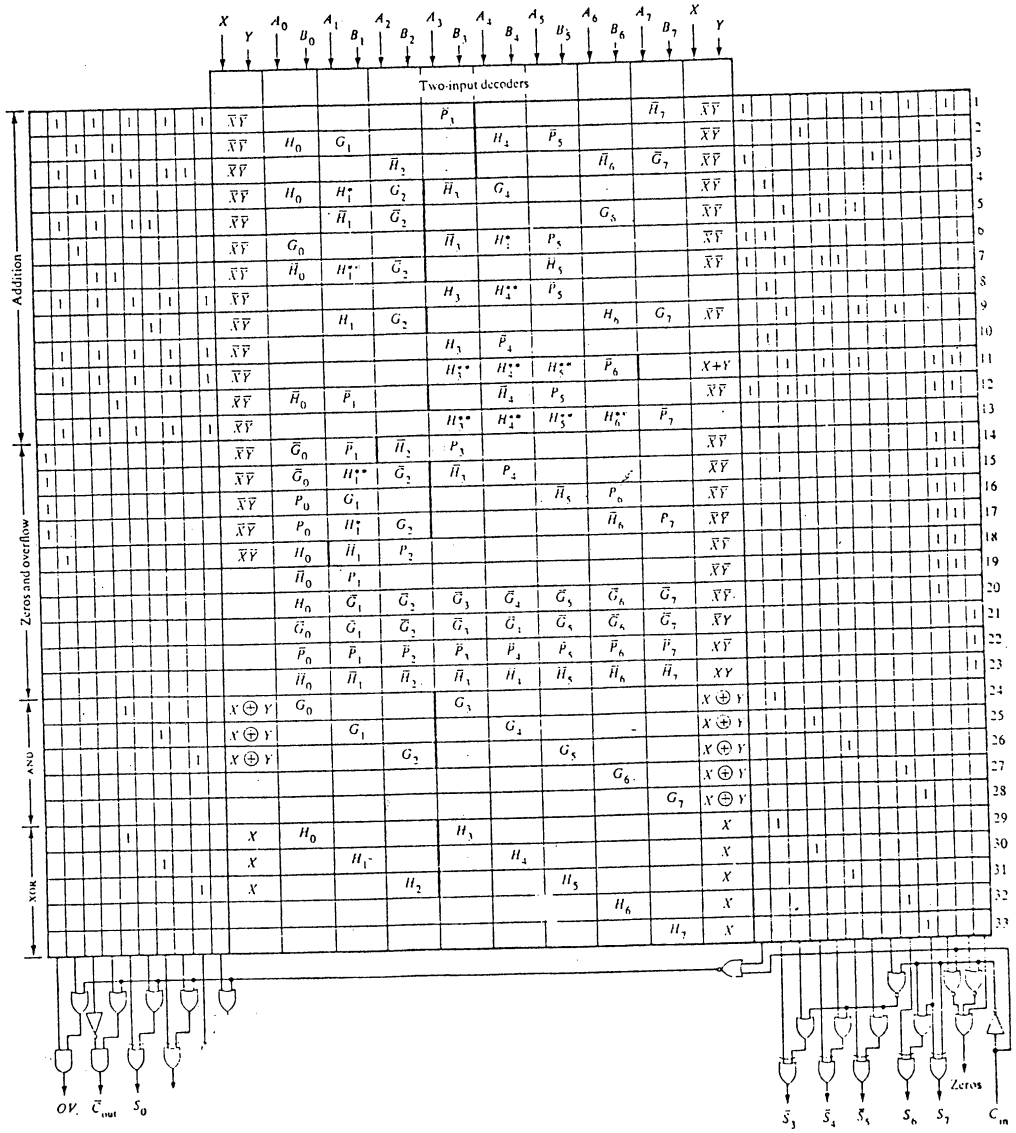


Figure 6.15 Eight Bit ALU.

6.3.5. Summary and Open Problems

Based on Rent's rule and a simple partition model of PLA, we have developed a procedure to estimate the bound of the saved area ratio δ.

Two possible extensions are

(1) To derive a tighter *upper bound* also as a function of Rent's exponent r (currently the upper bound is a constant $\frac{1}{2}$).

(2) To generalize the folding definition in such a way that several product terms can be arranged in one row [85]. This kind of generalization produces a bigger saved area ratio.

Another open question is how the folding process interacts with the process of logic minimization. Consider a system composed of a logic minimizer and a folding algorithm, as shown in Figure 6.16.

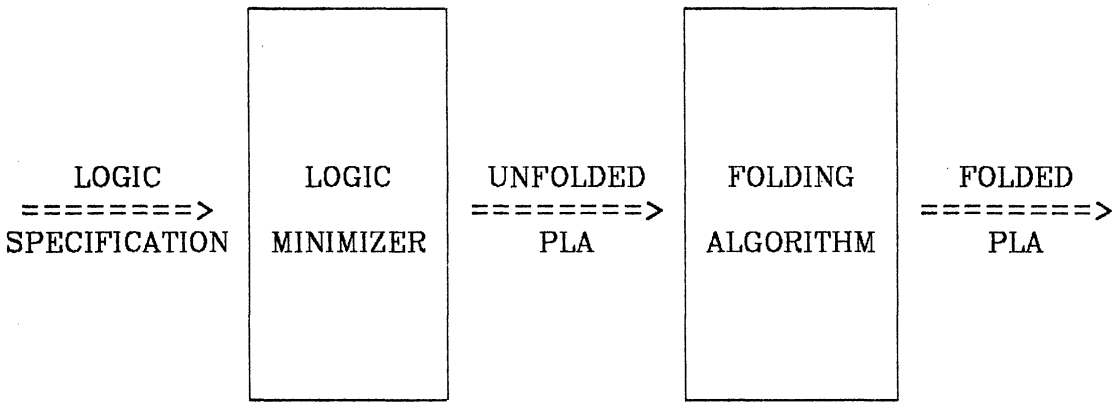


Figure 6.16 A PLA Minimization System.

