ACCELERATION OF SIMULATED ANNEALING

BUILDING BLOCK PLACEMENT PROCESS

by

Man Kuan Vai

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering and
Systems Science

1987

ABSTRACT

ACCELERATION OF SIMULATED ANNEALING BUILDING BLOCK PLACEMENT PROCESS

By

Man Kuan Vai

One of the important issues in VLSI design is the building block placement problem. The difficulty lies in optimal placement of many blocks on the chip so that those which are strongly connected are close to each other. Simulated annealing has been shown to be very effective in the placement problem; however, the process is very slow due to the numerous configurations generated at each temperature stage.

New methods are developed in this work to perform the building block placement task effectively. A new model for representing the building block placement problem in a CAE system is developed. This model reduces the size of the problem space so that a solution is found more rapidly. Moreover, this model guarantees that overlap between building blocks will not occur in the placement result.

An explicit mapping method is developed to map the building blocks of a placement problem to different regions of a chip. The power of parallel processing is applied to the placement process. The parallel placement method performs simulated annealing concurrently on different regions of a chip.

The architecture of a hardware placement engine implementing the parallel placement method is developed. This engine can be implemented as a coprocessor for a CAE system.

Experimental results on different design cases show that the placements are comparable to those obtained with the conventional algorithm; however, the computation time has been reduced by more than an order of magnitude.

To my parents

Mr. and Mrs. Man-Kit Vai

ACKNOWLEDGMENTS

I would like to thank my major advisor, Dr. Michael A. Shanblatt, for his invaluable insight and constant support during the development of this work. His guidance and encouragement were critical in the course of this research.

Comments from the committee members, Dr. David Fisher, Dr. Chin-Long Wey, Dr. Lionel Ni and Dr. Byron Drachman, helped me define the organization of this work. I am especially grateful for their encouragement and suggestions.

I also want to thank Dr. Stephanie Shanblatt for her blue pencil. I am grateful for the time she gave to polish my prose.

Finally, I owe many thanks to my best friend (and wife), Jin-Yu.

Table of Contents

List of Tables

List of Figures

CHAPTER I

INTRODUCTION

VLSI (Very Large Scale Integration) technology provides the capability of accommodating more than one million transistors on a single chip. This order of complexity has promoted the popularity of a divide-and-conquer design philosophy and hierarchical design methodologies have become prevalent in ASIC (Application Specific Integrated Circuit) designs [1, 2]. Hierarchical design methodologies simplify all critical design steps of a VLSI-based structure, including hardware specification, architecture development, physical layout, and simulation. The principle of hierarchical design is to predesign a set of circuits with different functions and complexities and store them in the library of a CAE (Computer-Aided Engineering) system. These library circuits are used as building blocks in an ASIC design process to construct larger systems. The physical layout process of an ASIC can then be described, from a simplified point of view, as the placement and interconnection of the desired building blocks on the chip.

Experiments show that the performance of a chip produced by using the building block concept is generally inferior to that of a full custom design in which the chip is built from scratch; however, the sub-optimal chip performance is compensated for by the savings in design complexity and turn-around time [3]. Moreover, the performance gap between the full custom and the building block approaches will be brought closer or even fully removed as computer-aided design tools are improved.

This research contributes to the physical layout process of ASIC design, specifically the building block placement problem. The ultimate goal of this work is to provide methods, with both software and hardware considerations, to accelerate and improve the building block placement procedure.

1.1 Problem Statement

The building block placement process belongs to the class of combinatorial optimization problems. The establishment of an optimal placement configuration has been shown to be NP-complete, which implies that the process of finding this configuration has a complexity that grows exponentially with the number of building blocks [4]. The difficulty of building block placement lies in optimally placing many components, or building blocks, on a chip so that those which are strongly connected are close to each other. Due to the NP-complete characteristic of the problem, heuristic algorithms have been developed to find a feasible solution in a finite time [5].

A typical heuristic placement process utilizes an iterative improvement strategy and proceeds as follows. An initial layout configuration is generated as the starting point of the heuristic search process. Small modifications are then made to the layout configuration at each step to generate a new configuration, which is evaluated according to an objective function. Traditional heuristic algorithms accept only configurations which improve the objective function. This criterion of acceptance often causes the process to be trapped into a local minimum of the objective function.

An algorithm which simulates the metal annealing process for application in the placement process has been developed to deal with the local minimum problem [6-8]. This algorithm proceeds similar to the conventional iterative improvement method except that a process control parameter, or a pseudo-temperature, is introduced into the placement process. The temperature is decreased very slowly from a large value to simulate the annealing process. New configurations are accepted if the objective function is improved and cases in which the objective function is worsened are accepted conditionally using a probability calculated from a Boltzman distribution function.

This algorithm has been shown to be very effective for the placement problem; however, it is very time consuming since numerous configurations must be generated and evaluated at each temperature step to simulate the slow cooling process. Due to the extremely large solution space in the placement problem, the long computing time of any heuristic algorithm searching for a combinatorial solution is always a major concern. This computation time problem is even more serious in the simulated annealing approach as the algorithm must ensure that every building block has a fair chance of being chosen for movement to cover all possible solutions in the problem space. Thus, there is a desire to provide a model which reduces the size of the solution space in the building block placement process. The simulated annealing process can then converge to a near-optimal solution more rapidly while maintaining good placement results.

Most of the conventional placement algorithms use nodes and edges as defined in graph theory to model the building blocks and their interconnections, respectively [9]. The interconnected graph is then algorithmically rearranged to produce a solution. One nice feature of using a connected graph to represent the placement problem is that many techniques developed in graph theory can be applied. However, there are some inherent disadvantages in this conventional model.

The information about the shapes and sizes of the building blocks cannot be readily considered using this model since all blocks are collapsed into dimensionless nodes. A placement algorithm utilizing this node-and-edge model can determine at most the relative locations between building blocks because the reconstruction of the nodes back into actual building blocks generally results in some overlap among adjacent blocks. Overlapping building blocks are, of course, unacceptable from a physical standpoint. The overlap can be eliminated by several heuristic methods during or after the placement process, but these methods are in themselves very time consuming and often degrade the placement result. Another problem with this model is that the locations of the block terminals, which are critical for the placement process to provide a routable configuration, cannot be considered.

In view of the inherent drawbacks of the node-and-edge model, it is desirable to provide a new model to represent the building block placement problem in a CAE system. This model should retain the advantages of the node-and-edge model but remove its drawbacks.

Much effort has been devoted to accelerate the slow building block placement process [10-13]. Due to the NP-complete nature of the problem, there is a limit in the acceleration that can be obtained by any algorithm that manages the entire set of building blocks simultaneously. As mentioned, the philosophy of divide-and-conquer is popular in VLSI design methodologies. Thus, it seems that a reasonable approach here is to decompose a design case with a large number of building blocks into several smaller problems. However, this approach to the placement problem has received little attention since, in general, the optimization at one level may increase the complexity at all other levels.

Partitioning the building block placement problem into several smaller subproblems naturally arises when parallel processing is considered. If the circuit under consideration has several non-intersecting sub-circuits, the decomposition task is rather straightforward. Unfortunately, this is often not the case in a practical ASIC design. A good method for applying the strength of parallel processing in the placement problem has yet to be developed.

Following the advances in integrated circuit technology and the development of computer-aided design tools, it is not uncommon to develop special hardware architectures to handle computationally intensive problems. Without the overhead of a general purpose computer, the building block placement process can take the advantage of a dedicated hardware accelerator, implementing a suitable placement algorithm, to achieve high performance. In view of this, there is a need to investigate and develop the architecture of a building block placement engine and its associated algorithm.

1.2 Approach

The goal of this work is to develop methods, with both software and hardware considerations, to perform the building block placement task effectively. Of particular interest in this research is the implementation of the simulated annealing approach, the advantages of which have already been established. Interestingly, most, if not all, of the results of this research can also be applied to other heuristic placement methods. The specific tasks performed in this work are listed below. These tasks and their relationships are graphically presented in Figure 1.1.

A new model for representing the building block placement problem in a CAE system is developed and tested. The purpose of this new model is to eliminate the drawbacks of the conventional node-and-edge model without sacrificing its advantages. In developing this model, a set of goals have been defined. The size of the problem space in the placement process has to be reduced so that the process converges to the final configuration rapidly. In addition, critical information about the building blocks, i.e., their sizes, shapes, and terminal locations, has to be considered in this model to

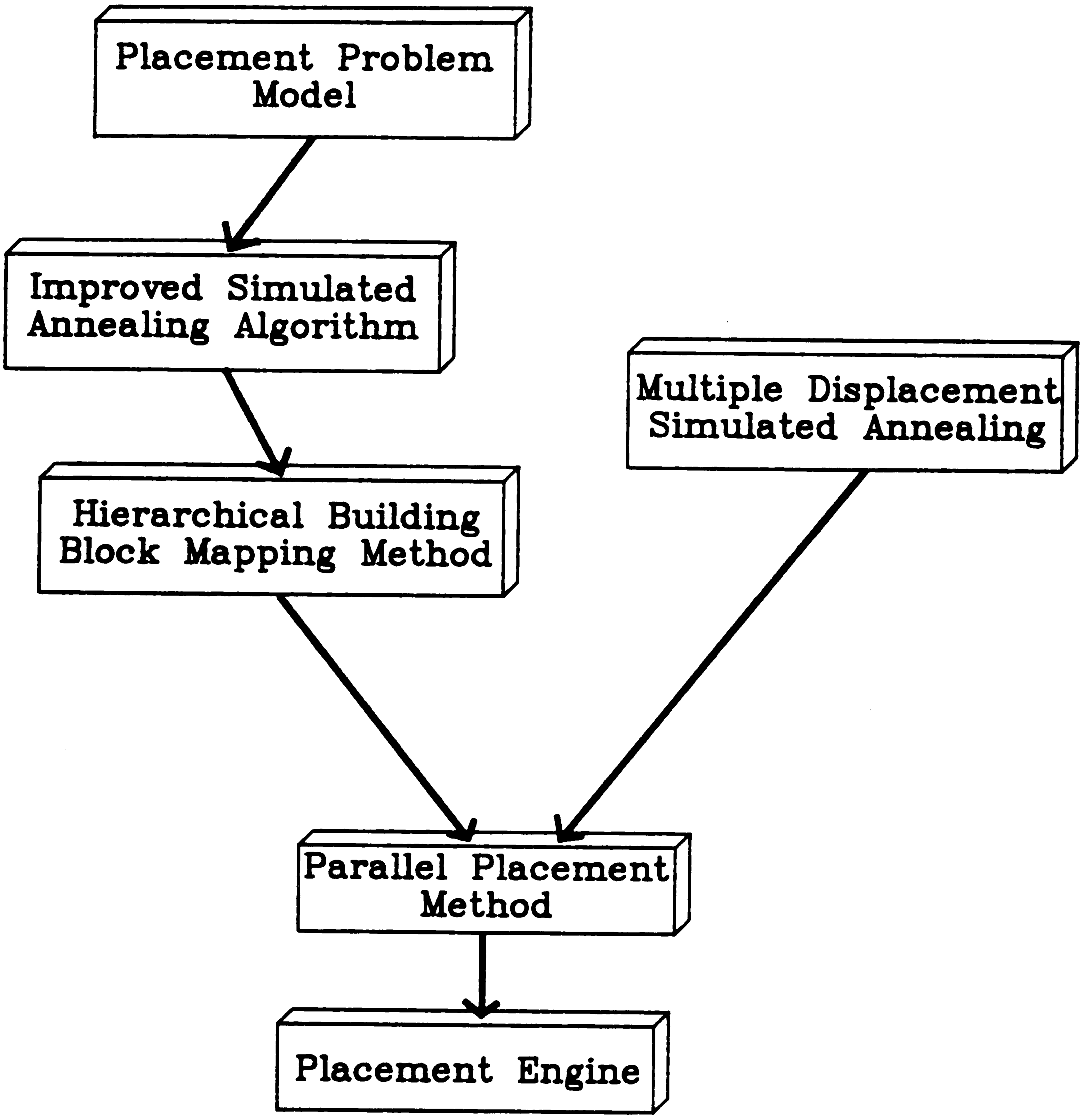

Placement Problem Model
Improved Simulated Annealing Algorithm
Multiple Displacement Simulated Annealing
Hierarchical Building Block Mapping Method
Parallel Placement Method
Placement Engine

Figure 1.1 The specific tasks performed in this research and their relationships.

provide a practical configuration.

To evaluate the performance of this new model, a simulated annealing placement algorithm using this model is developed and applied to test cases with different numbers of building blocks and connectivities. A program coded in the C language is used to implement this algorithm on a VAX 8600 operating under UNIX. The placement quality and the computation time of this algorithm are evaluated against a benchmark. The benchmark is established by a placement process using the concepts of the TimberWolf placement and routing package from the University of California at Berkeley VLSI design tools [7, 8].

A parallel placement method having multiple simulated annealing processes working concurrently on different regions of a chip is then developed. A scheme which hierarchically maps the building blocks of a placement problem to different regions of a chip is developed as a pre-process for this parallel placement method. A parallel environment is simulated to evaluate the performance of this placement method.

Finally, the concept of a hardware placement engine implementing the parallel version placement algorithm is developed. This architecture can be implemented as a VLSI structure and act as a coprocessor for a CAE workstation to handle the placement task.

1.3 Organization of This Dissertation

The remaining chapters of this dissertation are organized as follows. First, Chapter II provides the necessary background material related to this work, which addresses the importance of building block placement in VLSI design. The conventional placement methods, including the implementation of simulated annealing in the TimberWolf package, are also described in this chapter. The new model developed in this research for the placement problem and an improved simulated annealing placement algorithm implementing this model are then described in Chapter III. The concept of generating configurations with multiple block displacements in a simulated annealing process is discussed in Chapter IV, which eventually evolves into a parallel placement method applying concurrent simulated annealing. The architecture of a hardware placement engine specially designed for the parallel placement is proposed at the end of Chapter IV. Finally, Chapter V presents the test results of the approaches developed in this work and discusses the future trends in VLSI design automation and this research area.

CHAPTER II

BACKGROUND

The rapid evolution of the IC (Integrated Circuit) industry over the past two decades has created a new frontier for both electrical engineers and computer scientists. The potential of VLSI (Very Large Scale Integration) technology has not only stimulated the developments of many new algorithms and architectures, but has also provoked novel design concepts that are different from those used in traditional circuit design.

Only mass production could justify the high design cost of an integrated circuit at the beginning of the IC industry. The design history of two popular 16-bit microprocessors, the Motorola 68000 and the Intel 8086, which required 52 and 13 man-years of design effort, respectively, demonstrated the design complexity dilemma [14]. Few custom applications could afford a time-cost of this magnitude. Standard integrated circuits, due to their universal applications, have the advantage of sharing the one-time development cost among the large quantities of parts produced. Some of the standard integrated circuits, such as logic gates, have fixed functions and are commonly used as building modules in larger systems while others, such as microprocessors and programmable logic arrays, have fixed architectures but are programmable to perform different operations.

The recent advances in IC technology and the development of computer-aided design tools provide circuit or system designers with the opportunity of applying IC, or more specifically, VLSI technology, in their designs [15, 16]. It has been mentioned above that some of the standard IC's are programmable for specific applications, however, their universality and flexibility are often obtained at the cost of sacrificing performance in certain operations. It is now obvious that in order to fully utilize the potential of VLSI technology, the chip architecture must be designed specially for the intended application. Industry predictions are that ASIC (Application Specific Integrated Circuit) sales will be half of all IC sales by sometime in the 1990's [17]. The truth of this prediction relies on the development of powerful computerized VLSI design tools, the objective of which is to simplify the VLSI design task so that a fast turn-around time can be achieved with efficiently designed circuits.

