To the Graduate Council:

I am submitting herewith a dissertation written by Chandra Tan entitled "Physical Design of Area-Array Integrated Circuits." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosopy, with a major in Electrical Engineering.

Donald W. Bouldin

Dr. Donald W. Bouldin, Major Professor

We have read this dissertation
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

PHYSICAL DESIGN OF AREA–ARRAY INTEGRATED CIRCUITS

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Chandra Tan

May, 1997

Dedicated to

my parents, brothers and sister for their love and support

ACKNOWLEDGEMENT

I owe a great deal of gratitude to Dr. Bouldin for giving me the opportunity to work under his guidance; thank you for all your patience, supports, encouragement, and technical advice. Many of the research ideas presented in this dissertation originated during discussions with him and Dr. Dehkordi.

I would also like to thank:

- Dr. Bodenheimer and Dr Langston for serving on my committee and reviewing this dissertation.
- Dr. Peyman Dehkordi for assisting me with some other technical supports.
- the Defense Advanced Research Agency and Army Research Office for providing the resources and financial support for the research.
- the wonderful staff of our department.
- the staff of the Interlibrary Loan Services for getting the necessary refence material for this work.
- my wife, brothers, sister, sisters–in–law, and brother–in–law for their love and support and for standing by me during this period of my graduate study.
- my mother and father for their love and support and for the sacrifices they have made.

ABSTRACT

The continuing drive towards more complex integrated circuits (ICs) having lower cost, more inputs or outputs (I/Os), greater operating speed, increased function per chip, and smaller device geometries has pushed the connection requirements beyond the capability of the traditional perimeter–lead IC package. Due to these factors, flip–chip die attach processes have been developed to support pad–limited IC designs, allowing semiconductor designers to take advantage of this technology to place pad sites in an array on the surface of the die, rather than just around the perimeter. In this dissertation, a new framework for designing an area–array IC has been developed and implemented. The I/O circuitry/buffer and the pad are separated so that the buffer can be placed anywhere in the layout. In the input data preparation stage, the hierarchy of the logic design is flattened and the necessary information is extracted from the layout and imported to the pad generation tool. The Pad generation tool operates in three stages. The first stage is a placement of the possible area–array pads on the top metal layer of the design which may contain some prerouted wires and keepout areas. The second stage is the assignment of I/O ports to their closest pads using a bipartite weighted matching algorithm. The third stage physically connects together the I/O ports to the closest pads using a modified maze router based on A* search. Finally, the resulted layout of the area–array padframe is added to the chip core to generate an area–array IC.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

FIGURE **PAGE**

FIGURE PAGE

LIST OF TABLES

TABLE **PAGE**

CHAPTER 1

INTRODUCTION

Physical design of an integrated circuit (IC) is the process that begins with a logic module specification and synthesis information and adds geometric information that can be used in the fabrication of the IC. Traditionally, physical design consists of placement of the modules and routing of the interconnections between these modules.

In conventional physical design of ICs, the designer places all the active electronic components and combines them into a chip core. Then, for the IC terminations, the designer places the I/O pads around the periphery of this chip core and makes the connections between them. With this peripheral chip termination method, the I/O count capability is a function of the perimeter of the chip and the bond pad pitch. As a result, the chip becomes pad–limited when the space required by I/O pads exceeds the perimeter.

Today, the continuing drive towards more complex IC devices having lower cost, higher number of I/Os, greater operating speed, increased function per chip, and smaller device geometries has pushed the interconnection requirements beyond the capability of this traditional peripheral chip termination method. Due to these factors, die attachment processes based on flip–chip technology have been

developed to support pad–limited IC designs, allowing semiconductor designers to take advantage of this die–attachment technology to place pad sites in an array on the surface of the die, rather than just around the perimeter. Figure 1.1 shows the perimeter and area–array bonding techniques.

Area–array solder interconnections can be made using the flip–chip assembly technique. Known also as the C4 process (Controlled Collapse Chip Connection), flip–chip bonding was developed by IBM in 1964 and has been used for over 30 years to assemble IBM's mainframe computer modules [54]. Flip–chip gives the highest chip density of any packaging method to support pad–limited IC designs. Instead of placing the chips in space–wasting individual packages, flip–chip places them face down onto matching connections on a substrate or board by means of solder pads or bumps. Since the connections are under the chip, no additional space is required for bond wires. Connections may be made anywhere on the surface of the chip, using the whole chip area for connections, instead of just around the perimeter. The solder bumps provide shorter connections avoiding the performance penalties of conventional bond wires. Figure 1.2 shows the peripheral bonding IC and the area–array bonding IC.

In recent years, several commercial companies have started offering bumping and flip–chip services. Flip–chip technology is expected to grow at a compound annual growth rate of 38% through the year 2001 [3]. Flip–chip technology does have a major drawback in that not every IC design is built using this process, so a decision to use flip–chip for all the devices in multichip modules or boards may

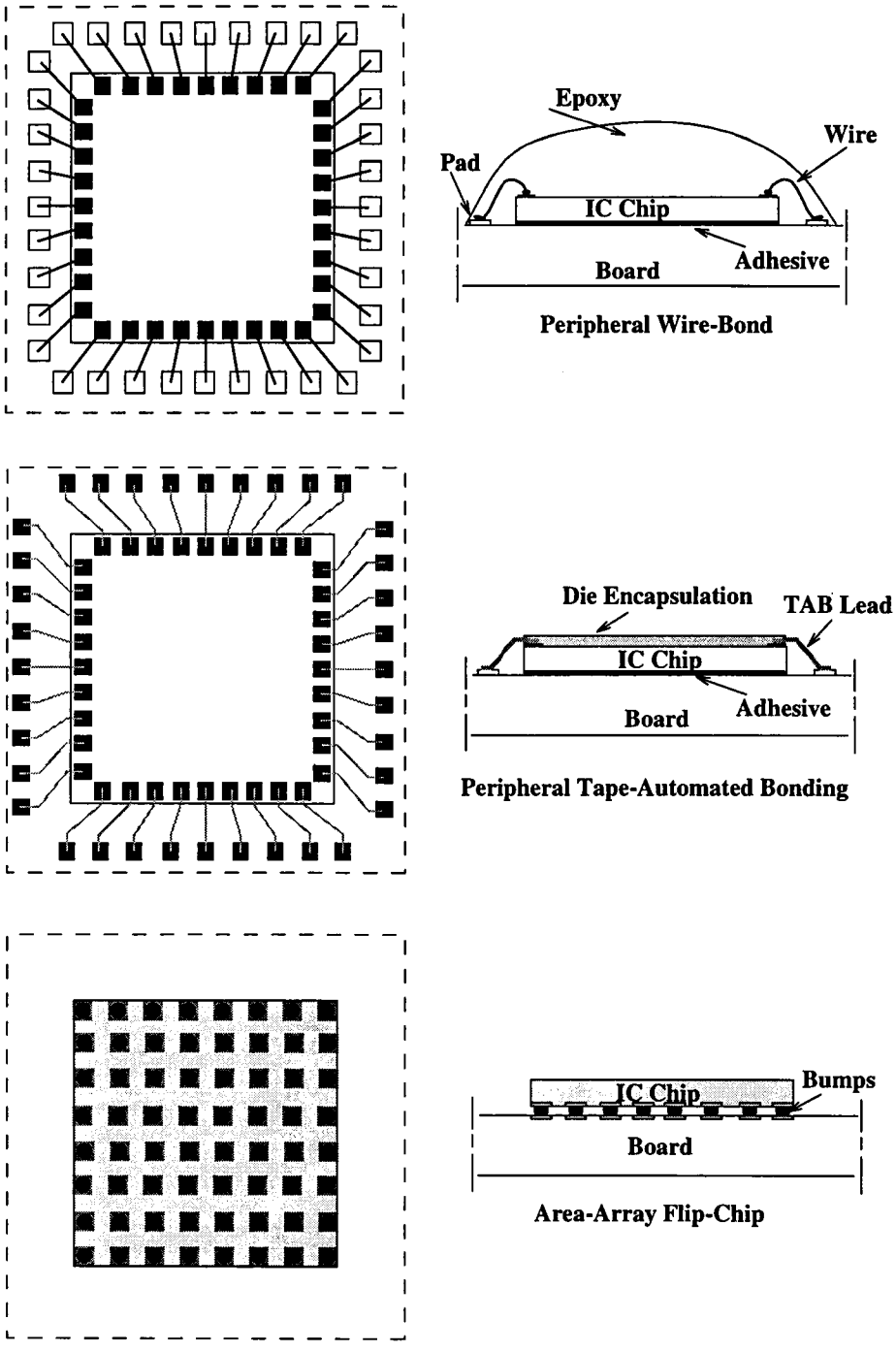
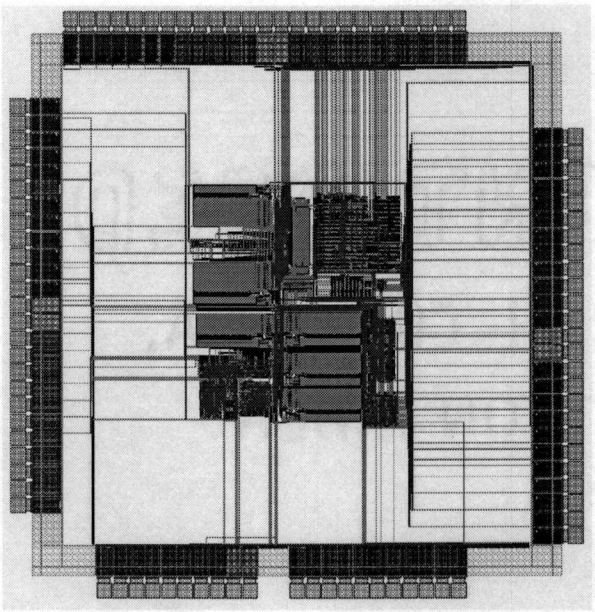
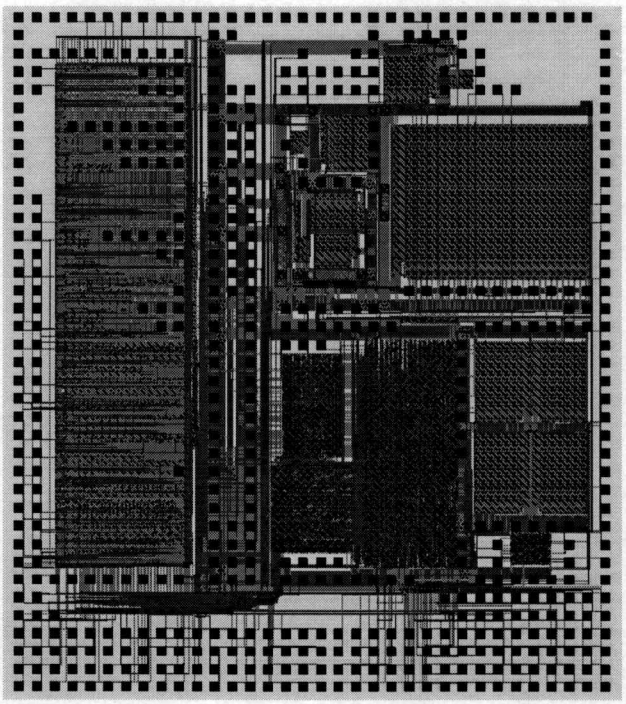


Figure 1.1: Perimeter and area–array chip–bonding technology.



(a)



(b)

Figure 1.2: (a) Peripheral bonding IC and (b) Area–array bonding IC.

require postprocessing the die to put solder bumps on the area–array pads. This postprocessing step involves the placement and routing of the area–array pads on the existing top metal layer of the die or on an additional distribution metal layer.

In this dissertation, a framework for designing an area–array IC is developed. In this framework, the I/O circuitry/buffer and the pad are separated so that the buffer can be placed anywhere in the layout. The proposed strategy is to generate the area–array padframe to be connected to the I/O ports of the IC. In the input data preparation stage, the hierarchy of logic design of the chip core is flattened and some necessary information is extracted from the layout to be imported to the pad generation tool. Pad generation operates in three stages. The first stage is a placement of the possible area–array pads on the top metal layer of the design which may contain some prerouted wires and keepout areas. The second stage is an assignment of I/O ports to their closest pads using a bipartite weighted matching algorithm. The third stage physically connects together the I/O ports to the closest pads using a modified maze router based on A* search. Finally, the resulting layout of the area–array padframe is added to the chip core to generate an area–array IC. For a chip with a small number of I/Os, the placement and routing of the area–array pads can be performed manually by the designer to achieve total placement control to prevent wasted space. However, in modern chips there can be hundreds of pads; placing them manually will be very time consuming and error prone. An automatic pad layout tool is developed in this

work for assisting the chip designer with all aspects of area–array pads.

1.1 Motivation

As the functions of IC chips have been remarkably enhanced, the number of pads for interconnections on a chip has increased in recent years. It is necessary to change the bonding method between IC chips and substrates.

VLSI is characterized by high integration and high speed. High integration brings about a remarkable increase of input/outputs (I/Os) that leads to the reduction of pad pitch and an increase in chip size. In the case of logic circuits, the relationship between the number of gates and the number of I/Os can be expressed by Rent's Rule.

The basic equation for Rent's Rule [54] is:

$$P_{I/O} = \alpha G^{\beta},$$

where $P_{I/O}$ is the number of signal I/O, G is the number of gates, and α and β are the coefficient and exponential factors, respectively, that vary with the type of logic being considered. Table 1.1 shows the Rent's constants for several types of logic function.

Figure 1.3 shows the plots of Rent's rule for circuits in Table 1.1. It can be seen that, even with the same number of gates, the required number of I/Os is very different. Gate arrays require the most number of I/Os followed by microprocessor ICs.

Table 1.1: The coefficient and exponential factors for different types of circuit.

Circuit Type	α	β
Gate Arrays	1.9	0.5
Logic Cells	3.2	0.434
Microprocessors	0.82	0.45
Functionally Complete Chips	7.0	0.21

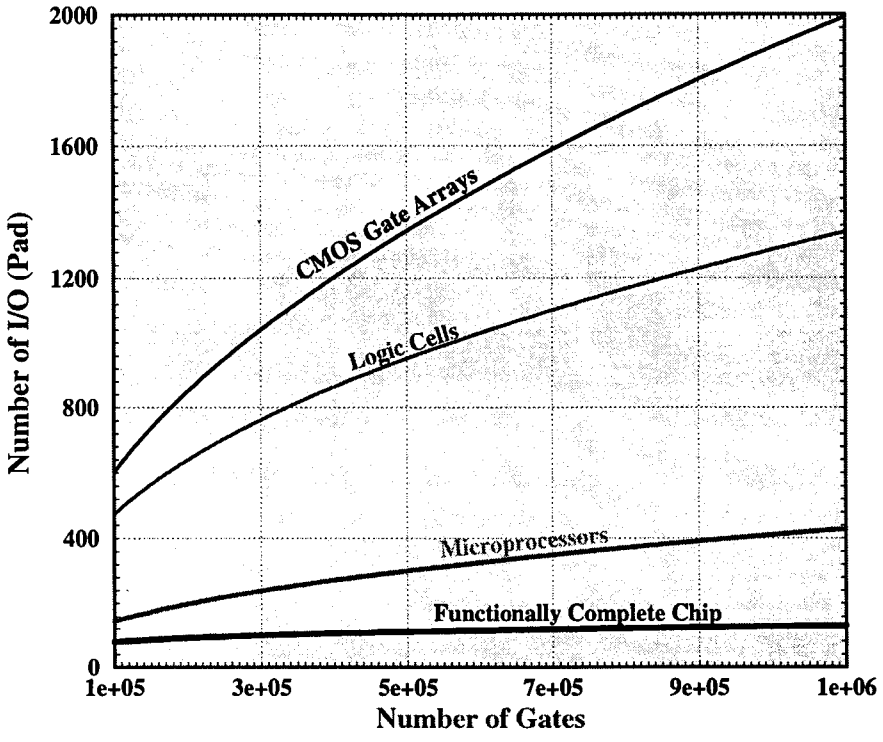


Figure 1.3: Number of required I/Os on a chip.

For a given chip size, the physically possible number of I/Os (pads) depends on the pad size, pad pitch, and pad arrangement. The number of physically possible pads available on an area–array bondings is given by

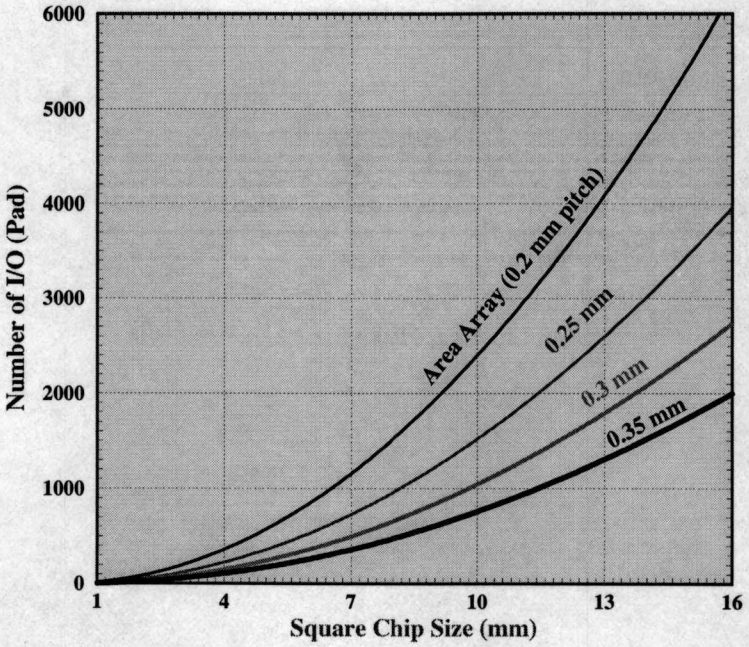
$$P_a = \left(\frac{x}{p} - 1\right)^2$$

With peripheral bondings, the pad count capability is a function of circumference of the chip and the pad pitch; this function is given by

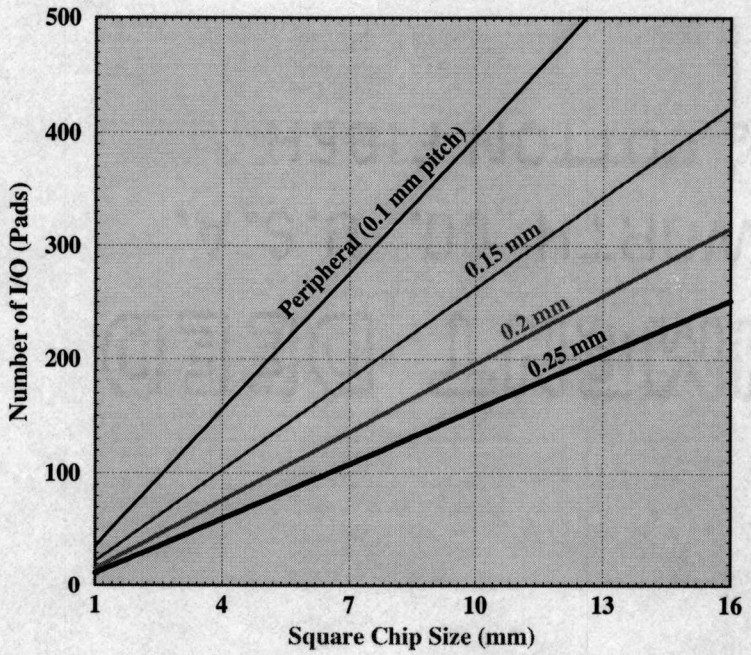
$$P_p = 4\left(\frac{x}{p} - 1\right)$$

In both equations above, p is the pad to pad pitch (mm) on the chip surface, and x is the side dimension of a square chip (mm). These two equations are plotted in Fig. 1.4 for several values of p and x.

Pad–to–pad spacing or pitch is primarily dependent upon substrate technology. Area–array bond pitches are typically larger than comparable wire–bond pitches. For chips less than $9\,mm^2$ or $3\,mm$ on a side, the peripheral bonds with pitch of $0.1mm$ provide greater interconnect density than the area–array technique with pitch of $0.25mm$. On the other hand, for chips larger than $9mm^2$, area–array interconnect density dominates. From Fig. 1.4, for chip size of 100 mm^2 or 10 mm on a side, the number of pads with $0.25mm$ pitch for area–array is approximately equal to 1600 and for the periphery, the number is approximately equal to 400 with $0.1mm$ pitch. This is close to a 400% increase in interconnect density.



(a)



(b)

Figure 1.4: Number of possible I/Os on a chip: (a) area–array and (b) peripheral connection.

Today a high I/O density requirement comes about because the I/O bus width on modern microprocessors continues to increase. Add to these signal I/Os, the additional I/O requirement for power terminals and the knowledge that chips will not continue to increase in area to accommodate the total I/O (the reason logic chips won't get much larger than they are today, up to 20 mm square, is due to yield–cost and on–chip performance limitation [13]), this leads to the realization that a denser area technique for chip connection will have to be employed. An example of this type of chip connection is the area–array pad technique. These high chip total I/O densities will be the impetus for MCMs because they have the high via and wiring densities that are required to support this high interconnection density.

The area–array solder interconnections can be done using the flip–chip assembly design. Flip–chip places the area–array ICs face down onto matching connections on a substrate or board. Since the connections are under the chip, no additional space is required for bonding. Connections may be made anywhere on the surface of the chip, using the whole chip area for connections, instead of just the edges. Shorter connections avoid the performance penalties of conventional bonding. This is a major benefit as smaller, denser, more powerful chips require increased I/O according to the Rent's rule. Flip–chip technology was developed by IBM in the 1960's, and has been used in millions of IBM circuits [5, 31]. As with any technology, despite all the advantages mentioned, flip–chip bonding has some limitations. One of these limitations is the physical design

issue of dies that have a large number of bumps in an area–array. Unless large quantities of dies are procured, semiconductor manufacturers are reluctant to add bumps to their dies or redesign for area arrays. Users are then faced with having to develop in–house processes or contract for outside services to prepare the wafers and insert bumps on the dies. Another limitation is the present lack of commercial chip place and route software tools for area–array interconnect.

Some studies have been conducted recently to show the impact of area–array flip–chip bonding on VLSI designs. The study conducted in [12] showed the reduction of die size and the increase of I/O count when peripheral wire–bond technology was replaced by area–array flip–chip technology. A conceptual trade–off analysis between peripheral and area–array bonding in MCMs is presented in [47]. The impact of partitioning an ultra–large single die into multiple smaller dies housed on a MCM is studied in [13]. The result of this study indicated a dramatic reduction of cost if the CPU studied were divided into three chips bonded using area–array technology and interconnected on MCM–D substrate. In addition, from Figure 1.3, note that among four types of circuits, the functionally complete chip requires the least number of I/Os and can thus be the driver for low cost MCM applications. From these studies, we can see the need for a design automation tool that can handle the physical design of an area–array IC.

1.2 Problem Statement

The fundamental features of the pad layout problem are two sets of terminals (I/O ports and bonding pads) placed arbitrarily in a single plane that need to connect to one another. The pad layout problem in area–array IC designs can be stated below: Given a "core" portion of the chip that already contains the I/O circuitries, place the possible uniformly spaced area–array pads on the top metal layer of the design which may contain some prerouted wires and keepout areas and route these pads to the I/O ports of the chip using a limited number of available connection layers, such that design rules are obeyed and some other objective functions are satisfied. Among many objective functions, the most important one is to minimize the routing wire length because it directly affects the circuit performance.

The specialized nature of this routing problem requires algorithms optimized to the task, which will be implemented in a program called Automatic Area–Array Pad Router (A3PR). As with any development of a layout tool, several considerations must be identified such as:

1. What is the strategy to place and route the I/O circuitries and how can the I/O ports and the keepout areas be identified in chip core design?

2. How should the chip core design be converted into A3PR's internal database and what kind of data structures should be used to represent the geometries contained in the design?

3. How should the physical constraints such as design rules, minimum pad sizes, and pad spacing guidelines be handled.

4. What kind of strategy should be used to identify the physical locations of the pads and how should these pads be placed?

5. What is the best routing algorithm to use to perform the routing from the I/O ports to the pads?

1.3 Goals and Expected Contributions

The main goal of this work is develop a design framework to assist the IC designer to layout area–array pads for flip–chip bonding. This work aims at developing an automatic placement and routing of area–array bond pads to the chip core circuitry. The method presented in this dissertation is very general and flexible and is designed to handle all the physical constraints of the pad layout problem for an arbitrary number of routing layers. The tool developed can be employed as a post–processor to any VLSI CAD tools.

1.4 Organization of The Dissertation

The dissertation is organized as follows. Chapter 2 provides some background on the basic concepts of chip attachment and a survey of previous work involving area–array IC design. It also has an overview of automatic physical design

methodologies used today for peripheral ICs especially for routing algorithms. Chapter 3 presents the methodology of automatic physical design of an intrinsic area–array IC. The various phases of area–array pad layout proposed in this dissertation are described individually. In Chapter 4, the implementation of the proposed area–array layout method using the framework of the *Magic* CAD tool [1, 41] is presented using a design of the "Quantizer" circuit taken from the tutorial of the Epoch design automation tool. The pad placement based on "corner stitching" data structure is described in detail. Also the pad routing based on a general purpose router built in *Magic* called *mzroute* is presented. The results of some intrinsic area–array ICs are included in this chapter. Finally, Chapter 5 summarizes the contribution of this dissertation, the conclusions found and suggestions for possible future work.

CHAPTER 2

BACKGROUND

2.1 Chip Attachment Method

The typical process used in the production of electronic systems consists of fabricating many identical ICs on a single wafer, separating and packaging the ICs and then externally connecting the packaged ICs on some type of interconnect medium, such as a printed circuit board (PCB). In an MCM, unpackaged ICs are mounted on a substrate which contains multiple layers of interconnect. The ICs are mounted to the substrate and electrically connected (by wire–bonding or other methods) to the substrate.

Multichip module packaging technology is an important technique which results in high chip density, small interchip propagation delay, low power consumption, and high interconnect density. A four–layer thin film MCM process may have a maximum interconnection density as high as 300–800 linear *cm*s of interconnections per square *cm* of die area [31]. In addition, an MCM can accommodate many different types of ICs, such as logic, memory, analog, etc.

All multichip modules require the attachment of bare die to the substrate. Since the die are unpacked, care must be taken during assembly to avoid damage to the unencapsulated silicon. There are three basic methods for assembly of the

bare die to the board as shown in Figure. 1.1:

- Chip–and–wire (wire–bonding)
- Tape automated bonding (TAB)
- Flip–chip attachment

Among these three methods, flip–chip gives the highest chip density of any packaging method. Instead of placing the chips in space–wasting individual packages, flip–chip places them face down onto matching connections on a substrate or board. Since the connections are under the chip, no additional space is required for bond wires. Connections may be made anywhere on the surface of the chip, using the whole chip area for connections, instead of just the edges. Shorter connections avoid the performance penalties of conventional bond wires.

2.2 Area–Array Flip–Chip Design

Flip–chip assembly is projected to replace wire–bonding as the predominant method of connecting to high performance, high pin–count integrated circuits. First explored by IBM more than 30 years ago with their now famous C4 (Controlled Collapse Chip Connection) flip–chip process [38], area–array flip–chip bonding is now finally coming of age. In the original IBM thermal conduction modules used in the main frame computers, each die had an area–array of 121 solder bumps [5]. In recent designs, arrays of 762 solder bumps per die have been

achieved [53]. In the late 1980's, Motorola realized flip–chip, area–array technology would provide the step function in chip–to–package interconnects essential for future generations of complex semiconductor devices. Subsequently, Motorola benchmarked the industry's flip–chip technologies with a goal to leverage the best process in future products. One of Motorola's first commercial area–array (C4) products is the PowerPC 601 RISC microprocessor in a ceramic quad flat pack [25]. The others included Motorola's 88110 RISC microprocessor and 88410 memory management unit which are both available in C4 Ceramic Ball Grid Array packages [32]. All the products were designed by adding a redistribution layer on the top of the chip to convert the peripheral connections to C4 area–array design. Figure 2.1 shows the redistributed area–array PowerPC 601. Presently, Motorola is designing new C4 products and converting older designs to C4. These products are utilizing C4 to meet performance and/or size goals. Figure 2.2 illustrates the methodology for converting the peripheral bonding IC to area–array bonding IC.