For the logic minimizer, either absolute or near minimization with respect to some cost function can be obtained. Some efficiently computational algorithms have been proposed for the minimization task [78], [79], [86]. However, if a folding algorithm is cascaded to the logic minimizer, and if two kinds of minimization procedures are taken, we get two final folded PLAs which have area $A_{fold\ with\ absolute\ minimization}$ and $A_{fold\ with\ near\ minimization}$ respectively. Let $\delta_{fold\ with\ absolute\ minimization}$ and $\delta_{fold\ with\ near\ minimization}$ denote the saved area ratios corresponding to two minimization procedures. The open question is whether $\delta_{fold\ with\ absolute\ minimization}$ is greater or less than $\delta_{fold\ with\ near\ minimization}$

CHAPTER 7

CONCLUSIONS

7.1. Contributions

The major problems of the VLSI technology is the management of the complexity of the digital integrated circuit. The design methodologies that help the designers to speed the design tasks are important. In this thesis, we classify the approach into two major categories, namely nonautomatic and automatic layout approaches. Each approach in the categories is time consuming. Among them, we study three kinds of layout approaches - gate array, programmable logic array, and general cell approach. Our philosophy in the thesis is try to estimate the figures of merit resulting from the design process or subprocess by extracting some characteristics of the design. If the answer is positive, we develop some model and do the estimates. If the answer is negative, we use a constructive approach by developing a set of efficient algorithms to estimate or even exactly compute the desired figures of merit.

Since the processes or subprocesses are time consuming, not every design instance deserves going through the processes. The first part of the philosophy has the merit of advising the designer to avoid wasting time by just getting a small amount of improvement. In this thesis, the results relating to the *gate array* and *PLA* follow this philosophy. The nature of the hierarchical descriptions in ADS cannot fit the *gate array* and *PLA* layout approach. We need a general model to map the design descriptions into physical structures. This motivates us to develop the *rectangle model* which is the same as the *general cell* approach. Since the *general cell* approach is a very complicated process, this motivates us to use the *constructive approach* which follows the second part

of the philosophy. In summary, the thesis investigates the techniques for reaching the goal outlined in Chapter 1 under the thesis philosophy in this section. We discuss the problems associated with the *gate array*, *PLA*, and *general cell* layout method.

The contributions of the thesis are:

(1) the development of a model for estimating the number of the folding pairs in PLA folding process. By a simple computation on the Rent's exponent δ, the model can give the lower bound of the number of the folding pairs. It is very useful to have the lower bound for the result obtained in an NP-complete problem. Experimental results validate our model.

(2) the identification that both the area and interconnection length of a digital system are function of Rent's exponent of that system under our gate array model. Our gate array model allows only one track in each vertical or horizontal channel. The digital system with bigger Rent's exponent would have bigger area and longer interconnection length.

(3) a new proposal for a general cell layout method and rigorous treatments for parts of the subproblems of general cell layout method. This includes

(a). a new treatment of the routability and channel routing order problem. The necessary and sufficient condition for the routability test are derived by using a directed graph model. Based on the model, an algorithm is developed for the generation of the channel routing order.

(b). a new channel router based on the dynamic manipulation of interval graph and constraint graph in which the channel routing problem is modeled.

Regarding to the general cell approach, we propose the following layout approach. Chapter 3,4,5 have detail study for this approach.

(1) Placement

The goal of the placement is to avoid an unroutable channel and to make the final area as small as possible. The necessary and sufficient condition for the routability test provides a guideline for avoiding unroutable channels. While the global allocation is a process to minimize the final area.

(2) Routing

The goal of the routing process is to have 100% complete interconnections and make the final area as small as possible. We break the routing process into the following processes:

(2-1) Finding a legal channel routing order.

(2-2) Performing topological net assignments.

(2-3) Solving the displacement problem for each channel.

(2-4) Performing channel routing by following the legal channel order.

(2-5) Performing the **T** shape router on each **T** shape intersection if any.

(2-6) Performing the **cross channel** router on each + shape intersection if any.

7.2. Future Research

For the PLA approach, we are not able to have a nontrival upper bound. It is interesting to find a nontrivial upper bound. Interesting extensions for this PLA folding process are mentioned in section 6.3.5.

For the gate array approach, our model has severe restriction on the available tracks for every vertical and horizontal channel. To generalize our results for the model without the track restriction is very useful in practical case.

With regard to the general cell approach, we propose a layout method and analyze it theoretically. We have programmed the rotability test, the channel routing order generation as well as the channel router. Works remain to be done in order to form a complete system.

In our proposed method, channel router is the major algorithm for detail routing. In order to use channel router, appropriate channel routing order should be generated. We treat this problem very formally. In some system, this problem can be avoided. For example, in MIT's PI system, it divides entire routing space into a set of rectangles. Each rectangle would be treated as a channel. However, this approach requires a global crossing placer to decide the terminals' positions along the boundary in the rectangles. It is intersting to have comparison on the result obtained from these two different methods.

APPENDICES

APPENDIX I.