An intimate relationship between engineering and computers has developed right from the dawn of IC technology. However, computers were primarily used for the storage of information and simulation through the 60's and early 70's. It was not until the mid 70's that CAE (Computer-Aided Engineering) became a necessity in the design of highly complex VLSI circuits [18, 19]. In the early 80's, CAE workstations with general IC layout capability emerged to cope with the increasing complexity of IC design [20]. However, many problems in the IC design process remain to be solved. For instance, the process of placing building blocks onto a chip requires hours or days of computation time to obtain a near-optimal result [7, 8]. This consideration has motivated the objectives pursued in this research.

Today, numerous automated or computer assisted VLSI design systems are available commercially. The following sections discuss the aspects of the state-of-the-art VLSI design processes with special attention paid to the building block placement problem and the physical layout stage.

2.1 Design Methodology

Design methodology is defined as a set of codified techniques that are applicable to the VLSI design process. Design functions of interest in VLSI design methodology can be categorized as follows [21]:

1. Chip specification and partitioning;
2. Chip design planning and initial implementation;
3. Subcircuit and module synthesis;
4. Simulation at different levels;
5. IC mask layout;
6. Design verification;
7. Testability in design and product;
8. Test sequence generation;
9. Database management;
10. Design documentation.

The ultimate objective of studying design methodology is to facilitate the creation of better designs in less time. The prevalent VLSI design methodology today, and in the foreseeable future, is hierarchical in nature [1, 2]. A set of universal circuits are designed, optimized, and stored in the library of a CAE system. The library circuits can be repeatedly accessed, modified, and used as building blocks to construct the desired system.

Figure 2.1 presents the overview of a typical VLSI design process. A top-down design flow is normally used to decompose the circuit under design into a network of smaller and simpler functional modules. Once a functional implementation strategy has been established, a bottom-up flow is used to complete the physical design of the chip.

The design process begins by converting an idea for a VLSI-based circuit into more concrete circuit specifications. An algorithm is developed to perform the required operations and a suitable architecture is designed to carry out the chosen algorithm. Simulations are used to verify the correctness and estimate the performance of both the algorithm and architecture. After the architecture of the circuit has been established, the design process enters the physical layout phase. Copies of the needed building blocks are fetched from the library, placed in some optimal manner, and interconnected as they will be on the chip. The layout is then simulated to verify its operation and performance with respect to the desired specifications. Finally layout mask information is sent to a silicon foundry for fabrication.

The structure and behavior of a chip must be completely described in a CAE environment. High level programming languages or Hardware Description Languages (HDLs) have been developed to describe circuits so as to facilitate the use of CAE equipment [22-26]. An ideal HDL should be able to describe all levels of chip behavior and structure. In practice, however, different HDL's are usually devised to effectively represent different levels of the design hierarchy. These representation methods must also interface with design operations such as placement and routing.

Three major design and fabrication approaches are commonly used in the design of a VLSI-based structure: gate array, standard cell, and full custom. (Gate array and standard cell approaches are sometimes referred to as semi-custom). The philosophy of the semi-custom approaches is to perform the design process as completely as possible before the design is customized for a specific application. The design process is highly simplified by the development of automatic semi-custom design tools. A fast turn-around time is also possible. Some degrees of design flexibility are, however,

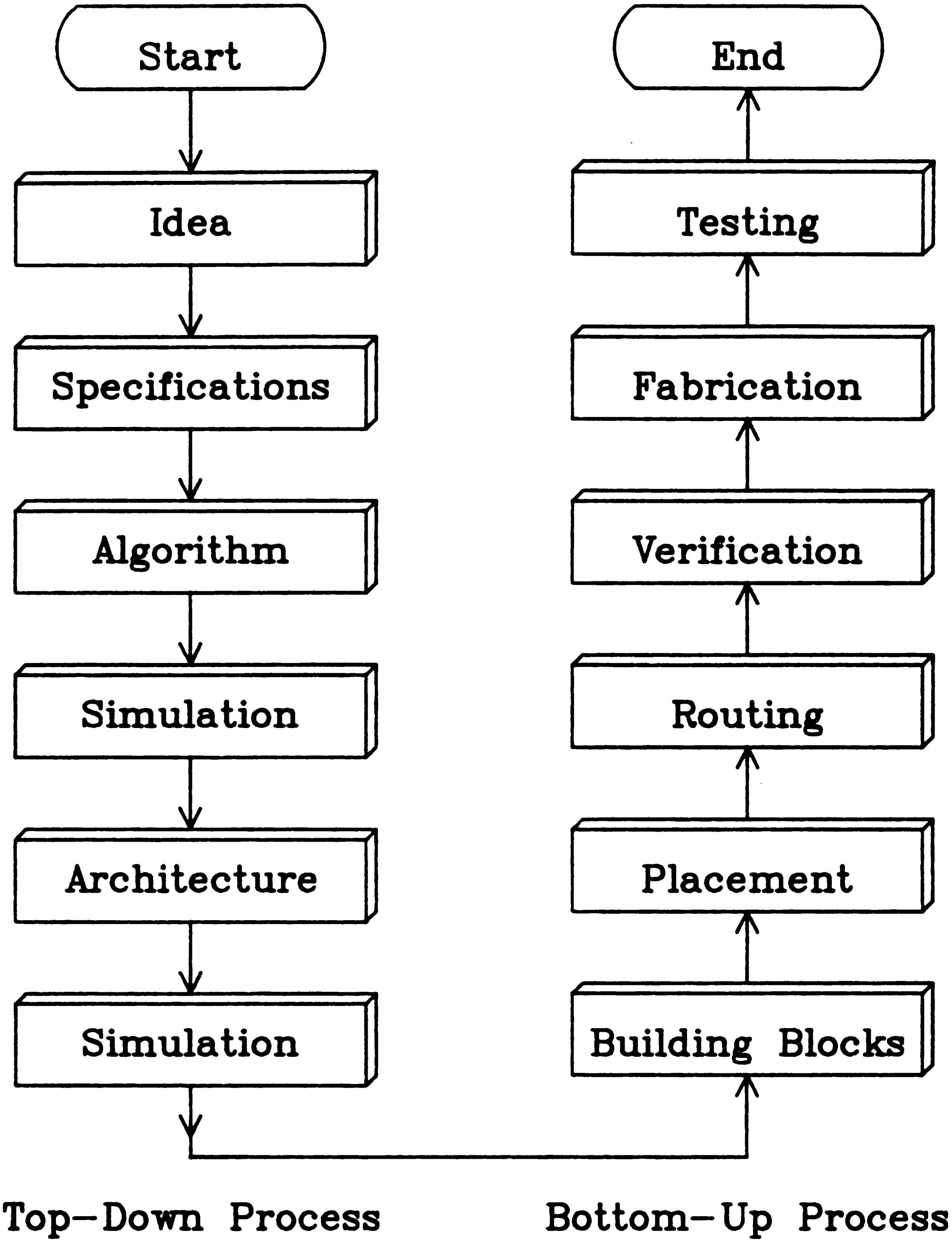


Figure 2.1 The general design process of a VLSI-based structure.

sacrificed to gain these advantages.

Prefabricated chips, containing cells of non-interconnected transistors and feed-throughs, are used in gate array designs. A typical floorplan of a gate array structure is presented in Figure 2.2. Cells of transistors are arranged in rows with feed-throughs inserted between these cells. The space between rows is called a channel which is provided for the interconnecting signal paths. The function of the feed-throughs is to allow interconnections between the signal paths in different channels.

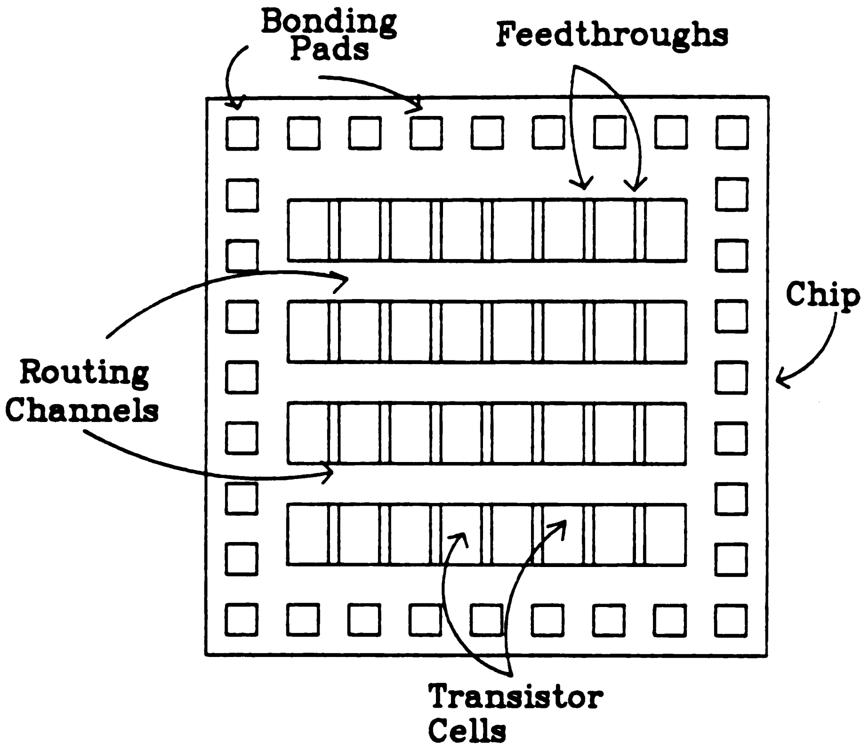


Figure 2.2 A typical floorplan of a gate array chip.

Macro-cells, built with the transistors on a gate array as basic components, are stored in a cell library of a CAE system. The layout procedure is simplified into fetching the desired macro-cells from the library and arranging them (placement) on the chip. The process is then completed by interconnecting these macro-cells.

Another semi-custom VLSI design approach is to use standard library cells. The major difference between the gate array and standard cell approaches is that the chips used in the gate array approach are pre-fabricated with unconnected transistors, while the chips for standard cell approach are fabricated anew for each design.

The layout task of a design using the standard cell approach also involves the placement and routing of building blocks, i.e., standard cells, fetched from a cell library. One distinct feature of the standard cells is that their heights are identical but their widths vary according to their complexities and functions. The identical heights of the standard cells allow them to be placed in rows by an appropriate placement algorithm. Channels are allocated between the rows for the signal paths that interconnect the standard cells to form the target circuit. A typical floorplan of the standard cell approach is shown in Figure 2.3.

Instead of using predesigned parts, the VLSI layout can also be done from scratch. A number of CAE tools have been provided for the layout of IC masks and simplified design rules have been developed to guide this type of low level layout process [16, 27, 28]. These design rules assure that the patterns generated are within the resolution of the fabrication process and that they do not violate the device physics required for the proper operation of transistors and interconnections formed by the process. Full custom design provides circuits with the best possible performance since the designer now has much more freedom in laying out the circuit. The design flexibility is gained at the cost of increased design complexity, and thus full custom design is only practical for circuits having a high degree of design regularity. An example of this type of circuit, a multiplier, which is constructed by an array of full adders is shown in Figure 2.4 [29]. The layout of the multiplier is simplified as it requires at the primary level the layout of a full adder, which serves as a basic building block.

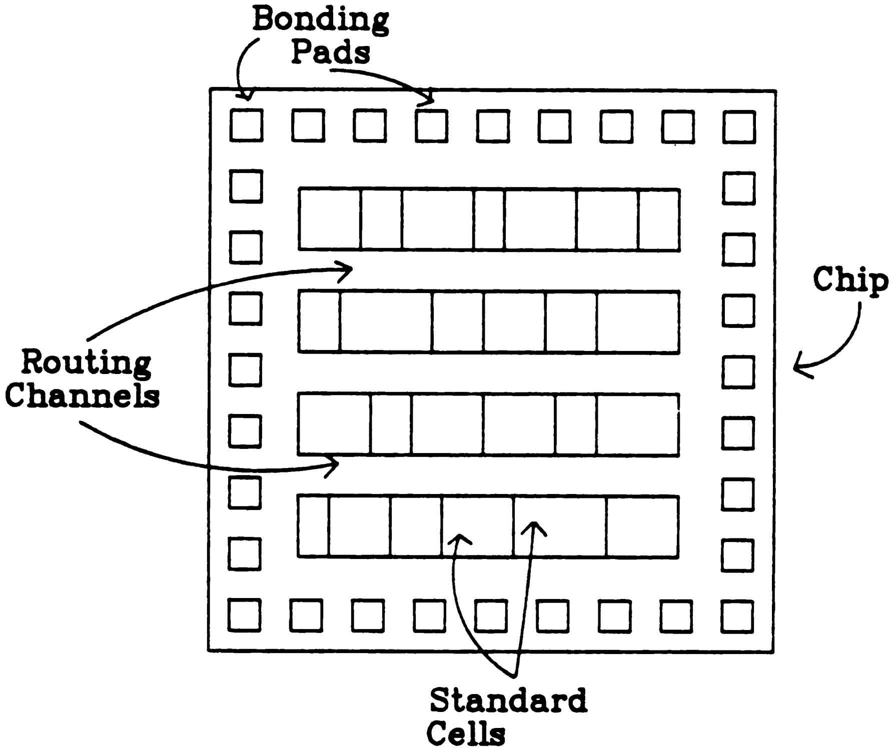


Figure 2.3 A typical VLSI floorplan using the standard cell approach.

The placement process for this circuit is straightforward since the entire architecture can be built by tesselating a few different types of modules with connections only to nearest neighbors.