Because existing chips designed for wire–bonding with peripheral–bond pads were used, it was necessary to relocate some bumps when the pads were too close to each other for proper patterning. In addition, for optimum assembly yield, for reduced stress, and for improved thermal performance of the assembly, more than one bump per I/O was sometimes used. In such cases, the barrier metal layer was used to provide connection to the I/O's and to define bump pads at alternate locations closer to the chips' centers. This resulted in bumps over

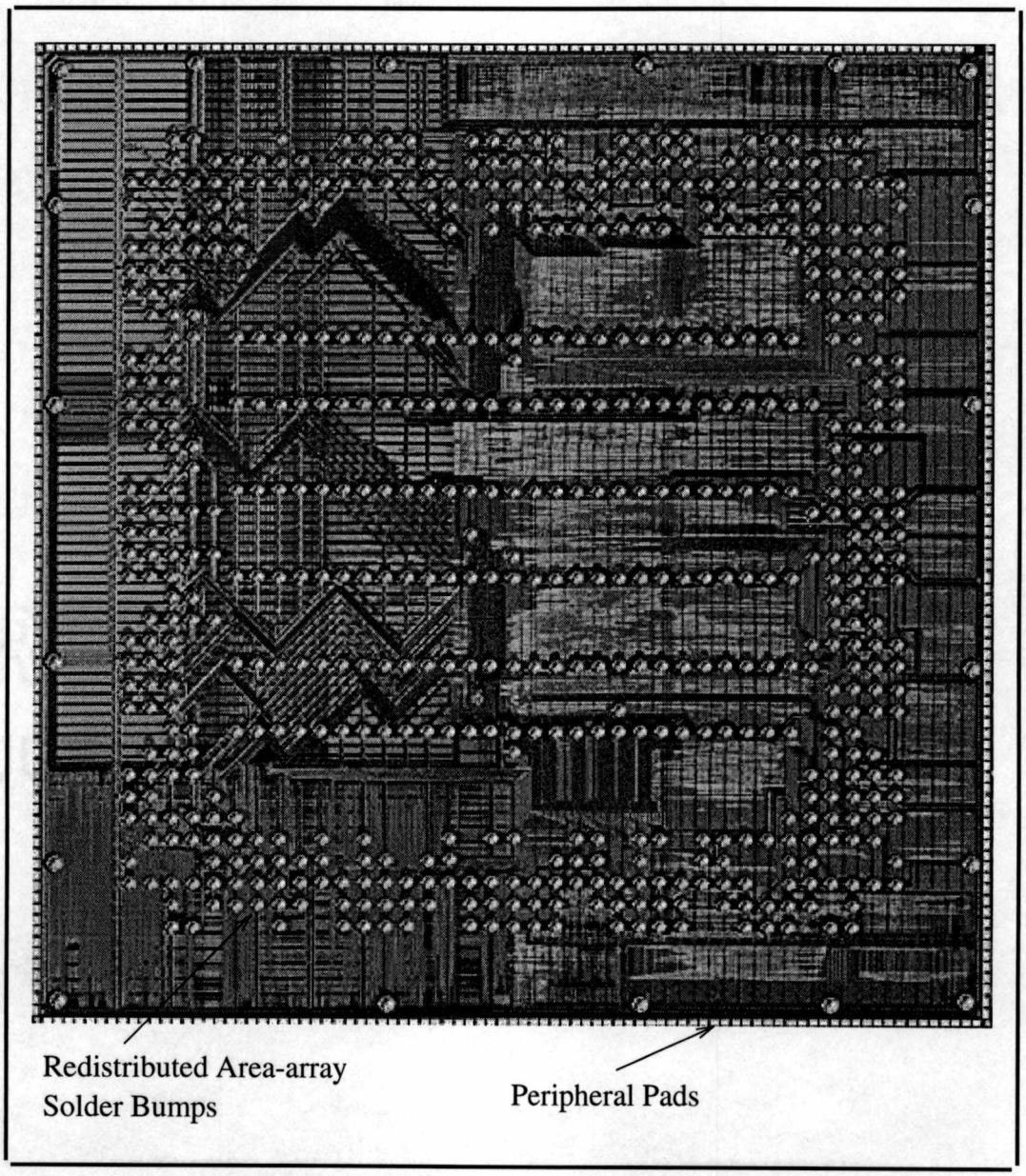


Figure 2.1: An example of area–array IC with redistribution. Die photo was obtained with permission from URL address: *http://micro.magnet.fsu.edu/chipshot.html.* (Copyright ©1995–1997 by Michael W. Davidson and The Florida State University).

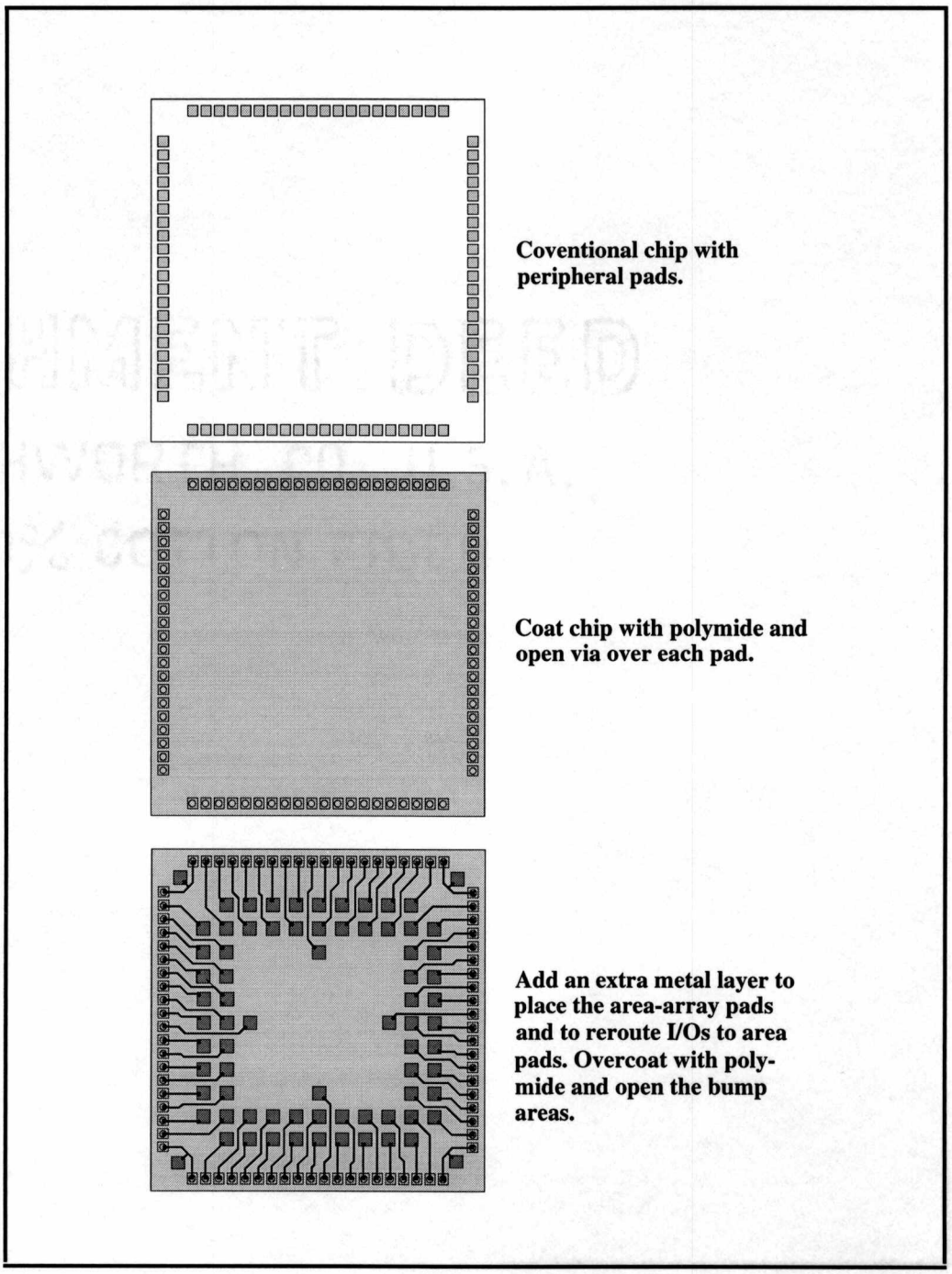


Figure 2.2: Illustration of conversion of peripheral bonding to area–array bonding by rerouting on an extra layer.

active circuit areas. Using the barrier metallization as a signal layer with chips designed specifically for bump processing, it is possible to design chips around circuit constraints rather than pad constraints; that is, it is possible to place buffer circuits where they make the most sense from a circuit design standpoint rather than where they are required for periphery–bond purposes. The mismatch of the pitch of area–array of bond pads on the IC and the substrate is very common. To enable flip–chip attachment, a flexible method for bridging this gap is required. By moving the pads on the substrate closer together to meet the configuration on the IC, or redistributing the pads on the IC to match that of the substrate as shown in Figure 2.3. Some algorithms have been developed to perform the routing from the peripheral bonding pads to the area–array C4 pads [9, 58, 7]. Figure 2.3 shows two methods of pad redistribution to accommodate a specific pitch. One of the methods shown in Figure 2.3(a) is to redistribute the peripheral pads to the desired flip–chip pad locations on the IC using an extra metal layer. Algorithms presented in [9, 58] can be used to do the routing. Another method shown in Figure 2.3(b) is to reroute the substrate bond pads using distribution layers to allow flip–chip bonding of the IC to the substrate. The work presented in [7] can be used for this purpose.

2.3 Present Approaches for Designing An Area–Array IC

Recently, most military and commercial electronic products are evolving toward lower cost, smaller form factor, lower weight and higher performance. All

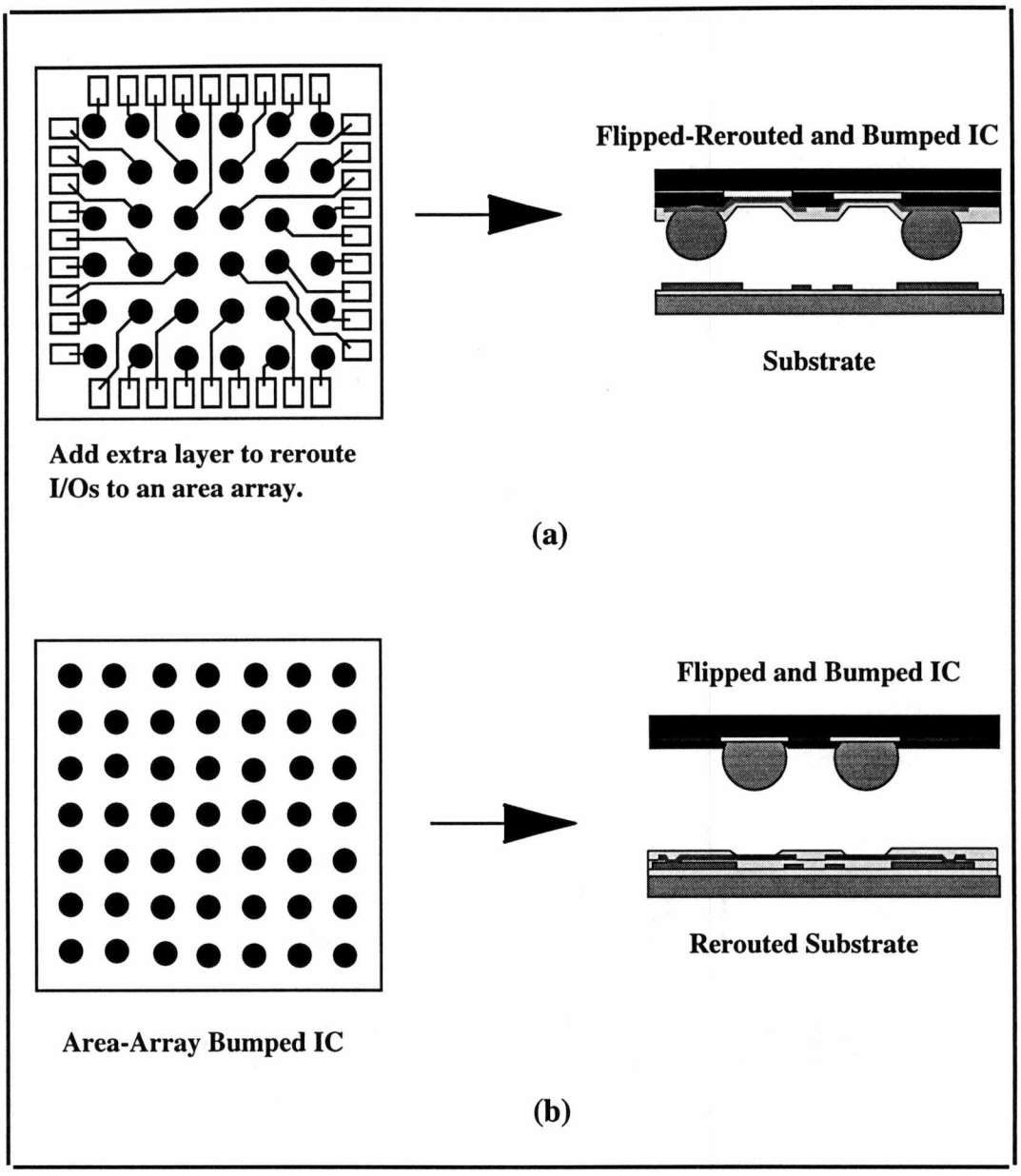


Figure 2.3: Two methods of redistribution for pitch matching: (a) Rerouting the pads to larger pitch on the IC and (b) Redistributing the pads on the substrate using distribution layers.

the improvements can best be realized at the chip level by eliminating the IC packages and transitioning from perimeter to area I/O.

Presently, there are two major approaches for designing an area–array IC: **intrinsic** and **extrinsic**. An intrinsic area–array IC is one specially designed and laid out for area–array bonding. Whereas, an extrinsic area–array IC is one originally designed and laid out for periphery–bonding but has been converted for area–array bonding by means of a redistribution layer. As shown in Figure 2.4, both types of ICs take advantage of the superior electrical performance of solder bumps and a smaller footprint as compared to wire–bond ICs.

In the first part of Figure 2.4, peripheral wire–bonded ICs are attached to the MCM substrate using wire–bond technology. This method uses large area for die footprints and the chips need to be placed further apart to accommodate the wire–bonding processes. The total delay from the core circuits to the MCM equal to the sum of the delay from the chip core to the peripheral pad, the delay in the buffering circuit, the contact (bonding pad), and the bonding wire. In the second part of Figure 2.4, the wire–bonding process is replaced by the C4 process to eliminate the long bonding wire to help performance. The delay of the solder bump is much less than that of bonding wire. In addition, the placement of the ICs requires less area because of the usage of flip–chip processes. Pitch mismatching between the IC and the substrate can be resolved using the redistribution process on the substrate layers [56]. In the third part of Figure 2.4, the conversion from the peripheral to area–array IC is done by redistributing the

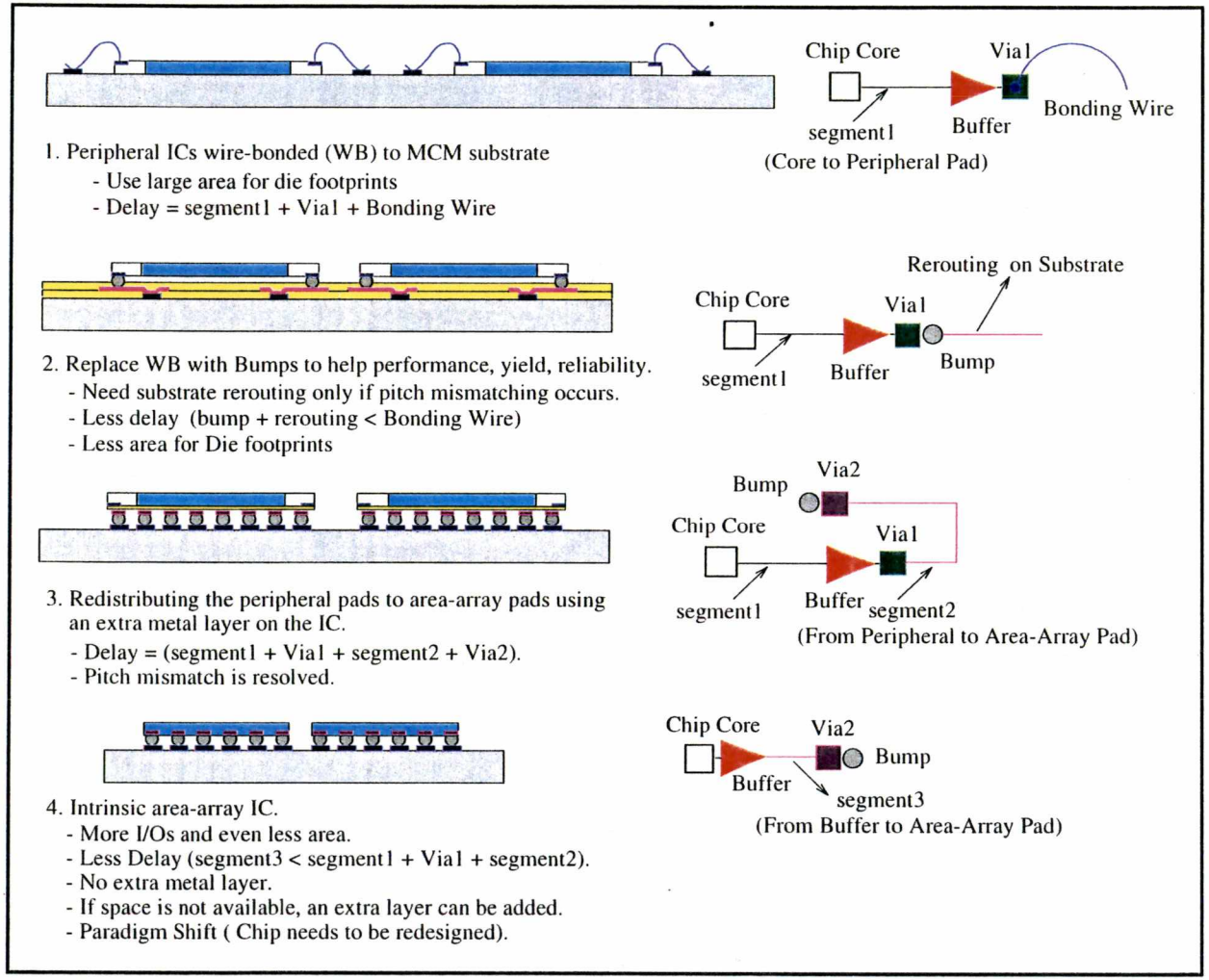


Figure 2.4: Why design an intrinsic area–array IC.

peripheral pads to area–array pads using an extra metal layer. In this case the pitch mismatch is resolved. This is a semi–optimal method for producing an area–array IC. It can be done on any existing peripheral IC. The advantage of the method is that the IC does not need to be redesigned and retested. However, the I/O count of the IC is still limited by the constraint of the perimeter of the chip. We call this method the **extrinsic** method. Recently there are some reports of converting the peripheral wire–bond IC to area–array flip–chip IC using a redistribution layer [9, 32].

In the last part of Figure 2.4, The IC is redesigned to eliminate the routing from the chip core to the peripheral pads. The I/O buffering circuits are included in the chip core. The area–array pads are placed on the existing top metal layer on the whole surface of the chip core. The interconnections from the I/O ports to their area pads are performed on the existing layers of the design. This approach eliminates the need for the redistribution layer as discussed above. Because there are more I/Os that can be placed on the entire surface of the chip, more power/ground and clock pads can be used to enhance the performance of the chip as well as to ease the on–chip routing [59]. This approach is called the **intrinsic** method. Several approaches for designing an intrinsic area–array IC have been reported recently [14, 18, 30]. The work presented in [11] quantitatively showed additional benefits can be gained by using intrinsic area–array ICs rather than using the extrinsic counterparts at the IC level.

Cascade Design Automation recently reported a preliminary version of a CAD

tool for designing area–array ICs [18]. This tool consists of area–pad power analyzer, area–pad floorplanner and area–pad router. In the area–pad power analysis phase, the power pads are placed at specific locations to optimize the power consumption with a relatively thorough power analysis. The area–pad floorplanner analyzes the position of the area pads and cell blocks and suggests the location for the I/O buffer cells for improving the system routing. Finally, an area router based on the maze running algorithm with A*–search is used to route the I/O buffers to the area pads. The routing is done on the upper two layers of metal which are not used during the block routing.

The paper presented by Kiamilev et. al. [30] demonstrated three methods of designing an intrinsic area–array IC. In this first method, an array of "pixel" cells with electronic processing circuitry, output pad driver and input pad receiver circuitry were replicated in a two–dimensional "pixel array" structure that matches the pitch of the area–distributed I/O padframe. The method can only be used for designing some special ICs. In the second method, which is similar to the approach proposed in this dissertation, the area–distributed padframe is integrated directly over the active circuitry of the IC. The design of the IC was done automatically using a CAE system. The placement and routing of the area–distributed padframe were done using full–custom manual layout which was very time consuming and error prone. This was the reason why they only attempted to design smaller ICs (less than 50K transistors). In their final method, the I/O pad circuits are placed and optimized separately from the core of the chip. The

area pads were placed directly over their corresponding I/O pad circuits to form a two–dimensional array structure called the pad interface module. This module was placed in the center of the IC. This method can be used to do a large IC design because it is compatible with current VLSI CAD tools, thus enabling the automation of placement and routing tasks. However, the density of this design is much lower than the design achieved using the second method.

The approach proposed in this dissertation is similar to the second method mentioned above. It allows the designer to use some of the CAE system to place and route the core of the IC. However, the placement and routing of the area padframe to the core of the chip are automated using our A3PR tool.

2.4 Routing Algorithms

A router determines the placement of the wiring paths that connect any electronic modules. Pins or terminals on any modules that are to be wired together are part of a set called a *net.* Nets are wired within the environmental constraints such as the number of layers, the minimum spacing between wires, the minimum width of the wire, etc. In addition, there are performance related requirements common to many routing tasks, such as finding the shortest possible connection to maximize the speed of the circuit. Maze routers [33, 37] form one broad algorithmic approach to routing tools. Other different algorithmic approaches also exist because particular routing approaches offer distinct advantages and disadvantages. Figure 2.5 shows a simplified router taxonomy derived from [43]

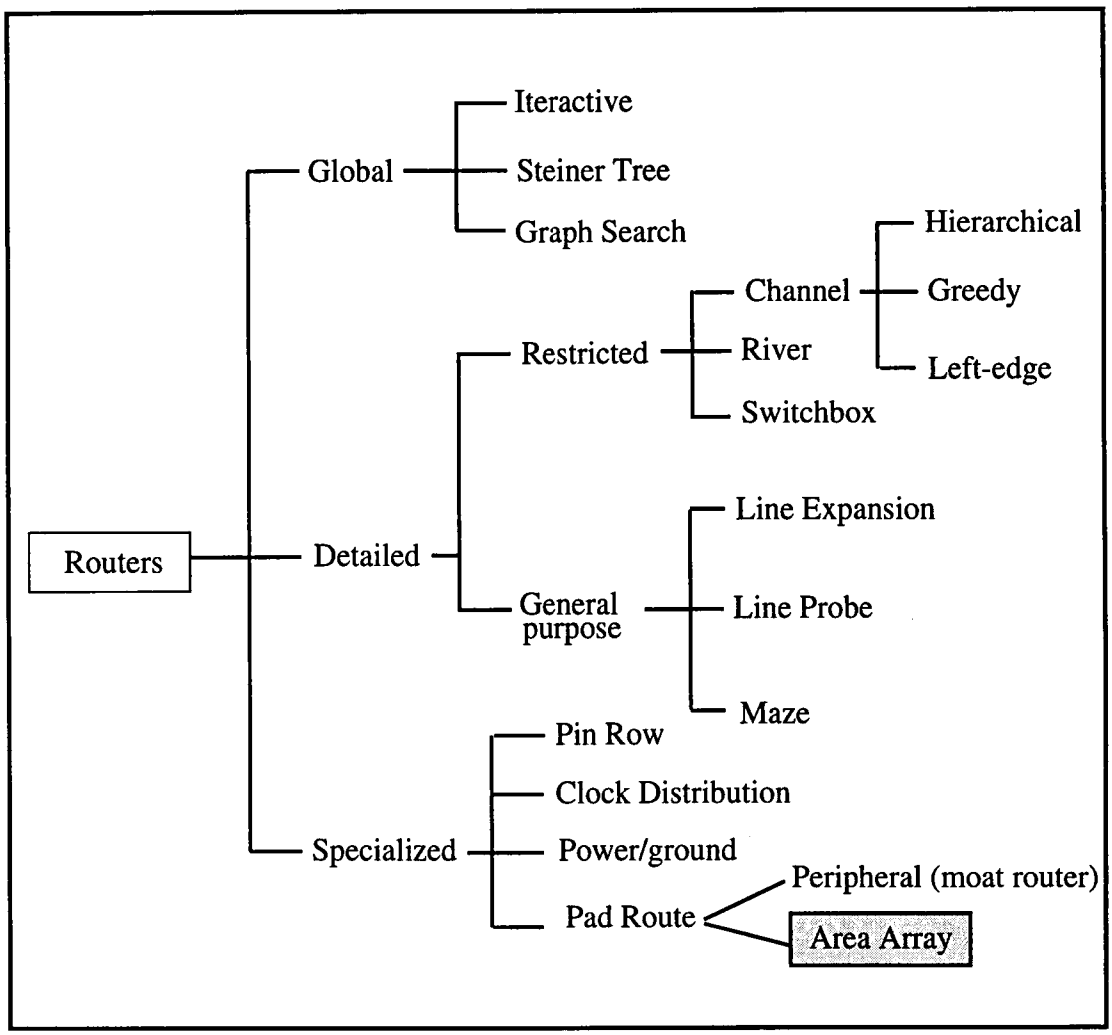


Figure 2.5: Taxonomy of routers.

In the following, some of these routers will be reviewed in more detail. A routing problem in integrated circuits dealing with several thousands of nets are decomposed into small subproblems using a divide and conquer approach. This problem is often organized hierarchically [6, 51]. First the total routing area is divided into regions. This is often referred to as channel definition step (Figure 2.6). Second, the nets are routed through the regions (Figure 2.7), but actual placement of the nets onto exact physical locations within each region is not performed. This assignment of nets to a subset of routing areas is done using a global router. The main consideration is the minimization of a congestion bottleneck, i.e., we try to avoid having too many wires trying to cross a region which might make the final detailed routing of the region impossible. Global routers tend to be one–wire–at–a–time routers. The algorithm may be either a line–probe [22] or maze router type or both. Third, because we now know which nets must cross each region, each region is then separately routed to derive the actual placement of the nets using restricted–area–routers such as channel and switchbox routers [8, 45, 57, 44, 20, 50].

In channel routing, the routing problem is decomposed into smaller problems of routing rectangular regions called channels. The routing region can be completely broken into channels to produce slicing, a structure–based layout as mentioned before. A channel (as shown in Figure 2.8(a)) has fixed pins on the top and bottom edges and floating pins "pseudo–terminals" on the left and right edges. When an edge has a floating pin on a net, it implies that the net cross-

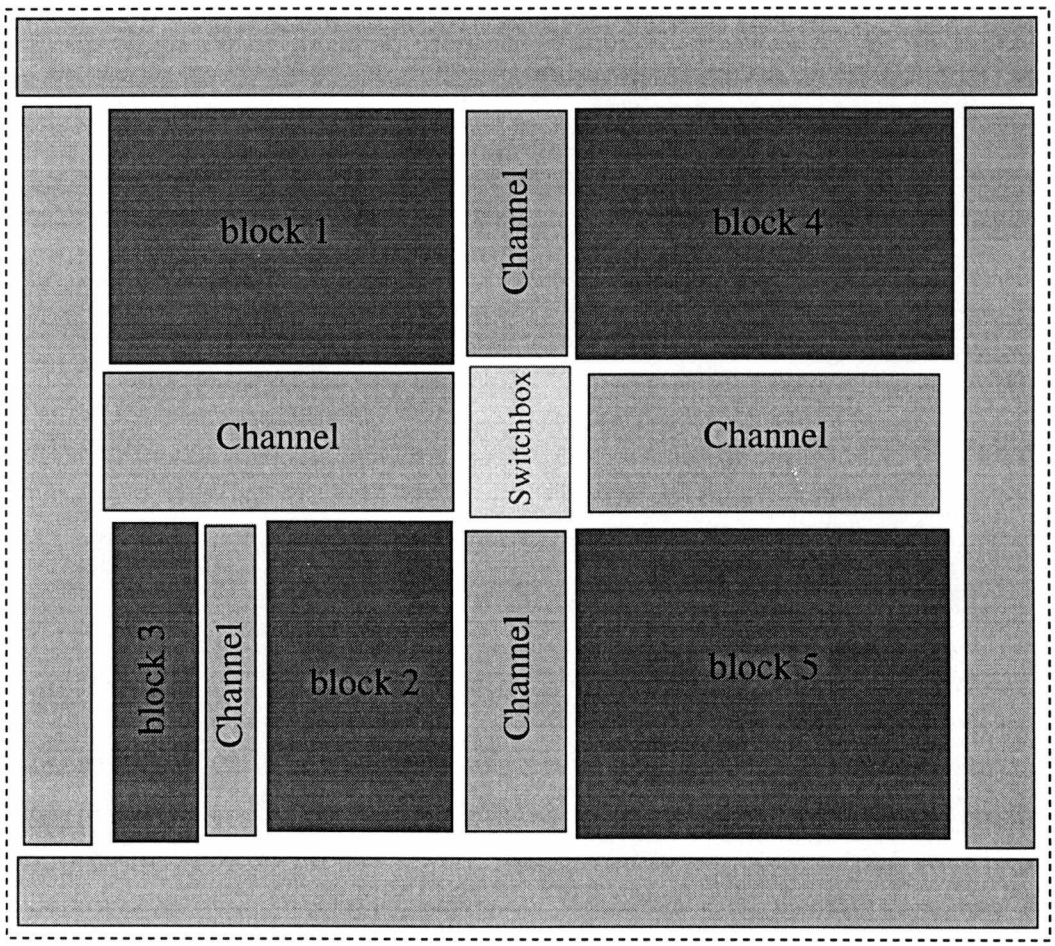


Figure 2.6: Channel definition: the routing regions are divided into channels and switchboxes.