Min-Cut Algorithm

```
  AA PART BB;HN;A;B;N
 [1]
'*****************************************************************'
 [2] 'TOTAL SET TO BE PARTITIONED:'
 [3] AA,BB
 [4] 'INITIAL PARTITION SETS:'
 [5] 'SET A:   ';AA
 [6] 'SET B:   ';BB
 [7] →(1=ρAA)/RETURN
 [8] N←2×(ρAA)[1]
 [9] HN←N÷2
 [10] A←AA
 [11] B←BB
 [12]
LPBEG:'>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'
 [13] 'NEW ITERATION BEGINS:'
 [14] P←1
 [15] AB←A,B
 [16] C←CD[AB;AB]
 [17] AI←(HNρ1),(HNρ0)
 [18] BI←(HNρ0),(HNρ1)
 [19] APP←HNρ0
 [20] BPP←HNρ0
 [21] CH←CD[A;B]
 [22] EA←AI/C+.×BI
 [23] IA←AI/C+.×AI
 [24] EB←BI/C+.×AI
 [25] IB←BI/C+.×BI
 [26] DA←EA-IA
 [27] DB←EB-IB
 [28] CA←HNρ0
 [29] LPPART:C←DA∘.+DB
 [30] C←C-2×CH
 [31] RET←MAXC C
 [32] A11←RET[1]
 [33] B11←RET[2]
 [34] APP[P]←A1←A[A11]
 [35] BPP[P]←B1←B[B11]
 [36] CA[P]←RET[3]
 [37] →(P=HN)/MAXYES
 [38] P←P+1
 [39] DA←DA+(2×HN↑C[;A11])-2×HN↑C[;B11+HN]
 [40] DB←DB+(2×HN↓C[;B11+HN])-2×HN↓C[;A11]
 [41] DA[A11]←-99999
 [42] DB[B11]←-99999
 [43] →LPPART
 [44] MAXYES:KP←MAXCS
 [45] 'CA   ';CA
```

```
[46] 'GSUM   ';GSUM
[47] 'KP   ';KP
[48] →(KP[1]>0)/REAR
[49] →NEXT
[50] REAR:APM←APP[1 TO KP[2]]
[51] BPM←BPP[1 TO KP[2]]
[52] IHN←(ρA)ρ0
[53] IHN[A⍳APM]←1
[54] A←((~IHN)/A),BPM
[55] IHN←(ρB)ρ0
[56] IHN[B⍳BPM]←1
[57] B←((~IHN)/B),APM
[58] '*** INTERMEDIATE PARTITIONS: ***'
[59] 'SET A:   ';A
[60] 'SET B:   ';B
[61] →LPBEG
[62] NEXT:'→→→ FINAL PARTITION: ←←←'
[63] 'SET A:   ';A
[64] 'SET B:   ';B
[65] ⍞'**********************************************************'
[66]
'==================== SUBPARTITION 1 ===================='
[67] AA2←(HN÷2)↑B
[68] BB2←(HN÷2)↓B
[69] AA2 PART BB2
[70]
'==================== SUBPARTITION 2 ===================='
[71] AA1←(HN÷2)↑A
[72] BB1←(HN÷2)↓A
[73] AA1 PART BB1
[74]
'**********************************************************'
[75] RETURN:'*** FINAL PARTITIONS: ***'
[76] 'SET A:   ';AA
[77] 'SET B:   ';BB
[78] →0
```

```
KP←MAXGS
[1] GSUM←HNρ0
[2] I←0
[3] KP←2ρ0
[4] LPMAXGS:→((I←I+1)>HN)/ENDMAXGS
[5] GSUM[I]←+/I↑GA
[6] →LPMAXGS
[7] ENDMAXGS:⌈/GSUM
[8] KP[1]←⌈/GSUM
[9] KP[2]←GSUM⍳⌈/GSUM
[10] →0
```

```
 RET←MAXG  C
[1]  RET←3⍴0
[2]  GMAX←⌈/⌈/C
[3]  ID←C⍳GMAX
[4]  RET[1]←⌈ID÷1↑⍴C
[5]  RET[2]←(((1↑⍴C)|ID)=0)×(1↑⍴C))+(1↑⍴C)|ID
[6]  RET[3]←GMAX
```

```
 TOV←N1  TO  N2
[1]  TOV←(N1-1)+⍳(1+N2-N1)
[2]  →0
```

APPENDIX II.

Channel Router

```
ROUTER
[1] ⍝ THIS IS A CHANNEL ROUTER WHICH IS GOOD FOR
[2] ⍝ BOTH VERTICAL AND/OR NONVERTICAL CONSTRAINTS.
[3] ⍝ CONSTRAINTS GRAPH ARE GENERATED BY THE FUNCTIONS
[4] ⍝ ∇ CONSTRAINTS AND ∇ CLOSURE. THE GRAPH IS STORED
[5] ⍝ IN ADJACENT MATRIX.
[6] ⍝ THE INPUT DATA INCLUDE ARRAYS U AND L CONTAINING
[7] ⍝ THE UPPER AND LOWER TERMINAL DISTRIBUTIONS, ARRAY
[8] ⍝ NESTDATA CONTAINING THE TWO EXTREME COORDINATES
[9] ⍝ ASSOCIATED WITH EVERY NET.
[10] ⍝ N IS THE NUMBER OF NETS
[11] '.....'
[12] MASK←N⍴1
[13] MASKTMP←N⍴0
[14] LTRT←0
[15] IC←0
[16] MASKAND←MASK
[17] ⍝ START ANOTHER TRACK UNTIL ALL
[18] ⍝ THE NETS HAVE BEEN ASSIGNED
[19] START:→((∨/MASK)=0)/ENDROUTE
[20] NET←⍴1
[21] ⍝ SEARCH ANOTHER NET WHICH IS FEASIBLE
[22] ⍝ TO THE CURRENT TRACK UNTIL IT IS FILLED.
[23] TRACK:→((∨/MASKAND)=0)/ENDTRACK
[24] →(LTRT=1)/RIGHT
[25]
LEFT: INDL←⌊/((NETDATA[;1]=(⌊/MASKAND/NETDATA[;1]))×MASK)/⍳N
[26] NETID←NETDATA[INDL;3]
[27] EXTREMR←NETDATA[INDL;2]
[28] AVNET←((~(VC[NETID;]∨⍉VC[;NETID]))∧EXTREMR<NETDATA[;1])/⍳N
[29] →DOROUTE
[30]
RIGHT: INDL←⌊/((NETDATA[;2]=(⌈/MASKAND/NETDATA[;2]))×MASK)/⍳N
[31] NETID←NETDATA[INDL;3]
[32] EXTREML←NETDATA[INDL;1]
[33] AVNET←((~(VC[NETID;]∨⍉VC[;NETID]))∧EXTREML>NETDATA[;2])/⍳N
[34] DOROUTE: NET←NET,NETID
[35] MASK[NETID]←0
[36] →(0=⍴AVNET)/ENDTRACK
[37] IDAVNET←NETDATA[;3]⍳AVNET
[38] MASKTMP[IDAVNET]←1
[39] MASKAND←MASK∧MASKTMP
[40] MASKTMP←N⍴0
[41] ⍝ RELATIONSHIP ELIMINATED
[42] TNI←0
[43] TN←(⍴NET)-1
[44] PREDID←(VC[;NETID]=1)/⍳N
[45] SUCCID←(VC[NETID;]=1)/⍳N
```

APPENDIX III.