One approach in between semi-custom and full custom is to construct the circuit by the placement and interconnection of mega-cells. This is similar to the standard cell approach in the sense that pre-designed library circuits are used to build the entire system hierarchically. However, the restriction on identical cell height is removed with mega-cells. Both the heights and widths of mega-cells can vary, even though the shape of a mega-cell is normally rectangular and arbitrary shapes are generally not allowed. The placement of mega-cells are no longer confined to rows or columns. An example layout of a chip using mega-cells is given in Figure 2.5, in which each

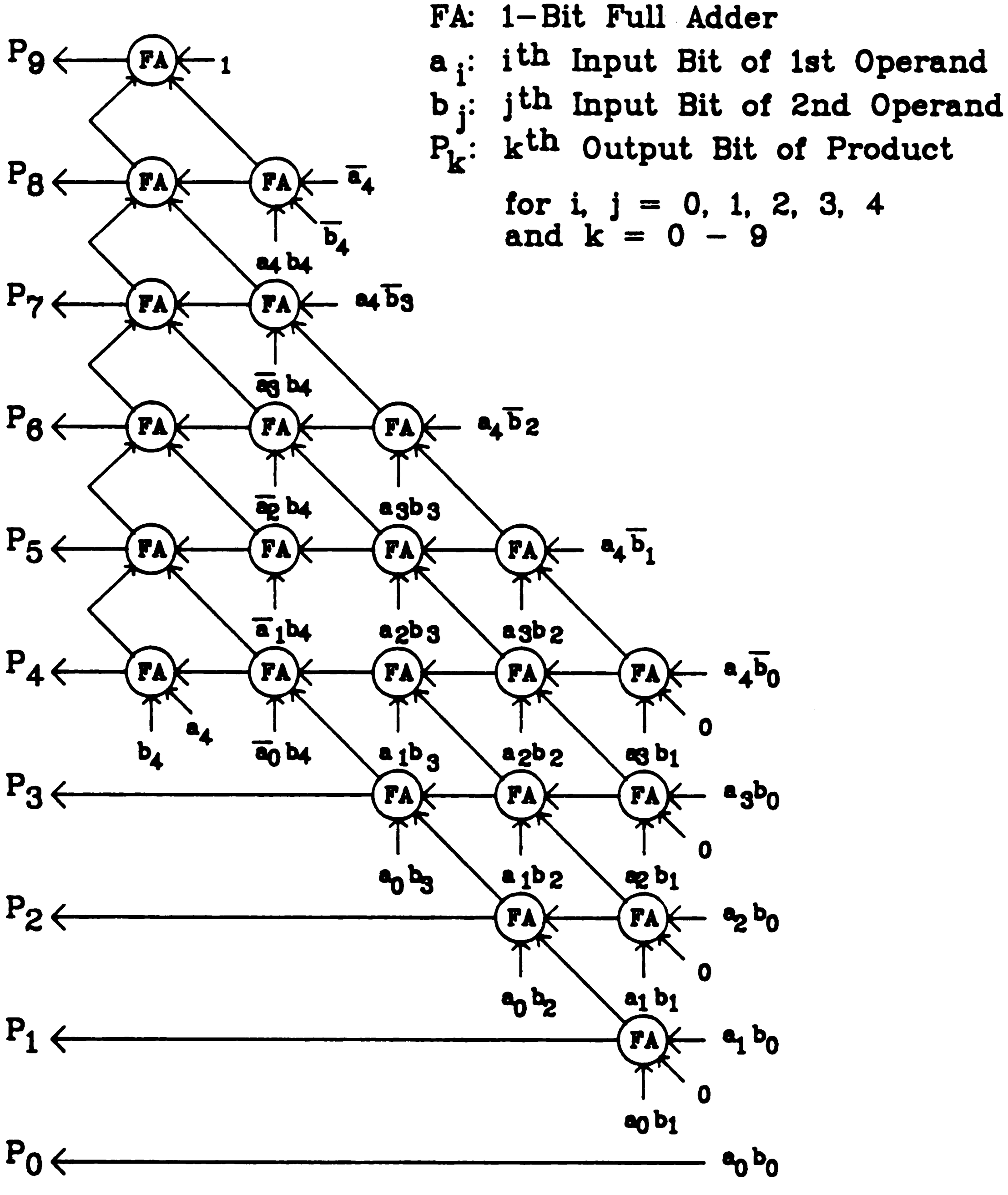


Figure 2.4 A multiplier constructed by an array of full adders [29].

rectangle (except for the bonding pads) is a mega-cell.

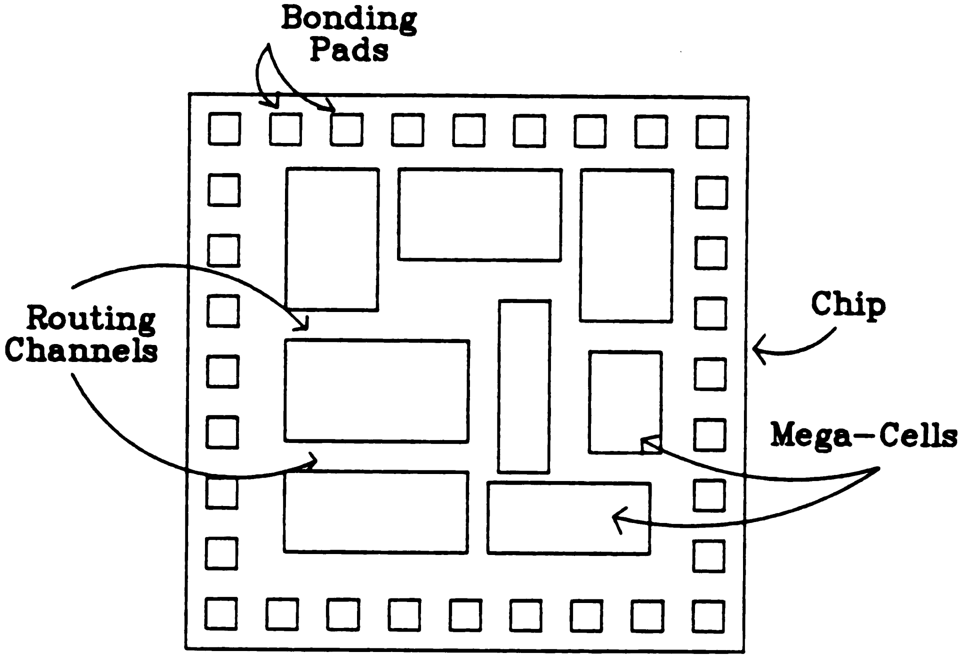


Figure 2.5 An example layout of a chip using mega-cells.

Since most of the restrictions on library cells are removed in this mega-cell design approach, the layout can be better optimized. But, the fact that more degrees of freedom are permitted in a mega-cell custom design makes the placement problem more difficult than that of other design approaches. The placement process for such mega-cells is of special interest in this work.

2.2 Building Block Placement in Design Automation

Most of the current design methods are based on the existence of a large set of universal library cells. Library cells can be the macro-cells in the gate array approach, the standard cells in the standard cell approach, the basic building blocks modules in a full custom design, or mega-cells. Copies of the necessary blocks for constructing the circuit are taken from the database library in the design process, and then must be placed and routed to ultimately construct the masks for the integrated circuit. An illustration demonstrating this layout concept is provided in Figure 2.6.

The information available at the placement stage is a set of functional blocks and their interconnections. The format of this information is, of course, dependent on the HDL used. However, the following information is generally essential for the placement process.

1. The physical information, such as the size, shape, and orientation, of each functional block is necessary for determining the locations of the building blocks. In addition, the lower bound of chip area required for implementing the circuit can be estimated by the total area of the building blocks.
2. The terminal information for each functional block describes the locations and functions of the terminals on the functional block boundaries. This information is useful since global routability is taken into consideration during the placement process.
3. An netlist is used to describe the interconnections between the terminals of the functional blocks. This information indicates the relationships between the blocks. Wiring distance, which is calculated using this information, is often used to evaluate the quality of a configuration.

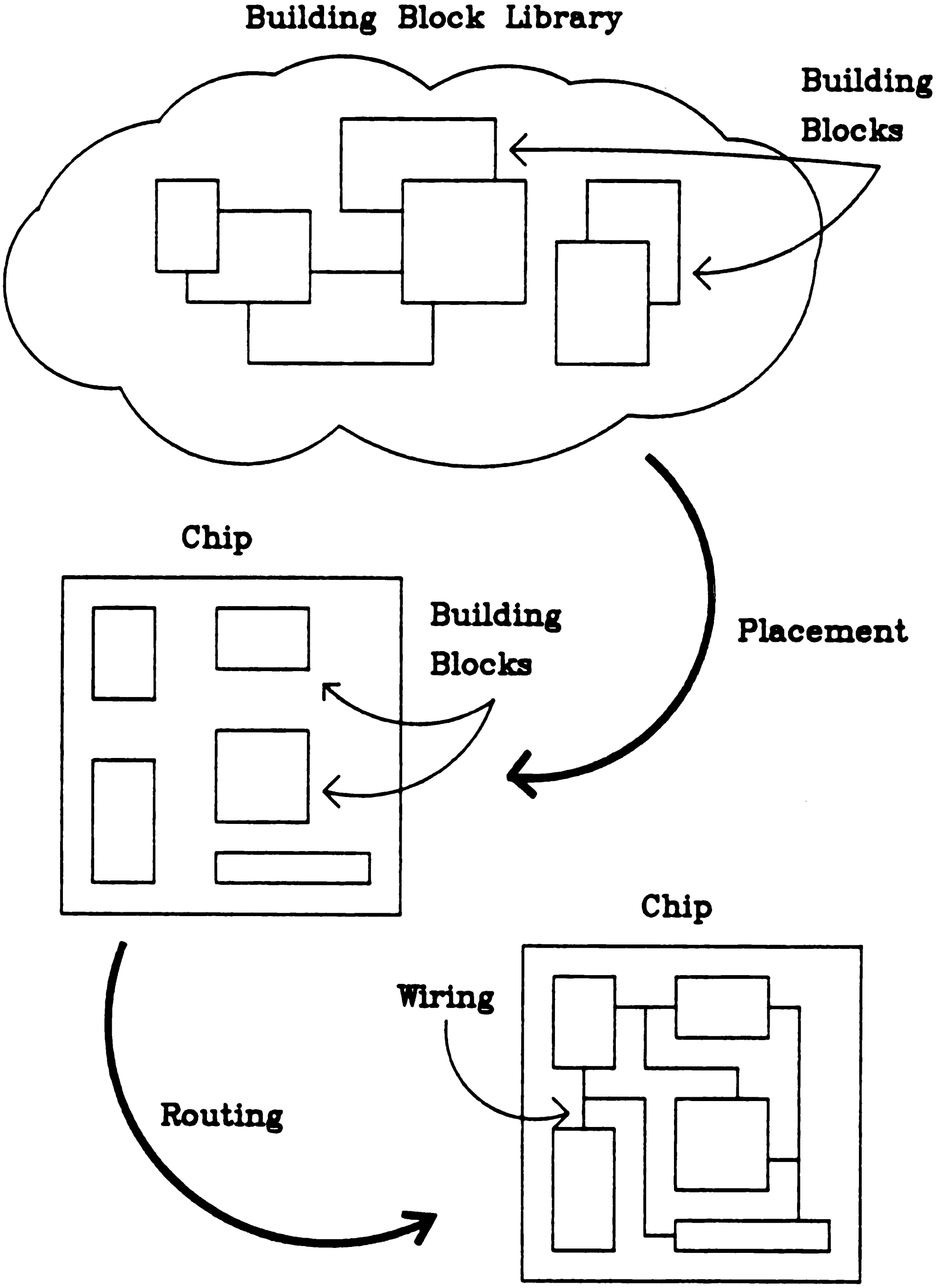


Figure 2.6 The concept of physical layout using building blocks from a database library.

The problem then is to place a set of building blocks within a limited area of the carrier, subject to various positional and electrical constraints. The ultimate goal of the placement process is to determine the optimal locations of the blocks fetched from the library on the chip so that those which are strongly connected are close to each other.

The placement problem did not originate in integrated circuit design and, in fact, has been studied with respect to many other applications long before the existence of VLSI technology. However, the problem is more difficult at the VLSI level due to the tremendous complexity of the circuit.

The task of finding a good placement configuration is not easy and it has been shown that the establishment of an optimal solution for a placement problem belongs to the class of combinatorial optimization problems and is NP-complete [4]. Therefore, heuristic methods are used to find a valid placement configuration in a finite time. Common heuristic algorithms solve the problem step-by-step and at each step they use an objective function to guide the selection of blocks to be placed or moved. The common criteria used for evaluating the objective function include wiring length, chip area, and the number of crossovers. Due to the fact that the delay time in a VLSI-based circuit depends heavily on the signal paths, the total wiring length is the most popular criterion for evaluating a placement configuration. Almost all the heuristic methods employ the branch and bound method to optimize the solution. The main difference between them lies in the particular search strategy and rules which are used to generate and evaluate a feasible solution.

Numerous placement algorithms have been proposed during the last decade [30-36]. Four representative methods that are conventionally used to solve the placement problem are described next. It should be noted that these methods and others are capable of finding a valid placement configuration for the design under consideration; however, there is no guarantee that the solution found is near-optimal or optimal.

2.2.1 Constructive Method

A constructive placement algorithm begins with the placement of one or a few "seed" building blocks, which are usually the critical components of the circuit under design. The other building blocks that are closely related to those placed are then sequentially added to the layout. Clusters of building blocks are established on the chip until all the components have been placed.

The reasoning behind this placement method is that by keeping the connected building blocks close to each other the wires between them are short and thus require as little area as possible. Problems arise when a building block is heavily connected to two or more other building blocks which have been placed at the opposite sides of a chip. The order of selecting building blocks to be placed is essential for the performance of this class of algorithms.

2.2.2 Min-Cut Method

Graph theory techniques are applied to the placement problem in the min-cut placement method [9, 35]. The building blocks and their interconnections are represented by nodes and edges as defined in graph theory. The circuit is then converted into a connected graph. The intention of min-cut placement is to emphasize a global placement strategy and to defer local considerations as long as possible. This is a top-down approach to placement rather than the bottom-up technique typically used by the constructive algorithms. The operation of this method is based on the partitioning of a connected graph. The goal of the partitioning is to generate two subgraphs such that the number of edges cut by this partitioning is minimal and that the difference between total block areas in the two subgraphs does not exceed a predefined threshold value. The cutting of the subgraphs is repeated until, at the end of the process, each subgraph contains only one node (building block). The relative locations of the building blocks are then established.

The main advantage of the min-cut method is that it considers the balance between block area in partitioned subgraphs. Unfortunately, this method can only locally minimize the count of net-cuts due to the sequential nature of the process. Moreover, the actual locations of the building blocks cannot be readily determined since, in addition to the area of the building blocks, their shapes, sizes, and terminal locations need to be considered to generate a practical configuration.

2.2.3 Force-Directed Method

The force-directed method, also called the attractive and repulsive method, applies Hooke's law from physics to the placement problem [35, 36]. Hooke's law states that if two particles are connected to each other by a spring, then there is an attractive force between these two particles that is equal to the spring constant times the distance between the two particles. The building blocks are treated as particles and the interconnections between blocks are considered as springs to generate an attractive force proportional to the connectivity between building blocks. In addition, repulsive forces are assumed to exist between blocks that are not directly connected. A system of particles, with forces interacting between them, thus represents the circuit. A set of simultaneous equations describes the relationships between the forces and the relative locations between the building blocks. These equations are solved, most commonly by numerical methods, to determine the relative locations of the building blocks.

The advantage of the force-directed method is that it provides a systematic approach to solve the placement problem. However, it is very time consuming to solve the set of simultaneous equations especially when the number of building blocks, and thus the number of variables, is large. Moreover, this method also suffers, as in the case of min-cut method, from ignorance of the actual sizes and shapes of the building blocks and their terminal locations.

2.2.4 Iterative Improvement Method

The iterative improvement method is of special interest in this research. As mentioned, the placement process is a combinatorial optimization problem involving a large solution space. Theoretically, the best solution exists under a certain well-defined cost function and can be guaranteed only by generating and evaluating all possible solutions (enumeration). However, the size of the solution space (all the possible configurations) in the placement problem is extremely large and it grows exponentially with the number of building blocks involved in the circuit. Due to the NP-complete nature of the placement process, it is impossible to perform an exhaustive search to locate the best solution in finite (reasonable) time.

Heuristics have been applied to the placement problem to find a good solution in a reasonable period of time. The iterative improvement method is comprised of two phases. An initial placement configuration is generated in the first phase by any constructive placement method or simply by randomly placing all the building blocks onto the chip. The initial placement configuration is then iteratively improved in the second phase of the process. A new placement configuration is generated at each step by introducing modifications to the present configuration. The modifications that are made to a placement configuration include:

1. the movement of a block to a new location;
2. the swapping of locations between two blocks;
3. the rotation of a block in multiples of 90^o;
4. the mirror imaging of a block.

A cost function is defined to guide the heuristic search in a direction that will improve the placement result. The delay time of the signal nets in a VLSI circuit plays an important role in the determination of its performance and thus the most common cost function used in this class of algorithms is the estimated total wiring length between the blocks. If the cost of a new configuration is lower than that of the previous one, the new one is accepted and further modifications are made to it. Otherwise, if the cost has increased in the new configuration, it is rejected and the previous one is retained for other modifications. The iterative improvement process continues until no further improvement can be obtained.