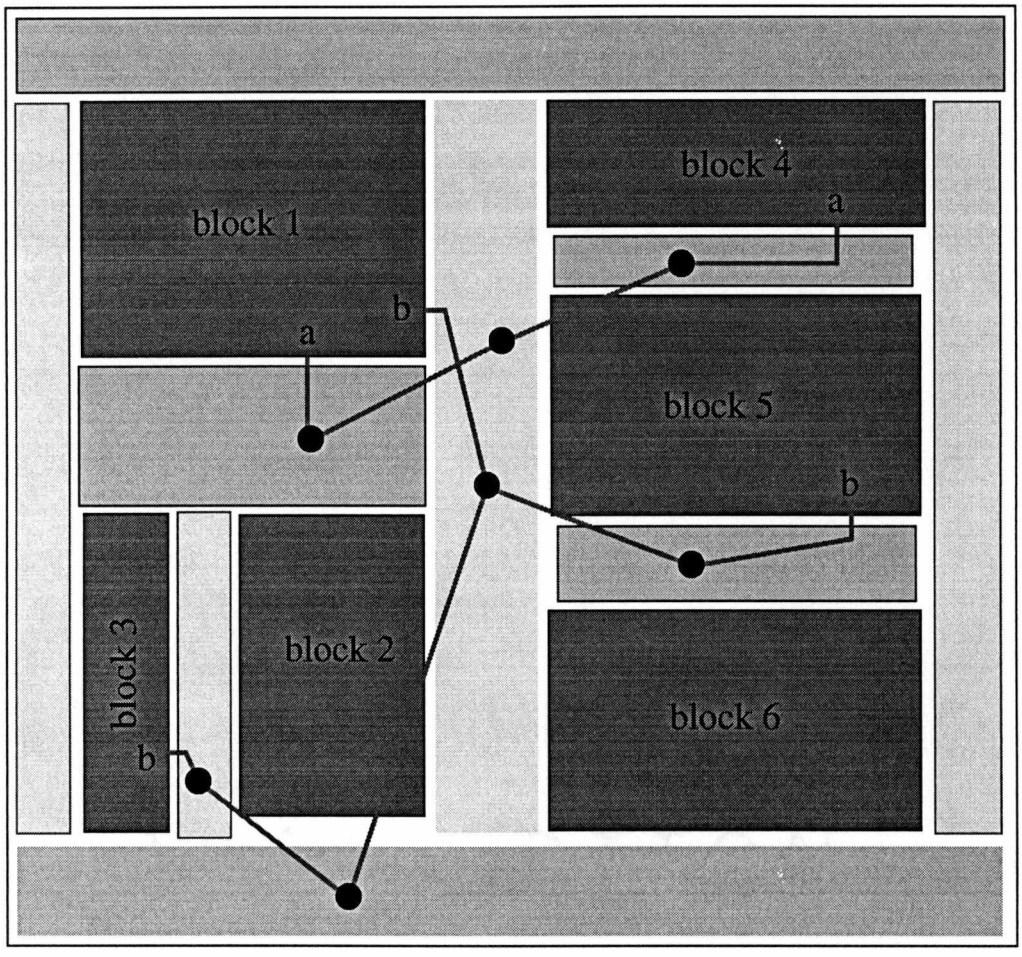


Figure 2.7: Global routing of net a and net b.

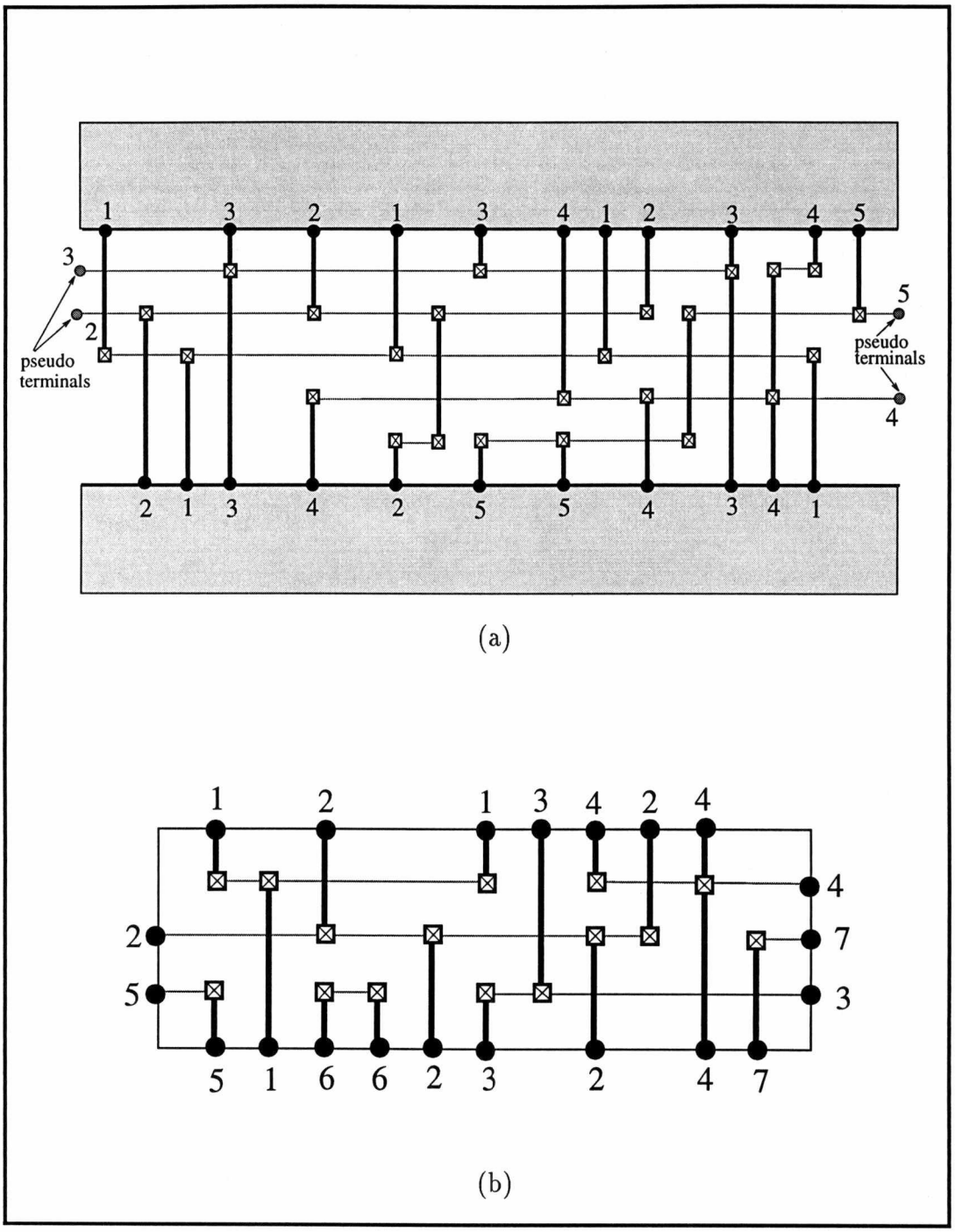


Figure 2.8: Restricted detailed routers: (a) channel router and (b) switchbox router.

es that edge, and the exact crossing position will be determined after routing. Many of the existing channel routers can handle irregular top and bottom edges enhancing the applicability of such routers.

A switchbox (Figure 2.8(b)) is a rectangular routing area with no obstruction inside and with signals entering from all four directions the channel routing problem is a particular case of the switchbox problem when the signals are allowed to enter from two opposite sides of the rectangle.

Area routing is a more general form of routing. In this form of routing, the routing region can be any irregular shape and can have obstacles and pins at arbitrary positions. The maze–running algorithm, which is also well known as the Lee's Algorithm [33], is one of the oldest general purpose area routers for connecting points on a plane and has been used as the underlining work–horse in many newer routing algorithms [50, 8]. The Lee–Moore method (or its modifications) is essentially a breadth–first search method (BFS) applied to a grid–graph. One of the reasons that Lee's algorithm is so popular is its guarantee of finding a best solution if one exists. However, its exhaustive searching nature makes it slow, especially when the problem size is large. The algorithm consists of three phases: 1) wave propagation, 2) backtrace, and 3) label clearance. In the first phase, a simulated wave is propagated out of the source by labeling all the empty cells next to the source "1," then labeling all the empty cells next to those labeled "1" as "2" and so on. The wave propagation continues until the target is reached or no more cells can be labeled. In the latter case, we conclude

there is no path between the source and the target. In the former case, a path can be found by performing the second phase. The backtrace process finds the shortest path by collecting a set of cells with distinct and decreasing labeling starting at the target and ending at the source. When there are multiple choices and the cell which includes the path does not change, the backtrace direction is preferred. Another weakness of Lee's algorithm is that the routing quality is dependent on the ordering of nets being routed. It is quite often that a net routed earlier will block another net being routed. Since trying out all possible orderings is computational intractable, heuristic ordering rules and/or rip–up reroute schemes are usually used in conjunction with Lee's algorithm to handle practical problems [10, 52]. The final phase clears all the unused labeled cells for the routing of subsequent nets.

Maze routing is actually a special case of the general single source shortest path problem (Dijkstra's algorithm) on weighted undirected graphs [16]. The main reason to distinguish maze routers from shortest path approaches lies in the regularity of the grid graph and the uniform weights associated with each edge. Hence, a maze router can be implemented using an efficient implementation of the shortest path algorithm given in graph theory.

This type of router often operates on a gridded model of the routing region and routes a single net at a time [33, 37]. The width of the grid is set to the pitch design rule for the routing layer. Pitch is defined as the sum of the minimum spacing plus minimum width for a layer. The center–to–center distance between

two adjacent routing tracks on the same layer must be greater or equal to the minimum pitch for the layer. A maze router starts from source port (V) and expands in a breadth–first manner labeling each grid with the current length until it hits the target port (P2) as shown in Figure 2.9.

Many enhancements [51, 39] have been made to improve the original Lee's algorithm, including: 1) starting point selection, framing, and double fan–out to speed up the search process; 2) coding schemes for labeling to reduce the storage requirement; and 3) extensions to multiple terminals and multiple layer interconnections.

Some area routers use the line search algorithm. This is a gridless algorithm in which horizontal and vertical lines are expanded from both the pins to be connected. If the lines originating from the two pins do not intersect because of obstacles, lines are expanded again from the points until they hit the obstacles. This process is repeated until they intersect. A path is obtained using a back–trace along the lines. The illustration of this router is shown in Figure 2.10.

Some routing problems in IC design need special attention. In synchronous VLSI systems, clock signals must reach each register at exactly (or almost exactly) the same time. Clock skew is due to the variations in the time of arrival of a clock signal at clocked elements in a digital system. The clock net is a global net which is over the whole chip. The main objective of a clock router is to route this net with minimum clock skew [26]. Routing the power and ground nets also

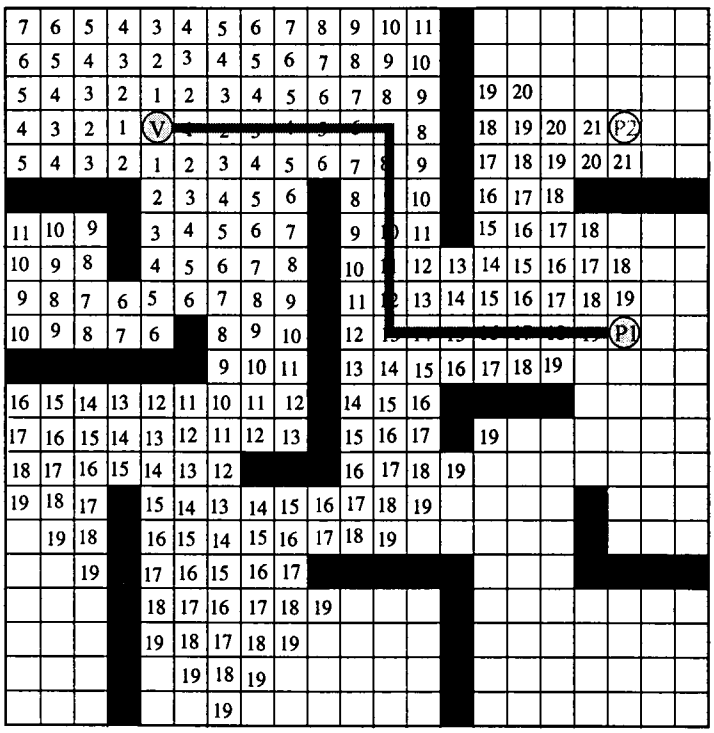


Figure 2.9: Lee's Maze Router.

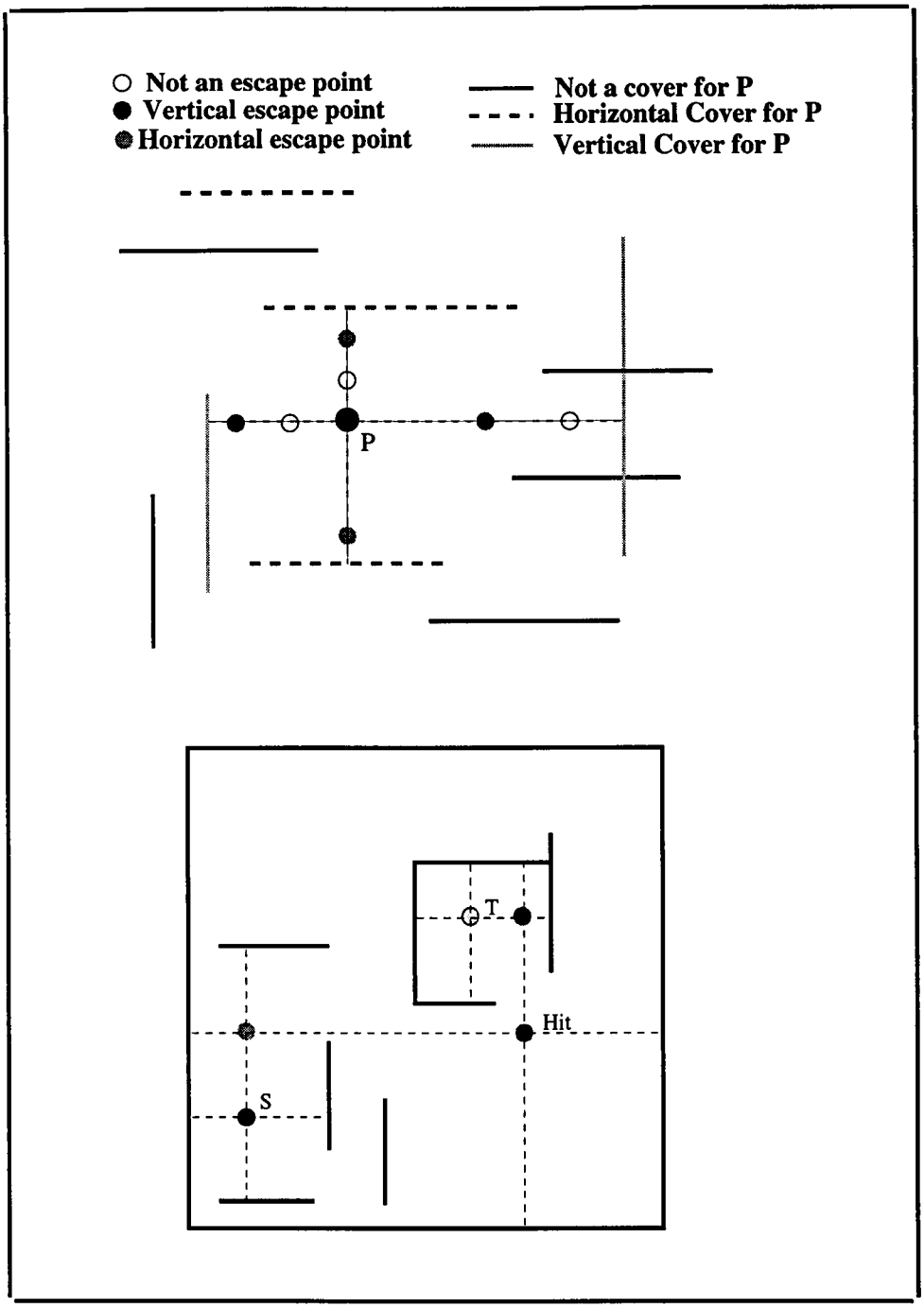


Figure 2.10: Line expansion router.

requires a special routing algorithm because of its predetermined layer, location and variable net width [24, 23]. The power nets usually form a tree whose root is a power pad and whose branches reach macrocells. All IC chips require bonding pads to connect the pins of the package to the silicon surface. Therefore, the last stage in detailed routing is typically to route the connections between these I/O pads and the core circuit. The area between the core and the peripheral pads is called the *moat* (Figure 2.11), and this routing task is consequently called moat routing [35, 34, 55, 19]. In this dissertation, we introduce another specialized router similar to the moat router but the interconnections are made for pairwise nodes (I/O port and area pad). We call this new router the Automatic Area–Array Pad Router (A3PR).

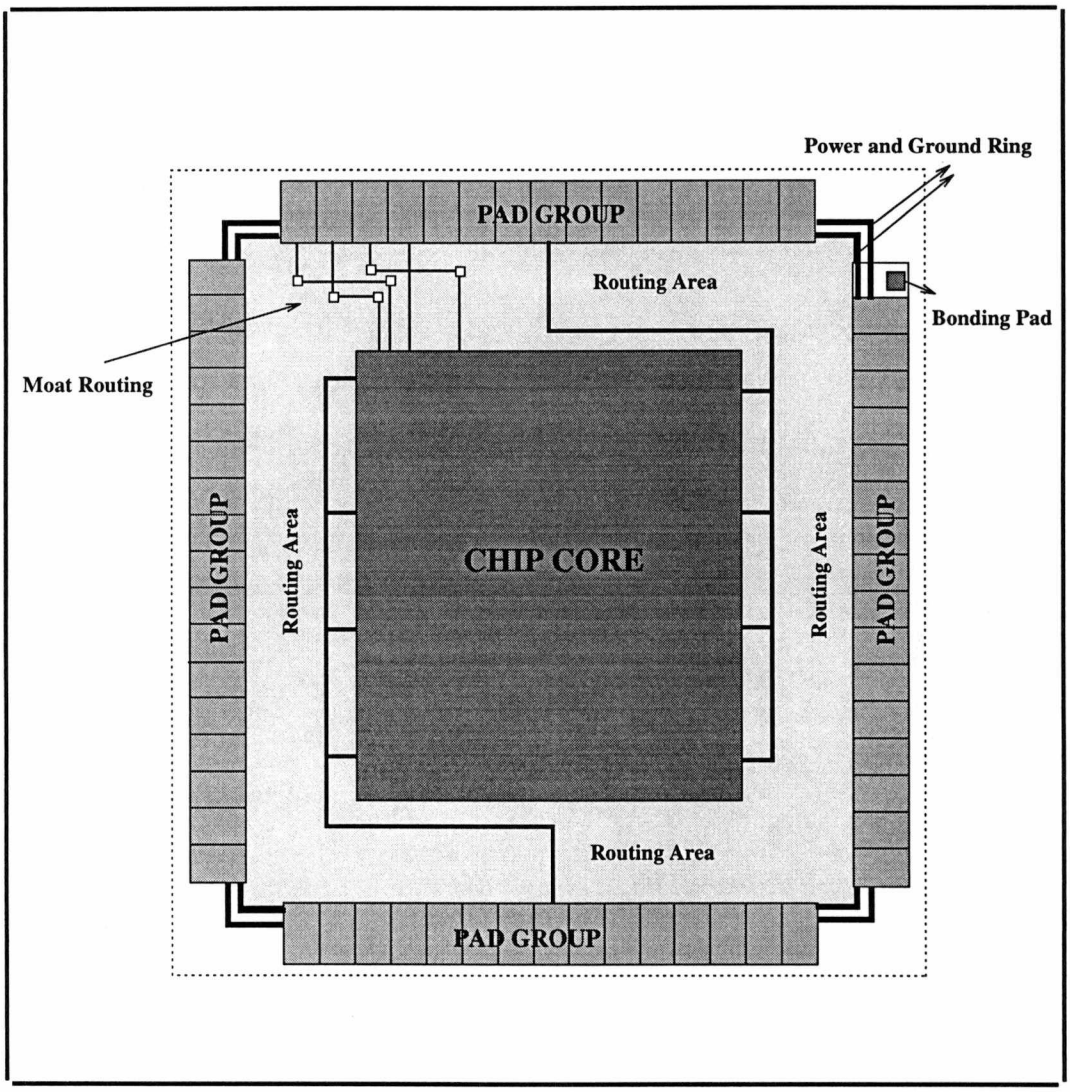


Figure 2.11: Moat Router.

CHAPTER 3

METHODOLOGY

3.1 Conventional Physical Design

Integrated circuits consist of many electronic components, built by layering several different materials in a well–defined fashion on a silicon wafer. For this or the circuit description has to be transformed into a geometric description, the layout of the chip. This process is called the physical design of a chip. Because physical design is a complex and time–consuming process, it is advisable to decompose the process into several subproblems and conquer one at a time. The subproblems of conventional physical design are shown in Figure 3.1 an are described in more detail below:

Partitioning: Because a VLSI chip may contain several million transistors, it is usually not viable to handle the entire circuit simultaneously due to time limitations. This large circuit has to be *partitioned* into several more manageable submodules such that their sizes are small enough to be handled by the existing physical design process. This partitioning is done with respect to the size of the blocks, the number of blocks and the number of interconnections required between the blocks. Many partitioning algorithms [49, 46] have been developed to reduce the number of net crossings and the signal

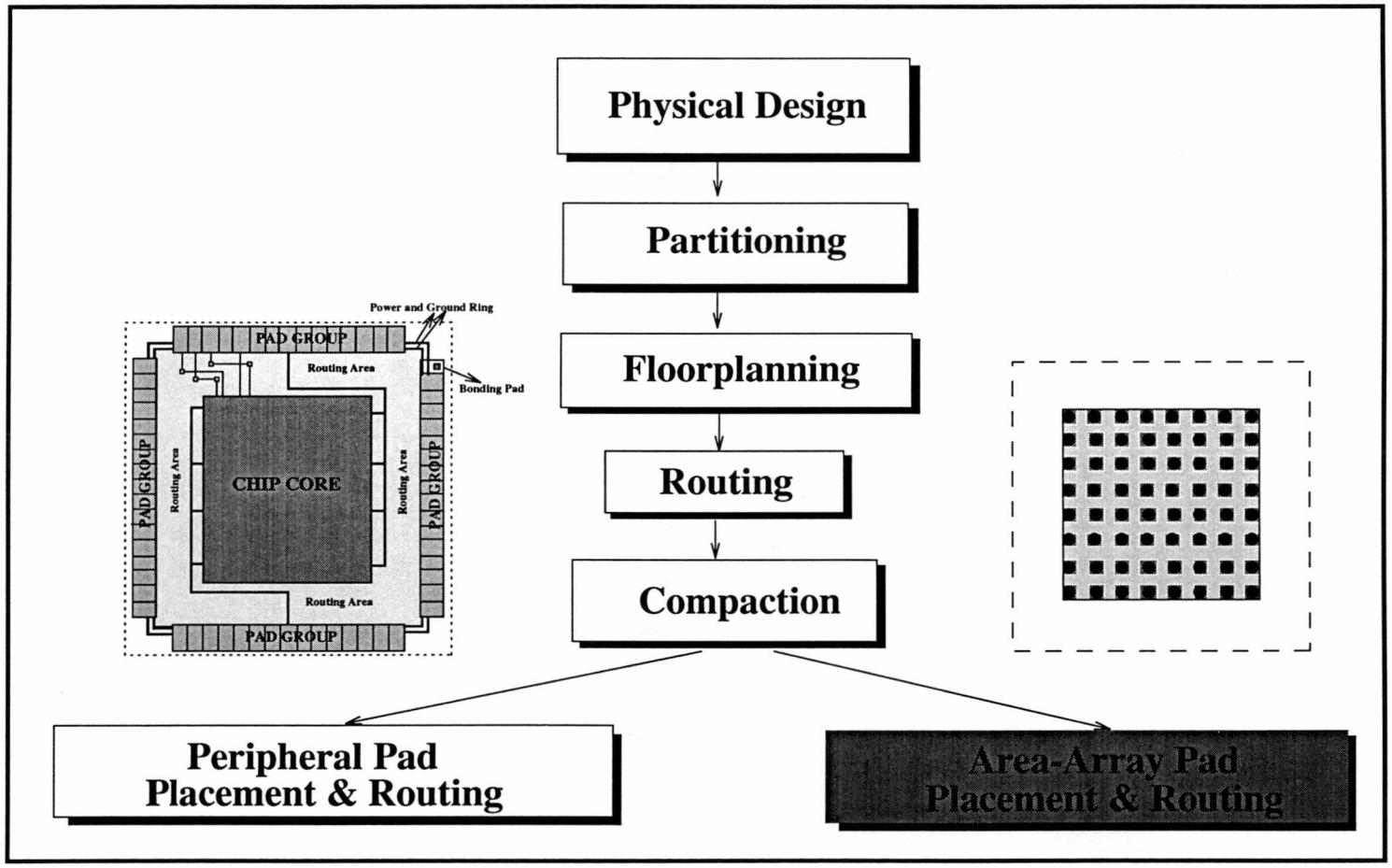


Figure 3.1: Physical design steps.

propagation delay on the nets between modules.

Floorplanning: In the floorplanning phase, the building blocks are exactly positioned on a chip. In this phase the aspect ratio of a block can be varied within a prespecified range. The goal of this phase is to find a minimum area arrangement for the blocks that allows completion of interconnections between them.

Routing: Given a placement of the modules done in the previous step, a routing algorithm subsequently performs exact wiring of the nets based on interconnection specifications given in a netlist. The routing problems in ICs dealing with a large number of nets are decomposed into two phases using a divide and conquer approach. In the first phase, called global routing, the coarse route of each net through a set of routing regions is assigned. The objective of this assignment is to plan the routing globally to reduce the chip area, improve routability, and balance congestion across the layout surface. In the second phase, the exact routing of the nets is done by a detailed router.

Compaction: The compaction phase is the task of compressing the layout in all directions such that the total area is reduced without any violation of the design rules.

Pad Routing: The last phase in the physical design of s VLSI chip with peripheral bonding is to place a bonding pad ring around the "core" portion

of the chip and connect the nets from the ring to the core. The bonding pads also contain circuitry for providing input buffering, input protection, and output drive capability for the chip. However, for an area–array IC, the bonding pads are placed on the whole surface of the chip; then the net from each of the I/O terminals is connected to its closest pad such that the total wire length is minimized.

3.2 Physical Design Flow of Area–Array ICs

In designing an area–array IC, we will adapt most of the conventional procedures presented in Section 3.1. We modify some of these sub–steps to incorporate some design guidelines to facilitate the placement of the area–array pads and the routing of the connections between the I/O terminals and the area–array pads. The placement and routing of the pads will be performed in the final stage. In other words, the algorithm developed in this work can be used as a post–processor to existing VLSI layout tools to design an area–array IC.

Figure 3.2 illustrates the various steps in designing area–array flip–chip IC. Given the core circuits designed following the design guidelines defined in Section 3.3, the process starts with the extraction of the top layer (more than one layer is also possible) of the core circuits of the chip. Then the area–array pads are placed at potential sites on the top layer and each of the I/O ports is routed to the closest pad. After routing, a passivation layer is applied to insulate the lines. Vias are opened in the passivation layer to allow placement of

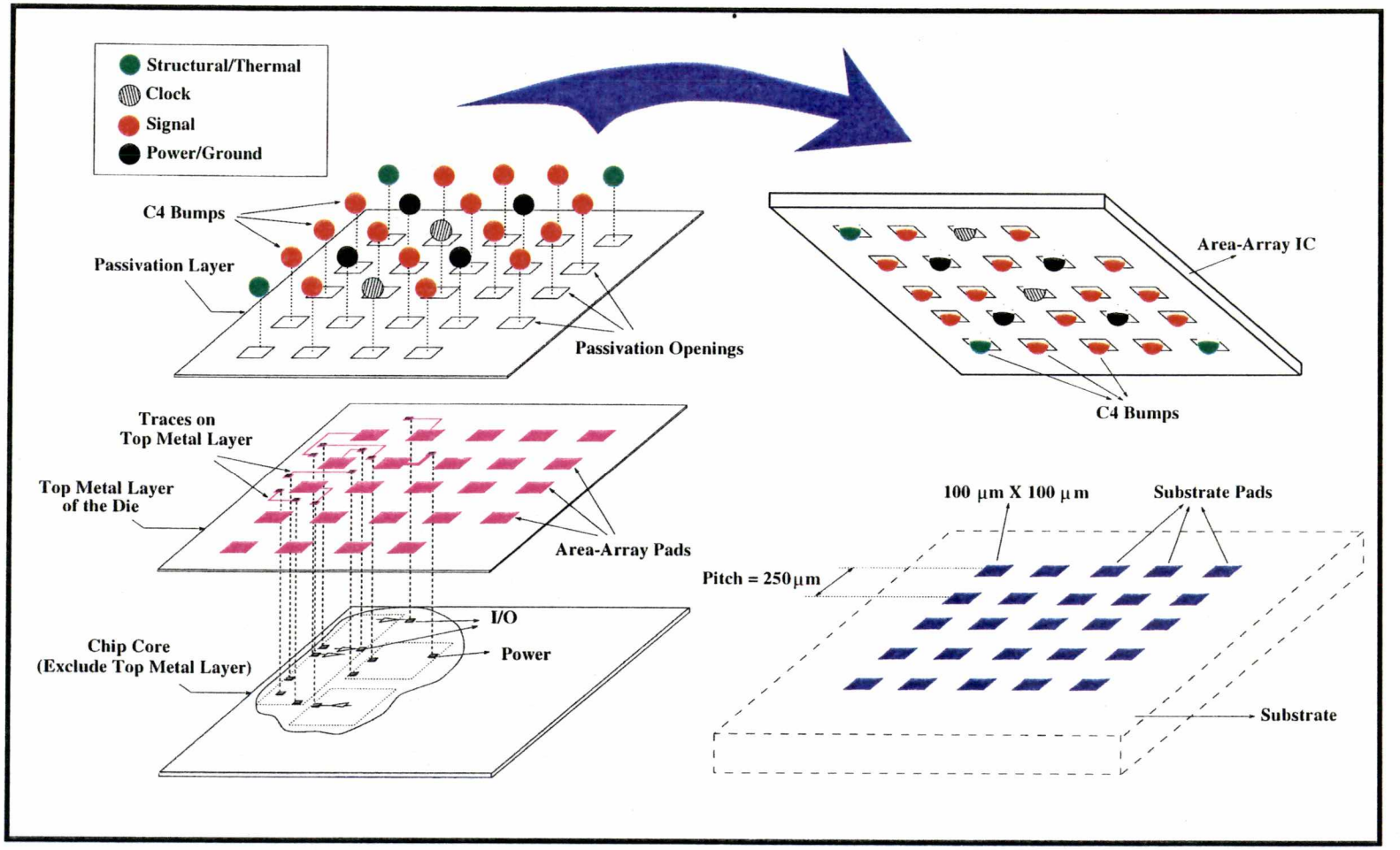


Figure 3.2: Area–array pad routing.

solder balls for flip–chip attachment. The design flow of our approach is given in Figure 3.3.