PLA Decomposition Algorithm

```
 CLUSTER V
[1] →((1↓V)=0)/ZERO
[2] VISITEDO[1↑V]←1
[3] →BEGIN
[4] ZERO:VISITEDI[1↑V]←1
[5] BEGIN:Q1←SOL1←⍴(S←1)
[6] Q2←SOL2←⍴(S←1)
[7] Q1←Q1,1↑V
[8] Q2←Q2,1↓V
[9] GO1:→(0=⍴Q1)/END
[10] NV1←1↑Q1
[11] NV2←1↑Q2
[12] Q1←1↓Q1
[13] Q2←1↓Q2
[14] SOL1←SOL1,NV1
[15] SOL2←SOL2,NV2
[16] →(NV2=0)/INV
[17] ADJ←(VOVI[;NV1])[;1]/⍳VI
[18] GO2:→(0=⍴ADJ)/GO1
[19] NADJ←1↑ADJ
[20] ADJ←1↓ADJ
[21] →(VISITEDI[NADJ]≠0)/GO2
[22] Q1←Q1,NADJ
[23] Q2←Q2,0
[24] ⍞ 'Q1  ';Q1
[25] ⍞ 'Q2  ';Q2
[26] VISITEDI[NADJ]←1
[27] →GO2
[28] INV:ADJ←(VOVI[NV1;])[1;]/⍳VO
[29] GO22:→(0=⍴ADJ)/GO1
[30] NADJ←1↑ADJ
[31] ADJ←1↓ADJ
[32] →(VISITEDO[NADJ]≠0)/GO22
[33] Q1←Q1,NADJ
[34] Q2←Q2,1
[35] ⍞ 'Q1  ';Q1
[36] ⍞ 'Q2  ';Q2
[37] VISITEDO[NADJ]←1
[38] →GO22
[39] END:' OUTPUTS  : ';SOL2/SOL1
[40] ' PRODUCTS:  ';(~SOL2)/SOL1
[41]    →0
```

```
 CONNECTS
[1] VISITEDO←VO⍴0
[2] VISITEDI←VI⍴0
```

```
[46] LPELIM:→((TNI←TNI+1)>TN)/ENDLPELIM
[47] PRED←(VC[;NET[TNI]]=1)/ιN
[48] SUCC←(VC[NET[TNI];]=1)/ιN
[49] →((0=ρPREDID)∨(0=ρSUCC))/LPE2
[50] VC[PREDID;SUCC]←1
[51] LPE2:→((0=ρPRED)∨(0=ρSUCCID))/LPELIM
[52] VC[PRED;SUCCID]←1
[53] →LPELIM
[54] ENDLPELIM:→TRACK
[55] ENDTRACK:'THIS TRACK CONTAINS'
[56] NET
[57] '.....'
[58] IC←IC+1
[59] LTRT←~LTRT
[60] MASKAND←MASK
[61] →START
[62] ENDROUTE:'TOTAL NUMBER OF TRACKS NEED IN THIS CHANNEL:'
[63] IC
[64] '.....'
[65] →0
```

```
CONSTRAINT
[1] ⍝ GENERATE CONSTRAINTS GRAPH
[2] ⍝ THE INPUT ARE THE ARRAY U AND L CONTAINING OF
[3] ⍝ UPPER AND LOWER TERMINALS DISTRIBUTION
[4] ID←(U≠0)∧(L≠0)
[5] IU←ID/U
[6] IL←ID/L
[7] I←0
[8] VC←(N,N)ρ0
[9] LPCON:→((I←I+1)>ρIU)/ENDCON
[10] VC[IU[I];IL[I]]←1
[11] →LPCON
[12] ENDCON:→0
```

```
CLOSURE
[1] ⍝ GENERATE THE TRANSITIVE CLOSURE OF VC
[2] K←0
[3] LPCLOSURE:→((K←K+1)>N)/ENDCLOSURE
[4] VC←VC∨(VC[;K])∘.∧VC[K;]
[5] →LPCLOSURE
[6] ENDCLOSURE:→0
```

```
[3]  I←N←0
[4]  LOOP:→((I←I+1)>VO)/ENDCONN
[5]  →(VISITEDO[I]≠0)/LOOP
[6]  V←I,1
[7]  N←N+1
[8]  '.....'
[9]  'CLUSTER NO. ';N;':'
[10] CLUSTER V
[11] →LOOP
[12] ENDCONN:'.....'
[13]     →0
```

APPENDIX IV.