The idea behind the application of the heuristic search in building block placement is easy to understand. However, the design of a good heuristic search placement process is not trivial. The following summarizes four major considerations in the design of an iterative improvement placement algorithm:

1. The set of possible movements in the solution space must be rich enough so that all reasonable solutions can be found. The movements must be relatively inexpensive to compute since thousands or more of such movements will be made before a solution is found.
2. The cost function must be incrementally computable, so that the time to evaluate each move is minimal. Moreover, the cost function must be physically meaningful.

3. The search procedure must be planned carefully. The performance of this type of algorithm depends heavily on the quality of the initial placement. A good strategy for guiding the layout modification is necessary.

4. The data structure representing the system for iterative improvement must support the movements and evaluation of the cost function efficiently. The data structure will determine the speed of execution of the algorithm.

2.3 Simulated Annealing

Most heuristic algorithms search for a solution only in the directions that improve the objective/cost functions. One inherent drawback of this type of heuristic search is that it can be easily trapped into a local minima of the objective/cost function. The example presented in Figure 2.7 is used to demonstrate this problem.

Consider the curve shown in Figure 2.7 as the cost function of an iterative improvement process and the circles indicate the costs of certain configurations. Since a new configuration is generated by introducing small modifications to the placement, its corresponding location on the curve is most likely to be somewhere near that of the original configuration. The traditional iterative improvement algorithms only accept configurations that have reduced the cost. This criterion of configuration acceptance implies that the process can only go downhill into a local minimum and any uphill movement is forbidden. Thus, the search process cannot climb over the peak of the curve to reach the global minimum.

Recently, an approach called simulated annealing has been proposed as a method to find a near optimal solution for NP-complete combinatorial problems [6]. Simulated

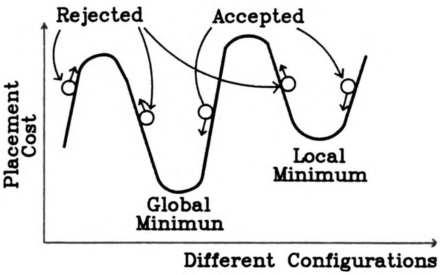


Figure 2.7 The local minimum traps of a cost function.

annealing associates the statistical mechanics which deal with the behavior of systems with many degrees of freedom in thermal equilibrium at a finite temperature, to combinatorial optimization which finds the minimum of a given function depending on many parameters. This new approach has found a major application in the building block placement problem [7, 8].

Annealing is defined as a process of slow cooling after subjecting a piece of metal to heat in order to prevent checking, cracking, and warping. When the metal is solidified from its liquid phase, low temperature is not a sufficient condition for achieving the lowest energy state. The metal must be cooled slowly to allow the large number of atoms in the material to settle in a stable manner. The likelihood of the piece of metal at a certain temperature being in a given energy state is governed by the Boltzman distribution for that temperature. As the temperature decreases, the distribution becomes concentrated on the lower energy states until, when the temperature finally approaches zero, only the minimum energy state has non-zero probability.

An analogy was noticed between the annealing process and the search of a combinatorial solution space [6]. The conventional iterative improvement strategy forbids the changes of states that increase the cost function. From the point of view of annealing, it is much like rapidly quenching a material to zero temperature. The local minima encountered in an iterative improvement process are analogous to the metastable states that dominate after rapid cooling.

An algorithm can be used to provide an effective simulation of a collection of atoms in equilibrium at a given temperature. In each step of this algorithm, an atom is given a small random displacement and the difference, $\Delta E(s_i)$, in the energy of the system between the present and previous states, s_i and s_{i-1}, is computed. If $\Delta E \leq 0$, the displacement is accepted as in the traditional heuristic search and the modified configuration is used in the next step. Cases in which $\Delta E > 0$ is treated using probability and accepted conditionally. The probability that the new configuration will be accepted is

$$P(s_i) = e^{-\Delta E(s_i)/T}, \tag{2.1}$$

where $P(s_i)$ is the probability of having a state s_i at a certain temperature, $\Delta E(s_i)$ is the energy difference between state s_i and previous state s_{i-1}, and T is the temperature. Random numbers, uniformly distributed in the interval 0 and 1, are convenient means of implementing the random part of the algorithm. One such number is generated and compared with $P(s_i)$. If it is less than $P(s_i)$, the new configuration is accepted; otherwise, the original configuration is retained.

A configuration has a higher probability of being accepted at higher temperature for the same cost increase. On the other hand, when the temperature is lower, the probability of accepting a cost increasing configuration is smaller and the system concentrates at configurations having lower cost. The same example cost function shown in Figure 2.7 is used to demonstrate this concept in Figure 2.8. The salient feature of simulated annealing is to allow the exploration of the solution space in a "wrong" direction, which worsens the cost function. The probability of accepting a move depends on the temperature applied at a certain simulated annealing stage. It has been proved that, under certain movement assumptions, the simulated annealing approach asymtotically produces the global optimal solution with probability one [7].

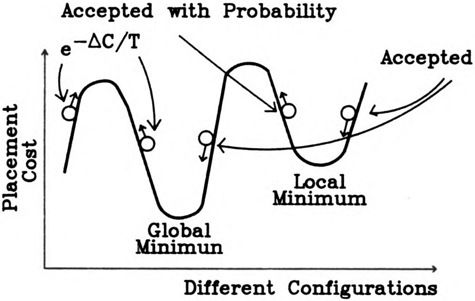


Figure 2.8 The hill climbing capability of a simulated annealing algorithm.

2.4 Application of Simulated Annealing to the Placement Process

Simulated annealing is an excellent way to attack the VLSI placement problem. An analogy between the placement problem and metal annealing is given in Table 2.1. In order to apply the concept of simulated annealing to the placement problem, a control parameter (pseudo-temperature) is introduced into the building block placement process.

Table 2.1 Analogy between building block placement and metal annealing problems.

VLSI Placement	Metal Annealing
Building blocks	Atoms
Movements, rotations, mirror imaging	Displacements
Cost function	Energy
Control parameter (pseudo-temperature)	Temperature

The placement process implemented in the TimberWolf placement and routing package is used here to discuss the application of simulated annealing [8]. The function provided in Figure 2.9 gives the general structure of the algorithm. Note that the important part of the algorithm is the function "accept" which is given in Figure 2.10.

A placement process using simulated annealing proceeds similar to the traditional iterative improvement methods except that the pseudo-temperature decreases very slowly from a large value during the process. The selection of new configurations is

```
Placement(x,c){
    /* configuration x and cost function c */
    while ("stopping criterion" is not satisfied){
        generate a new configuration x';
        evaluate c(x'); /* new cost */
        if (accept(c(x'),c(x)){
            /*
                accept returns 1 if an acceptance criterion
                has been satisfied and 0 otherwise
            */
            x = x';
        }
    }
}
```

Figure 2.9 The TimberWolf placement algorithm [8].

based on the following considerations:

1. A random number between one and the total number of blocks is generated. A block is selected according to this random number.

2. Experimental investigation has revealed that the ratio of single block displacements to block interchanges has a pronounced effect on the quality of the final placement. Experiments have revealed that a ratio of about 5 to 1 yields the best results. A second random number is selected between 1 and the number of blocks multiplied by this ratio.

3. If the two numbers selected both represent blocks, then the pair of blocks are interchanged to generate a new configuration.

4. If the second number selected does not represent a block, then the first number selected governs the generation of a new configuration. The block

```
accept(c(x'),c(x)){
    /*
      given the cost of a new configuration x' and of the
      previous configuration s, return 1 if the cost variation
      passes a test. T is a pseudo-temperature.
    */
    Δc = c(x') - c(x);
    if (Δc ≤ 0){
        return(1);
    }else{
        y = exp(-Δc / T);
        r = random(0,1);
        /*
          random(0,1) is a function which returns a pseudo
          random number between 0 and 1 (with uniform
          distribution).
        */
        if (r < y){
            return(1);
        }else{
            return(0);
        }
    }
}
```

Figure 2.10 The "accept" function in the TimberWolf placement algorithm [8].

indicated by the first number is moved to a new location.

A placement configuration is accepted if the cost is reduced as in the conventional heuristic methods. Cases in which the cost is increased are treated using the probability $e^{-\Delta C_i/T}$, where ΔC_i is the cost difference between the present and the previous configurations and T is the temperature. This probability is compared to a random number generated between 0 and 1. If the probability is larger than the random number, the new configuration is accepted. If the movement or interchange of building blocks is rejected, the orientation change of the first block selected is performed to generate yet another configuration. The acceptance of this configuration generated by changing the orientation of the first block is determined in the same manner as the block movement. At each temperature, an appropriate number of changes are applied to the configuration to simulate the slow cooling procedure. The number of moves evaluated at each temperature must be large enough so that every block has a fair chance of being chosen for movement thereby ensuring that all solutions in the problem space can be reached. Experiments have shown that more than 100 new configurations per building block are necessary to fully explore the solution space [7, 8]. A rule of thumb for decreasing the temperature is to use the formula

$$T_2 = \alpha T_1, \tag{2.2}$$

where T_1 and T_2 are the present and next temperatures, respectively, and α is a constant in the range of 0.8 to 0.95. In the current implementation of TimberWolf, the parameter α is changed during the placement process. The best results have been obtained when α is large (approximately 0.95) during the stages of the algorithm when the cost function is decreasing rapidly and α is given its lowest values at the initial and latter stages of the algorithm (usually 0.80). The value of α is gradually increased from its lowest value to its highest value, and then gradually decreased back to its lowest value. The process is stopped when the cost function's value has not changed for 4 consecutive stages.

This method has been shown to be very effective in the placement problem; however, the process is very time consuming since numerous moves must be generated at each temperature to simulate the "slow" cooling procedure. An example is used to demonstrate the magnitude of the computation complexity in this approach. Consider a simulated annealing process starting from $T = 300$. If the rate of decrease, α, is set as 0.9, there are more than 50 temperature steps before $T \leq 1$. Assuming that 100 new configurations are generated and evaluated at each temperature, more than 10^5 new configurations must be generated for a moderate design case with 100 building blocks.

2.5 Placement Problem Model

As long as a computer program is used to handle the building block placement problem, the effectiveness and efficiency of this operation is at least partially determined by how the problem is represented in memory. A minimum requirement on a problem model for building block placement is that the model itself must be easily manipulated by a computer program. In addition, the model must be able to represent all the necessary information about the building blocks and their interconnections to obtain a meaningful placement configuration.

2.5.1 Node-and-Edge Model

Many placement algorithms use nodes and edges to model the building blocks and their interconnections, respectively. Under this model, the circuit is converted from a network of interconnected building blocks into a connected graph. An example of applying this technique to represent a circuit is shown in Figure 2.11. The original circuit, presented as a network of interconnected functional blocks is converted into a connected graph.

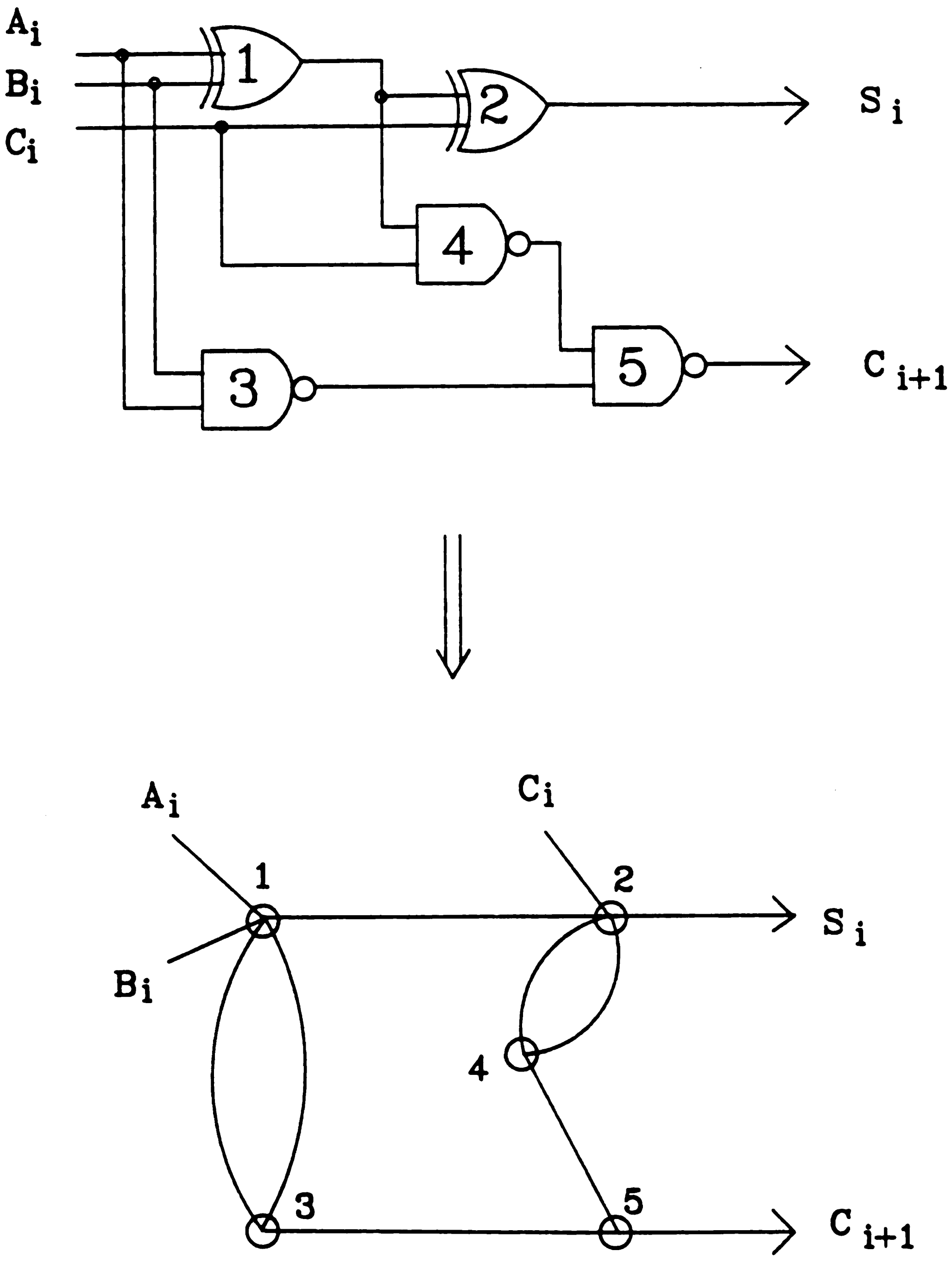


Figure 2.11 An example of representing a circuit by a connected graph.

The advantage of this node-and-edge model is that many techniques and principles from graph theory can readily be applied. The placement algorithms manipulate the connected graph algorithmically to produce a solution for the placement problem. Moreover, due to the simplicity of this model, it can be implemented easily in a computer program. However, there are some inherent drawbacks to this conventional node-and-edge model:

1. The shapes and sizes of the building blocks are totally ignored when they are converted into dimensionless nodes.
2. The orientations of the building blocks cannot be readily considered in an algorithm that utilizes this model since it is difficult to represent operations such as the rotation of a dimensionless node.
3. The locations of the building block terminals cannot be easily presented by dimensionless nodes.

Ignoring the shapes and sizes of building blocks is acceptable as long as all the building blocks are the same size and are allowed to be placed only in some predetermined fixed location. This is the case in a gate array design in which everything except the final metal interconnection is prefabricated. However, this is not true in the other design approaches. Without knowledge of the shapes and sizes of the building blocks, the algorithm can determine their relative locations, but the blocks often overlap when the nodes are changed back to their real shapes and sizes.