3.3 Design Guidelines

To facilitate the design of a VLSI chip with area–array bonding technology, the chip designer needs to follows some design guidelines given as followings:

1. The pad and the buffering circuitry are separated. The I/O buffers need to be placed and routed to the active components in the chip core during the first three sub–steps of the physical design.

2. The I/O ports of these buffers must be labeled with a special attribute. For instance, label the I/O ports of signal and clock terminals with a suffix "_IO" and label the power and ground nets that need to be connected to bonding pads with suffix "_PIO".

3. The I/O ports must consist of one of the layers or contacts required in the pad routing stage in our tool.

4. Any of the submodules in the design which prohibit the placement of the pads should be assigned as pad keepout areas. Name these submodules with suffix "_KO".

5. The mask geometries of the chip core need to be stored in a file in CIF (Caltech Intermediate Form). CIF consists of a set of textual commands to

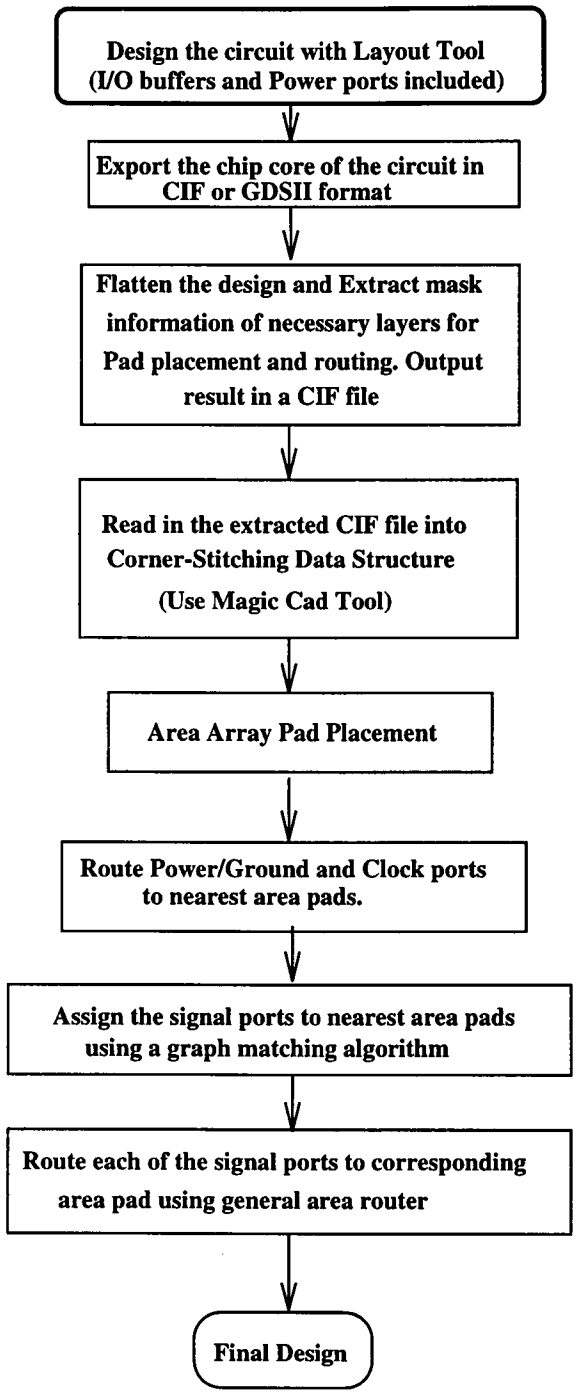


Figure 3.3: Area–array flip–chip pad router design flow.

describe the layers. It is concise and is both human and machine readable. It describes only the geometry on IC photomasks in terms of rectangles, polygons, wires and circular pads. No higher level graphics such as text are provided. It has hierarchy, and is therefore easy to generate and can be used to build complex graphics. The language is structured, allowing for multiple CIF files to be aggregated. The detailed structure of the CIF is given in [36].

3.4 Data Preparation

Most of the VLSI circuits are designed using a hierarchical approach. Cells in a hierarchical layout may contain mask information, or they may contain other cells, known as *subcells*. Hierarchical physical design partitioning allows the design to be worked on in parallel, while ensuring that any conflicts between the separate designs are reduced to a minimum when combined on the final layout. If a chip is broken into cells along functional boundaries, it is usually much easier for designers to understand. In addition, hierarchical designs are modular; cells may be placed or moved as entire units, and a single cell may be used in many different places in the design. Hierarchy benefits CAD tools by reducing the amount of work they must do to process repetitive structures.

Although hierarchical structure has advantages, sometimes it is a hindrance. Sometimes tools need to have all material in a particular area, without caring which cell the material came from. This occurs, for example, when generating

masks to fabricate the chip, and also during operations such as circuit extraction and routing. A common way of finding all material in an area is to flatten a hierarchical layout by converting it into a single set of mask layers that represents the entire hierarchy. Before we proceed to the pad placement and pad routing steps, we need to flatten the chip core layout and extract the number of specific routing layers, the pad keepout areas and chip core boundary as well as the I/O port sites. This step is done using a program called *cifextr* (a modified version of a public domain program called *cifflatten* [2] developed at Univ. of California at Berkeley). The bounding box of each pad keepout area is stored on a special layer and the chip core boundary is marked with its lower–left and upper–right corners.

3.5 Area-Array Pad Layout Process

The area–array pad layout process is subdivided into pad placement, pad assignment, and pad routing phases. During the pad placement phase, the pads are placed onto the top–metal surface such that they do not overlap the prerouted wires and so that they meet the minimum spacing constraint. The pad assignment phase consists of matching each I/O port with an area pad in order to minimize the maximum Manhattan distance. In the subsequent routing phase, the paths of the wires are determined. The wiring that connects the I/O ports to the desired pads has to be performed using the specified number of upper metal layers as given by the designer. The following terminology is used throughout

this dissertation (shown in Figure 3.4):

- area pad: the bonding pad (the square on uppermost metal under an overglass opening that the solder ball is attached).
- pad pitch: the minimum spacing between the center of two area pads.
- padframe: a set of bonding pad locations and their minimum sizes, located in an array over the surface of the chip core of fixed size.
- I/O terminal/port: a point inside the chip core where interconnection to area pad must be made.
- chip core: the central block area containing the active logic of the circuit.
- core–limited design: a chip design where all the area pads can be placed inside the boundary of the chip core (Figure 3.4(a)).
- pad–limited design: a chip design where the size of the chip is determined by the area occupied by the area–array pad (Figure 3.4(b)).

3.5.1 Pad Placement

The pad placement algorithm takes as input the layout information on extracted layers—prerouted wires, pad keepout areas, and the chip–core boundary. It calculates the center of the chip core and then using this center as a reference point, generates a set of possible C4 bonding sites based on the user–specified pad size and pad pitch. Pad size is defined usually by the minimum size to which

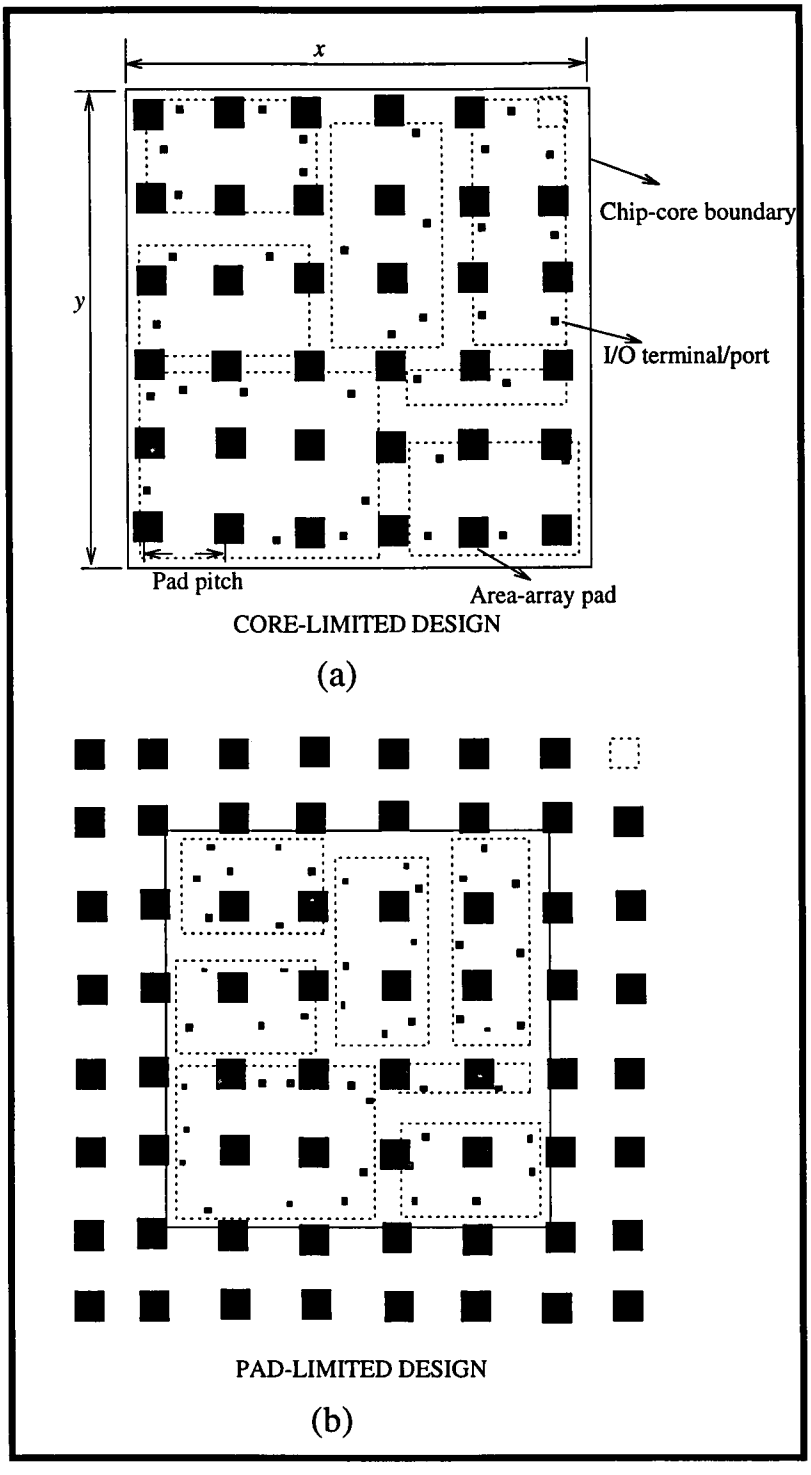


Figure 3.4: Terminology used for pad placement and routing.

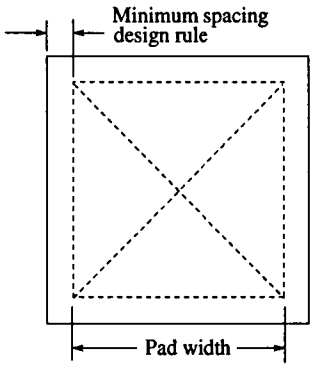


Figure 3.5: A square box used in area searching in pad placement stage.

a solder ball can be attached. This is usually of the order of 80 to 150 μm square. The spacing or pitch of pads is primary dependent upon substrate technology. This tends to be in between 150 to 350 μm. If the bonding site is inside the core boundary and not covered by any pad keepout areas, searching box is used. This searching box, shown in Figure 3.5, is a square area created by adding a minimum spacing width to all four sides of the area pad to ensure that no spacing violations can be created when placing the pads. Using an area–searching technique, a search is performed for any prerouted wire under the area covered by this box. Also, a check is made to see whether this box is overlapping any keepout area. The pad centered at this location is then placed if no wire is found and no overlapping occurred. If the bonding site is outside the core boundary, the pad is placed directly at this site. This process is continued until the number of pads placed is at least 10% larger than the number of I/O ports. The pad location at the upper–right corner of this array is purposely left blank to identify chip orientation. Figure 3.6 shows this pad placement method.

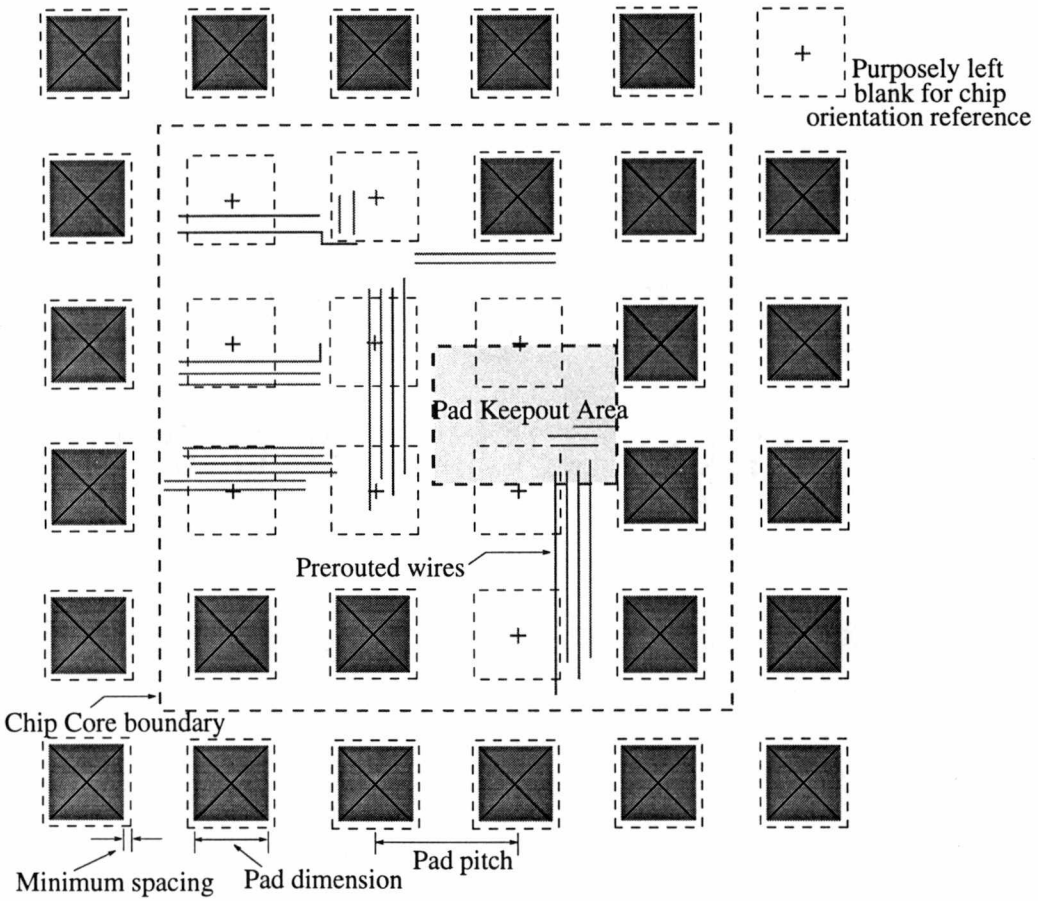


Figure 3.6: Pad placement strategy.

3.5.2 Pad Assignment

Some of the I/Os that need to be at certain predefined pad locations can be assigned by the designer manually. These assigned pairs are routed sequentially using the maze router with the I/O port as the source and the area pad as the target. In this case, the source is a point and the target is a square. The remaining I/Os are also routed sequentially using the maze router but at each routing only the source (I/O port) is defined and the target (area pad) is the one first reached by the wave expansion stage of the maze router; this pad is the target with the shortest path to the source. Figure 3.7 illustrates this process; even though the pad located on the top–right corner has the shortest direct distance to the source, the shortest path which avoids the obstacles is the path from the source to the pad labeled "closest pad".

The result obtained using this approach depends heavily on the order of the I/Os being routed. To illustrate this problem, an example is shown in Figure 3.8 in which the squares are area pads and the dots are I/O ports. If the numeral associated with each dot is the ordering of the routing sequence, the resulted pairings of the I/O ports and area pads are shown as dashed lines with arrows pointing from the dots to the squares. We can see the result obtained in this example is not optimized with respect to the maximum Manhattan distance, since port 15 and port 16 which are routed last have to connect to distant pad A and pad C, respectively.

Therefore, a pad assignment method that minimizes the maximum Manhat-

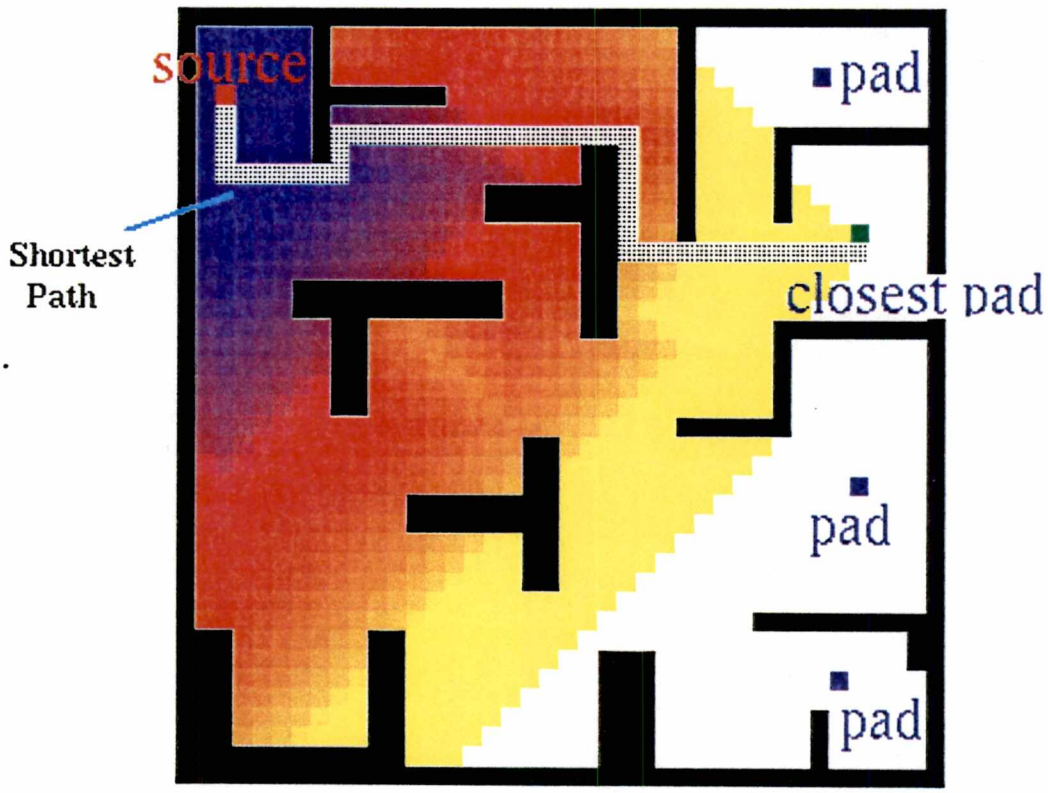


Figure 3.7: Lee's algorithm wave propagation.

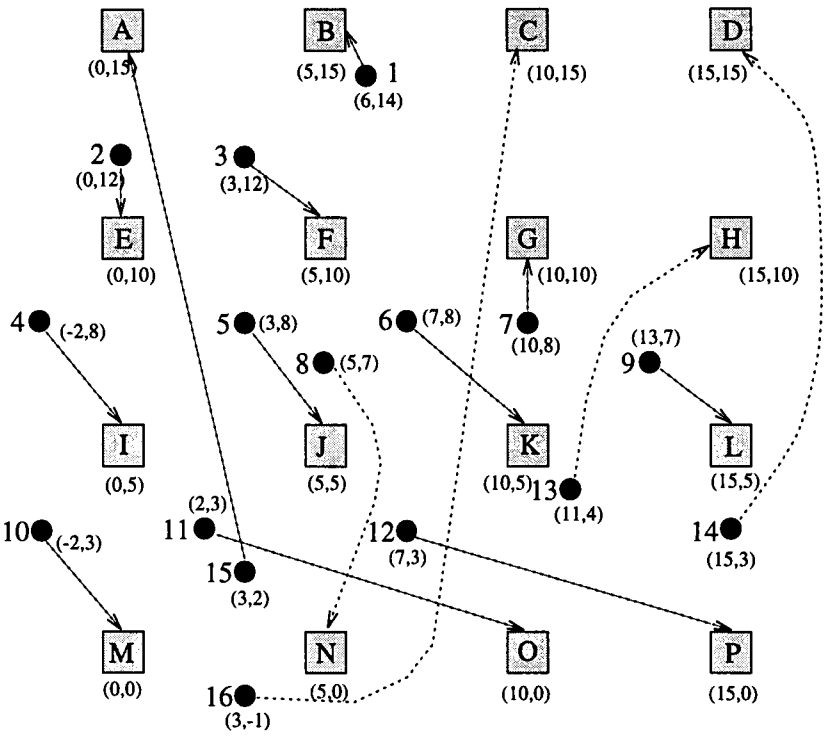


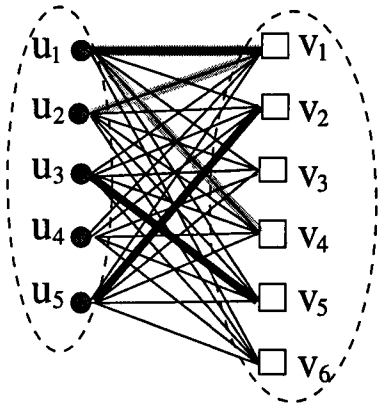
Figure 3.8: Example of the result obtained without pad assignment.

tan distances is required to resolve the problem mentioned above. Assignment of an I/O port to an area pad should be done in such a way the wire length of that net is as small as possible. Using this criterion, many I/Os may compete for the same pads. To find a good assignment of I/Os to area pads, a linear assignment method is used. This assignment problem can be formulated as a bipartite weighted matching problem. This problem has been studied very extensively in the field of network optimization and operation research. Some of the most efficient algorithms for solving this matching problem are based on the successive augmenting path method [15, 17, 21, 28, 27, 42].

Before continuing with the description of this method, the problem may be defined as: Given two set of points $P = \{p_1, p_2, \ldots, p_m\}$ corresponding to the location of the I/O ports and $Q = \{q_1, q_2, \ldots, q_n\}$ corresponding to the location of the area pads on the Manhattan plane, and a non–negative weight $w(p_i, q_j)$ as the weight between each pair of source p_i and sink q_j to indicate the Manhattan distance between this pair, the *pad assignment problem* is defined as finding a matching, M, that will minimize the total weight. The Manhattan distance between p_i and q_j is defined as

$$|p_{xi} - q_{xj}| + |p_{yi} - q_{yj}|.$$

A *bipartite graph* is an ordered triple $G = (U, V, E)$ such that U and V are sets, $U \cap V = \emptyset$, and $E \subseteq \{\{u, v\} : u \in U, v \in V\}$. The *vertices* of G are the elements of $U \cup V$. The *edges* of G are the elements of E. A bipartite graph is a complete bipartite graph if every edge in U is connected to every edge in V. If



$$M = \{(u_1,v_1),(u_3,v_5),(u_5,v_2)\}$$

$$P = \{(u_2,v_1),(u_1,v_1),(u_1,v_4)\}$$

Figure 3.9: Bipartite graph definition.

U has m elements and V has n, then we denote the resulting complete bipartite graph by $K_{m,n}$. Figure 3.9 shows $K_{5,6}$. A *matching* in G is a pairwise disjoint set $M \subseteq E$. Here pairwise disjointness means that no two edges in M have a common vertex, *i.e.* we have $e_1 \cap e_2 = \emptyset$ for all $e_1, e_2 \in M$ such that $e_1 \neq e_2$.

With respect to a given matching $|M|$, an edge, e, is a matched edge if $e \in M$. An edge, e, is a free edge if $e \in E - M$. Likewise, a *matched vertex* is a vertex which is an endpoint of a matched edge. A *free vertex* is a vertex that is not the endpoint of any matched edge. An *alternating path* is a path (edges) which begins at a free vertex and whose edges alternate between matched and unmatched edges. An *augmenting path* is an alternating path which starts and ends with unmatched edges (and thus starts and ends with free vertices). It should be noted that an augmenting path must be of odd length. The minimum-cost

matching algorithm will attempt to increase the size of a matching by finding the shortest augmenting path. For more background on bipartite matching, see Papadimitriou and Steiglistz [42].

Let p denote a minimum-cost augmenting path with respect to matching M, and P denotes the set of edges in p, then $M \oplus P$ is called the *symmetric difference* of M and P. $M \oplus P$ is the set of elements that are in one of M or P, but not both, i.e. $M \oplus P = (M - P) \cup (P - M)$. It can be shown that $M \oplus P$ has the following properties: (1) it is a matching; (2) $|M \oplus P| = |M| + 1$. For example, consider the matching M shown in Figure 3.9, the edges $(u_1, v_1), (u_3, v_5)$, and (u_5, v_2) are matched (shown in heavy edges). One of the augmenting paths with respect to matching M is $P = \{(u_2, v_1), (u_1, v_1), (u_1, v_4)\}$. The symmetric difference $M \oplus P$ and the cardinality of new matches are $M \oplus P = \{(u_2, v_1), (u_1, v_4), (u_3, v_5), (u_5, v_2)\}$, and $|M \oplus P| = 3 + 1 = 4$, respectively.

The first algorithm developed specially for solving the assignment problem was the classical "Hungarian Method" proposed by Khun [29]. The time complexity of the Hungarian method is $O(m^2 n)$, where $m = |U|$ and $n = |V|$. The matching M, which corresponds to the assignment, can also be formulated as a linear programming problem:

$$\min \sum_{\{u_i, v_j\} \in E} c_{ij} x_{ij} \qquad \text{subject to}$$

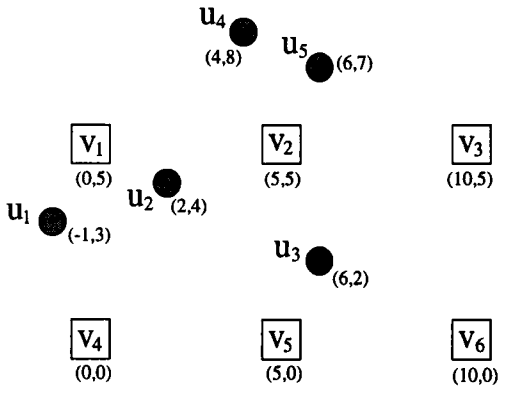
$$\sum_{v_j \in V} = \begin{cases} 1, & \text{for } u_i \in U(M) \\ 0, & \text{otherwise} \end{cases}$$

$$\sum_{u_i \in U} = \begin{cases} 1, & \text{for } v_j \in V(M) \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} \in \{0,1\} \qquad \text{for } \{u_i, v_j\} \in E.$$

In this dissertation, a bipartite weighted matching method of solving the linear assignment is chosen for its speed and simplicity in implementation. One the most efficient bipartite weighted matching algorithms is based on the successive shortest augmenting path method described above. Jonker and Volgenant [27] developed a shortest augmenting path is adapted to solve the pad assignment problem in this dissertation.

Figure 3.10 illustrates the step to reduce a pad assignment problem to a bipartite weighted matching problem. In this example, we have 5 I/Os and 6 pads; the cost matrix C contains the cost, c_{ij} = Manhattan distance between i^{th} I/O and j^{th} pad. Figure 3.10(c) shows a weighted complete bipartite graph corresponding to this pad assignment problem. Figure 3.11 illustrates the successive steps to find the shortest augmenting path and matching (heavy edges). The initial matching M contains $\{u_1v_1, u_3v_5, u_5v_2\}$. Each of these matched edges is obtained from the least cost entry in each column of cost matrix, C. In the next step, a minimum cost augmenting path, P, is computed and the new matching, M', is obtained by finding the symmetric difference of P and M. This step is repeated until there is no augmenting path. Figure 3.12 shows the comparison of the results obtained with matching algorithm and the results obtained based on the shortest Manhattan distance. The matching algorithm not only performs

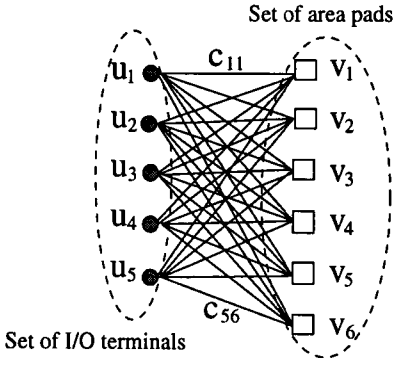


(a)

C_{ij}, Cost matrix based on the Manhattan distance

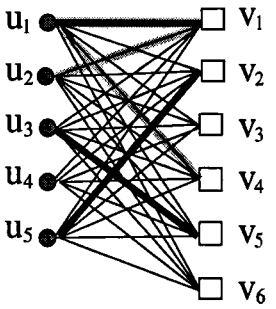
u_i \ v_j	1	2	3	4	5	6
1	3	8	13	4	9	14
2	3	4	9	6	7	12
3	9	4	7	8	3	6
4	7	4	10	12	9	14
5	8	3	6	13	8	11

(b)



(c)

Figure 3.10: An example to illustrate the necessary steps to reduce a pad assignment problem to a bipartite weighted matching problem.