PLA Folding Algorithm

```
 FOLD;I;V;VD;VH
[1] ⍝ INITIALIZATION
[2] A←(N,2)⍴0
[3] V←N⍴1
[4] PAIR←N⍴1
[5] VD←N⍴0
[6] I←0
[7] VD←(0≠G)+.×(N⍴1)
[8] LOOP1:→((∨/(V≠0))=0)/END
[9] ⍝ SELECT A COLUMN
[10] VV←V MIN VD
[11] ⍝ POSSIBLE FOLDING CANDIDATES WRT VV
[12] VH←VV COMPRESS G[VV;]
[13] VH←VH×PAIR
[14] LOOP2:→((∨/VH)=0)/ENDL2
[15] ⍝ A FOLDING CANDIDATE WRT VV
[16] UU←VH MAX VD
[17] →(UU=0)/ENDL2
[18] 'LET US TRY THE PAIR ';VV;'    ';UU
[19] I←I+1
[20] A[I;]←VV,UU
[21] G[VV;UU]←-1
[22] ⍝ CHECK ALTERNATING CYCLE IN AUGUMENTED GRAPH
[23] CHKCY
[24] →(CY≠0)/DELE
[25]
'THERE IS A CYCLE AND WE DONT RECOMMAND THE PAIR';VV;'    ';UU
[26]
'.....'
[27] VH[UU]←0
[28] A[I;]←0,0
[29] G[VV;UU]←0
[30] I←I-1
[31] →LOOP2
[32] ⍝ DELETE UU AND VV FROM THE GRAPH
[33] DELE:V[UU]←0
[34] V[VV]←0
[35] G[VV;UU]←-1
[36] PAIR[VV;UU]←0
[37] 'WE SUGGEST THE FOLDING PAIR    ';VV;'    ';UU
[38]
'.....'
[39] →LOOP1
[40] ENDL2:V[VV]←0
[41] →LOOP1
[42] END:'THE FOLDING PAIR ARE'
[43] A
```

```
[44] IFOLD←I
[45] →0
```

```
 OUTPLA ORDER;AV;AU;NDI
[1] NDI←ND-IFOLD
[2] CUT←NDI⍴0
[3] P1←PN←NDI⍴0
[4] MASK←ND⍴1
[5] P←(N,NDI)⍴'X'
[6] I←0
[7] LPOUT:→((I←I+1)>IFOLD)/ENDOUT
[8] VV←A[I;1]
[9] UU←A[I;2]
[10] AV←DATA[ORDER;VV]
[11] AU←DATA[ORDER;UU]
[12] MASK[VV,UU]←0
[13] MED←((AU⍳'0')⌊AU⍳'1')-1
[14] P[;I]←(MED↑AV),(MED↓AU)
[15] P1[I]←VV
[16] PN[I]←UU
[17] CUT[I]←MED
[18] →LPOUT
[19] ENDOUT:P1[I UPTO NDI]←PN[I UPTO NDI]←MASK/⍳ND
[20] P[;I UPTO NDI]←MASK/DATA
[21] P1
[22] '      '
[23] P
[24] '      '
[25] PN
[26] '      '
[27] 'CUT AT'
[28] CUT
[29] →0
```

```
 OUTPLA ORDER;AV;AU;NDI
[1] NDI←ND-IFOLD
[2] CUT←NDI⍴0
[3] P1←PN←NDI⍴0
[4] MASK←ND⍴1
[5] P←(N,NDI)⍴'X'
[6] I←0
[7] LPOUT:→((I←I+1)>IFOLD)/ENDOUT
[8] VV←A[I;1]
[9] UU←A[I;2]
[10] AV←DATA[ORDER;VV]
[11] AU←DATA[ORDER;UU]
[12] MASK[VV,UU]←0
[13] MED←((AU⍳'0')⌊AU⍳'1')-1
[14] P[;I]←(MED↑AV),(MED↓AU)
```

```
[15] P1[I]←VV
[16] PN[I]←UU
[17] CUT[I]←MED
[18] →LPOUT
[19] ENDOUT:P1[I UPTO NDI]←PN[I UPTO NDI]←MASK/⍳ND
[20] P[;I UPTO NDI]←MASK/DATA
[21] P1
[22] '      '
[23] P
[24] '      '
[25] PN
[26] '      '
[27] 'CUT AT'
[28] CUT
[29] →0
```

```
 SORT VTOP;I;W
[1] NUM[VTOP]←1
[2] ADJTOP←(1=P[VTOP;])/⍳ND
[3] NADJTOP←⍴ADJTOP
[4] I←0
[5] LPSORT:→((I←I+1)>NADJTOP)/ENDSORT
[6] W←ADJTOP[I]
[7] →(NUM[W]=1)/ERRSORT
[8] SORT W
[9] →LPSORT
[10] ERRSORT:→(LAB[W]≠0)/LPSORT
[11] 'THERE IS A CYCLE'
[12] →0
[13] ENDSORT:JTOP←JTOP-1
[14] LAB[VTOP]←JTOP
[15] ITOP←ITOP+1
[16] ORDER[ITOP]←VTOP
[17] →0
[18] W
[19] Q
```

```
 IJ←I UPTO J
[1] IJ←(I-1)+⍳J+1-I
```

```
 ROW AO
[1] N←1↑⍴AO
[2] G←(N,N)⍴0
[3] IN←(⍳N)∘.,⍳N
[4] G←(IN≠⌽IN)∧((AO)∨.∧⍉AO)
[5] CY←1
[6] ⍎ FOLD
```

```
[7] →0
```

```
 COLM AO
[1] N←1↓⍴AO
[2] G←(N,N)⍴0
[3] IN←(⍳N)∘.-⍳N
[4] G←(IN≠⍉IN)∧((⍉AO)∨.∧AO)
[5] CY←1
[6] FOLD
[7] →0
```

```
 CYGEN;I;L;K;CYV;ONEV
[1] CYV←(STH+1)⍴0
[2] CYV←(STH↑PATH),S
[3] ONEV←((⍴CYV)-1)⍴0
[4] I←0
[5] LOOPCY:→((I←I+1)>STH)/JMPCY
[6] L←CYV[I]
[7] K←CYV[I+1]
[8] ONEV[I]←G[L;K]
[9] →LOOPCY
[10] JMPCY:CY←+/ONEV
[11] →0
```

```
 FLAG←FLIP CYCLE VC;I;AN;ADJ;W;GFLIP
[1] FLAG←0
[2] STH←STH+1
[3] PATH[STH]←VC
[4] AVAIL[VC]←0
[5] AN←+/(1=(G[VC;]×(-1)*FLIP))
[6] ADJ←AN⍴0
[7] ADJ←(1=(G[VC;]×(-1)*FLIP))/⍳N
[8] →(FLIP≠0)/BEGIN
[9] AN←+/(∨/((-1)=G[ADJ;]))
[10] ADJ←(∨/((-1)=G[ADJ;]))/ADJ
[11] BEGIN:I←0
[12] LOOPC:→((I←I+1)>AN)/END
[13] ⎕ 'S    ';S;'ADJ    ';ADJ;'VC  ';VC;'AVAIL  ';AVAIL
[14] W←ADJ[I]
[15] →(W=S)/PATHST
[16] →(AVAIL[W]=0)/LOOPC
[17] GFLAG←0
[18] GFLIP←~FLIP
[19] GFLAG←GFLIP CYCLE W
[20] FLAG←FLAG∨GFLAG
[21] →(CY=0)/ENDCYCLE
[22] →LOOPC
```

```
[23] PATHST:'|------->'
[24] 3(STH↑PATH),S
[25] FLAG←1
[26] CYGEN
[27] →(CY=0)/ENDCYCLE
[28] →LOOPC
[29] END:→(FLAG=1)/UN
[30] I←0
[31] LOOPB:→((I←I+1)>AN)/END2
[32] W←ADJ[I]
[33] →(∨/VC=B[W;])/LOOPB
[34] BI←0
[35] BI←IB[W]+1
[36] B[W;BI]←VC
[37] IB[W]←BI
[38] →LOOPB
[39] UN:UNMARK VC
[40] END2:STH←STH-1
[41] AVAIL[VC]←1
[42] ENDCYCLE:→0
```

```
CHKCY
[1] PATH←N⍴0
[2] STH←0
[3] S←0
[4] SN←+/∨/(-1)=C
[5] LOOPD:→((S←S+1)>SN)/ENDD
[6] ⍝ AVV←(S-1)↓⍳N
[7] ⍝ AVAIL←N⍴0
[8] ⍝ AVAIL[AVV]←1
[9] AVAIL←N⍴1
[10] CY←1
[11] B←(N,N)⍴0
[12] IB←N⍴0
[13] F←0
[14] FFLIP←1
[15] F←FFLIP CYCLE S
[16] →(CY=0)/ENDD
[17] →LOOPD
[18] ENDD:→0
```

```
UNMARK U;J;IIB
[1] AVAIL[U]←1
[2] J←0
[3] IW←0
[4] IIB←IB[U]
[5] LOOPU:→((J←J+1)>IIB)/ENDU
[6] IW←B[U;J]
[7] B[U;J]←0
[8] IB[U]←IB[U]-1
```

```
[9] →(AVAIL[IW]≠0)/LOOPU
[10] UNMARK IW
[11] →LOOPU
[12] ENDU:→0
```

```
CHKCY
[1] PATH←N⍴0
[2] STH←0
[3] S←0