Even though the locations of the building blocks are determined, different configurations can still be generated without changing their locations. These different configurations are generated by the rotation and/or mirror-imaging of one or more building blocks. Figure 2.12 shows eight different orientations that a building block can assume. The circled vertex is the lower left corner of the original orientation.

The orientations of the building blocks and their terminal locations also affect the subsequent routing process. All possible orientations of the building blocks have to be considered in placement process to enhance routability. Without taking the routability of a placement configuration into consideration, it is impossible to generate a good placement result.

2.5.2 TimberWolf Model

The rudimentary node-and-edge model can, of course, be enhanced to carry all the desired information. The placement process of the TimberWolf package [7, 8] takes the sizes and terminal locations of the building blocks into consideration. Since the sizes of building blocks are carried in the model, different orientations of the building blocks can be considered in the placement algorithm. Also, the inclusion of terminal locations improves the routability of the final placement result. However, this enhanced version is not popular since the problems encountered in the node-and-edge model are only partially solved. Building blocks generally have different sizes implying that overlap is very likely to occur with the movement or rotation of nodes. In fact, if cells are allowed to overlap with each other they will be attracted to the same location in the final configuration since that obviously produces the shortest wiring length when all the interconnections are assumed to go on a two dimensional plane.

Overlap can be eliminated during the placement process in a number of ways. In the TimberWolf model, a new configuration is generated by either exchanging two blocks or moving a block to another location. Overlap in the TimberWolf package is dealt with by introducing a penalty function. When two building blocks overlap, a penalty is assessed which is proportional to the square of quantity of the amount of

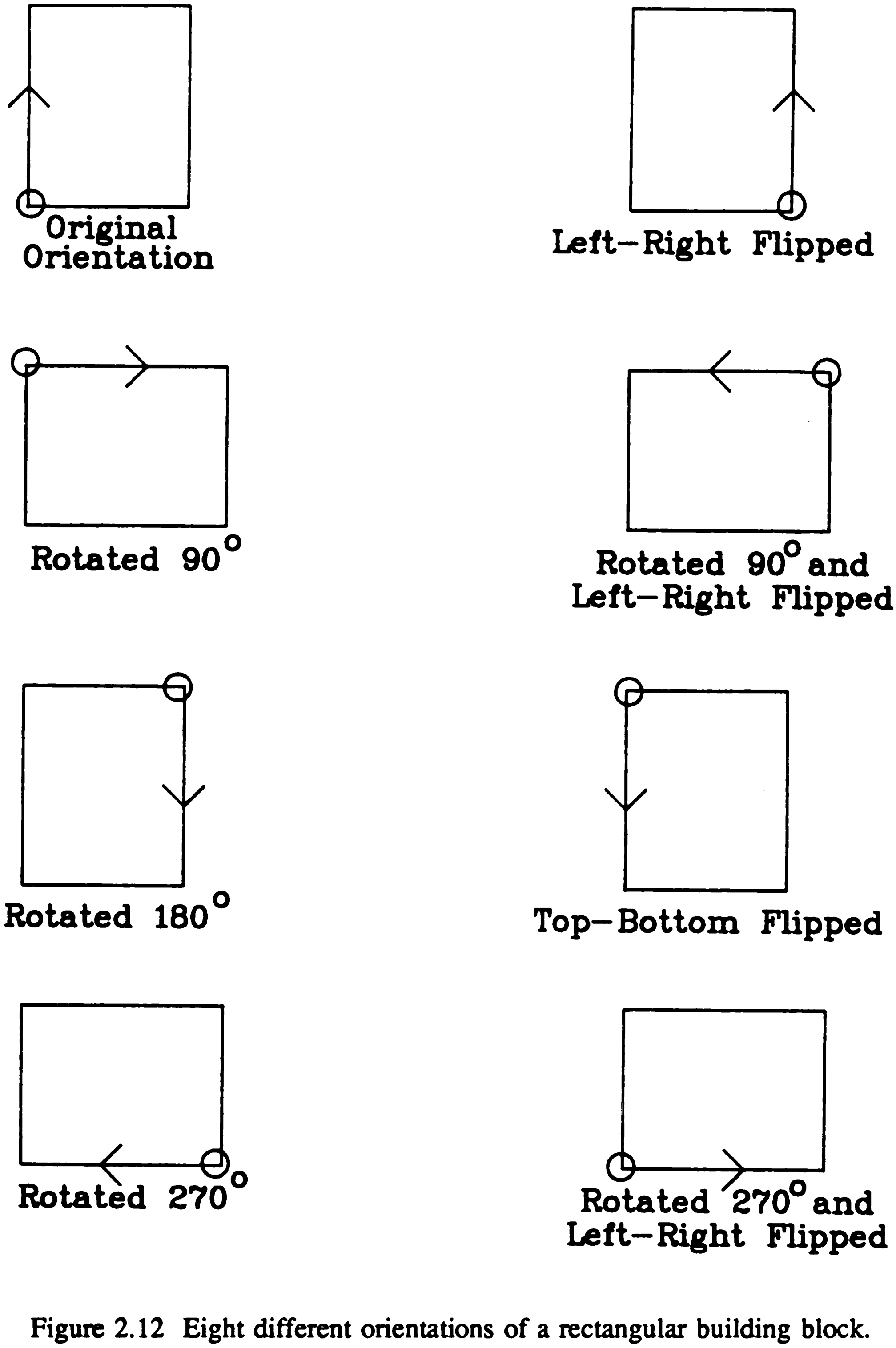


Figure 2.12 Eight different orientations of a rectangular building block.

overlap plus an offset parameter. The offset parameter is chosen to ensure that when the temperature approaches zero, then the total amount of overlap approaches zero. This method requires the tedious calculation of overlap area and the corresponding penalty term, making the already slow simulated annealing process even more sluggish.

Another way to avoid block overlap is to check whether or not there is sufficient space before a block is moved or rotated. When it is determined that there is not enough space for a certain operation, the operation is canceled or the blocks in the neighborhood can be moved apart to generate the necessary space. This check-before-move strategy is very time consuming and often results in inferior placement configurations. If the desired operation is canceled, the placement process is handicapped since the solutions that it can reach are limited. The area of the final placement is often oversized due to this limitation. Moreover, the strategy to generate necessary space by moving building blocks apart degrades the result by changing the determined block locations. Alternately, overlap can be eliminated after the process by expanding the final configuration. This method suffers the same problems of being time consuming and clearly reduces the placement quality.

2.6 Parallel Placement

Simulated annealing is an approach to the placement problem that has proven to be particularly successful. Unfortunately, the run-time of a placement process based on the simulated annealing algorithm is often unacceptable [10]. Due to the expanding complexity of VLSI circuits, and thus the increased number of building blocks considered in the placement process, a practical limit is introduced by any algorithm that attacks the entire problem space simultaneously. The philosophy of divide-and-conquer and its application in VLSI design have been discussed in the previous sections. It seems that a reasonable next step in the research of the placement problem is to apply this philosophy and divide a large problem into smaller more amiable problems. The need to partition a placement problem arises naturally from the application of parallel processing, but is is also advantageous, from the standpoint of computation speed, for a sequential process. If a placement algorithm for n blocks has a computation complexity of $O(n^2)$, quite common for the heuristic methods such as the iterative improvement approach, the computation time required by the process will be reduced after the problem is partitioned into k sub-problems. This consideration is summarized in the following equation.

$$\text{If } n = \sum_{i=1}^{k} n_i, \text{ then } \sum_{i=1}^{k} n_i^2 < (\sum_{i=1}^{k} n_i)^2. \tag{2.3}$$

Even though the partitioning of a placement problem has advantages with respect to computation speed, it has received little attention. This is because the quality of the partitioning result will be extremely important to the optimality of the final placement configuration. It is well-known that optimization at one level may increase complexity at all other levels. Some approaches for implementing the placement process using simulated annealing in a multiprocessor environment, however, have been proposed [10-13].

The key issue in the parallel implementation of a placement algorithm is its partitioning across communicating processors. Two approaches have been proposed to accelerate the simulated annealing algorithm [10, 11]. The first approach is simply to reduce the per-move computing time by dividing the task of computing a single move into several cooperating subtasks that execute in parallel on a multiprocessor. The second approach is to compute several complete moves in parallel.. No matter which of these approaches is applied, the annealing process is still characterized as a long sequence of accept/reject decisions, but in each time slot many moves are evaluated in parallel. The move decomposition and parallel move strategies are not mutually exclusive. With sufficient parallel resources, one could evaluate several moves in parallel and also divide each of these parallel moves into cooperating subtasks.

The work of a move consists of selecting a feasible block displacement, evaluating the cost change, deciding to accept or reject, and updating a database. Not all these activities, however, can proceed in parallel. The concept of "serializable subset" of moves is introduced in the parallel move evaluation approach. The moves evaluated in parallel must be serializable so that the moves which are accepted in parallel are not contradictory (e.g., moving the same block to two different locations). The simplest scheme to guarantee the set of serializable moves is to take all the rejected moves and also the accepted move found first, and abort all other accepted moves. The determination of the subsets of serializable moves is quite difficult since the algorithm has to guarantee that the configurations are reachable from each other in a finite number of steps, which is a necessary condition for the convergence of simulated annealing. This approach has a major intrinsic limitation. The greater the number of processors, the more difficult it is to find the serializable subsets [10].

Another approach is to partition the building blocks in a placement problem into groups and have each group annealed by a processor [12]. If the circuit under design comprises several independent subcircuits of which the number is smaller than or equal to the number of available processors, the partitioning will be rather straightforward. Unfortunately, this is often not the case in a practical design. In a realistic problem, simulated annealing must also be applied to the partitioning and is combined with the placement process. The clustering of the blocks is dynamically changed during the placement process by utilizing the concept of a center of mass. The blocks are imagined as particles of mass proportional to their area and their distances to the center of mass are used to calculate the cluster-cost. If a block is given by a processor to another processor that owns blocks which are in closer proximity then the cluster-cost decreases. In addition to heavy interprocessor communication traffic created by the movement of blocks from one processor to others, this approach is non-applicable if the partitioning has to be done before the placement process when no information about the locations of the blocks is available.

A special multiprocessor architecture has also been proposed to implement a parallel placement algorithm using the strategy of conventional iterative improvement [13]. An adjacent pairwise exchange method is applied in these placement methods. The proposed architecture is formed by a two-dimensional array of processors, each of which has direct connection to its immediate neighbor processors. An initial placement is generated by randomly assigning a building block to a processor. Non-intersecting pairs of neighboring blocks are selected for exchanges. Blocks are exchanged if the total wiring length associated with one pair of blocks can be decreased by the exchange. Exchanges are continued until the improvement rate becomes lower than a given value or until a given time has elapsed.

A unique problem of oscillation exists in this parallel pairwise exchange placement method. Oscillation refers to the phenomenon that a number of blocks perform endless exchanges of locations. The reason for oscillation is that erroneous decisions are possible when numerous pairs of modules are considered simultaneously for exchange. Because the algorithm cannot predict the movements of other blocks, it can only assume that they will not change locations. The decision to exchange block locations based on this assumption may turn out to be incorrect. Even though some special patterns for selecting the pairs of blocks to be exchanged have been suggested to reduce the adverse effect of oscillation, this method still suffers from the sequential broadcast of the new block locations [13]. Also, the approach cannot deal with placement problems which have more building blocks than the number of processors in the array.

CHAPTER III

PLACEMENT PROBLEM MODEL

The intimate relationship between computer-assisted tools and VLSI design has been discussed in the previous chapter. As the complexity of a VLSI circuit continues to grow, many operations in the design process, such as the building block placement process, cannot even be considered without the assistance of a computer. As long as a design task is handled by a computer program, the effectiveness and efficiency of this operation are partially dependent on how the task itself is represented.

Some conventional models for representing the building blocks and their interconnections in a placement problem have been discussed in Section 2.5. Their advantages and disadvantages have also been addressed in that section. A minimum requirement on a problem model for the placement process is that the model itself must be algorithmically manageable by a computer program. In addition, the model should be able to carry all the necessary information about the building blocks and their interconnections to facilitate the production of a meaningful placement result.

A new model for representing the building block placement problem in a CAE system is described in this chapter. This is followed by a discussion of the improvements made to the simulated annealing placement algorithm by applying this model. This model guarantees that the building blocks will not overlap during the placement process, thus saving the computation required for eliminating the overlap. Best of all, this model significantly reduces the solution space of the heuristic search in a placement process. The process can then converge to a near optimal solution more rapidly.

3.1 Solution Space of Building Block Placement Problem

A few observations can help to explain the time consuming feature of the simulated annealing placement process. It was shown in Section 2.4 that, numerous moves, most of which will be rejected, must be generated at each temperature stage to fully explore the solution space. For the same number of components, the size of the solution space is determined by the number of locations that a building block can assume on the chip.

In the design of a VLSI circuit, the gate array approach has the smallest solution space since all gate cells are prefabricated on the chip and have fixed locations. For instance, a cell can only be swapped with another cell and cannot be moved arbitrarily. The allowable changes to a placement configuration are quite limited and thus the solution space is small. The solution space of a standard cell placement is larger, since even though a cell can only be placed in a row with standard height, many rows and numerous locations at each row can be chosen. The solution space of a mega-cell custom design is the largest of these approaches. This is easily understandable since all the building blocks involved in the placement process have different heights and widths and they have complete freedom to be placed in any location on the chip.

In order to demonstrate the magnitude of the size of solution space for the placement problem, the number of possibilities for placing a building block with a size of 100 X 100 square units into an area of 1000 X 1000 square units with different design approaches are compared. Three different layout styles are considered in this comparison.

Case 1: The first layout style considered is a gate array like design. Slots, which determine the possible locations of the building blocks, are defined on a chip.

The changes that can be made to a configuration are restricted to moving a building block to an empty slot, swapping the locations of two building blocks, and the left or right mirror imaging of a building block. It is noted that, according to these rules of introducing modifications to a configuration, no overlap occurs between blocks when they are moved or flipped.

Case 2: In the second layout style, the chip is divided into rows with identical heights as in the standard cell design approach. The building blocks are restricted to be placed in the defined rows, but they can assume any location in a row as long as they do not cross the border of the chip. The only orientation change allowed in this case is the left or right mirror imaging of a building block. The modification of a configuration using this style generally results in some amount of overlap between building blocks.

Case 3: In the third style, building blocks have different sizes and shapes as in the mega-cell design approach. A building block can be placed at any location on the chip as long as it does not overlap the boundary of the chip. The building blocks are free to rotate by multiples of 90^o and to be mirror imaged. In addition to the possible block overlap that has to be dealt with, this design style creates the largest solution space.

The result of the solution space comparison for these cases is listed in Table 3.1. The data in Table 3.1 were calculated by considering the coordinates at which the "origin" (lower left corner) of a building block can be placed without overlapping the boundary of the defined chip area. The area required for the routing channels is not considered in the calculations; however, the possible orientation changes, such as rotating or flipping of the blocks, are taken into account.

Table 3.1. The possible placements of a block in three different design styles.

Design Style	Locations	Orientation Changes	Possible Placements	Overlap
Case 1	100	2	200	No
Case 2	9,000	2	18,000	Yes
Case 3	810,000	8	6,480,000	Yes

The second column in Table 3.1 lists the number of different locations that can be taken by a building block. The third column gives the number of allowable orientations that a building block can assume at a certain location. The next column is the product of the numbers given in columns 2 and 3, which indicates the number of possible placements for the building block when changes in both its location and orientation are taken into account. The possibility of having overlap between blocks is described in the last column. In order to fully explore this tremendous solution space, numerous different configurations must be generated and evaluated during the placement process. This is the main reason for the time consuming feature of the simulated annealing approach.