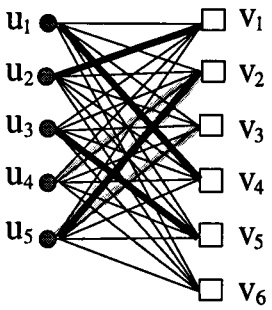


Initial Matching

M = (heavy edges)

$P = \{(u_2,v_1),(u_1,v_1),(u_1,v_4)\}$

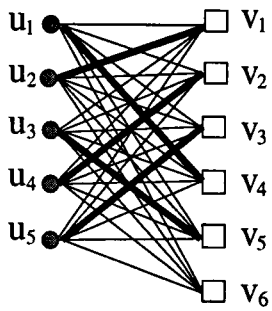
(a)



$M' = M \oplus P$

$P' = \{(u_4,v_2),(u_5,v_2),(u_5,v_3)\}$

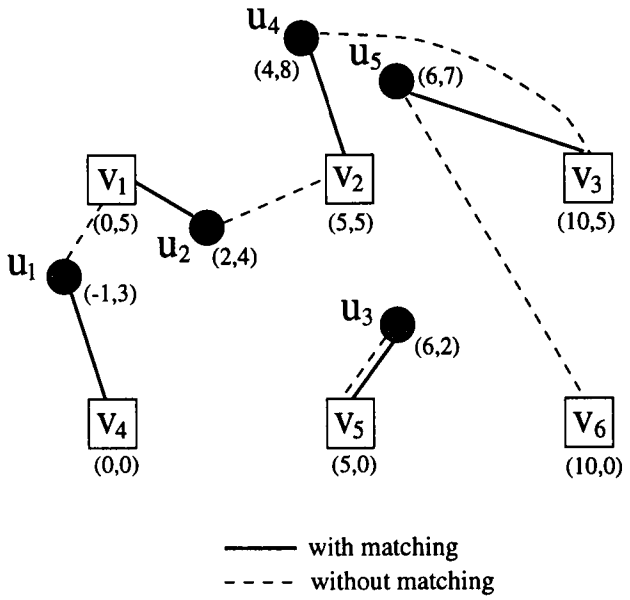
(b)



$M'' = M' \oplus P'$

(c)

Figure 3.11: Minimum–cost matching found using a successive shortest augmenting path algorithm.



without matching	with matching
u_1—V_1 = 3	u_1—V_4 = 4
u_2—V_2 = 4	u_2—V_1 = 3
u_3—V_5 = 3	u_3—V_5 = 3
u_4—V_3 = 9	u_4—V_2 = 4
u_5—V_6 = 11	u_5—V_3 = 6
total 30	total 20

Figure 3.12: The results obtained with matching (heavy lines) Vs the results obtained based on shortest distance criteria (light dashed lines).

better pad assignment independent of the net ordering but it also tries to keep all the distances between matched pairs as close as possible. Finally, Figure 3.13 shows the results obtained using pad assignment. Comparing these results to those obtained in Figure 3.8, it is evident that a lot of improvement can be achieved using pad assignment method. Using the pad assignment, the resulted total Manhattan distance is 81 and the longest distance is 10 (distance between port 9 and pad D). Without the assignment, the resulted total Manhattan distance is 125 and the longest distance is 23 (distance between port 16 and pad C). This pad assignment phase generates a two terminal (I/O port and area pad) netlist to be input into the pad routing phase.

3.5.3 Pad Routing

Pad routing is performed on the few top metal layers defined by the designer. Thus, the routing region is represented by different, arbitrary, rectilinear polygons on each of its multiple routing layers which contain an arbitrary blockages due to the prerouted wires. In addition, the I/O ports may be positioned anywhere within the routing region or on its border. Because of these conditions, channel or switchbox routing methods can not be applied to solve this routing problem because the usual channel or switchbox routing areas do not exist. For this reason, the general area router is adapted to solve this pad routing problem.

The general area routing problem can be viewed as a collection of search problems which find routes and lay down nets between each pair of I/O ports

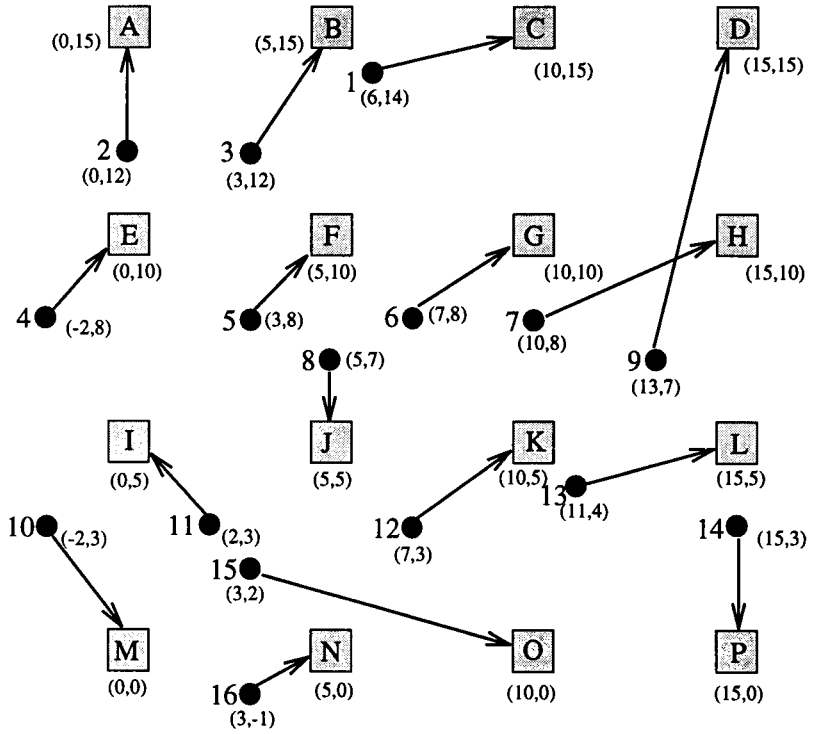


Figure 3.13: Example of the result obtained with pad assignment.

and area pads. There are many search algorithms for solving the routing problem as reviewed in Section 2.4.

When search algorithms are used to solve the routing problem, they typically operate in two phases. The routing region is represented in the form of a rectangular grid of cells, with some cells free and others blocked. Blocked grid cells represent either obstacles or previously routed interconnections. A free grid cell is available for routing. The grid size is determined such that wires can be routed on adjacent grid lines without touching. The router routes one net at a time along the grid. The first phase starts at the source cell and searches for the target cell by expanding a wavefront, usually by going in several directions at the same time. All cells on the wavefront can be reached from the source with the same cost, they are within the same routing distance from the source. The first wavefront contains only the cell at the source point. While expanding a wavefront each free cell that can be reached from a cell in the wavefront with lower cost becomes part of the next wavefront and an data structure is built that keeps track of how each cell was reached. Once the target cell has been reached, the second phase is started. On the other hand, if the first phase exhausts all possibilities without reaching the target cell, then no route exists between them, and there is no reason to do the second phase.

In the second phase, the minimal–cost path from the target to the source is retraced using the data structure obtained in the first phase and actually laying down the electrical connections. There might be several paths with the same

minimal cost between source and target. Generally, minimal–cost path with the least number of bends is preferred.

In this work, this flexible maze router is used to perform the pad routing. We start with defining the routing area as a graph. The routing graph is a three–dimensional grid. Each horizontal section of the grid corresponds to one of the layers available for interconnections. On each layer, the routing area is partitioned into rectangular portions and is represented by a node in the routing graph. An edge links two nodes if a wire segment can be laid out between points of the area associated with them. A common way to construct a routing graph is to organize it as a grid (Figure 3.14), whose size is constrained by the minimum allowed wire/via spacing and minimum allowed wire/via width. Each edge of this graph has the same cost. The cost of a path is defined as the sum of the costs of the edges in the path. In this way the routing problem is reduced to the search of a minimum cost path. The complexity of this search grows as a quadratic function of the circuit size.

In this implementation we adapted a version of a maze router based on Lee's algorithm [33]: To speed up the routing, there are several options for performing the search/expansion process. Some of these search methods developed in artificial intelligence such as *depth–first, breadth–first*, A^* etc. are shown in Figure 3.15.

Search methods are best explained using a grid. Consider a grid in Figure 3.15 where obstacles are marked dark, and the source and target of the routing are

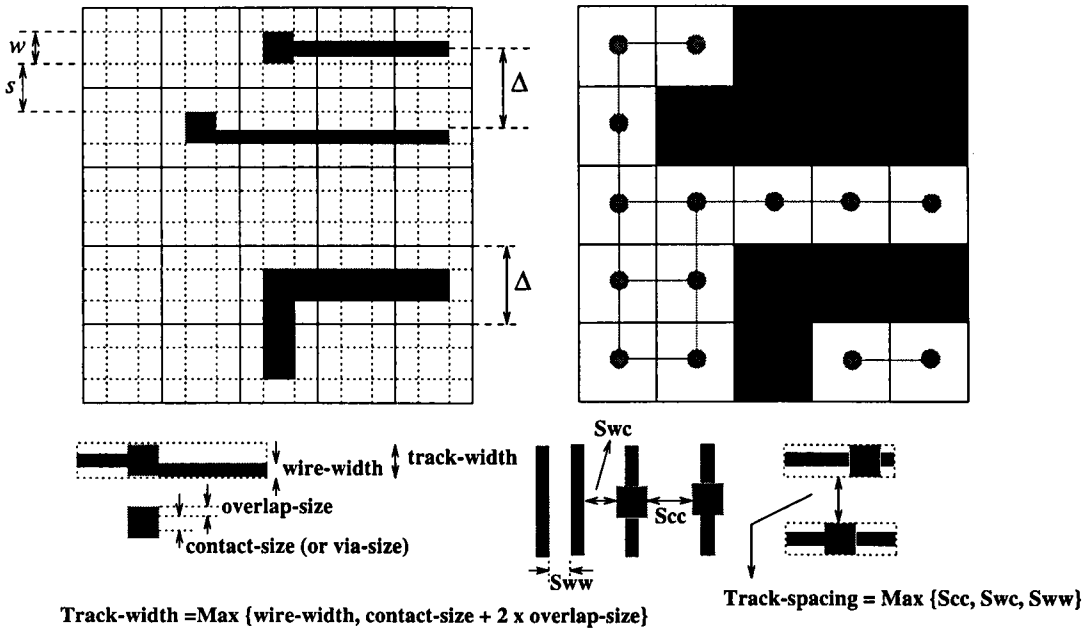
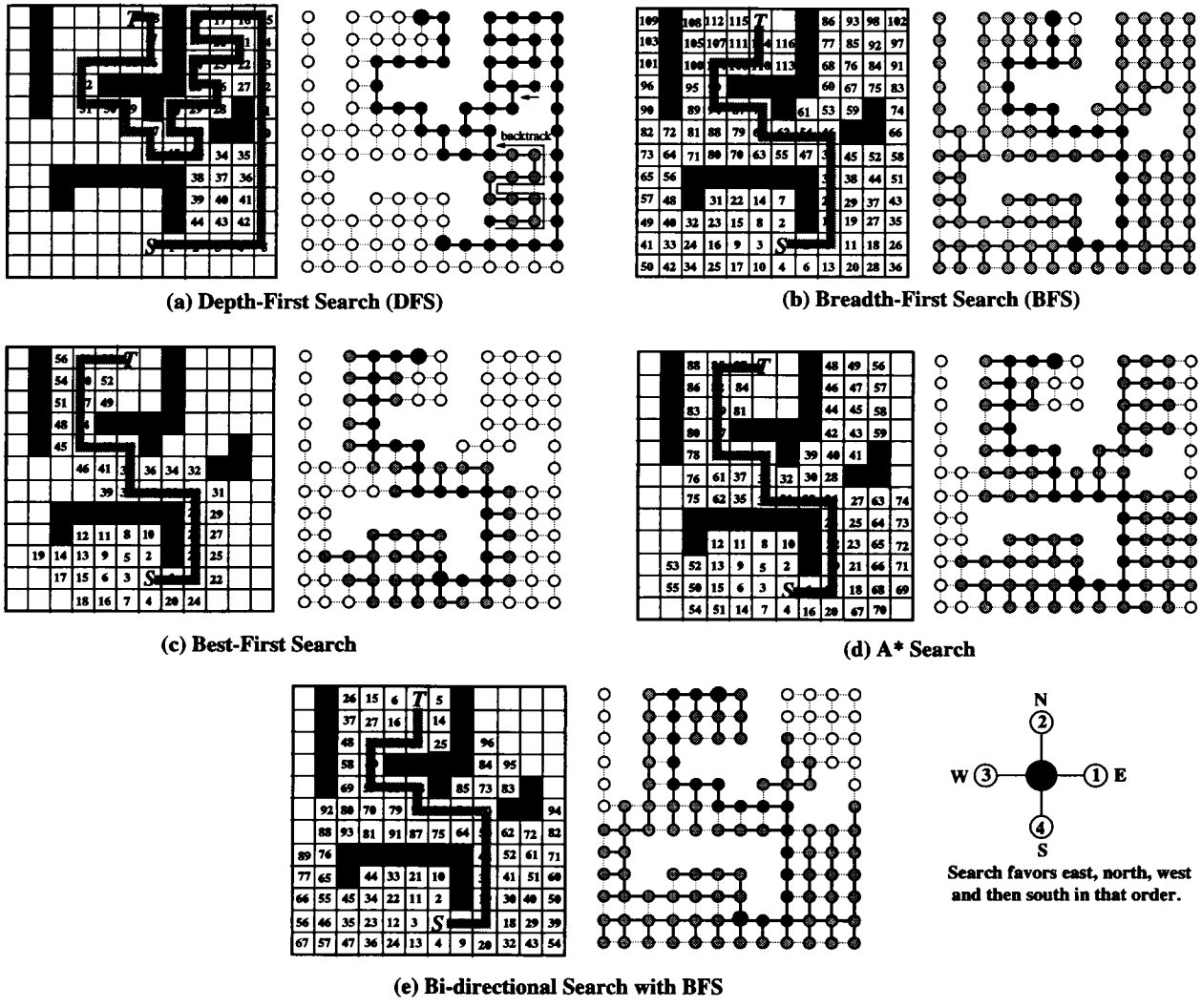


Figure 3.14: Grid construction.



(a) Depth-First Search (DFS)

(b) Breadth-First Search (BFS)

(c) Best-First Search

(d) A* Search

(e) Bi-directional Search with BFS

Figure 3.15: Different search strategies.

marked with S and T, respectively. The distance between two nodes is 1. The *depth–first* search (shown in Figure 3.15(a)) works by always generating a successor of the most recently expanded node, until a dead–end or the target node is reached. In the former case, the backtracking from this dead–end node is performed to the next most expanded node and generating one of its successor. In Figure 3.15(a), node, labeled "44", is one of the dead–end node. The backtracking from this node ends at node, labeled "33", and the search continues from this node. If we search all nodes at distant i from the S before looking at nodes at distant $i + 1$, we are performing a *breadth–first search* (Figure 3.15(b)); this search strategy is used in the Lee's maze router in its wave propagation stage. Both depth–first and breadth–first are uninformed or brute–force search method. If instead, we search preferentially those nodes that we know are "closer" to the target, we perform a heuristic or informed search. There are a number of search algorithm that utilize heuristic evaluation functions to reduce the search space, including *best–first* search and A* search [43]. The heuristic evaluation function used in best–first search (shown in Figure 3.15(c)) is an estimate of the cost of reaching a target from the given node. This function can be the Manhattan distance between the given node and the target node. The expansion is performed on least costly node on the wavefront. If the evaluation function used in the expansion is the total cost to get to a given node from the source node plus an estimate of the distance to the target node, we perform a A* search (Figure 3.15(d)). The *bi–directional* search is performed to trade space for time by

from the source node and backward from the target node simultaneously, until a common node is found on the both search wavefronts. This search can be used with any of the above search methods. Figure 3.15(e) shows the bi–directional search combined with the breadth–first search.

In the area pad routing problem where the connections are not specified between the I/O terminals and the pads, the breadth–first search is the only method which can be used to find the shortest path, but it examines a large part of the space and hence is very slow if the search space is large. The reason for its inefficiency is that no global information is utilized that can help the wavefront expand in a promising direction; thus, searching is done in all directions. In addition to this inefficiency, another drawback is the net–ordering problem described in the previous subsection. For these reasons, the pad assignment stage is used to generate the netlist that specifies the pair–wise connections between the I/O terminals and the area pads. With this netlist, we now can use an A*–search method to speed up the routing process. The A* utilizes a heuristic function to guide its search towards the target. The heuristic function is a cost function from any point to the target. The cost can be an estimated distance from the point to the target. Let v be a node reached by a wave front. On v a cost function is defined:

$$f(v) = g(v) + h'(v)$$

where $g(v)$ is the cost of an optimal path from S to v and $h'(v)$ is an estimate of the cost $h(v)$ of an optimal path from v to T. $h'(v)$ can be defined as the

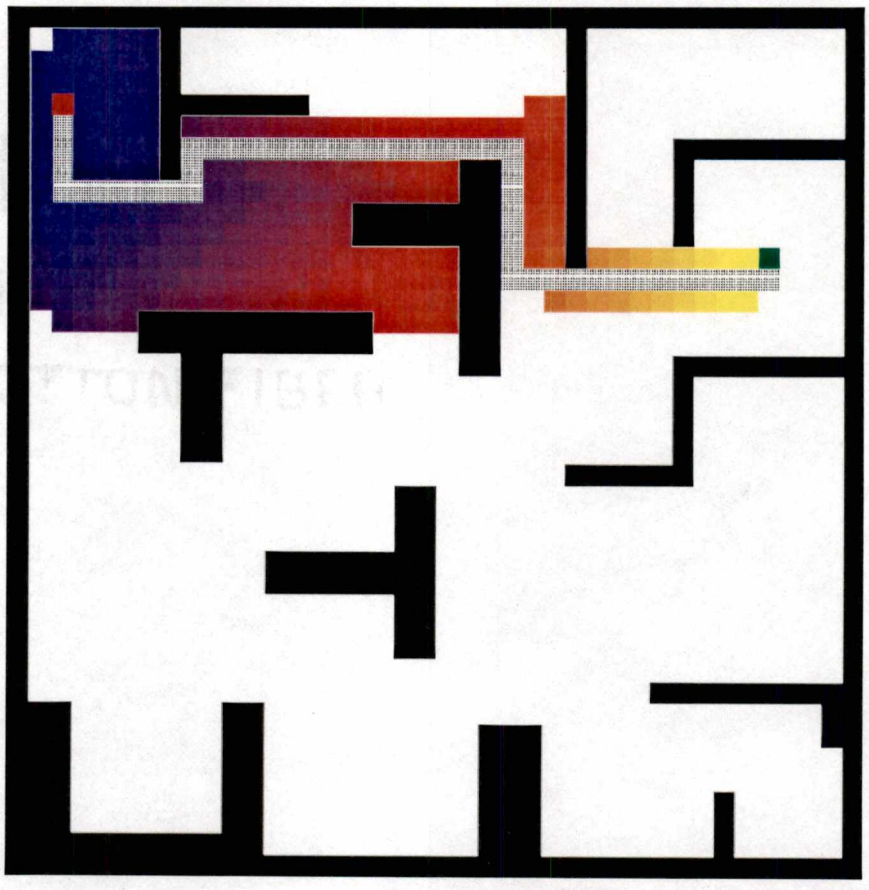


Figure 3.16: A* algorithm wave propagation.

Manhattan distance between node v and the target, T. The function $f(v)$ gives the cost of a path between S and T, constrained to pass through v. At each step of the propagation phase, the one which minimizes $f(v)$ is propagated. Figure 3.16 shows the implementation of the A*–search method for finding an optimal path. The setup used is the same as that shown in Figure 3.7. Notice that the number of nodes explored in the propagation phase is less than that using BFS search method shown in Figure 3.7.

CHAPTER 4

IMPLEMENTATION AND RESULTS

The framework for automatic area–array IC placement and routing, A3PR, has been implemented within the *Magic* CAD tool [1, 41] in C and run on a SUN Sparcworkstation. Given the mask geometries of the chip core stored in CIF (Caltech Intermediate Form), A3PR will generate the area–array padframe of the IC. The pads are large areas of metal that are left unprotected by openning cuts on the overglass layer so they can be soldered to the bumps using the C4 process. Pad size is defined usually by the minimum size to which the solder bumps can be attached. The spacing of the pads is defined by the minimum pitch of the substrate. Both of these parameters are specified by the designer when running A3PR. Before the tool proceeds to the pad placement, pad assignment, and pad routing steps, the chip core layout is flattened to extract the geometric information on the top few layers (as specified by the designer), the pad keepout areas, the locations of I/O terminals, and the boundary of the core circuits. Utilizing this extracted information, the pad placement algorithm calculates the center of the chip core and then using this center as a reference point, generates a set of the potential area–array bonding pads on the top metal layer satisfying the user–specified bond pitch. Using a bipartite weighted matching algorithm,

the I/O ports are assigned to area pads with the objective of minimizing the total wire length. This assignment generates a two–terminal netlist to be input into the pad routing routine to make the complete wiring. In the routing phase, the power/ground and clock I/O pads are routed first before the signal I/Os. The unrouted nets are routed manually with the computer–assisted interactive router [4]. The unused area pads location on the boundary of the padframe are designated as dummy pads. The final padframe (area–array pads along with the wire connections) is saved into a CIF file to be appended to the CIF file of the original design to form an area–array IC. In the next few sections, each of the substeps in the implementation of A3PR is described in more detail. The "Quantizer" chip (shown in Figure 4.1) taken from the tutorial example of the Epoch physical design tool is used as the implementation example.

4.1 Data Structure

In designing any IC, a key question to be resolved is how to represent the layout internally. The data structure used in the layout is a two–dimensional representation and part of the spatial data structure. Ousterhout [40] proposed a data–structuring technique called "Corner Stitching" for Manhattan layouts. In this technique, the layout space is modeled as one or more planes partitioned into rectangular tiles, which are stitched together by pointers at their corners as shown in Figure 4.2. The tile representation has to obey the *maximal horizontal rule* that no space tile has another space tile immediately to its right or left. Each

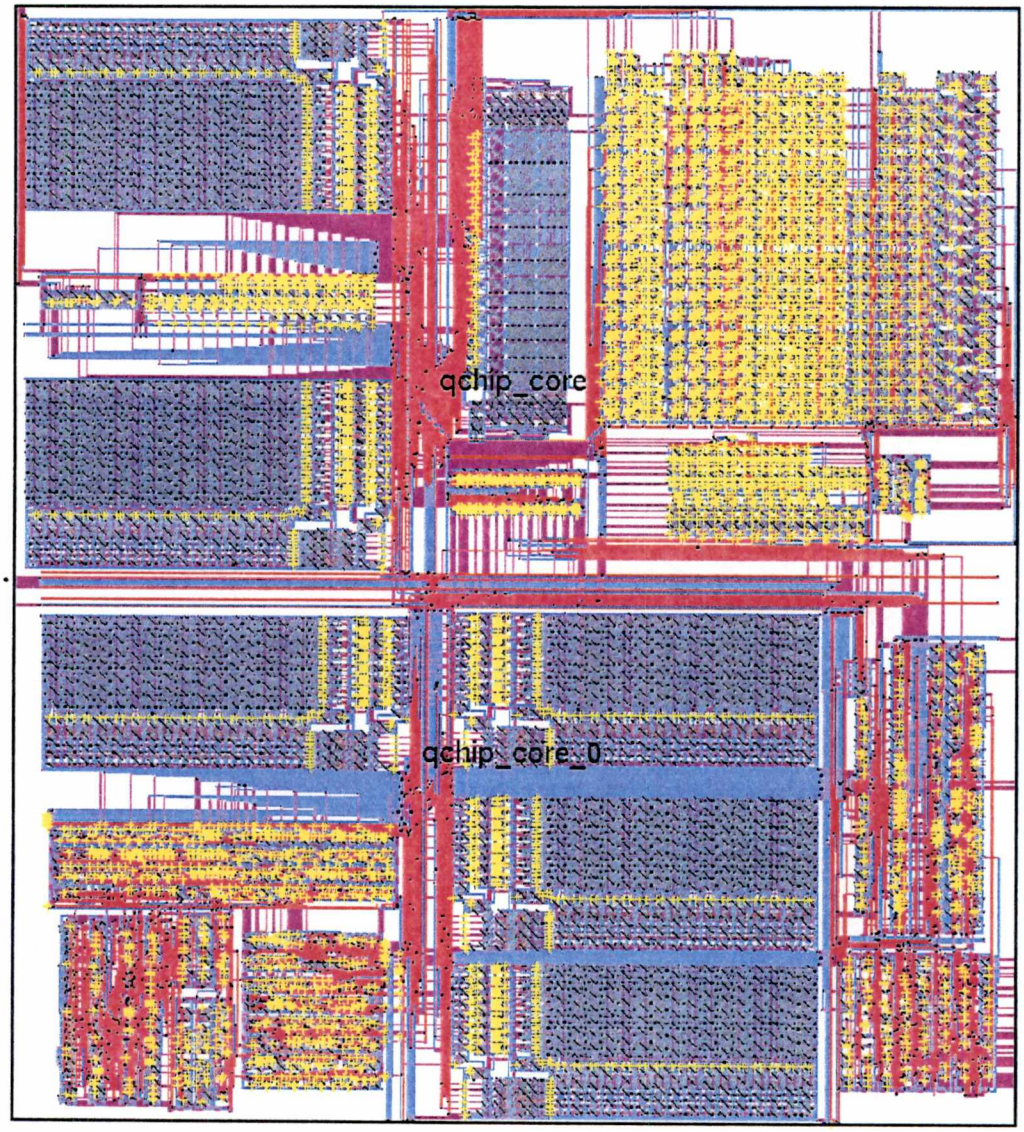


Figure 4.1: The "Quantizer" chip used as example to illustrate the implementation of A3PR.

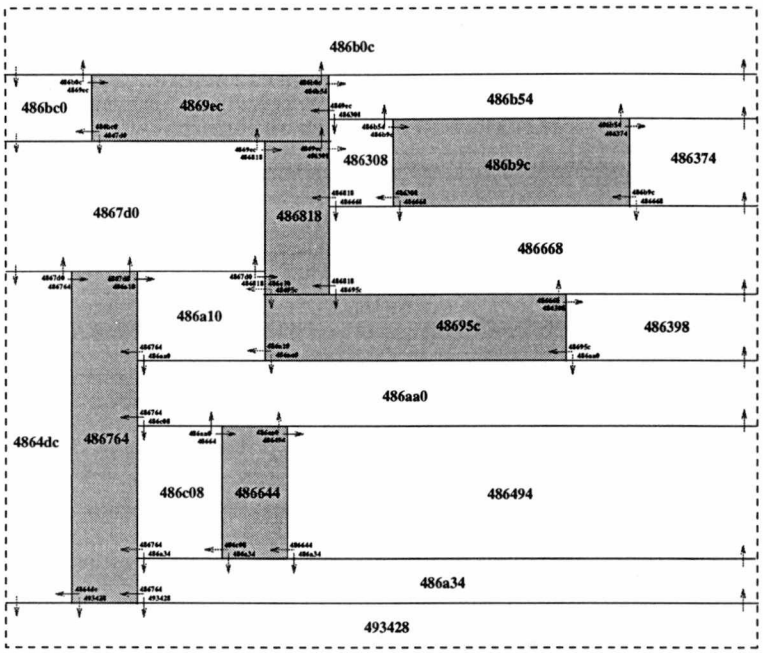
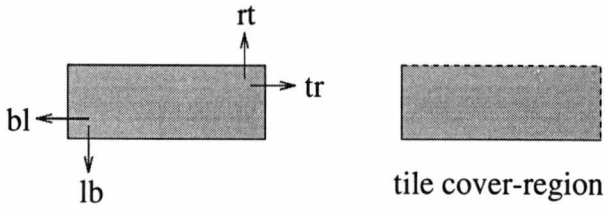


Figure 4.2: Rectangular geometries represented in corner stitching.

point lies within exactly one tile; a tile contains its left and bottom edges but not its top or right edges. Tiles of the same type are stored in maximal horizontal strips to prevent fragmentation and to provide a canonical form. Each tile on a plane is represented using four corner stitches, two at its lower–left corner and two at its upper–right corner. The two stitches at the lower–left corner, namely *bl* and *lb*, denote the neighboring bottom–most tile on the left and the left–most tile at the bottom. Similarly, two stitches at the upper–right corner of the tile point to the right–most tile at top (*rt*) and top–most tile on the right (*tr*). Figure 4.3 shows the representation of a tile. This representation has led to a simple search algorithm.

Database routines [40, 48] traverse these pointers to provide fast point search-




```
typedef structure tile
{ ClientData   ti_body;   /* Body of tile */
 struct tile   *ti_lb;
 struct tile   *ti_bl;
 struct tile   *ti_tr;
 struct tile   *ti_rt;
 Point          ti_ll;    /* Lower left coordinate */
 ClientData     ti_client;
  } Tile;
```

(Adapted from Magic layout tool)

Figure 4.3: Corner stitching data structure. Each tile has four pointers that are linked to its neighbors. Two are in the top–right corner: *rt* pointing up and *tr* pointing right. The other two are in the lower–left corner: *lb* pointing down and *bl* pointing left.

ing, fast area enumeration, fast neighbor finding, and fast database updates. To find the tile which includes the given point, $p(x, y)$, shown as a grey dot in Figure 4.4, the algorithm starts the searching from the current (reference) tile (tile with pointer *486a34*). It then moves vertically as shown by the heavy dashed line to find the tile whose vertical range contains the y coordinate (tile with pointer *486374*). The search now moves horizontally to find the tile whose horizontal range contains the x coordinate (tile with pointer *4867d0*).