[4] SN←+/∨/(-1)=G
[5] LOOPD:→((S←S+1)>SN)/ENDD
[6] ⍝ AVV←(S-1)↓⍳N
[7] ⍝ AVAIL←N⍴0
[8] ⍝ AVAIL[AVV]←1
[9] AVAIL←N⍴1
[10] CY←1
[11] B←(N,N)⍴0
[12] IB←N⍴0
[13] F←0
[14] FFLIP←1
[15] F←FFLIP CYCLE S
[16] →(CY=0)/ENDD
[17] →LOOPD
[18] ENDD:→0
```

```
VV←V MIN VD
[1] MI←⌊/((0≠(VD×V))/(VD×V))
[2] VV←(VD×V)⍳MI
```

```
UU←VH MAX VD
[1] →(0=(∨/(0≠(VH×VD))))/ENDMAX
[2] MA←⌈/((0≠(VH×VD))/(VH×VD))
[3] UU←(VD×VH)⍳MA
[4] →0
[5] ENDMAX:UU←0
[6] →0
```

```
VH←VV COMPRESS VT
[1] VH←(0=VT)
[2] VH[VV]←0
```

```
ND TOPSORT P;VTOP
[1] LAB←NUM←ORDER←ND⍴0
```

```
[2] JTOP←ND+1
[3] ITOP←0
[4] LPTOP:→(0=+/(0=NUM))/ENDTOP
[5] VTOP←NUM\0
[6] SORT VTOP
[7] →LPTOP
[8] ENDTOP:'A SUGGESTED ORDER ΛRE :'
[9] ORDER←φORDER
[10] ORDER
[11] →0
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within apertures," *Proc.of 8th Design Automation Workshop*, pp. 155-169, 1971.

[2] D. Kuck, *The Structures of Computers and Computations*. John Wesley, 1979.

[3] C. Mead and L. Conway, *Introduction to VLSI System*. John-Wesley, 1980.

[4] J. Soukup, "Circuit layout," *Proceeding of the IEEE*, vol. 69, no. 10, pp. 1281-1304, Oct. 1981.

[5] D. Gibson and S. Nance, "SLIC - Symbolic layout of integrated circuits," *Proceedings of 13th Design Automation Conference*, pp. 434-440, June 1976.

[6] R. Larsen, "Versatile mask generation techniques for custom microelectronics devices," *Proceedings of 15th Design Automation Conference*, pp. 193-198, June 1978.

[7] J. Williams, "STICKS - A graphical compiler for high level LSI design," *AFIPS conference Proceedings*, vol. 47, pp. 289-295, June 1978.

[8] Y. Cho, Korenjak, and Stockton, "FLOSS: An approach to automated layout for high volume density," *Proceedings of 14th Design Automation Conference*, pp. 138-141, June 1977.

[9] A. Dunlop, *Integrated Circuit mask compaction*. PhD Thesis, Carnegie-Mellon University, 1979.

[10] M. Hsueh, *Symbolic layout and compaction of integrated circuits*. PhD Thesis, University of California, Berkeley, 1980.

[11] N. Weste, "Virtual grid symbolic layout," *Proceedings of 18th Design Automation Conference*, pp. 225-233, June 1981.

[12] M. Schmookler , "Design of large ALUs using multiple PLA macros ," *IBM Journal of Research and Development* , vol. 24 , no. 1 , pp. 2-14 , July 1980.

[13] B. Vergnieres , "Macro generation algorithms for VLSI custom chip design ," *IBM Journal of Research and Development* , vol. 24 , no. 5 , pp. 612-621 , May 1980.

[14] R. Wood , "A high density programmable logic array chip ," *IEEE Trans. on Computers* , vol. C-28 , no. 9 , pp. 602-608 , Sept. 1979.

[15] H. Fleisher and L. Maissel, "An introduction to array logic," *IBM Journal of Research and Development*, vol. 18, no. 5, pp. 98-109, 1975.

[16] K. Khokhani and A. Patel, "Chip layout problem - a placement procedure for LSI," *Proceedings of 14th Design Automation Conference*, pp. 291-297, 1977.

[17] K. Chen, M. Feuer, K. Khokhani, N. Nan, and S. Schmidt, "Chip layout problem - routing procedure," *Proceedings of 14th Design Automation Conference*, pp. 298-302, 1977.

[18] S. Goto, "A two dimensional placement algorithm for masterslice LSI layout problem," *Proceedings of 16th Design Automation Conference*, pp. 11-17, 1979.

[19] R. Nair, S. Hong, S. Liles, and K. Villani, "Global wiring on a wire routing machine," *Proceedings of 19th Design Automation Conference*, pp. 224-231, 1982.

[20] K. Khokhani, A. Patel, W. Ferguson, J. Sessa, and D. Hatton, "Placement of variaable size circuits on LSI masterslices," *Proceedings of 18th Design Automation Conference*, pp. 426-434, 1981.

[21] W. Heller, W. Mikhail, and W. Donath, "Prediction of wiring space requirements for LSI," *Journal of Design Automation and Fault Tolerant Computing*, vol. 2, pp. 117-144, 1978.

[22] A. El Gamal, "Two dimensional stochastic model for interconnections in masterslice integrated circuits," *IEEE Trans. on Circuits and Systems.* , vol. CAS-28, pp. 127-138, Feb. 1981.

[23] G. Persky, D. Deutsch, and D. Schweikert, "LTX-A minicomputer-based system for automated LSI layout," *Journal of Design Automation and Fault-Tolerant Computing*, vol. 1, pp. 217-255, May 1977.

[24] B. Preas, *Placement and routing algorithms for hierarchical integrated circuit layout*. PhD Thesis, Stanford University, 1979.

[25] U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," *Proceeding of 16th Design Automation Conference*, pp. 474-482, 1979.