3.2 New Placement Problem Model

The discussion presented in Sections 2.5 and 3.1 suggest that the simulated annealing placement process is handicapped by two problems:

1. The large solution space of the placement problem; and
2. The avoidance or elimination of overlap between building blocks.

In view of these considerations, two strategies can be applied to accelerate the simulated annealing placement process: reducing the size of the problem space so that less configurations have to be generated at each temperature stage without affecting the coverage of the solutions and getting rid of the overlap area calculations. A new model for representing the circuit in a placement problem is developed. This model guarantees that overlap between building blocks does not occur in the placement process. This model also greatly reduces the solution space of the problem so that a good solution can be found more rapidly.

The VLSI design style of interest in this work is Case 3 as presented in Table 3.1, which is the most difficult one. It has been shown in Table 3.1 that, if a design style similar to the one discussed in Case 1 is followed, the size of the solution space can be reduced by several orders of magnitude. Moreover, no area overlap between building blocks will ever occur.

The chip, or in general, the carrier, is divided into an array of equally-sized square slots. The dimension of each slot is determined by the size of the largest building block in the design. The array is then used to guide the placement process. An example of a layout generated by this fixed slot model (FS model) is shown in Figure 3.1. A building block can only be placed into an empty square. In this manner, no overlap can occur when the blocks are moved, rotated, or flipped. The overhead of eliminating the overlap or evaluating the penalty term is thus totally removed. It is also very easy to show that the problem space of the heuristic search has been reduced by dividing the chip into an array of slots for the building blocks. Since the possible locations available for a block are significantly reduced, a comparable coverage of the solution space can be obtained with fewer new configurations generated.

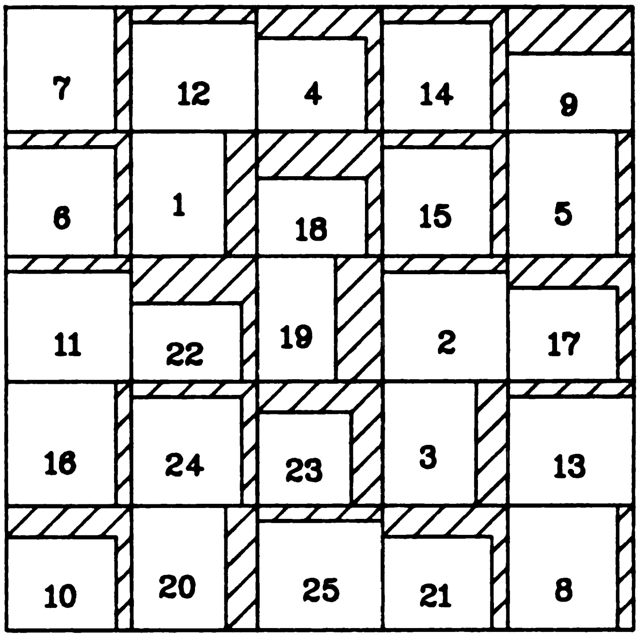


Figure 3.1 A placement configuration generated by using the FS model.

The original simulated annealing process described in Section 2.4 can be easily modified to apply this problem model. This modified algorithm (FS algorithm) is provided in Figure 3.2. The distinct feature of this algorithm is in providing an array of slots on the chip to guide the movement of the blocks.

This approach works well if the sizes of all the building blocks are approximately the same and their length to width ratios are close to unity. However, this is not

```
Place(){
    Calculate the largest dimension of the building blocks;
    /* the result is used to define the slots on the chip */
    Divide the chip into an array of slots;
    /* each of which is square and large enough to hold the largest block */
    Randomly place all the blocks onto the empty slots;
    While (stopping criterion is not satisfied)
        Rearrange the configuration by simulated annealing;
        /* modifications: moving a block to an empty slot,
                          swapping the locations of two blocks,
                          changing the orientation of a block */
}
```

Figure 3.2 The FS building block placement algorithm.

always the case in a practical placement problem. The argument provided above for the solution space is still valid, but since the dimension of the slots has to be determined by the largest building block, a slot holding a relatively small building block will have some wasted area. This situation is demonstrated in Figure 3.3 and the wasted area is indicated by the shaded regions. Building block No. 1 is the largest block in this placement example and thus its size is used to divide the chip area into an array of slots. When the relatively small building blocks, e.g., block Nos. 2 and 4, are placed into their slots, some area in the specific slots is wasted. The unusable area, of course, increases the lengths of the interconnections between these building blocks and others.

This problem is solved by a modification of the FS model. The relatively large building blocks are partitioned into connected submodules of approximately the same size. Weighted pseudo-interconnections are provided between the sibling submodules of the same parent building block. The chip is then divided into an array of slots according to the size of the submodules. An example of applying this fixed slot with

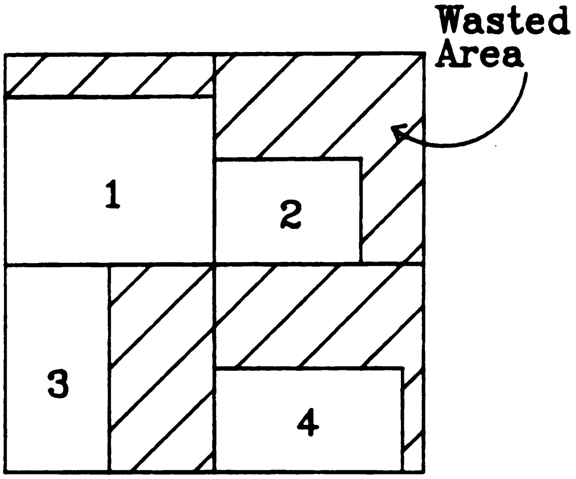


Figure 3.3 The chip area wasted in the FS model.

partitioned building block (FSWPB) model to the placement problem in Figure 3.3 is shown is Figure 3.4. The partitioning of the blocks is guided by the objective of minimizing the total amount of unused area within the slots when the submodules are placed into the array. In general, the average block size can be used for this purpose. The partitioning of building block No. 1 is illustrated in Figure 3.4. This large building block is partitioned into four submodules, indicated by 1a to 1d, with weighted pseudo-interconnections assigned between them. The chip area is redivided to generate slots that fit the size of the submodules.

The submodules generated by the partitioning are treated as independent blocks in the annealing process. The validity of this model depends on its capability of bringing the submodules of a building block together at the end of the process to eliminate the pseudo-interconnections. The cost of a configuration is calculated by the total estimated wiring length including those of the pseudo-interconnections multiplied by their respective weights. Since the pseudo-interconnections between the modules are weighted more heavily than the real interconnections in the original netlist, the change

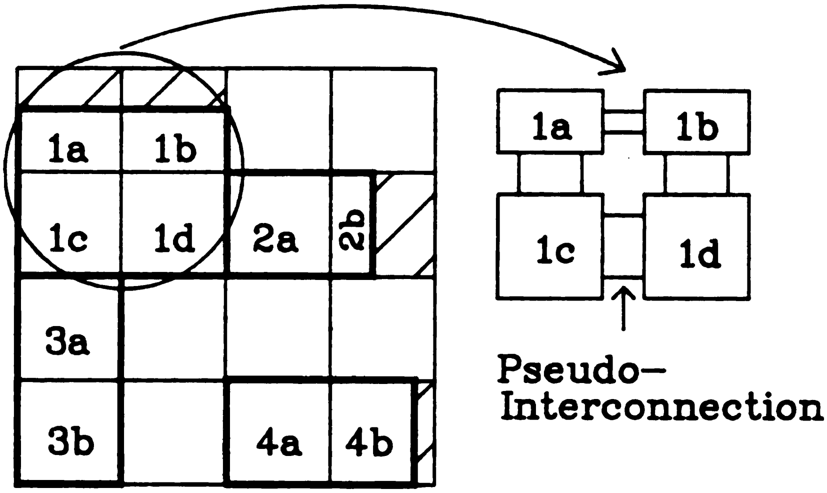


Figure 3.4 An example of applying the FSWPB placement model.

in the wiring length of a pseudo-interconnection will more strongly affect the cost of a configuration than the same change in a real interconnection. For example, the separation of two submodules belonging to the same parent will strongly increase the cost and will likely be rejected, especially at low temperatures. Also, the nearness of two submodules generated from the same parent will greatly improve the cost and will have a better chance of acceptance.

In order to guarantee that, at the end of the placement process, the submodules of the same parent are placed such that the pseudo-interconnections between them have zero wiring distances, the weight applied on the pseudo-interconnections is changed dynamically during the annealing process. The weight increases as the temperature decreases and thus their effect on the cost evaluation gets stronger as the process progresses. At the later stages of the annealing process, even a small displacement of a submodule can greatly affect the cost.

It has been shown that, under certain assumptions, only the optimal configuration determined by a well-defined cost function exists with probability 1 as the temperature approaches 0 in a simulated annealing process [7]. Since the weight on the pseudo-interconnections is inversely proportional to the temperature, the weight approaches ∞ as the temperature approaches 0. A small separation between two sibling submodules will cause the cost of the pseudo-interconnection between them to be enormous. Such a configuration is, of course, far from optimal and will not be permitted.

Another advantage of this placement model is its potential to allow building blocks with arbitrary Manhattan shapes to be used in the design process. The handling of some example Manhattan building blocks is given in Figure 3.5. A set of rectangular submodules can be generated by partitioning the building blocks.

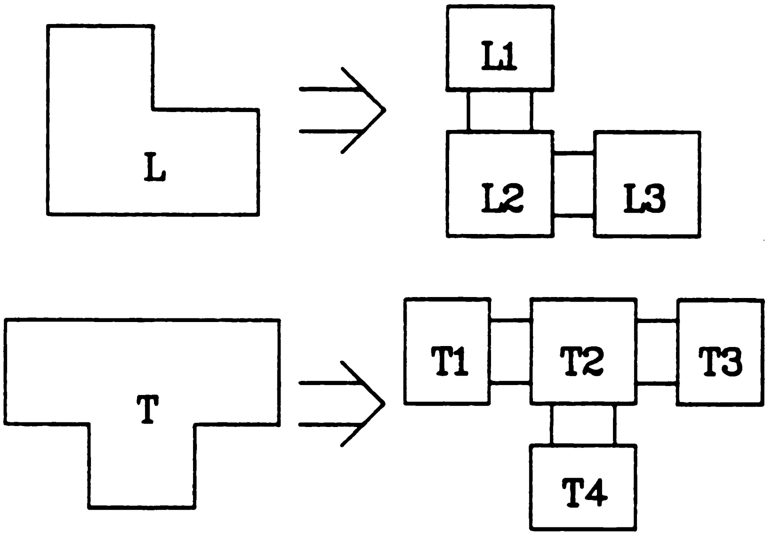


Figure 3.5 The partitioning of Manhattan building blocks.

The modified simulated annealing placement algorithm presented in Figure 3.2 is further improved to include the partitioning of large building blocks into interconnected submodules. This algorithm is described in Figure 3.6. The fact that a larger problem with increased numbers of blocks and connectivities is created after the partitioning of the building blocks into submodules has an adverse effect on the computation time required for the placement process. The placement process is divided into two stages to minimize this problem. The stopping criterion for the annealing process is temporarily ignored prior to a user-specified switching temperature which defines the boundary between the first and second stages. In the first stage, the FS model is applied to find the relative locations of the building blocks, the configuration of which is used as the initial placement for the second stage. In the second stage, relatively large blocks are partitioned into connected submodules which have sizes approximately equal to those of the relatively small blocks. Weighted interconnections between the sibling submodules are added to the original netlist. The chip is then redivided into slots that fit the newly defined submodules. The simulated annealing process then continues, with all the submodules considered independently for movement and rotation to generate new configurations, until the stopping criterion is satisfied.

3.3 Validity of the FSWPB Placement Problem Model

The principles of the FSWPB placement problem model and its application to the simulated annealing process were discussed in the previous section. The theoretical background of this model is expanded here to verify its validity. The necessary mathematical tools, adapted from stochastic processes for this purpose, are first discussed [37].

A stochastic process deals with a collection of random variables. That is, if the steps in the process are indicated by $\{s \in S\}$, where S is the index set, there exists a set of random variables $\{X_{s,} s \in S\}$. X_s is referred to as the state of the process at step s. When S is a countable set the stochastic process is said to be a discrete time

```
FSWPB(){
    Calculate the largest dimension of the building blocks;
    /* the result is used to define the slots on the chip */
    Divide the chip into an array of slots;
    /* each of which is square and large enough to hold the largest block */
    Randomly place all the blocks onto the empty slots;
    While (T > Switching Temperature)
        /* the Switching Temperature is specified by the designer */
        Rearrange the configuration by simulated annealing;
        /* modifications: moving a block to an empty slot,
                           swapping the locations of two blocks,
                           changing the orientation of a block */
    Calculate the average dimension of the building blocks;
    Partition the blocks into approximately equally-sized submodules;
    Generate weighted interconnections between submodules of the same parent;
    Redivide the chip array to fit the submodules;
    While (Stopping Criterion is not satisfied)
        Rearrange the configuration by simulated annealing;
        /* submodules are considered independently */
}
```

Figure 3.6 The FSWPB building block placement algorithm.

process. The state space of a stochastic process is defined as the set of all possible values that the random variables X_s can assume. Thus, a stochastic process is a family of random states that can be used to describe the evolution through the time of the process. Suppose that whenever the process is in state i, there is a fixed probability P_{ij} that it will next be in state j. This condition can be expressed as

$$P(X_{s+1} = j | X_s = i, X_{s-1} = i_{s-1}, \dots, X_1 = i_1, X_0 = i_0) = P_{ij} \qquad (3.1)$$

for all states $i_0, i_1, \dots, i_{s-1}, i, j$ and all $s \geq 0$. Such a stochastic process is known as a Markov chain. Equation (3.1) may be interpreted as stating that, for a Markov chain, the conditional distribution of any future state X_{s+1}, given the past states $X_0, X_1, \dots, X_{s-1}$ and the present state X_s, is independent of the past states and depends only on the present state.

The value P_{ij} represents the probability that the process will, when in state i, next make a transition into state j. Since probabilities are nonnegative and the process must make a transition into some state, it follows that

$$P_{ij} \geq 0,\ i, j = 0,\ 1, \ldots,\ |S|, \tag{3.2a}$$

and

$$\sum_{j=0}^{|S|} P_{ij} = 1,\ i = 0,\ 1, \ldots,\ |S|. \tag{3.2b}$$

There exists a limiting probability that a Markov chain process will be in state j after a large number of transitions. This value is independent of the initial state. Some properties of the states of a Markov chain need to be considered in order to define them more precisely. State j is said to be *accessible* from state i if $P_{ij}^{n} > 0$ for some $n \geq 0$. This implies that state j is accessible from state i if and only if, starting in i, it is possible that the process will ever enter state j. Two states i and j that are accessible to each other are said to *communicate*. Two states that communicate are said to be in the same *class*. Any two classes of states are either identical or disjointed. A Markov chain is said to be *irreducible* if there is only one class, i.e., if all states communicate with each other. State i is defined as a *recurrent* state if the probability that, starting in state i, the process will ever reenter state i is 1. A recurrent state i has *period d* if $P_{ii}^{n} = 0$ whenever n is not divisible by d and d is the largest integer with this property. A state with period 1 is called *aperiodic*. If state i is recurrent, then it is said to be *positive recurrent* if, starting in i, the expected time until the process returns to state i is finite. Positive recurrent, aperiodic states are called *ergodic*. If n is large enough, P_{ij}^{n}, the probability of being at the n-th iteration in state j, starting from state i, depends only on the state itself and is totally independent on the initial state j. This important result is the backbone of the simulated annealing approach and is described in the following theorem [37].