For pad placement, an area search algorithm is used to see whether there are any solid tiles within the area covered by the pad searching box. This operation

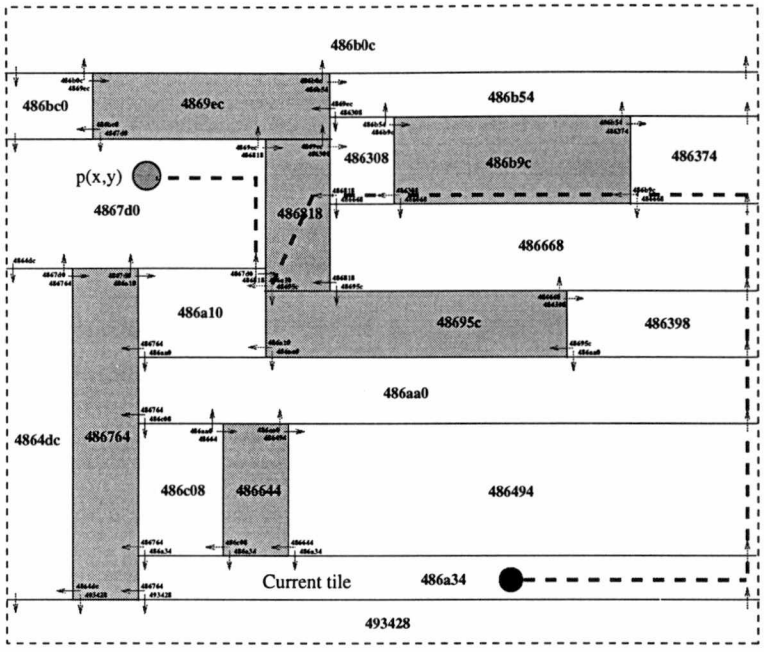


Figure 4.4: Point searching method in corner stitching.

can easily be performed using the corner–stitching data structure as follows [40]:

1. Find the tile that contains the lower left–corner of the searching box using the point–finding algorithm described above. If the tile is solid, the searching is done.

2. Otherwise, check the right edge of this space tile to see if it is inside the searching box. If it is, due to the maximal horizontal rule, the tile to the right of this space tile must be a solid tile or the tile representing the edge of the layout.

3. If the right tile is a solid tile, then the search is done. Otherwise, move upwards to the next tile touching the left edge of the searching box by

traversing upwards with the *rt* pointer and then traversing left using the *bl* pointer until the desired tile is found.

4. Repeat step 2 and 3 until a solid tile is found or until the top of the searching box is reached.

Figure 4.5 shows the example of the area searching algorithm. The tile containing the lower–left corner of the searching box is tile *486a34* and its right edge is beyond the searching area. Thus, the search moves upwards to tile *486494*. This tile already contains the left edge of the searching box and its right edge is outside the searching area. The search moves upwards to tile *486398* which does not contain the left edge of the searching box. Thus, the search moves directly to tile *48695c*, which is a solid tile.

Figure 4.6 shows the insertion of a new block (tile *486590*). This insertion starts by checking to see that there are no existing solid tiles in this area using the area search routine. Next, an insertion routine create a new tile, *486590*, then all the neighboring tiles are updated using tile splitting and merging routines. The detailed description of the algorithm is given in [48].

4.2 Data Extraction and Problem Setup

Given the hierarchical design of core circuits of the chip stored in CIF, the first step is to flatten and to extract the geometric information on the top few layers (as specified by the designer), the pad keepout areas, the locations of I/O

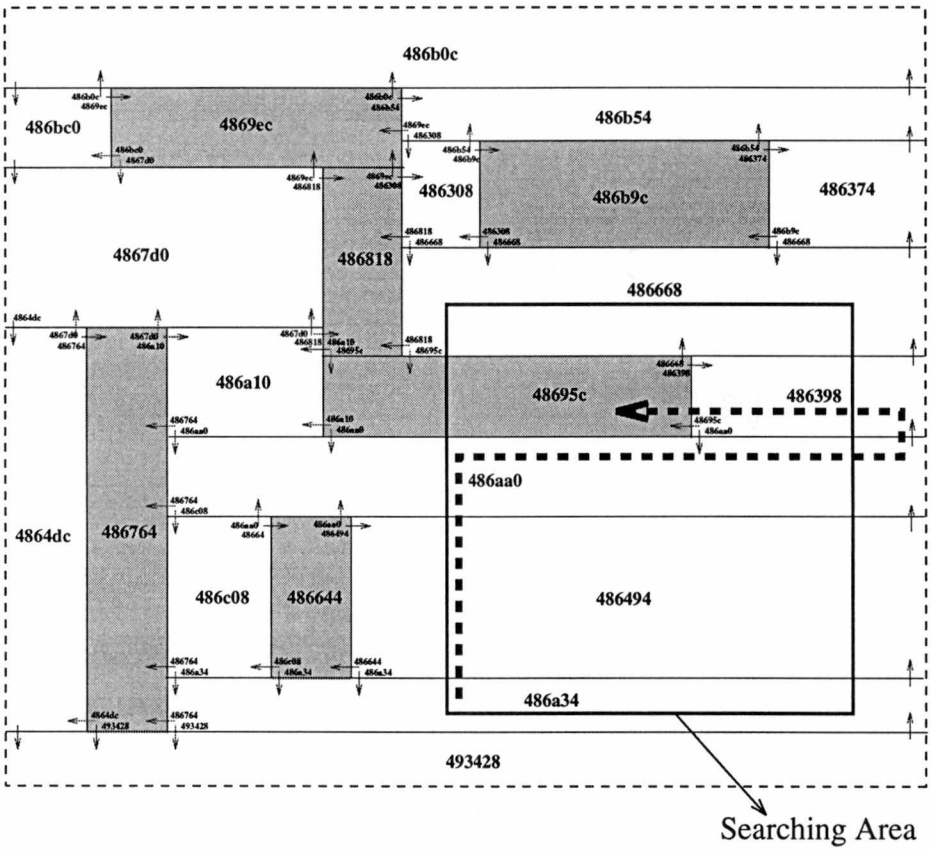
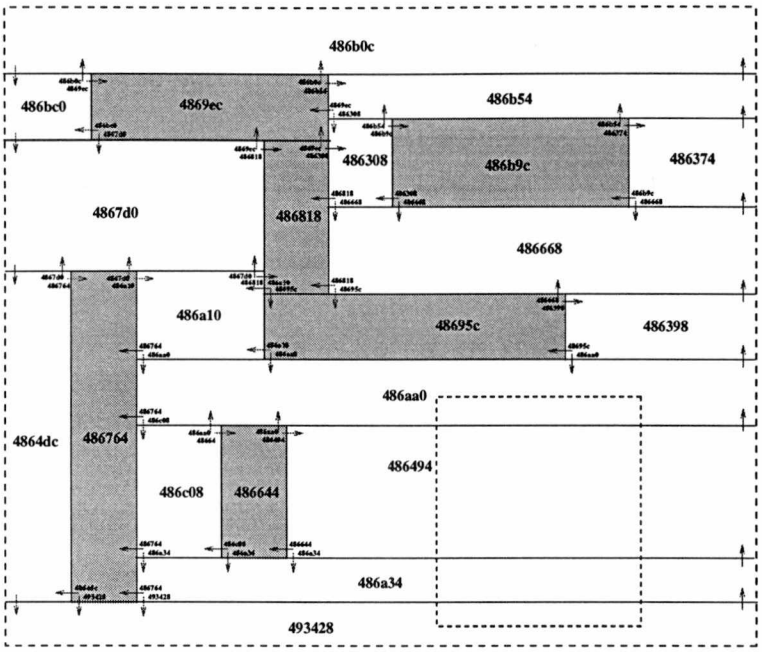
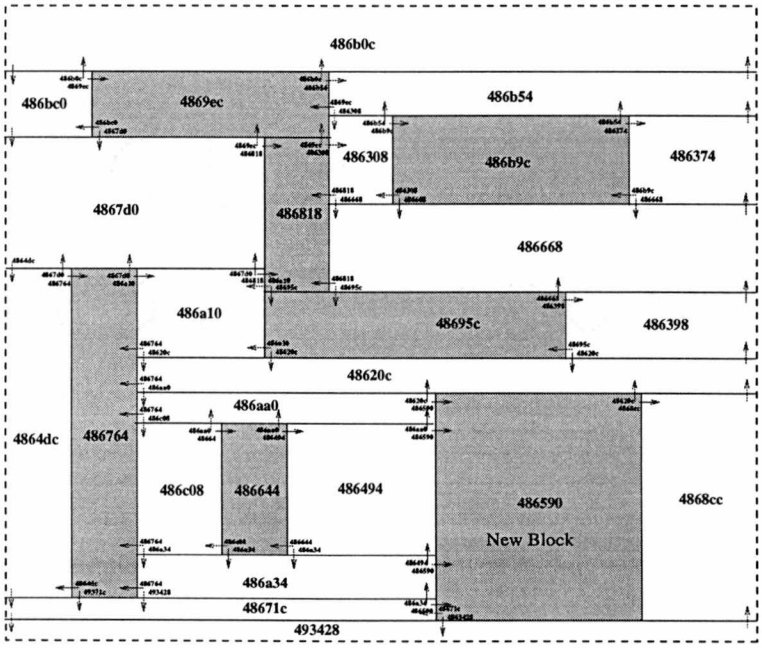


Figure 4.5: Area searching method in corner stitching.



(a)



(b)

Figure 4.6: Example of area insertion in corner stitching.

terminals, and the boundary of the core circuits. To accomplish this task, we modified a public domain program called *cifflatten* [2] developed at Univ. of California at Berkeley. We call this modified program *cifextr*. The input to the program are the name of input CIF file, the number of layers to be extracted, and the output CIF file. The processed CIF file contains the mask geometries on the layers specified by the designer, the location of the primary I/O ports, the pad keepout areas, and the boundary of the chip core. Pad keepout areas usually are the areas covered by the boundary of some subcells with labels containing special attributes (suffix "_KO" in this case). Hence, we just need to create a box corresponding to the bounding box of these subcells on a special layer in CIF. Because we use *Magic* to perform the layout of the padframe, we assign this special layer to the "fence" layer. To mark the chip–core boundary, we create two labels *LL_CORNER* and *RT_CORNER* corresponding to the coordinates of the lower–left and the upper–right corners of the bounding box of the chip core, respectively.

After bringing up *Magic*, the first step is to import the layer information from the flattened CIF file into the internal layout representation in Magic with a modified built–in CIF reading routine. The I/O port locations are recorded to be input into the pad assignment step. Figure 4.7 shows the layout of the top two metal layers extracted from core design of the Quantizer shown in Figure 4.1.

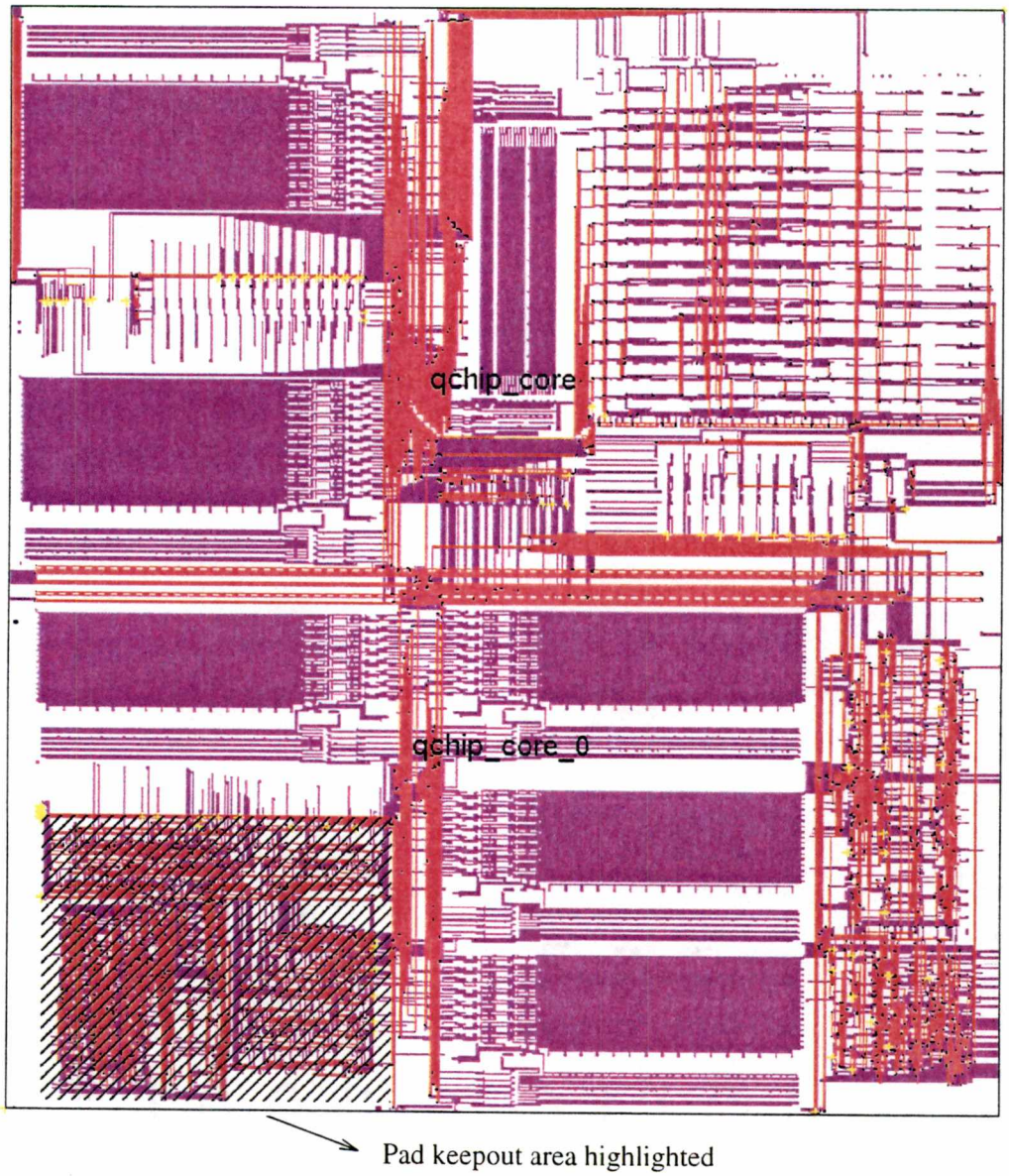


Figure 4.7: The layout of top two metal layers extracted from the core design of Quantizer.

4.3 Pad Placement

This section describes the implementation of the pad placement algorithm based on *corner–stitching* method. The pad placement accesses *Magic*'s internal database (corner–stitching data structure) and create the resulting pad placement directly into the database. The coordinates of the lower–left and upper–right corners of the chip–core boundary are obtained by finding the location of labels *LL_CORNER* and *RT_CORNER*. From these two coordinates, center point of the chip core is calculated and used as the reference point to find an array of potential pad sites under pad pitch constraint. The pad placement algorithm employs two database routines in *Magic* to perform the placement of the pads. The pads are placed on the top metal layer one at a time. In each step, the point–finding routine is used to locate the potential pad site. A searching box shown in Figure 3.5 is placed with its center corresponding to this pad site. Using the area searching routine, search the area which is overlapped by this searching box to see whether any solid tile exists. If there is no solid tile for both top metal plane and keepout area plane, then place the area pad at this pad site. The placement is started from the center of the core area and grows outward to form an array of fixed–grid pads with spacing corresponding to pad pitch. The placement process is terminated when the outermost row and column of the array are outside the boundary of the chip core and the number of pads is 10% larger than the number of I/O ports. The upper–right corner pad location is purposely left blank for chip orientation. After placing all the area pads, the keepout areas are clear for

routing. Figure 4.8 shows the result of applying the pad placement algorithm to the Quantizer. The pad size is 80 μm and the pitch is 120 μm. The number of pads placed is 124 which 10% larger than the number of I/O ports, 112.

4.4 Pad Assignment

Some of the I/Os which need special attention (need to be at certain location) may be assigned manually by the designer and routed with the computer–assisted router bulit in *Magic* called *iroute*. The remaining I/Os are assigned area pads using a bipartite weighted matching algorithm which minimizes the total costs as explained in Section 3.5.2. The inputs to this algorithm are a set of I/O port locations and a set of area pad locations. The outputs of the algorithm are the pairwise assignments of I/O ports to area pads. Thess assignments are imported into the routing routine as a two–terminal netlist.

4.5 Pad Routing

The routing algorithm implemented in this work is a maze–routing algorithm with A*–search. Instead of writing a new code for this implementation, we adapted the maze–router called *mzroute* built in *Magic*.

In the first step of the routing phase, the net ordering is performed on the netlist to give the net closer to the center of the chip–core boundary the highest priority. Based on this ordering, at each net routing, each I/O port is selected as

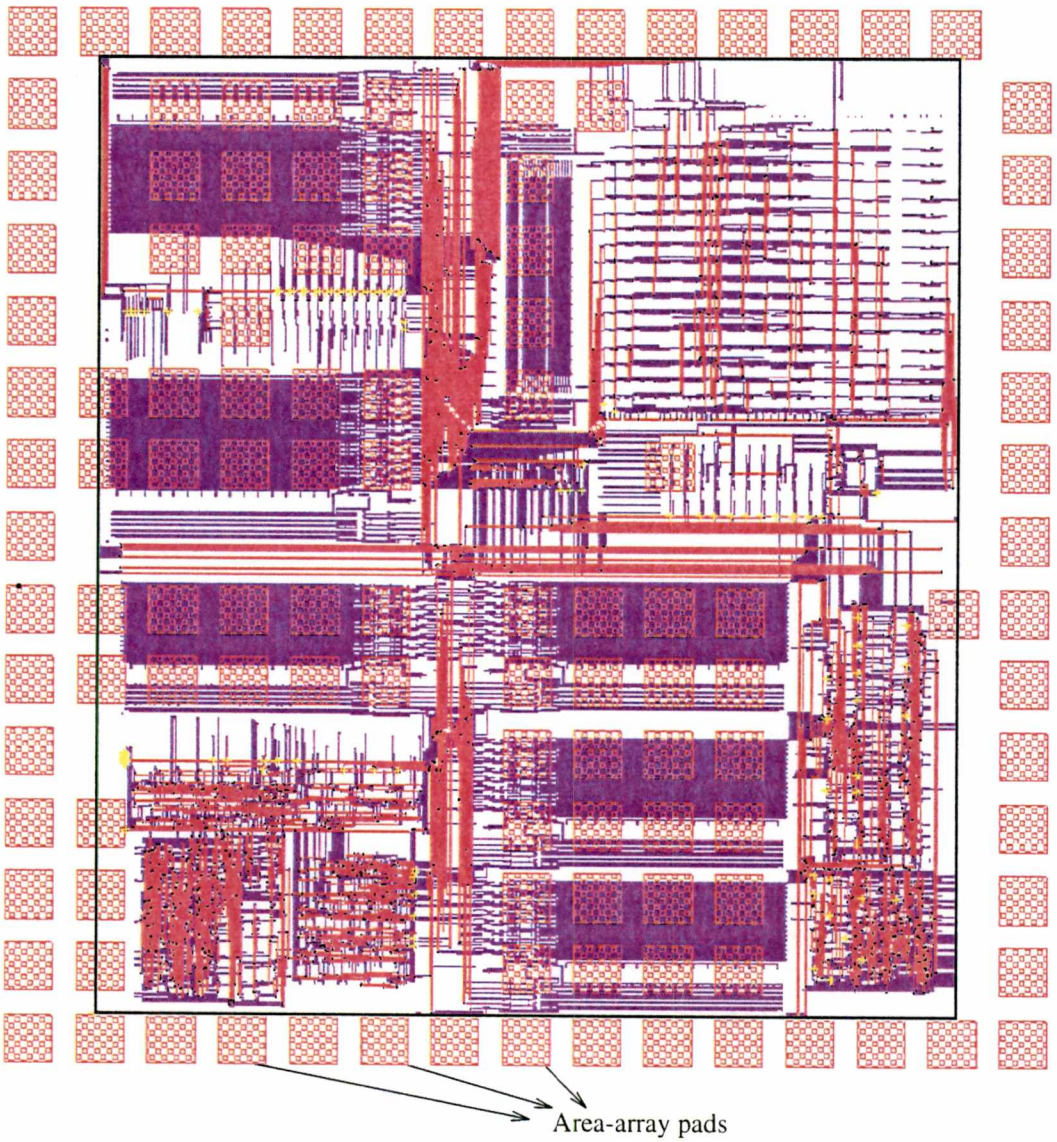


Figure 4.8: The area–array pads placed using placement algorithm. The pad size is 80 μm and the pitch is 120 μm.

the source point and an area pad as the destination target. Then *mzroute* was called with these parameters to perform the routing. If the routing failed, this net was recorded into an array. If the routing of the net was done, a glass cut and a label with suffix "_pad" were added to the targeted area pad. When all the nets had been processed, the unrouted nets were routed manually with computer assistance. Again, *iroute* can be used to help the designer to do the manual routing. To ease the routing, the pad routing routine provided a command to help the designer to find the location of the unrouted port location and its assigned area pad by zooming into the area containing the I/O port and highlighting both the I/O port and area pad. When all the nets were routed, the unused area pads that are located at the periphery of the pad array were assigned as dummy pads. The remaining unused area pads were erased from the layout. Figure 4.9 shows the result of the routing on Quantizer chip.

4.6 Output Generation

When placement and routing of area–array pads were completed, A3PR generated an area–array padframe by erasing all the layout information stored in the extracted CIF file and keeping the area–array pads and their routed wires to the I/O ports intact. This area–array padframe was then stored in a CIF file to be appended to the CIF file of the original core design to form a complete area–array IC. Figure 4.10 shows the area–array padframe of Quantizer chip and Figure 4.11 shows a complete area–array Quantizer IC. The chip has 112 bonding

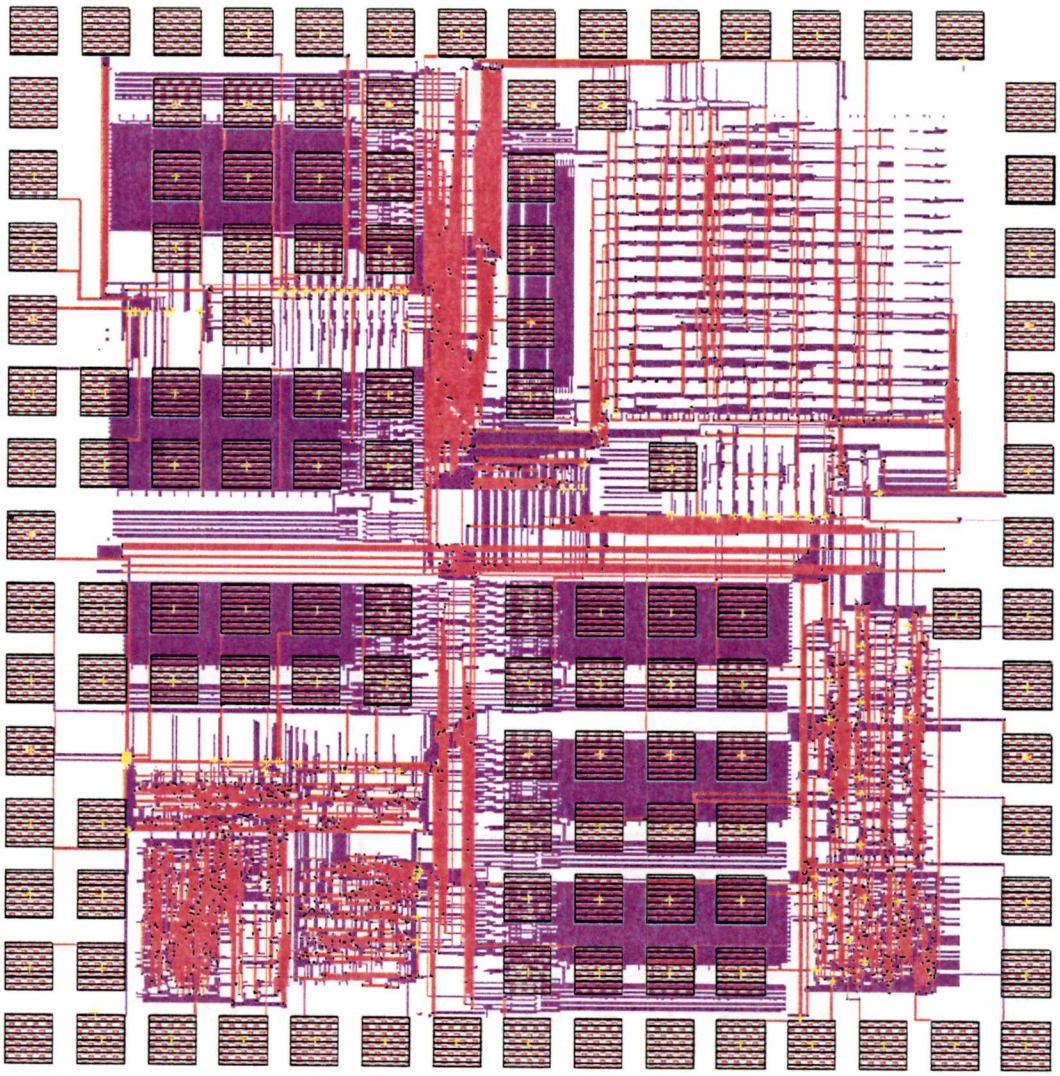


Figure 4.9: The completed routing result for the Quantizer chip.

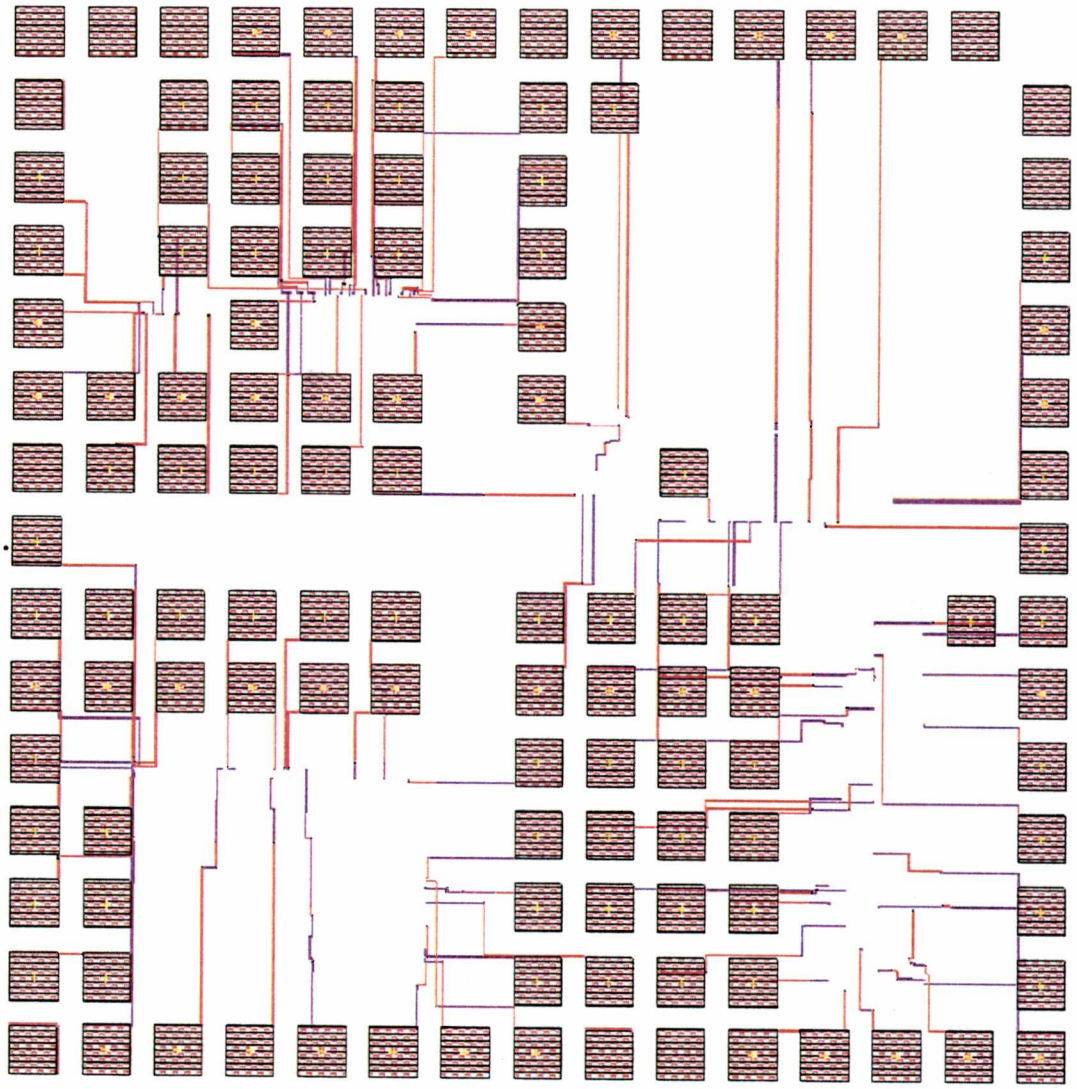


Figure 4.10: The area–array padframe of Quantizer chip extracted from the layout resulted from the routing step.