[26] R. Malladi, G. Serrero, and A. Verdillon, "Automatic placement of rectanglar blocks with the interconnection channels," *Proceeding of 18th Design Automation Conference*, pp. 419-425, 1981.

[27] M. Wiesel and D. Mlynski, "Two-dimensional channel routing and channel intersection problems," *Proceeding of 19th Design Automation*, 1982.

[28] G. Wipfler, M. wiesel, and D. Mlynski, "A combined force and cut placement algorithm for hierarchical VLSI layout," *Proceeding of 19th Design Automation*, 1982.

[29] H. Rothermel, M. Wiesel, and D. Mlynski, "Routability test with variable wire width for hierarchical chip design using a new database system," *Proceeding of 1982 ISCAS*, 1982.

[30] G. Odawara, K. Lijima, and T. Kiyomatsu, and J. Hassett, "Automated layout in ASHLAR: An approach to the problems of general cell layout for VLSI," *Proceeding of 19th Design Automation*, 1982.

[31] R. Rivest, "The 'PI' system," *Proceeding of 19th Design Automation Conference*, 1982.

[32] R. Rivest and C. Flduccia, "A 'greedy' channel router," *Proceeding of 19th Design Automation Conference*, 1982.

[33] K. Sato et al., "MILD-A cell based layout system for MOS-LSI," *Proceeding of 18th Design Automation Conference*, pp. 828-836, 1981.

[34] D. Johannsen, "Bristle blocks: A silicon compiler," *Proc. of Design Automation Conference*, pp. 310-313, June 1979.

[35] W. E. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Trans. Circuits Syst.*, , vol. CAS-26, pp. 272-277, April 1979.

[36] M. Feuer, "Connectivity of random logic," *IEEE Trans. on Computers*, pp. 29-33, Jan. 1982.

[37] B. S. Landman and R. Russo , "On a pin versus block relationship for partitions of logic graphs ," *IEEE Trans. on Computers* , vol. C-20 , no. 12 , pp. 1469-1479 , 1971.

[38] R. W. Floyd, "The compilation of regular expressions into integrated circuits," *Proceeding of 21th IEEE Symposium on Foundation of Computer Science*, p. ?, May 1981.

[39] C. E. Leiserson, "Area efficient graph layouts (for VLSI)," *Proceeding of 21th IEEE Symposium on Foundations of Computer Science*, pp. 270-281, Sep. 1980.

[40] W. Donath, "Equivalence of memory to 'Random Logic'," *IBM Journal of Research and Development*, vol. 58, no. 5, pp. 401-407, 1974.

[41] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *BSTJ*, vol. 49, no. 2, pp. 291-307, Feb. 1970.

[42] R. Russo, "On the tradeoff between logic performance and circuit-to-pin ratio for LSI," *IEEE Trans. on Computers*, vol. 21, no. 2, pp. 147-153, 1972.

[43] S. Kang and W. VanCleemput, "Automatic PLA synthesis from a DDL-P description," *18th Design Automation Conference*, pp. 391-397, 1981.

[44] L. Glasser and P. Penfield, Jr., "An interactive PLA generator as an archetype for a new VLSI design methodology," *IEEE International Conference on Circuits and Computers*, Oct. 1980.

[45] G. Hachtel, et al. , "An algorithm for optimal PLA folding ," *IBM Research Report RC8668* , Jan. 1981.

[46] D. Dietmeyer and M. Doshi, "Automated PLA synthesis of the combinational logic of the DDL descriptions," *Journal of Design Automation and Fault Tolerant Computing*, vol. 3, pp. 241-257, 1980.

[47] M. Hanan and J. Kurtzberg, "Placement techniques," *Design Automation of Digital System: Theory and Techniques (ed. M. Breuer)*, pp. 213-282, 1972.

[48] M. A. Breuer, "Min-Cut Placement," *Journal of Design Automation and Fault Tolerant Computing*, vol. 1, no. 4, pp. 343-362, Oct. 1977.

[49] P. Agrawal and M. Breuer, "Some theoretical aspects of algorithmic routing," *Proc. of 14th Design Automation Conference*, pp. 23-31, June 1977.

[50] D. W. Hightower, "The interconnection problem: A tutorial," *IEEE Computer*, vol. 7, no. 4, pp. 18-32, April 1974.

[51] C. Lee, "An algorithm for path connections and its applications," *IEEE Trans. on Electronic Computers*, vol. EC-10, pp. 346-365, Sept. 1961.

[52] F. Rubin, "The Lee connection algorithm," *IEEE Trans. on Computers*, vol. C-23, pp. 907-914, 1974.

[53] J. Soukup, "Fast maze router," *Proceedings of 15th Design Automation Conference*, pp. 100-102, 1978.

[54] D. Hightower, "A solutions to the line routing problems on the continuous plane," *Proceedings of 6th Design Automation Conference*, pp. 1-24, 1969.

[55] K. Sato and T. Nagai, "A method of specifying the relative locations between blocks in a routing for building block LSI," *Proc. of 1979 ISCAS*, pp. 673-676, 1979.