Theorem 3.1: For an irreducible ergodic Markov chain $\lim_{n \rightarrow \infty} P^{n}_{ij}$ exists and is independent of i. Furthermore, if

$$\pi_j = \lim_{n} \rightarrow \infty \; P^{n}_{ij}, \quad j \in S \tag{3.3}$$

then π_j is the unique nonnegative solution of

$$\pi_j = \sum_{i=0}^{|S|} \pi_i P_{ij}, \quad j \in S \tag{3.4}$$

and

$$\sum_{j=0}^{|S|} \pi_j = 1. \tag{3.5}$$

Proof: Given that $\pi_j = \lim_{n \rightarrow \infty} P^{n}_{ij}$ exists and is independent of the initial state i, it can be seen (heuristically) that the π's must satisfy Equation (3.5). Now, derive an expression for $P\{X_{n+1} = j\}$ by conditioning on the state at time n. That is,

$$P\{X_{n+1} = j\} = \sum_{i=0}^{|S|} P\{X_{n+1} = j | X_n = i\} P\{X_n = i\} = \sum_{i=0}^{|S|} P_{ij} P\{X_n = i\} \tag{3.6}$$

Letting $n \rightarrow \infty$, and assuming that the limit can be brought inside the summation, equation (3.6) leads to equation (3.4) which is restated below

$$\pi_j = \sum_{i=0}^{|S|} \pi_i P_{ij}. \tag{3.7}$$

The set $\{\pi_i, i = 1,\ldots,|S|\}$ determined by Theorem 3.1 is called the *stationary probability distribution* of the Markov chain. The stationary probability distribution is very important since it completely characterizes the asymptotic behavior of a Markov chain.

Markov chains can be used as mathematical tools to describe the FSWPB placement algorithm; however, the above developments cannot be applied directly. In fact, these results in general are valid for transition probabilities that are independent of time. Note that all the probabilities defined in the annealing process depend on the pseudo-temperature T which is updated during the evolution of the process and hence is dependent on time. However, when the temperature is kept constant, i.e., in the inner loop of the annealing process, the transition probabilities are constant and the associated Markov chain is stationary.

The FSWPB strategy is that the submodules from the same parent will come together, thereby eliminating the lengths of their pseudo-interconnections at the end of the simulated annealing process. To prove this it must be shown that the process will generate, asymptotically with probability one, a global optimal solution for the placement problem. This goal is stated in Theorem 3.2.

Theorem 3.2: If the simulated annealing process applying FSWPB model produces asymptotically with probability one a global optimal solution for the placement problem, then

$$\lim_{T \to 0} \left[\sum_{i=1}^{l} L_i \right] = 0, \tag{3.8}$$

where T is the pseudo-temperature of the annealing process, L_i is the length of a pseudo-interconnection indicated by i, and l is the number of pseudo-interconnections created in the FSWPB model.

Proof: Based on the proposed strategy for dynamically changing the weights on the pseudo-interconnections, when T approaches 0 the weight applied to the pseudo-interconnections approaches ∞. Any configuration having a pseudo-interconnection with non-zero length will therefore not qualify as an optimal configuration and the probability of its existence at the end of the process is zero. This implies equation (3.8) and completes the proof.

The remaining task in developing the mathematical model for the FSWPB placement model is to show that the simulated annealing process as applied to this model produces asymptotically with probability one a global optimal solution for the placement problem. The following discussion applies to all non-zero temperatures in the annealing process. The strategies of generating new configurations in the placement process ensures that for each pair of configurations, say i, j, there must be integers $m, n \geq 0$, so that $P_{ij}^{n} \neq 0$, and $P_{ji}^{m} \neq 0$. In other words, the Markov chain induced by the FSWPB algorithm is irreducible. The condition for making the Markov chain aperiodic is satisfied by the acceptance rule in the FSWPB algorithm since there is always a state i, i.e., the optimum solution for which $P_{ii} > 0$. The Markov chain associated with the FSWPB algorithm is obviously positively recurrent since it is finite. So, the Markov chain is irreducible and ergodic. According to Theorem 3.1, the placement process with the set of configurations (states), S, described by this Markov chain thus has a stationary probability distribution, $\{\pi_i(T), i = 1, ..., |S|\}$. Theorem 3.3 and Corollary 3.3.1 prove that the probability of having a global optimal placement configuration at the end of the process is 1.

Theorem 3.3: If the set of global optimal configurations is indicated by I, then the simulated annealing process utilizing the FSWPB model produces asymptotically with probability $\frac{1}{|I|}$ a certain global optimal configuration when the pseudo-temperature approaches 0.

Proof: According to the principle of simulated annealing, at a certain temperature, T, the probability of the system being in a configuration i with cost $c(i)$ is described by the Boltzman distribution $e^{-c(i)/T}$. Let $\pi_i(T)$, for all $i \in S$, be defined according to the general simulated annealing approach by

$$\pi_i(T) = \frac{e^{-c(i)/T}}{\sum_{j \in S} e^{-c(j)/T}} \quad \text{for all } T,\ i \in S. \tag{3.9}$$

For the set of global optimal configurations I,

$$\pi_{i'}(T) = \frac{1}{|I| + \sum_{j \in (S-I)} e^{-(c(j)-c(i'))/T}} \quad \text{for all } i' \in I. \tag{3.10}$$

For all $i' \in I$ and $j \in (S-I)$, $(c(j) - c(i')) > 0$ and hence $\sum_{j \in (S-I)} e^{-(c(j)-c(i'))/T}$ approaches 0 when T approaches 0, so

$$\lim_{T \to 0} \pi_{i'}(T) = \frac{1}{|I|}, \quad i' \in I. \tag{3.11a}$$

If the same reasoning is applied for a state $j \in (S - I)$, at least one element of the sum becomes ∞ as T approaches 0. As such,

$$\lim_{T \to 0} \pi_j(T) = 0, \quad j \in (S-I). \tag{3.11b}$$

This completes the proof of the Theorem.

Corollary 3.3.1: The probability of producing an optimal configuration for the placement problem is 1 when the temperature approaches 0 at the end of the FSWPB algorithm.

Proof: This result is trivial when equation (3.11a) is applied. The probability of having an optimal placement configuration when the temperature approaches 0 is $\sum_{\check{i} \in I} \frac{1}{|I|} = 1$.

The last consideration reveals that results obtained in Theorem 3.3 and Corollary 3.3.1 require the algorithm to perform an infinite number of iterations in the inner loop which is not practical. However, if the pseudo-temperature, *T*, is decreased slowly enough so that $\pi_i(T)$ can be treated as a continuous function, then the continuity of $\pi_i(T)$ implies that the stationary probability distribution for a particular value of temperature, *T*, is a good approximation for the stationary probability distribution for all the values of temperatures sufficiently close to *T*. The method of generating the next temperature in the annealing process by multiplying the present temperature by a constant between 0 and 1 meets this requirement.

In conclusion, the submodules of a building block will come together to give its location with probability 1 at the end of the simulated annealing process. The validity of the FSWPB model is thus proved.

CHAPTER IV

PARALLEL PROCESSING OF BUILDING BLOCK PLACEMENT

With the great advances in VLSI technology, it has become cost-effective to include a large number of general purpose processors in one computing system. As a result, parallel algorithm implementation is increasingly feasible and its applicability to VLSI design automation has recently received more and more attention. One of the main reasons is that many problems in this field, such as the building block placement problem, are combinatorial in nature. Solving these problems requires so much computation time that a sequential process is often not sufficient. More advanced technology can, of course, be used to build a faster machine; however, this approach is beyond the scope of this research. Implementation of improved placement methods on a sequential machine were discussed in the previous chapter. The two aspects considered in this chapter to further accelerate the placement process are parallel processing and a hardware accelerator which implements the parallel version of the placement method.

The concept of parallel processing is not new. There are many examples where powerful achievements have been accomplished by a collective endeavor. Parallel processing can be defined as the creation and operation of multiple processes, executed concurrently in more than one processing unit. The ultimate objective of parallel processing is to improve the computation time by an amount proportional to the number of processors. This must be done within the architectural constraints and involves trade-offs in computation time, memory space, and communications.

Limits do exist when the power of parallel processing is applied to combinatorial optimization problems [38]. The NP-complete characteristic of a placement algorithm cannot be changed by the application of parallel processing. If the placement algorithm has an exponential complexity of computation and is not polynomially solvable by a sequential computer then the advantage of parallel processing is insignificant unless an exponential number of processors are used. This is, of course, impractical for large problems. The computational efficiency of solving the placement problem can be improved in a parallel processing environment; however, the solvable problem space cannot be expanded. Approximate solutions will still serve as the goal of a parallel placement algorithm so that it can reach a good solution in a reasonable amount of time.

The approach of generating a configuration by making multiple (parallel) block displacements in a simulated annealing process is first discussed. The results obtained from this study evolve into a parallel placement method applying concurrent simulated annealing processes on different "regions" of a chip. An explicit mapping scheme is developed to hierarchically map the building blocks into the different regions. Each of the chip regions can be handled independently by a simulated annealing process. The parallelism in this placement method is also discussed. A hardware accelerator implementing the parallel placement method is proposed as the conclusion of this chapter.

4.1 Simulated Annealing Placement with Multiple Displacements

Three exclusive types of parallel processing schemes are currently in use: pipelining, array processing, and multiprocessing [39]. Different forms of parallel processing are emphasized in these schemes. Temporal parallelism is exploited by overlapping computations in pipeline computing. Multiple processing units operating under the same instruction stream are used in array processing to achieve spatial parallelism. The multiprocessing schemes have multiple instruction streams over a set of interactive processors with shared resources such as memories.

The term "block displacement" appearing in the discussion to follow is intended to cover both the rotation and movement of a block. A simulated annealing process making multiple displacements to generate a new configuration was studied. The rationale behind this experiment is based on the observation that only one or two blocks are displaced to produce a new configuration in the original simulated annealing process, while a large number of atoms are displaced simultaneously in the metal annealing process. Therefore, in order to produce a more realistic simulation of the metal annealing process, a large number of building blocks should be moved or rotated between two consecutive configurations. A sequential machine is very inefficient in simulating this effect since all the displacements must be done sequentially and if a multiple displacement is rejected, many time slots will have been wasted.

Intuitively, multiple block displacements can be made to the layout simultaneously in one time slot by a parallel machine. However, the significance of this approach is its ability to produce new configurations in a simulated annealing process. In order to discuss its effect, the concept of utilizing a two-dimensional processor array to represent the chip area is introduced in Figure 4.1. Each node processor in the array is provided with basic arithmetic/logic functions and a local memory. Communication paths are provided between a node processor and its immediate neighbors for interprocessor communications. A master controller is connected to all the node processors through a global bus for control and communication purposes.

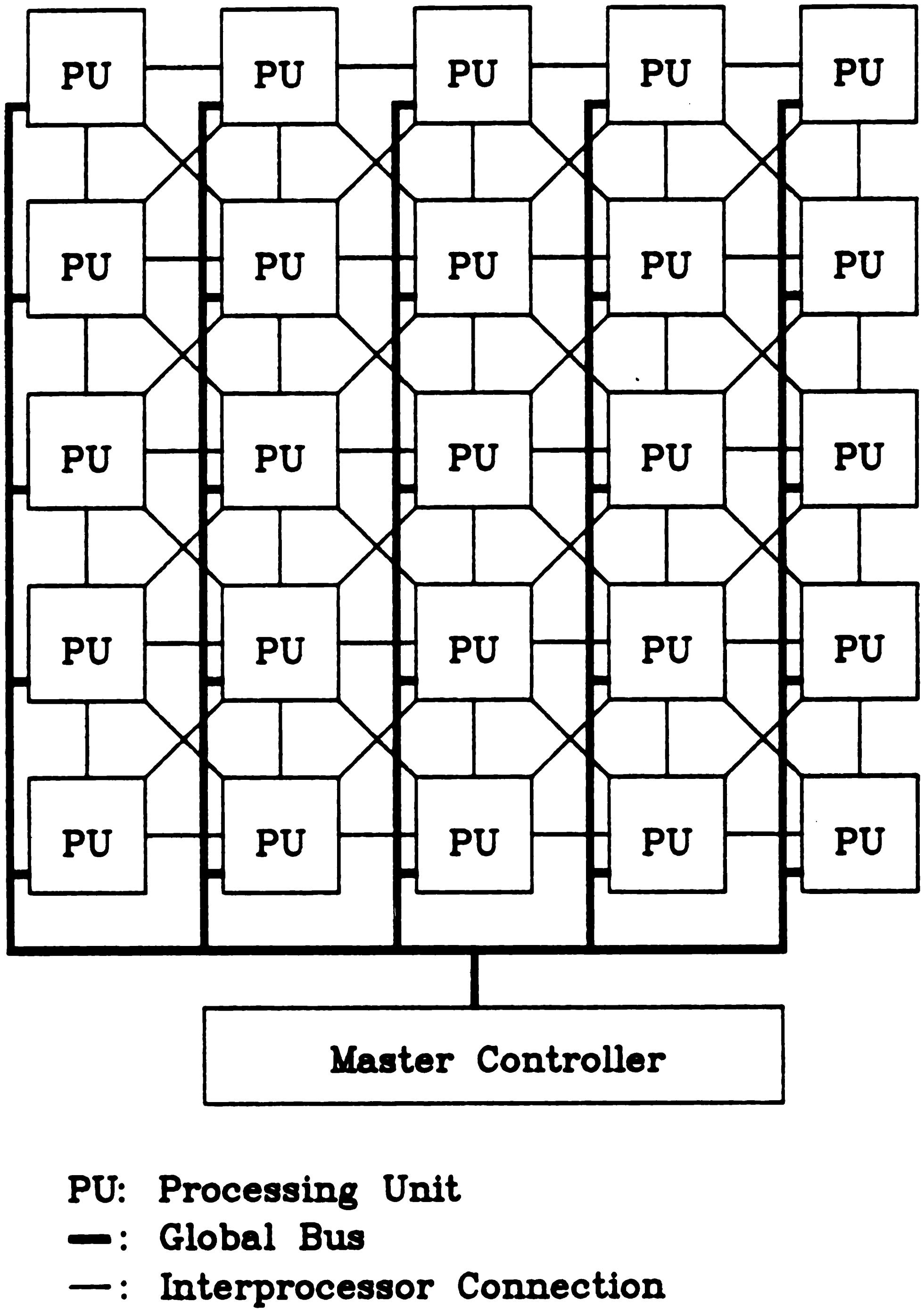


Figure 4.1 The concept of utilizing a processor array for the placement problem.

The FS placement model discussed in Chapter III can be implemented on this processor array. Each slot on the chip, as defined by the FS placement model, is mapped onto a node processor and thus the entire chip area is represented by the processor array. The initial placement is generated by randomly assigning one building block to an empty node processor. An occupied node processor possesses information about its resident block, including the identification of its resident block and the locations of the blocks that have connectivity with its resident block. The node processor is then able to calculate the wiring lengths of the interconnections involved with its resident block.

The movement of a building block to an empty slot is performed by transferring its entire set of information to the node processor that represents that slot. Exchanging the locations of two blocks is done similarly by exchanging their information between their host node processors. All displacements are restricted to those between neighboring slots to reduce the communication traffic between the node processors.