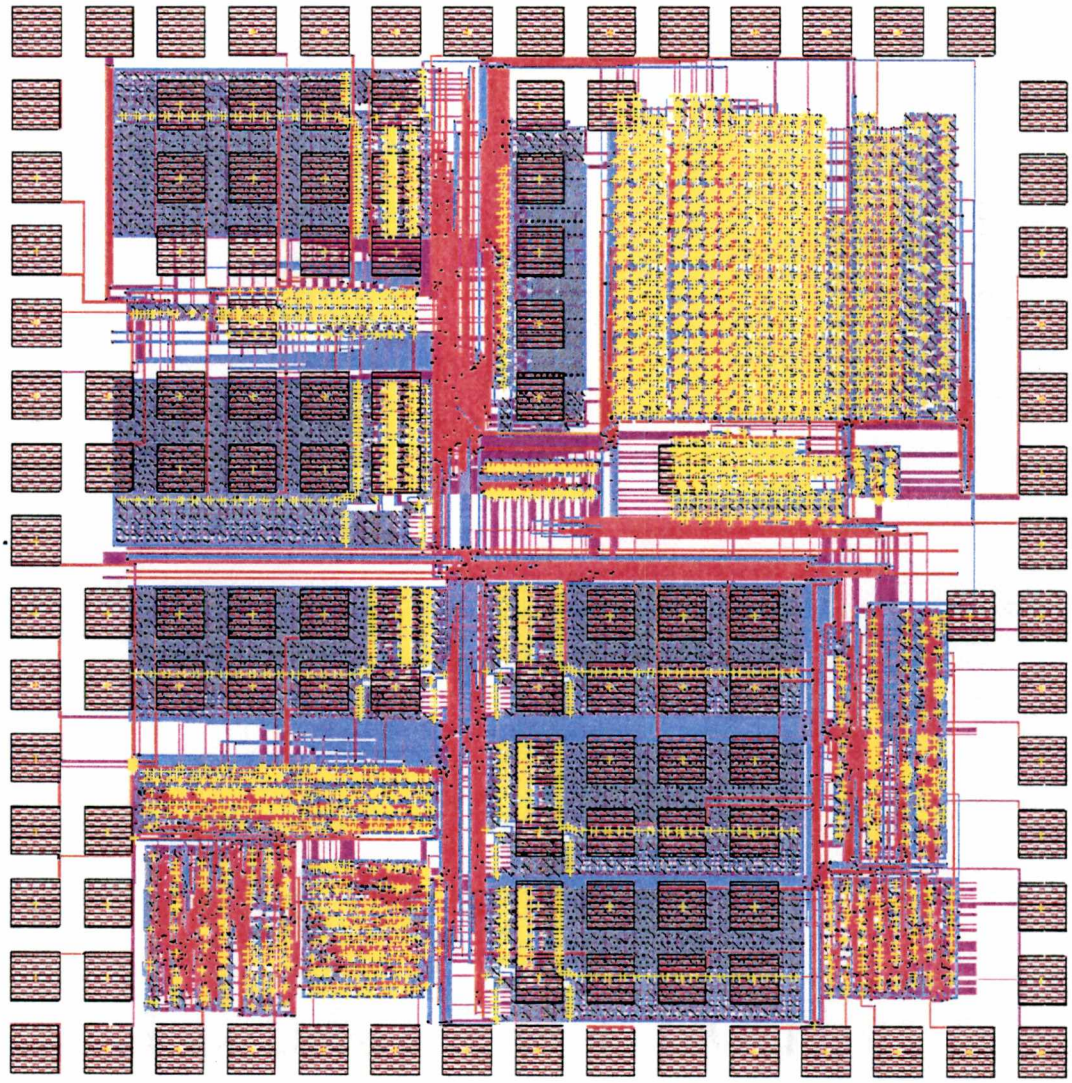


Figure 4.11: The complete area–array IC of Quantizer.

pads with size of 80 μm and pitch of 120 μm. All except two of the nets were routed automatically. The total size of this version is 1.76 $\times$ 1.76 mm. The Quantizer chip with peripheral pads was shown in Figure 4.12. Its size is 3.88 $\times$ 3.87 mm. The number of peripheral pad is 96 with size of 100 μm and pitch of 110 μm. The core of this chip is 1.48 $\times$ 1.64 mm.

4.7 Results and Discussions

For demonstration of the methodology of designing an intrinsic area–array IC, we conducted experiments are on the design of three partitioned chips from the SUN MicroSparc design resulting from the study conducted in [13]. The SUN Microsparc is a RISC processor design consists of about 750,000 transistors. The design is described at the functional unit level consisting of the following: integer unit (IU), floating point unit (FU), memory management unit (MMU), data cache (D-CACHE), instruction cache (I-CACHE), s-bus controller (S-BUS-CTL), memory interface unit (MEM-INTF), clock control and buffers (CLK-CTL), and miscellaneous control logic (MSC). The partitioned resulting from [13] is summarized in Table 4.1.

All three chips were resynthesized using 0.5 μm technology rules (hp14b) and laid out using the EPOCH physical design automation tool. The core circuits were exported to CIF files. The size of the core circuits for ChipA, ChipB, and ChipC were $5957.5 \times 5806.6 = 3.46e+07\ \mu m^2$, $6809.6 \times 6508.2 = 4.43e+7\ \mu m^2$, and $1500.8 \times 2119.2 = 3.18e+6\ \mu m^2$, respectively. Figure 4.13 and Figure 4.14 shows

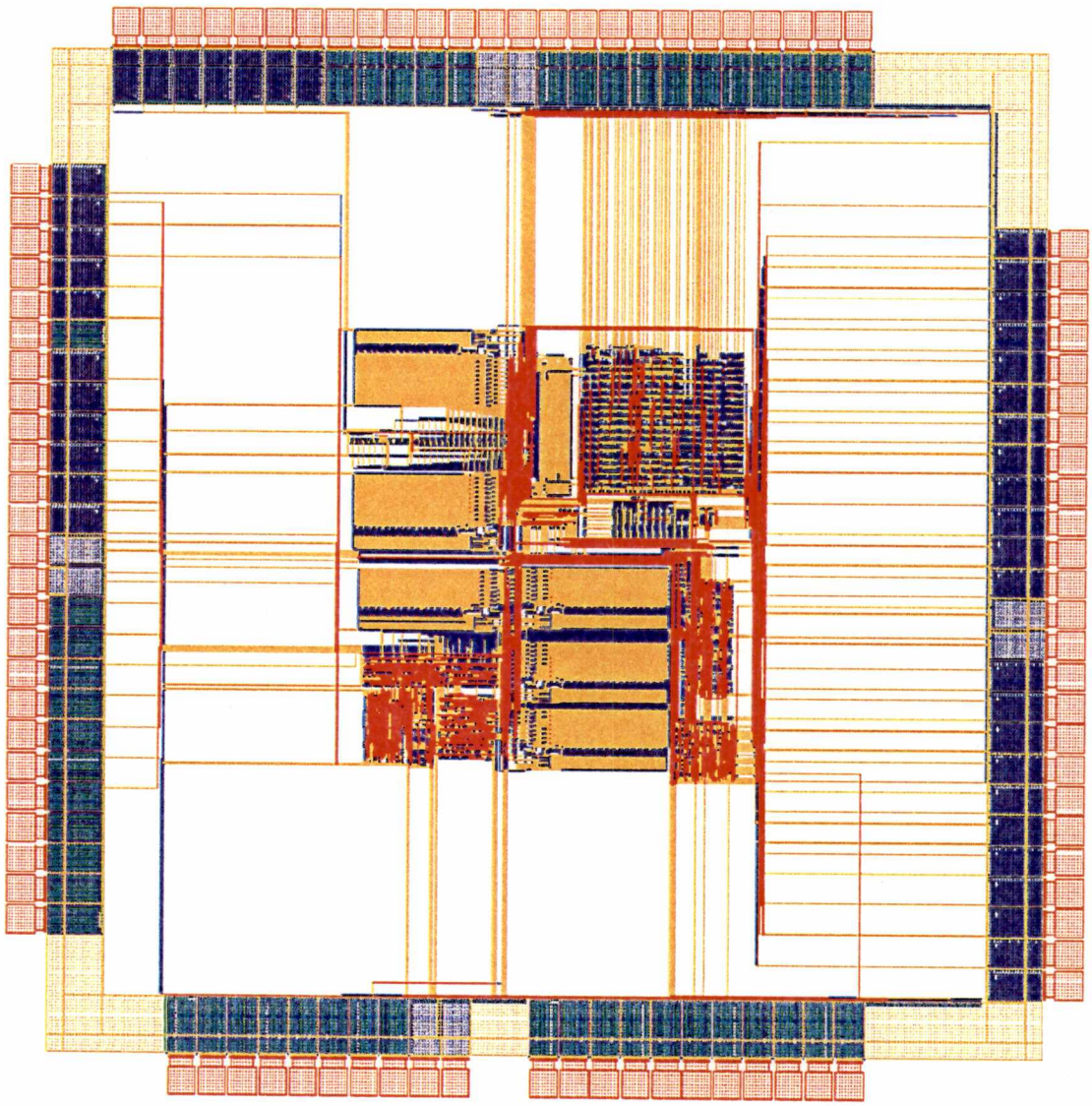


Figure 4.12: The peripheral IC of Quantizer.

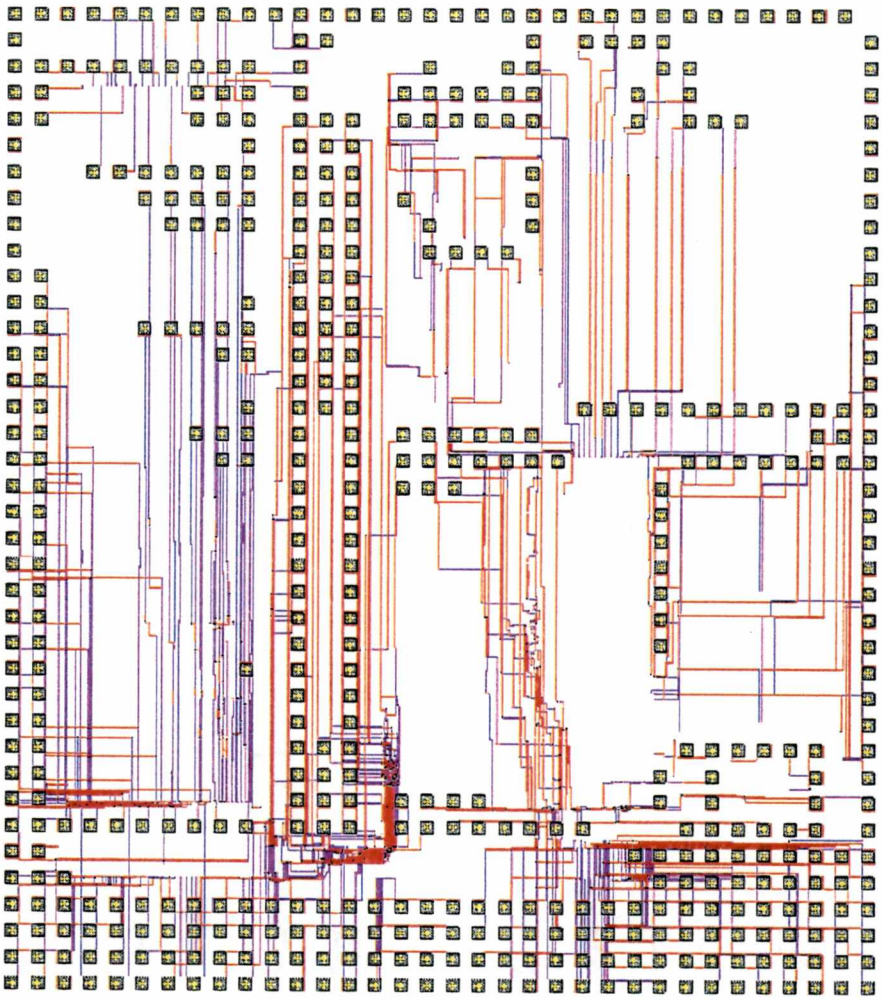


Figure 4.13: The area–array padframe of ChipA.

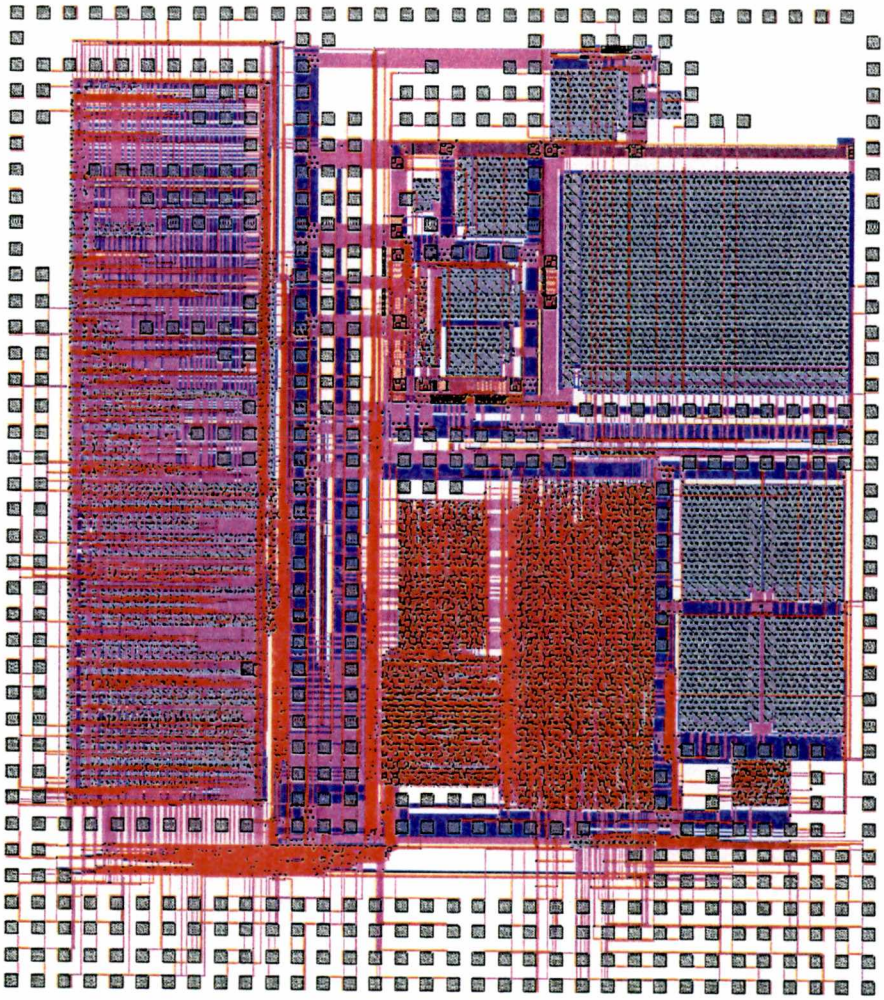


Figure 4.14: The complete area–array IC of ChipA.

Table 4.1: The three partitioned chips of SUN MicroSparc processor.

Chip	# of I/Os	Modules
A	485	D-CACHE, I-CACHE, MMU
B	298	FU, IU
C	414	MEM-INTF, SBC, CLK-CTL, MISC

the area–array padframe and the complete area–array IC of ChipA, respectively. The pad size is 100 μm and the pitch is 200 μm. Metal 3 was the top metal. The area pads were not placed over the area of RAM modules of the chip. The pad routing was done on metal 2 and metal 3 layers. The routing of the area pads took approximately 8 hours on an UltraSparc server with 512 Mbytes internal memory. There were 46 nets which were not routed automatically. It took another hour or so to route these nets manually using *iroute* with the assistance provided in our method. The reason it took so long to route the I/O ports to their assigned area pads was due to the clustering of I/O port locations at a certain region in the core area. In this case, most of the I/O ports were located at the bottom–left of the core area. Hence, some of the I/Os were assigned pads that are located at the top half of the core area. Therefore, the routing of these nets consumed a lot of time and memory space. If the I/O ports were scattered all over the core area, the routing would be faster. The total area of this chip is $6700 \times 7500 = 5.025e + 7\ \mu m^2$. The IC of ChipA with the peripheral pads is

shown in Figure 4.15. The area of this chip is $14279.3 \times 14129.3 = 2.018e{+}8\ \mu m^2$. From these results, we can see that the size of the area–array IC is almost 1/4 of the size of the peripheral IC.

The next design was ChipB of MicroSparc RISC processor. Figure 4.16 and Figure 4.17 shows the area–array padframe and the complete area–array IC of ChipB, respectively. The pad size is 100 μm and the pitch is 225 μm. Metal 3 was the top metal. The pad routing was done on metal 2 and metal 3 layers. The routing of the area pads took approximately 6 hours on the UltraSparc for the same reason explained in ChipA case. There were 16 nets which had to be routed manually. The total area of this chip is $7300 \times 6850 = 5.0e + 7\ \mu m^2$. The IC of ChipB with the peripheral pads is shown in Figure 4.18. The area of this chip is $9839.3 \times 10065.5 = 9.90375e + 7\ \mu m^2$. From these results, we can see that the size of the area–array IC is almost 1/2 of the size of the peripheral IC.

The last design was ChipC of the MicroSparc RISC processor. Figure 4.19 and Figure 4.20 shows the area–array padframe and the complete area–array IC of ChipC, respectively. The pad size is 80 μm and the pitch is 120 μm. Metal 3 was the top metal. The pad routing was done on metal 2 and metal 3 layers. The routing of the area pads took approximately 3 hours on UltraSparc machine. There were 40 nets which had to be routed manually. The total area of this chip is $3440 \times 3440 = 1.18e + 7\ \mu m^2$. Figure 4.21 shows the area–array IC of ChipC with the area pads placed on metal 4 layer. The routing was done on metal 2 and metal 3 as well as metal 4 layers. The size of this chip is 1.6 times smaller than

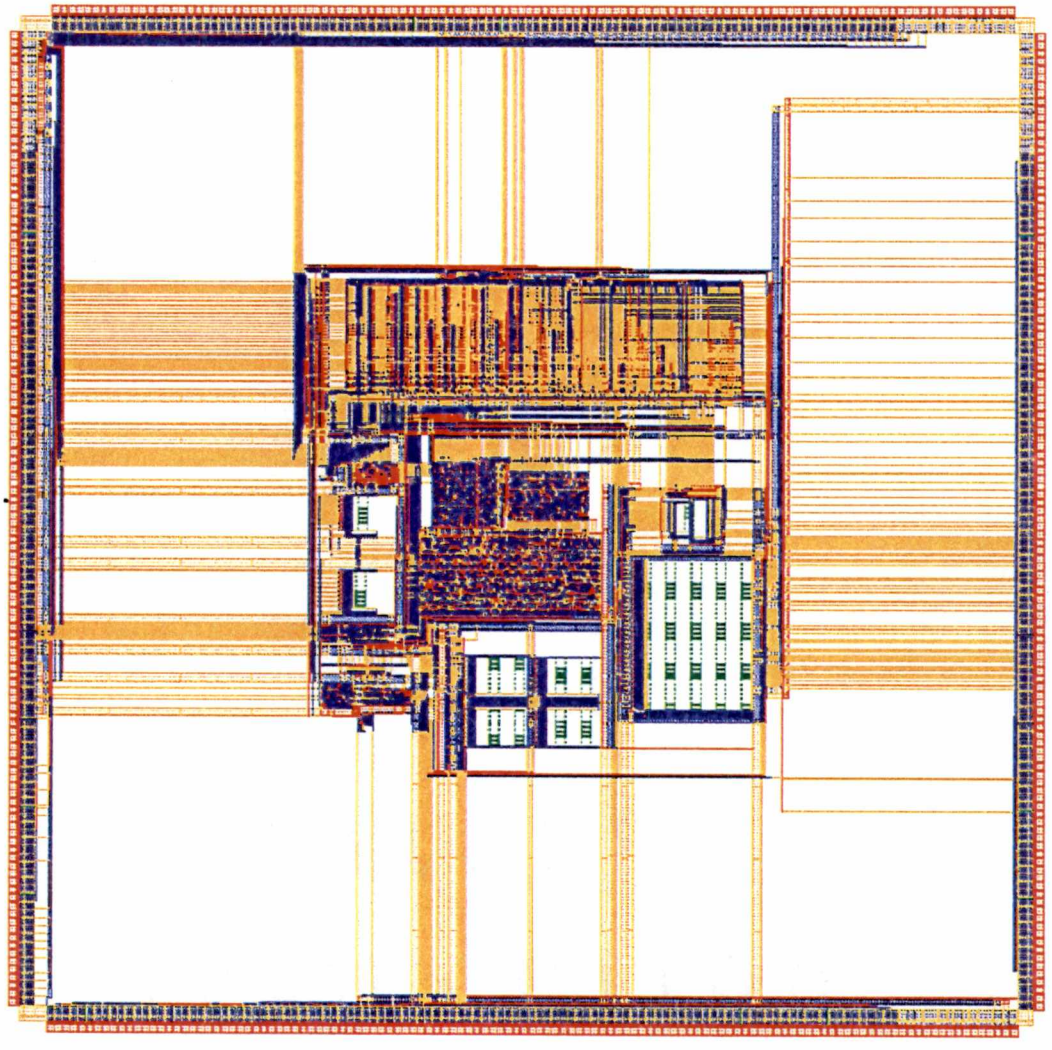


Figure 4.15: The peripheral IC of ChipA.

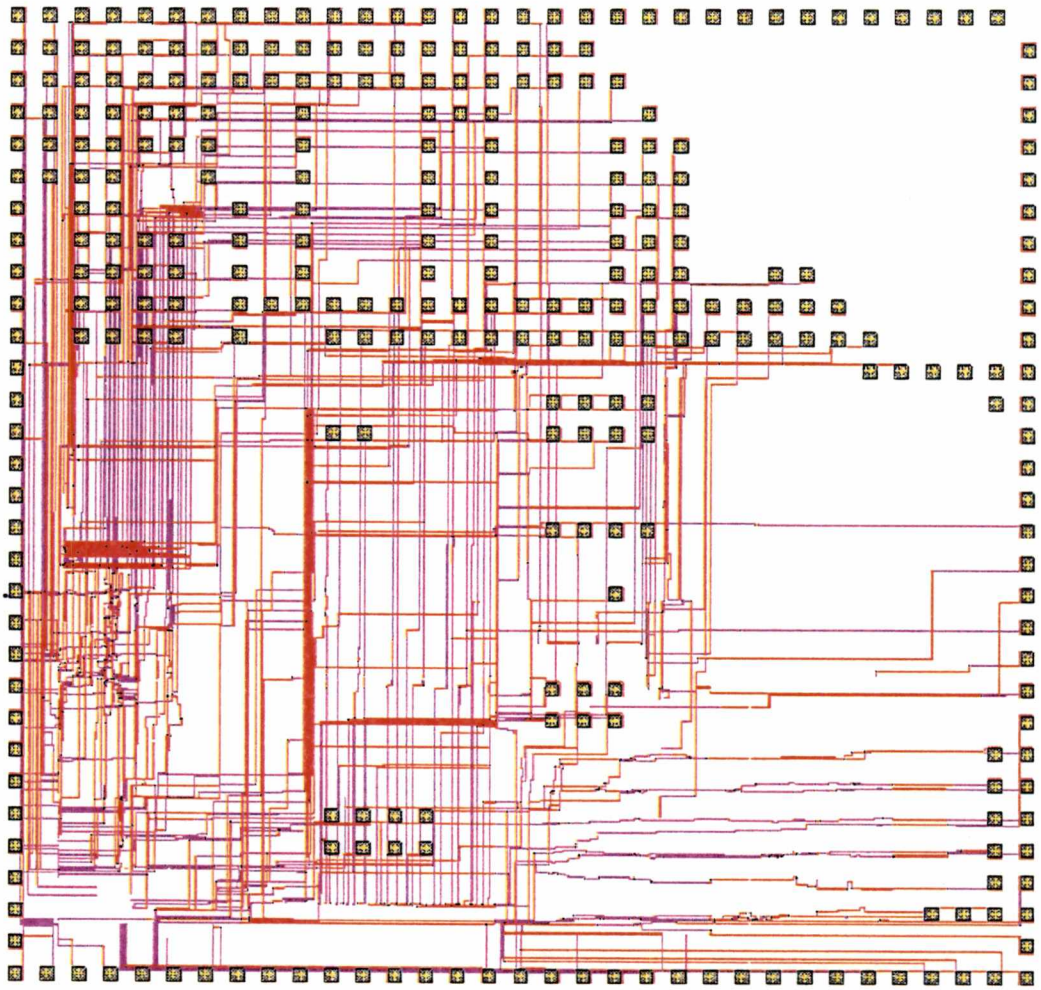


Figure 4.16: The area–array padframe of ChipB.

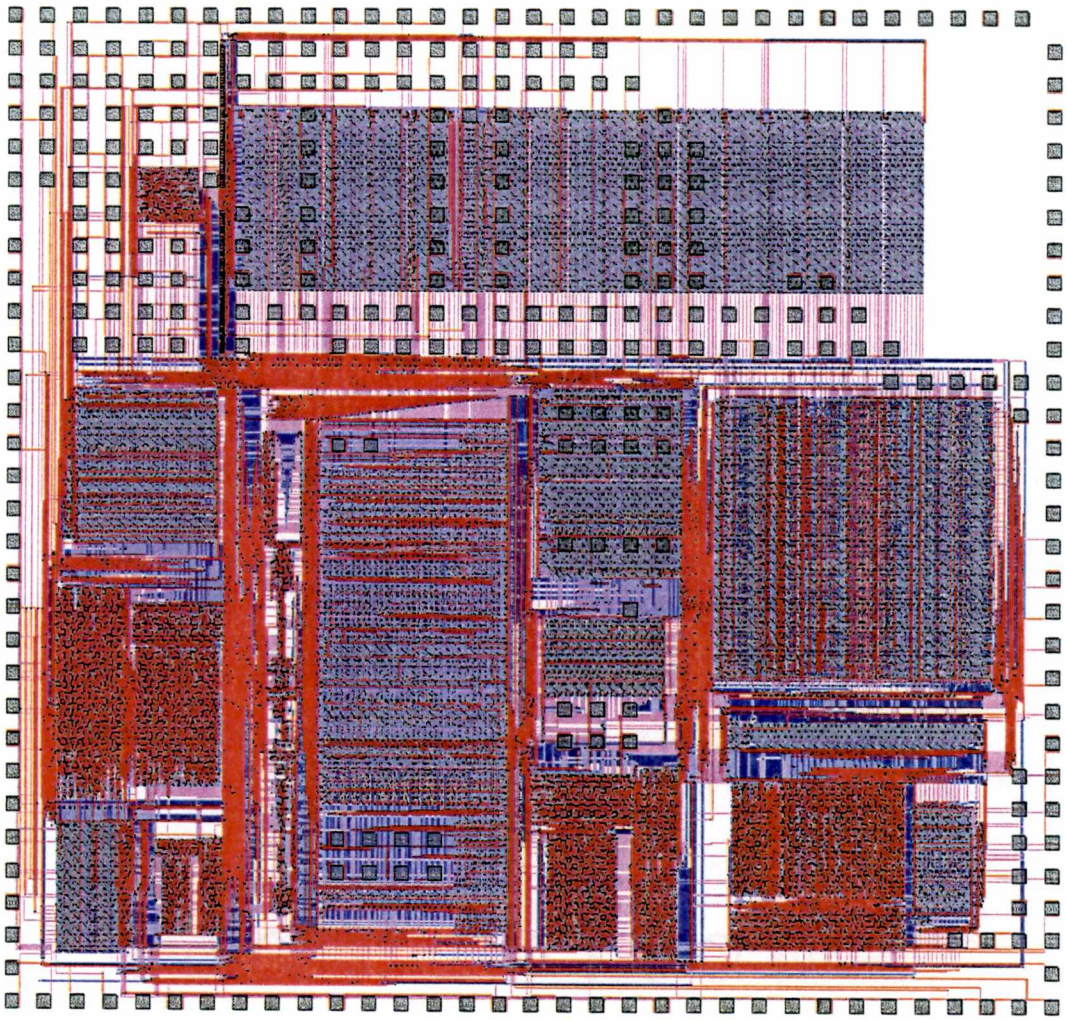


Figure 4.17: The complete area–array IC of ChipB.

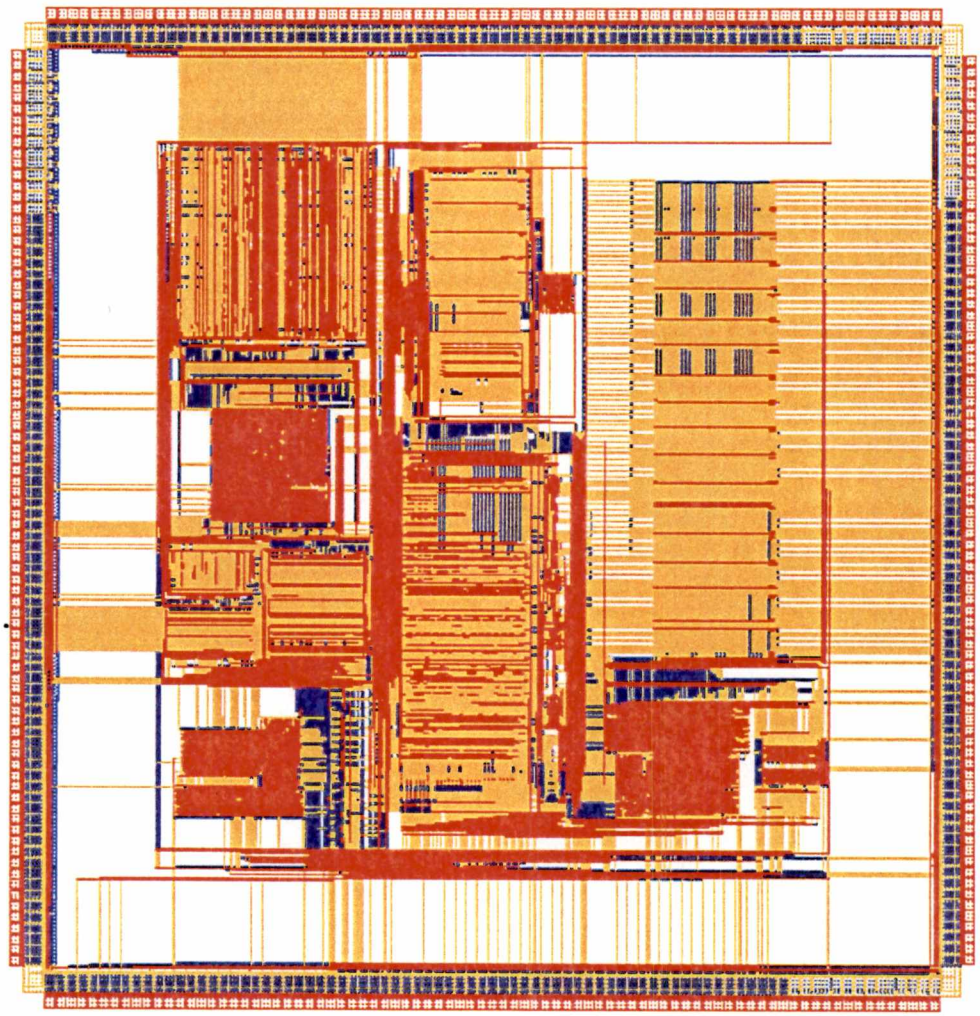


Figure 4.18: The peripheral IC of ChipB.