[56] H. Kawanishi, et al., "A routing method of building-block LSI," *Proc. of 7th Asilomar Conference on Circuits, Systems, and Computers*, pp. 119-123, Nov. 1973.

[57] Z. Syed, *On routing for custom integrated circuits*. PhD Thesis, University of Southern California, July 1981.

[58] J. M. Smith, "Generalized Steiner network problems in two and three dimensions", Feb. 1982, private communication.

[59] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-1, no. 1, pp. 25-35, 1982.

[60] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[61] C. Berge, *Graphs and Hypergraphs*. North-Holland, 1976.

[62] F. Gavril, "Algorithms for minimum coloring, maximum clique, minimum coloring by cliques and maximum independent set of a chordal graph," *SIAM J. Computing*, vol. 1, pp. 180-187, 1972.

[63] D. Rose, R. Tarjan, and G. Leuker, "Algorithmic aspects of vertex elimination on graphs," *SIAM J. Computing*, vol. 5, no. 2, pp. 266-283, 1976.

[64] D. Dilworth, "A decomposition theorem for partially order sets," *Ann. Math.*, vol. 51, pp. 161-166, 1950.

[65] U. Gupta, D. Lee, and J. Leung, "An optimal solution for the channel assignment problem," *IEEE Trans. on Computers*, vol. C-28, no. 11, pp. 807-810, Nov. 1979.

[66] B. Kernighan, D. Schweikert, and G. Persky, "An optimum channel-routing algorithm for polycell layouts of integrated circuits," *Proc. of 10th Design Automation Conference*, pp. 50-59, 1973.

[67] D. Deutsch, "A 'dogleg' channel router," *Proc. of 14th Design Automation Conference*, pp. 425-433, 1976.

[68] A. LaPaugh, *Algorithms for integrated circuit layout: An analytic approach*. PhD Thesis, MIT., 1980.

[69] T. Asano, et al., "A graph-theoretical approach to the routing problem," *Electronics and Communications in Japan*, vol. 56-A, no. 12, pp. 1-7, 1973.

[70] T. Asano, H. Horino, and T. Amano, "Realizability of wiring for building-block type LSI," *Electronics and Communications in Japan*, vol. 56 -A, no. 9, pp. 10-17 , 1973.

[71] T. Asano , T. Kitahashi, and K. Tanaka, "On a method of realizing minimum-width wirings," *Electronics and Communications in Japan*, vol. 59-A, no. 2, pp. 29-39, 1975.

[72] M. Wada, "A dogleg optimal channel router with completion enchansements," *Proc. of 18th Design Automation Conference*, pp. 762-768, July 1981.

[73] C. Leiserson and R. Pinter, "Optimal placement for river routing," *VLSI Systems and Computations*, pp. 126-142, Oct. 1981.

[74] D. Atkins et al. , "Overview of an arithmetic design system," *Proc. of 18th Design Automation Conference*, pp. 314-321 , July 1981.

[75] F. Mileto and G. Putzolu, "Average values of quantities appearing in boolen function minimization," *IEEE Trans. on Computers*, vol. EC-13, pp. 87-92, April 1964.

[76] S. Muroga, *Logic design and switching theory*. John Wiley & Sons, New York, 1979.

[77] Y. Kambayashi, "Logic design of PLA," *IEEE Trans. on Computers*,, vol. 609-617, no. 9, September 1979.

[78] R. Cuttler, *Algebraic derivation of minimal sums for functions of a large number of variables*. PhD Thesis, Dept. of Computer Science, University of Illinois, Apr. 1980.

[79] S. Hong, R. Cain, and D. Ostapko, "MINI: a heuristic approach for logic minimization," *IBM Journal of Research and Development*, pp. 443-458, 1974.

[80] Z. Arevalo and J. Bredeson, "A method to simplify a boolean function into a near minimal sum of products ffor programmable logic array," *IEEE Trans. on Computers*, vol. C-27, no. 11, pp. 1028-1039, 1978.

[81] T. Sasao, "Multiple-valued decomposition of generalized boolean functions and the complexity of programmable logic arrays," *IEEE Trans. on Computers*, vol. C-30, no. 9, pp. 635-643, September 1982.

[82] L. Glasser , "The analog behavior of digital integrated circuits ," *Proc. of 18th Design Automation Conference* , pp. 603-612 , July 1981.

[83] P. Penfield Jr. , "Signal delay in RC tree networks ," *Proc. of 18th Design Automation Conference* , pp. 613-617 , July 1981.

[84] D. Greer, "An associative logic matrix," *IEEE Journal of Solid State Circuits*, vol. SC-11, no. 5, pp. 679-691, 1976.

[85] J. Paillotin , "Optimization of the PLA area ," *Proc. of 18th Design Automation Conference* , pp. 406-410 , July 1981.

[86] J. Roth, "Methods in design optimization-array logic," IBM Research Report RC-4943, 1974.