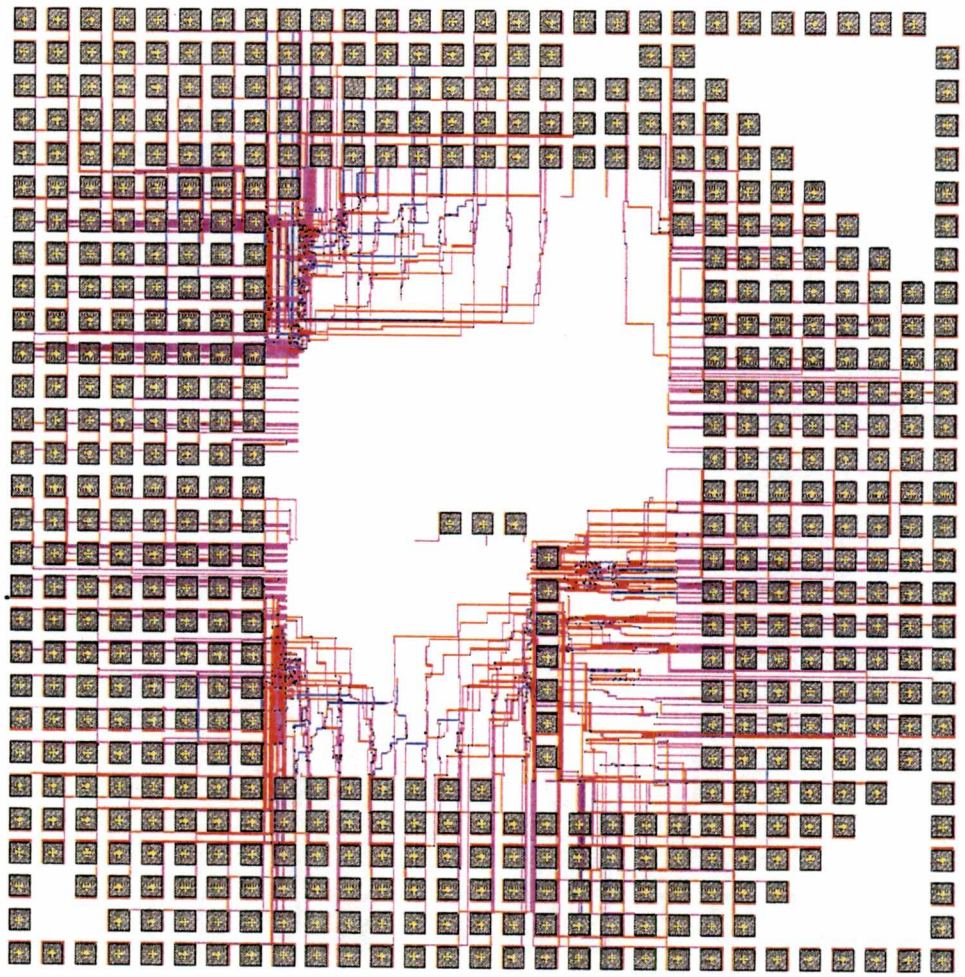


Figure 4.19: The area–array padframe of ChipC.

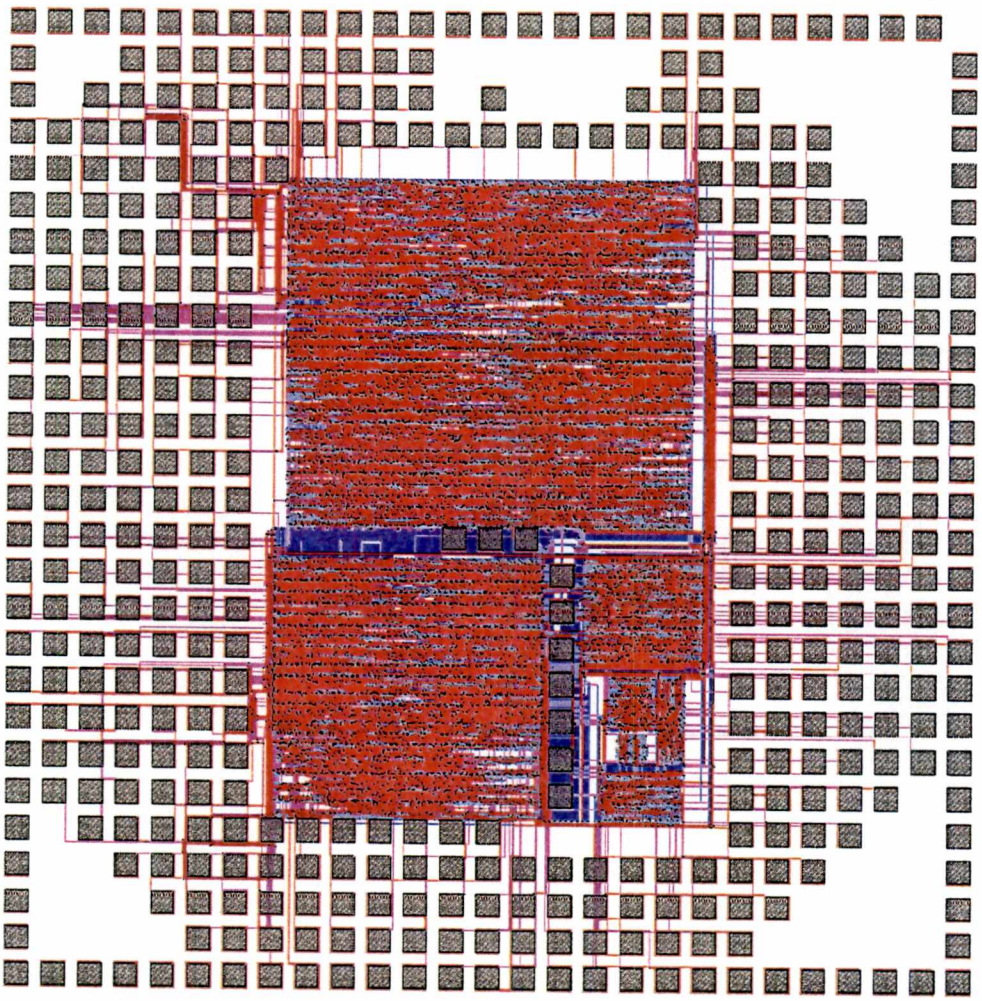


Figure 4.20: The complete area–array IC of ChipC.

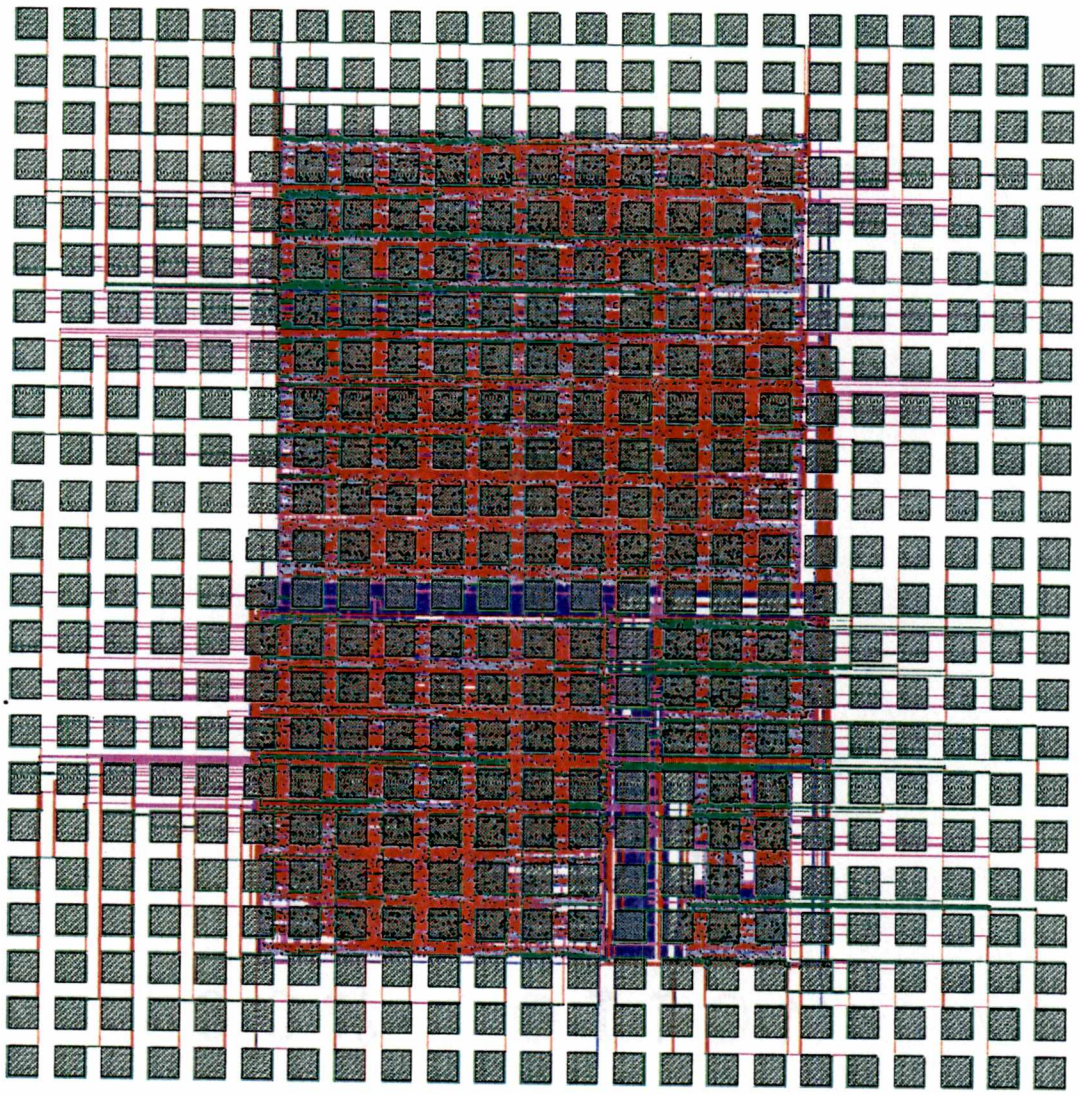


Figure 4.21: The complete area–array IC of ChipC (the area–array pads are placed on metal 4).

that obtained without using metal 4. The size is $2720 \times 2720 = 7.398e+6\ \mu m^2$. So adding an extra layer for placement of the area pads can reduce the chip size. The IC of ChipC with the peripheral pads is shown in Figure 4.22. The area of this chip is $11487 \times 11159 = 1.28e+08\ \mu m^2$. From these results, we can see that the size of the area–array IC is almost 1/10 of the size of the peripheral IC.

We purposely used different pad sizes and pitches for the above three different designs to show the flexibility of our tool. Comparison of chip sizes for area–array and peripheral IC for the three chips above are summarized in Table 4.2. From the results shown in this table, we can conclude that a signification reduction in chip size can be achieved by converting the peripheral IC with high I/O count to an intrinsic area–array IC.

Table 4.2: Comparison of chip sizes for area–array and peripheral IC for ChipA, ChipB, and ChipC.

Chip	# of I/Os	Peripheral IC			Area–array IC		
		pad size (μm)	pad pitch (μm)	chip size (μm^2)	pad size (μm)	pad pitch (μm)	chip size (μm^2)
A	485	100×100	110	20.18e+8	100×100	200	5.025e+7
B	298	100×100	110	9.9e+7	100×100	225	5.0e+7
C	414	100×100	110	12.8e+8	80×80	120	1.18e+7

More in depth analysis of this tool need to be performed to characterize the results in terms of time and space requirements for placement and routing of the area–array pads.

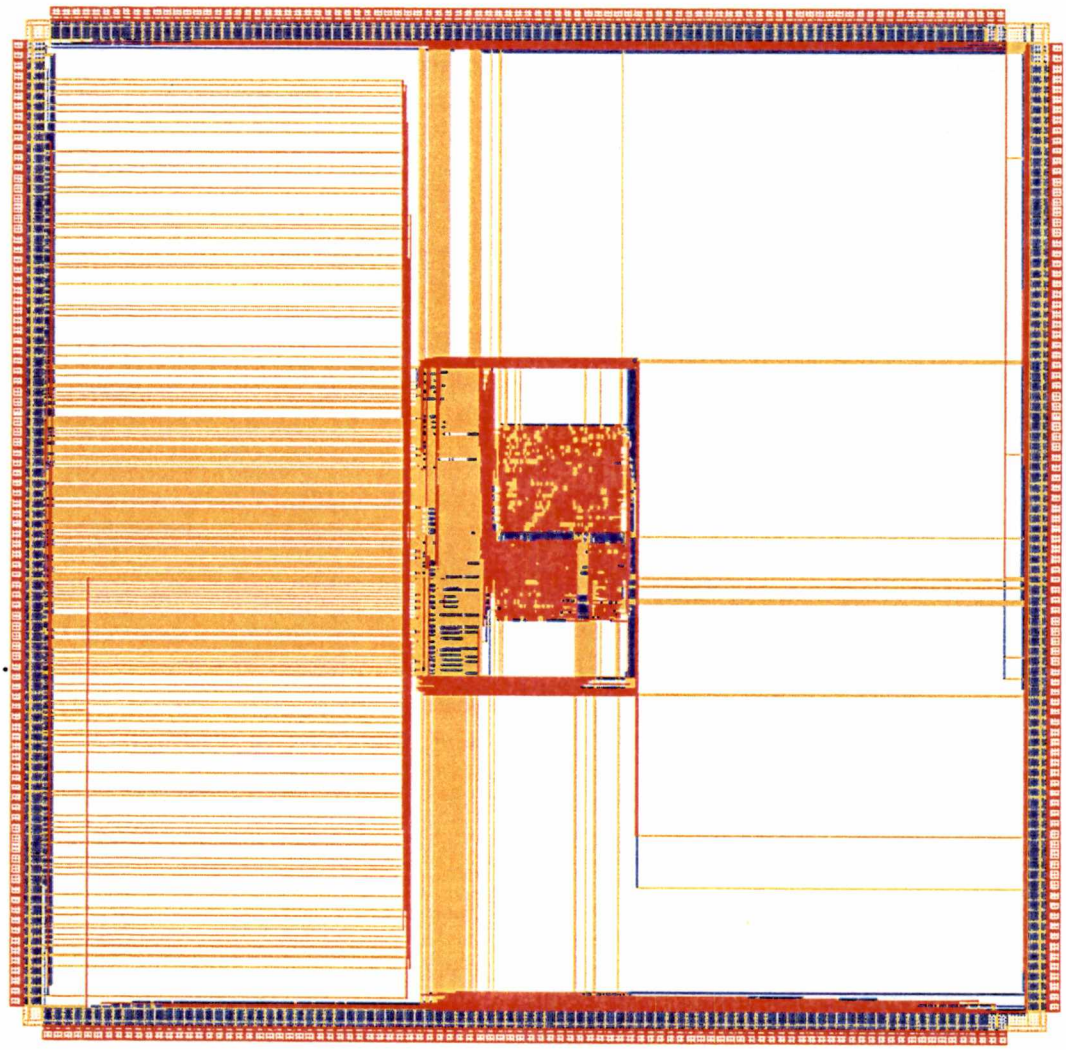


Figure 4.22: The peripheral IC of ChipC.

CHAPTER 5

SUMMARY, CONCLUSION AND FUTURE WORK

5.1 Summary

The framework for designing an intrinsic area–array IC, has been developed and implemented in this dissertation. This framework is divided into several stages: design guideline definition, data preparation, pad placement, pad assignment, pad routing, and output padframe generation. For each stage, the solution method and its implementation has been presented. The results from the pad generation tool for several design examples are satisfactory and promising.

We present the development of an area–array pad router which automates the placement and routing of the area–array pads on the IC. This is a post–processing tool which allows IC layouts generated by any other tool to be processed for area–array I/O pad placement and routing. This tool is different from a previously reported tool [9] since ours will place and route the pad using existing layers on the IC (if possible) before adding a new redistribution layer. Our method is also different from that presented [18] that we place the area-array pads on the top metal layer with some prerouted wires and routing of the I/O ports to these area pads are performed on the top few metal layers specified by the designer. Our approach is similar to the second method presented in [30]. However, in

our approach, the placement and routing of area-array pads are performed automatically. Unlike the physical design of a periphery bonded IC which places and routes the I/O cells (pads and I/O buffers) around the die after the active components of the chip have been laid out, the I/O buffers for area-array pads and the active components are laid out at the same time. The area-array bonding pads will then be placed on the top metal layer of the die and routed to their corresponding I/O buffers. In the implementation, we used the EPOCH physical design automation system developed by Cascade Design Automation to synthesize and layout the core circuits of the design following the guidelines provided in our method. The layout was then exported to a CIF file to be imported into our tool. An area-array padframe was generated and then appended to the core design to form an intrinsic area-array IC. The tool took approximately 6 minutes to place and route 112 area pads in the padframe of the Quantizer chip. The success rate of the routing was close to 99% so only 1% of the nets need to be manually routed. Even for a large design with close to 500 I/Os (ChipA), it took less than 8 hours to finish placing and routing the area pads with 46 unrouted nets. It took another hour or so to route these nets manually using *iroute* with the assistances provided in our method. For ChipB with 298 I/Os, the tool less than 6 hours to finish placing and routing the area pads with 16 unrouted nets. For ChipC with 414 I/Os, the tool less than 3 hours to finish placing and routing the area pads with 40 unrouted nets. Our method can also be incorporated into any physical design CAD tool to provide area-array pad placement and routing

capability.

5.2 Conclusion

Placing and routing area-array pads by hand will be virtually impossible and very time consuming as the number of I/Os become large. As flip–chip design approaches with high I/O count become more common in the future, there will be a greater demand to place and route area-array pads automatically for hundreds of I/Os. The framework presented in this dissertation for designing an intrinsic area-array IC will be useful in such a case.

5.3 Future Work

In this work, we did not touch the design issue of I/O drivers and receivers. We have assumed that these I/O circuits had been placed and routed in the chip core. Further work in this area needs to be done in the future. Currently, the routing only supports the rectilinear model that only allows horizontal and vertical path. In the future, we plan to extend the capability to handle the diagonal routing model. More in depth analysis of this tool need to be performed to characterize the results from pad placement and pad routing in terms of time and space.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Home page of Magic — A VLSI Layout System. At URL address: *http://www.research.digital.com/wrl/projects/magic/magic.html.*

[2] World wide web home page of MOSIS user-contributed files. At URL address: *http://www.mosis.org/contrib.html.*

[3] World wide web home page of Flipchip Technologies. At URL address: *http://www.flipchip.com*, August 1996.

[4] M. H. Arnold and W. S. Scott. An Interactive Maze Router with Hints. In *Proc. 25th Design Automation Conf.*, pages 672–676, Anaheim, CA, June 1988.

[5] A. J. Blodgett and D. R. Barbour. Thermal Conduction Module: A High-Performance Ceramic Package. *IBM Journal of Research and Development*, 26(1):30–36, 1982.

[6] M. Burstein and R. Pelavin. Hierarchical Wire Routing. *IEEE Trans. on Computer-Aided Design*, CAD-2(4):223–234, 1983.

[7] J.-D. Cho. *High-Performance Physical Design in Multilayer Packages.* PhD thesis, NorthWestern University, Evanston, Illinois, June 1993.

[8] J. P. Cohoon and P. L. Heck. BEAVER: A Computational-Geometry-Based Tool for Switchbox Routing. *IEEE Trans. on Computer-Aided Design*, CAD-7(6):684–697, 1988.

[9] J. Darnauer and W. W. Dai. Fast Pad ResΦdistribution for Periphery-IO to Area-IO. In *Proc. Multichip Module Conf.*, pages 38–43, Santa Cruz, CA, March 1994.

[10] W. A. Dees and P. G. Karger. Automated Rip-up and Reroute Techniques. In *Proc. 19th Design Automation Conf.*, pages 432–439, 1982.

[11] P. Dehkordi and D. Bouldin. Design for Packageability: Early Consideration of Packaging from a VLSI Designer's Viewpoint. *Computer*, 1:76–81, Apr. 1993.

[12] P. Dehkordi and D. Bouldin. Design for Packageability: The Impact of Bonding Technology on the Size and Layout of VLSI Dies. In *Proc. IEEE Multichip Module Conf.*, pages 153–159, 1993.

[13] P. Dehkordi, K. Ramamurthi, D. Bouldin, H. Davidson, and P. Sandborn. Impact of Packaging Technology on System Partitioning: A Case Study. In *Proc. Multichip Module Conf.*, pages 144–149, Santa Cruz, CA, 1995.

[14] P. Dehkordi, C. Tan, and D. Bouldin. Intrinsic Area-Array ICs: What, Why, and How. In *Proc. Multichip Module Conf.*, pages 120–124, Santa Cruz, CA, February 1997.

[15] U. .Derigs. A Shortest Augmenting Path Method for Solving Minimal Perfect Matching Problems. *Networks*, 11:379–390, 1981.

[16] E. W. Dijkstra. A Note Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[17] M. Engquist. A Successive Shortest Path Method for the Assignment Problem. *INFOR*, 20:370–384, 1982.

[18] R. Farbarik, X. Liu, M. Rossman, P. Parakh, T. Basso, and R. Brown. CAD Tools for Area-Distributed I/O Pad Packaging. In *Proc. Multichip Module Conf.*, pages 125–129, Santa Cruz, CA, February 1997.

[19] J. Ganley and J. P. Cohoon. A provably good moat routing algorithm. In *The Sixth Great Lakes Symposium on VLSI*, pages 86–91, Ames, Iowa, March 1996.

[20] G. T. Hamachi. An Obstacle-Avoiding Router for Custom VLSI. Technical Report UCB/CSD 86/291, Univ. of California, Berkeley, CA, April 1986.

[21] J. Hao and G. Kocur. *Network Flows and Matching : First DIMACS Implementation Challenge*, volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter An Implementation of a Shortest Augmenting Path Algorithm for the Assignment Problem. American Mathematical Society, Providence, R.I., 1993.

[22] D. W. Hightower. A Solution to the Line–Routing Problem on the Continuous Plane. In *Proc. of 6th Design Automation Conf.*, 1–24 1969.

[23] J.-M. Ho, M. Sarrafzadeh, G. Vijayan, and C. K. Wong. Pad Minimization for Planar Routing of Multiple Power Nets. *IEEE Trans. on Computer-Aided Design*, CAD-9(4):419–426, April 1990.

[24] J.-M. Ho, G. Vijayan, and C. K. Wong. Routing Multiple Power and Ground Nets on a Single Layer. Technical Report IBMC 13549, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, N.Y., October 1987.

[25] W. Huang and J. Castro. CBGA Package Design for C4 PowerPC Microprocessor Chips: Trade-off between Substrate Routability and Performance. In *IEEE 44th Electronic Components & Technology Conf.*, pages 88–93, Washington, D. C., May 1994.

[26] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. Clock Routing for High-Performance ICs. In *Proc. of 27th Design Automation Conf.*, pages 573–579, 1990.

[27] R. Jonker and A. Volgenant. A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems". *Computing*, 38:325–340, 1987.

[28] R. M. Karp. An Algorithm to Solve the $m \times n$ Assignment Problem in Expected Time $O(mn \log n)$. *Networks*, 10:143–152, 1980.

[29] H. W. Khun. The Hungarian Method for the Assignment Problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[30] F. E. Kiamilev, A. V. Krishnamoorthy, J. R. R. G. Rozier, G. F. Aplin, C. D. Hull, R. Farbarik, and R. E.Oettel. Design of ICs for Flip-Chip Integration with Optoelectronic Device Arrays. In *Proc. Multichip Module Conf.*, pages 163–167, Santa Cruz, CA, February 1997.

[31] J. U. Knickerbocker, G. B. Leung, W. R. Miller, S. P. Young, S. A. Sands, and R. F. Indynk. IBM System/390 Air Cooled Alumina Thermal Conduction Module. *IBM Journal of Research and Development*, 35(3):330–341, May 1991.

[32] G. Kromann, D. Gerke, and W. Huang. A Hi-Density C4/CBGA Interconnect Technology for a CMOS Microprocessor. In *Proc. IEEE 44th Electronic Components & Technology Conf.*, pages 22–28, Washington, D. C., May 1994.

[33] C. Y. Lee. An Algorithm for Path Connections and Its Applications. *IRE Trans. Electronic Computers*, pages 246–265, September 1961.

[34] E. R. Lettang. Padroute: A Tool for Routing the Bonding Pads of Integrated Circuits. Master's thesis, University of California, Berkeley, CA, June 1989.

[35] R. K. McGehee. A Practical Moat Router. In *Proc. the 24th Design Automation Conf.*, pages 216–222, 1987.

[36] C. A. Mead and L. A. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[37] E. F. Moore. Shortest Path Through A Maze. In *Annals of Computation Laboratory of Harvard University.*, volume 30, pages 285–292, Cambridge, MA. Harvard Univ. Press, 1959.

[38] K. C. Norris and A. H. Landzberg. Reliability of Controlled Collapse Interconnections. *IBM Journal of Research and Development*, 13:266–271, 1969.

[39] T. Ohtsuki. *Layout Design and Verification*, volume 4 of *Advances in CAD for VLSI*, chapter Maze–Running and Line–Search Algorithms. Elsevier Science Publishing Company, North-Holland, Amsterdam, 1986.

[40] J. K. Ousterhout. Corner Stitching:A Data Structuring Technique for VLSI Layout Tools. *IEEE Trans. on Computer-Aided Design*, CAD-3(1):87–100, 1984.

[41] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor. A Collection of Papers on Magic. Technical Report CSD-83-154, Univ. of California, Berkeley, CA., December 1983.

[42] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[43] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison–Wesley, Reading, MA., 1984.

[44] B. Preas and M. Lorenzetti, editors. *Physical Design Automation of VLSI Systems.* The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1988.

[45] J. Reed, A. Sangiovanni-Vincentelli, and A. Santamauro. A New Gridless Channel Router: YACR2 . *IEEE Trans. on Computer-Aided Designs*, CAD-4(3):495–465, July 1985.

[46] R. L. Rivest and C. M. Fiducia. A 'Greedy' Channel Router. In *Proc. 19th Design Automation Conf.*, pages 418–424, Los Alamitos, CA., June 1982. IEEE Computer Society Press.

[47] A. M. Sait and H. Youssef. *VLSI Physical Design Automation: Theory and Practice.* McGraw-Hill Book Company, New York, NY., 1995.

[48] P. Sandborn, M. Abadir, and C. Murphy. The Tradeoff Between Peripheral and Area Array Bonding of Components in Multichip Modules. *IEEE Trans. on Components, Packaging and Mfg. Tech. - Part A*, 17(2):249–256, June 1994.

[49] M. A. Shand. Algorithms for Corner Stitched Data-Structures. *Algorithmica*, 2(1):61–80, 1987.

[50] N. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, Bosto, MA, second edition, 1995.

[51] H. Shin and A. Sangiovanni-Vincentelli. Mighty: A Detailed Router Based on Incremental Routing Modifications. *IEEE Trans. on Computer-Aided Design*, CAD-6(6):942–955, November 1987.

[52] J. Soukup. Circuit Layout. In *Proc. of the IEEE*, pages 1281–1304. IEEE, October 1981.

[53] K. Suzuki, Y. Matsunaga, T. Kagata, and T. Ohtsuki. A Fast Maze Router with Application to Iterative Rip-up abd Reroute. *IEEE Trans. on Computer-Aided Design*, CAD-5(4):466–476, 1986.

[54] R. R. Tummala and J. Knickerbocker. Advanced Cofired Multichip Technology at IBM. In *Proc. IEPS.*, San Diego, CA, September 1991.

[55] R. R. Tummala and E. J. Rymaszewski, editors. *Microelectronic Packaging Handbook.* Van Nostrand Reinhold, 1989.

[56] D. C. Wang. Pad Placement and Ring Routing for Custom Chip Layout. In *Proc. 27th Design Automation Conf.*, pages 193–199, 1990.

[57] I. Yee, R. Miracky, J. Reed, and B. Lunceford. Flexible Manufacturing of Multichip Modules for Flip Chip ICs. In *Proc. Multichip Module Conf.*, pages 130–132, Santa Cruz, CA, February 1997.

[58] T. Yosimura and E. Kuh. Efficient Algorithms for Channel Routing. *IEEE Transactions on Computer Aided Design*, CAD-1(1):25–35, January 1982.

[59] M.-F. Yu and W. W.-M. Dai. Single-Layer Fanout Routing and Routability Analysis for Ball Grid Arrays. Technical Report UCSL–CRL–95–18, Univ. of California, Santa Cruz, CA., April 1995.

[60] Q. Zhu and W. M. Dai. Hierachical Clock Routing Scheme for Multichip Modules Based on Area Pad Interconnection. Technical Report UCSC-CRL-93-45, Board of Studies in Computer Engineering, University of California at at Santa Cruz, Santa Cruz, CA, 1993.

VITA

Chandra Tan graduated with first rank in his class in May 1977 from Ir. H. Juanda High School, Tebing Tinggi. He worked for his father in family business for 5 years before coming to United Stated in September 1992. He joined the South Georgia College, Georgia and graduated with a Associate Degree with honor. He received a Bachelor of Science degree with highest honor, Master of Science degree and Doctor of Philosophy degree in Electrical Engineering in 1986, 1989 and 1997, respectively from the University of Tennessee, Knoxville. He was a Teaching and a Research Assistant at the Computer Vision and Research Laboratory from 1987 to 1995. He was a Graduate Research Assistant at Microelectronic Systems Research Laboratory. His research interests are in VLSI circuit design, CAD tool development, Computer Vision, Image Processing, and Pattern Recognition.