After a new configuration is produced by making multiple concurrent block displacements, the simulated annealing algorithm enters its cost evaluation stage. It has been mentioned that a node processor can calculate the wiring distances associated with its resident block; however, its information about the blocks in other slots may be out-of-date since they may have just been displaced from their locations. The node processors can, of course, broadcast their up-to-date information to all other processors before the evaluation is done. This creates a lot of undesirable communication traffic on the global bus. Recall, however, that the block displacements are in their immediate neighborhoods and hence their previous locations are good approximations of their new locations. The node processors can thus use their slightly out-of-date information to perform the cost calculation.

The next step is to determine the acceptability of the newly produced configuration. This can be done either globally or locally. In the global cost evaluation, the node processors report their associated costs to the controller. The controller sums up the costs and decides whether or not to accept the configuration. It is possible that the controller will receive different costs for the same connection since their calculations are carried out redundantly and independently in different node processors using slightly out-of-date information. The average of the costs for the same connection can be used by the controller to make its decision.

This global cost evaluation scheme has been simulated on a VAX-8600 to evaluate its performance. The simulation results indicate that this process has a problem in convergence to a good placement of the building blocks. Moreover, the computation time of this processor array is found to be worse than that of a sequential machine due to the overhead required for the sequential broadcast of block information from each node processor. The reason for this is intuitive; a multiple displaced configuration may include some cost improving displacements as well as some cost increasing ones. It is very likely that the cost increasing moves will dominate the evaluation of the configuration acceptance, especially at the later stage of the annealing process. This prevents good displacements from being accepted.

Another possible scheme for evaluating a configuration generated by multiple displacements is to consider the cost changes separately for each displacement. The acceptability of an individual displacement is locally determined by the node processor involved. An analysis of the possibility of accepting a configuration with this evaluation scheme reveals that it is only an approximation to simulated annealing in which global cost evaluation is used. The following discussion is based on the assumption that all block displacements happen within their immediate neighborhoods so that the calculations based on the slightly out-of-date information are acceptable.

Since the block displacements are evaluated locally and independently, it is likely that only some of the block displacements will be accepted and appear in the new configuration. A configuration produced by the local cost evaluation scheme is considered and compared to its probability of being accepted in a multiple displacement simulated annealing process using global cost evaluation. Suppose *n* blocks are displaced to generate a new configuration and let ΔC_i indicate the cost change due to the displacement of block *i*. Three cases are considered to compare the possibility of accepting the same configuration in both schemes.

Case 1: The cost changes induced by the block displacements are $\Delta C_i > 0$, for $i = 1,\ldots, n$, which imply that all of them are cost increasing moves. In the local cost evaluation scheme, the probability of accepting the displacement of block *i* is $e^{-\Delta C_i/T}$. The probability of having all the displacements appearing in the next configuration is the same regardless of whether the cost is evaluated locally or globally. This probability can be expressed as

$$P_1 = \prod_{i=1}^{n} e^{-\Delta C_i/T} = e^{-(\sum_{i=0}^{n} \Delta C_i)/T} . \qquad (4.1)$$

In the local cost evaluation, there is also a non-zero probability of accepting *k* out of the *n* block displacements in the new configuration. Without the loss of generality, the accepted block displacements are numbered from 1 to *k*. The probability of accepting these block displacements in the local cost evaluation scheme is

$$P_2 = \prod_{i=1}^{k} e^{-\Delta C_i/T} \prod_{j=k+1}^{n} (1 - e^{-\Delta C_j/T}) = e^{-(\sum_{i=0}^{k} \Delta C_i)/T} \prod_{j=k+1}^{n} (1 - e^{-\Delta C_j/T}). \qquad (4.2)$$

Case 2: The cost changes resulting from the displacements are $\Delta C_i \leq 0$, $i = 1,\ldots, n$. These changes indicate that none of the block displacements has worsened the cost function. Since $\sum_{i=1}^{n} \Delta C_i \leq 0$, a configuration with these block displacements is accepted in either scheme with probability 1.

Case 3: Mixed displacement costs are produced in this case. Without the loss of generality, it is assumed that the cost increasing displacements are numbered from 1 to m, while the others improve or at least do not change the cost function. These cost changes can be expressed as $\Delta C_i > 0,\ i = 1,..., m$ and $\Delta C_j \leq 0,\ j = m+1,..., n$. In the local cost evaluation scheme, the displacements of blocks j, $j = m+1,..., n$, will definitely be accepted. In addition, assume k out of the m cost increasing displacements are accepted. The probability of accepting such a configuration in the local cost evaluation is

$$P_3 = \prod_{i=1}^{k} e^{-\Delta C_i/T} \prod_{j=k+1}^{m} (1 - e^{-\Delta C_j/T}) = e^{-(\sum_{i=0}^{k} \Delta C_i)/T} \prod_{j=k+1}^{m} (1 - e^{-\Delta C_j/T}). \tag{4.3}$$

If the cost is evaluated globally and $\sum_{i=1}^{m} \Delta C_i + \sum_{j=m+1}^{n} \Delta C_j \leq 0$, the cost improving displacements dominate the cost evaluation and the probability of accepting all displacements is 1. On the other hand, if $\sum_{i=1}^{m} \Delta C_i + \sum_{j=m+1}^{n} \Delta C_j > 0$, the global effect of all the displacements is cost increasing and the probability of accepting them is

$$P_4 = e^{-(\sum_{i=1}^{m} \Delta C_i + \sum_{j=m+1}^{n} \Delta C_j)/T}. \tag{4.4}$$

The global cost evaluation scheme considers all the displacements at the same time and they are either totally accepted or totally rejected. In the local cost evaluation, those displacements that improve the cost will definitely be accepted, while the cost increasing displacements are handled using probability. This is considered an advantage of the local cost evaluation, since it will not miss the individual displacements that improve the cost. This feature is especially helpful at the later stages of the process by avoiding the inferior result obtained in the global cost evaluation scheme. The capability of locating the optimal configuration is thus enhanced.

The major improvement of the local cost evaluation lies in the fact that several simulated annealing processes are working in parallel on different regions of the chip. This improves the speed of convergence since it is similar to evaluating multiple moves at the same time. But, this approach also suffers from some inherent drawbacks. The necessity of broadcasting the block information every time a new configuration is accepted creates a large amount of communication traffic on the global bus. From the standpoint of hardware, a vital drawback of this approach is that it needs a node processor to represent each slot. The number of slots defined on a chip is directly proportional to the chip area and the size of the building blocks. The number of processors needed to solve a large placement problem may thus become impractical. Even though future technology may allow the inclusion of a large number of processors in the same system, the sequential broadcast of information will waste many working cycles of the processors.

A variation of this multiple displacement simulated annealing with local cost evaluation is developed to remove these problems. The details of this algorithm and a hardware accelerator implementing this algorithm are described in the next sections.

4.2 Parallel Building Block Placement Using Simulated Annealing

In order to reduce the number of node processors the multiple displacement simulated annealing with local cost evaluation proposed in the previous section is extended. Instead of assigning a single slot to a node processor, a node processor is assigned to a region of the chip and it executes the simulated annealing process for that region. If the building blocks that will eventually be placed into a certain area can be predicted beforehand, they can be handed over to the node processor that is assigned to work at that region. Unfortunately, the exact location of a building block within a region is not known to the processors working on other regions unless the time consuming broadcast scheme is used. However, if the area of a chip assigned to a node processor is small, the displacement of any building block within that region will be in a small neighborhood. Whenever a cost evaluation takes into account the blocks in other regions, the center of a region can be used as an approximation of the locations of the building blocks within this region. Assuming that the mapping of the building blocks is satisfactory, then the simulated annealing processes on different regions can be performed independently and hence no communication is required between the node processors. Furthermore, a hardware accelerator implementing the processor array will be more flexible for the placement problem since the number of slots or the area of the region assigned to each node processor can be easily changed.

The first step in developing this parallel simulated annealing placement process involves finding an appropriate method to map the building blocks into their proper regions. The mapping plays an important role in the algorithm and will have a critical effect on the quality of the final placement. If the blocks mapped into one region remain there through the entire placement process, they are bound to become neighbors in the final placement. This may or may not be as good a configuration as might be determined by a sequential placement process depending upon the quality of the initial building block mapping. On the other hand, if the building blocks are allowed to migrate between different regions during the placement procedure, the requirement of inter-process communication will be tremendous and the presumption of no inter-processor communication is ruined.

A scheme for explicitly mapping the building blocks to different regions on the chip is developed in this research. The ultimate objective of this mapping scheme is to predict and ensure the spatial localities of the building blocks in a layout as would be produced by a sequential placement process. This means that the blocks mapped into a region are neighbors within a placement configuration obtained sequentially. The adverse effect of making multiple displacements and evaluating the cost changes independently will be minimized if the spatial localities of the building blocks are predicted and retained by the mapping process. This observation is stated and discussed in the following proposition.

Proposition 4.1: If the building blocks mapped to a certain region on a chip appear as neighbors in the placement generated sequentially, then in the parallel placement method having independent simulated annealing processes working on different chip regions, the spatial localities between these blocks are retained. In addition, if the spatial localities are retained for all the regions and the relative locations of these regions are substantially equal to that obtained by a sequential simulated annealing placement method, then the adverse effect of performing simulated annealing independently in different regions is minimized.

Remark: For easy of discussion, the configurations generated by a sequential simulated annealing process and a parallel simulated annealing process are referred to as sequential placement and parallel placement, respectively. If the number of blocks mapped to a region approaches the size of the entire problem, the cost of parallel processing, or the cost difference between the sequential and the parallel placements, approaches 0. This is very easy to see since both placements will be exactly the same when all the blocks are mapped into a single region. When more than one region is defined on the chip, the cost of a placement can be calculated by considering two factors:

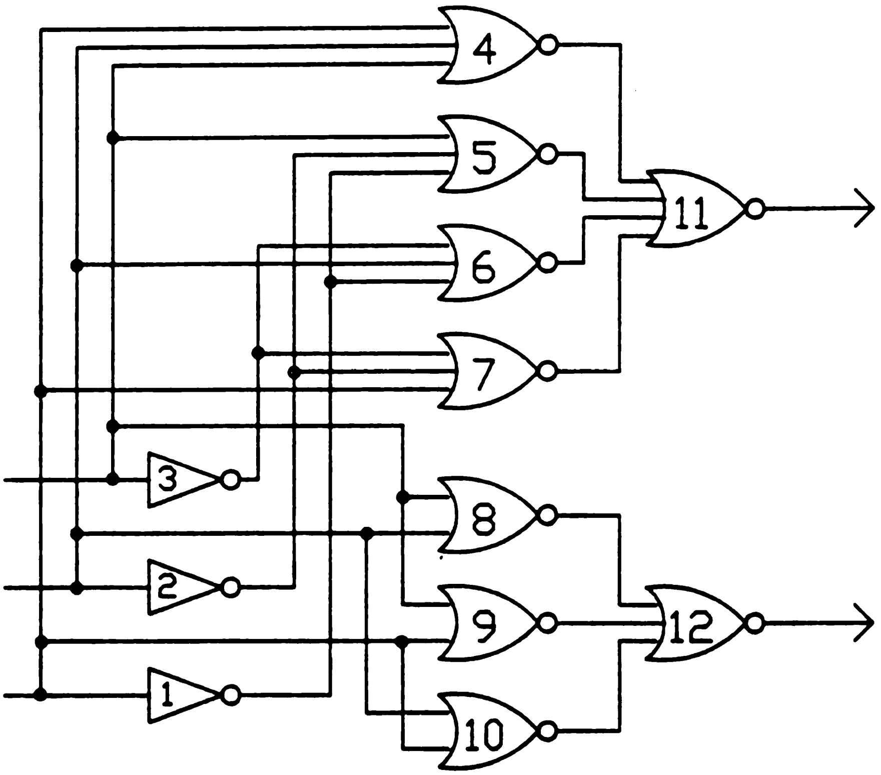
1. the sum of regional costs, each of which is associated with only the building blocks in a region; and
2. an inter-regional cost which describes the cost induced by the connectivity of building blocks in different regions.

Based on the assumption that the blocks mapped to a region appear as neighbors in the sequential placement, the regional costs in the parallel placement are close to those in the sequential placement. If the relative locations of the regions in the parallel placement are approximately the same in the sequential placement, then the inter-regional costs calculated in both placements are also close to each other. The total placement costs in both methods are substantially equal and thus the adverse effect of parallel processing is minimized.

The mapping of the building blocks to different regions of a chip is performed in two steps. The building blocks are first grouped into clusters according to their predicted spatial localites as would be found by sequential placement and then the clusters are mapped to different regions on the chip. In order to guarantee the satisfaction of Proposition 4.1, the clustering scheme must follow the guidance used in the sequential placement process. Since the most common cost function for placement algorithms is the estimated wiring length between the building blocks, the clustering scheme in this work is based upon the interconnections between the blocks. The relationship between any pair of blocks can be described by the connectivity between them. A coherency matrix, which represents the spatial locality between any two blocks, is used to guide the clustering of the building blocks. The coherency between two blocks is calculated by considering their interconnections. The rules for the generation of the coherency matrix are described below.

1. If a connection exists exclusively between two blocks, the coherency assigned to this connection is 1 unit.
2. If the connection is shared by more than two blocks, the coherency between any pair of blocks in the group defined by this connection is 1 unit divided by the number of blocks connected.
3. The coherency between any pair of blocks is calculated by summing up all the coherencies induced by the interconnections between them. If there is no connection between two blocks, their coherency with respect to each other is defined as 0.

An example is used to demonstrate the generation of a coherency matrix for a circuit. The schematic diagram of the example circuit and its corresponding coherency matrix are shown in Figure 4.2. Each element h_{ij} of the matrix H describes the coherency between blocks i and j. As shown in Figure 4.2, the coherency matrix is symmetric about the diagonal and can be treated as a triangular matrix since the coherency between blocks i and j is the same as that between blocks j and i.



$$\begin{array}{cc}
 & \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} &
\left[\begin{array}{cccccccccccc}
- & 0 & 0 & .2 & .2 & .33 & .33 & .2 & .2 & 0 & 0 & 0 \\
0 & - & 0 & .2 & .33 & .2 & .33 & .2 & 0 & .2 & 0 & 0 \\
0 & 0 & - & .2 & .33 & .33 & .2 & 0 & .2 & .2 & 0 & 0 \\
.2 & .2 & .2 & - & .2 & .2 & .2 & .2 & .2 & .2 & 1 & 0 \\
.2 & .33 & .33 & .2 & - & .33 & .33 & .2 & .2 & 0 & 1 & 0 \\
.33 & .2 & .33 & .2 & .33 & - & .33 & .2 & 0 & .2 & 1 & 0 \\
.33 & .33 & .2 & .2 & .33 & .33 & - & 0 & .2 & .2 & 1 & 0 \\
.2 & .2 & 0 & .2 & .2 & .2 & 0 & - & .2 & .2 & 0 & 1 \\
.2 & 0 & .2 & .2 & .2 & 0 & .2 & .2 & - & .2 & 0 & 1 \\
0 & .2 & .2 & .2 & 0 & .2 & .2 & .2 & .2 & - & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & - & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & -
\end{array}\right]
\end{array}$$

Figure 4.2 The generation of a coherency matrix for an example circuit.

The clustering of the building blocks in a placement problem also belongs to the class of combinatorial optimization problems and finding optimal clusters is as difficult as the placement problem itself. Due to the near optimal characteristic of the simulated annealing approach in solving NP-complete problems, it is adopted again for the clustering scheme developed in this research. The coherency matrix is used to guide the clustering operation. The algorithm for grouping building blocks into different clusters is presented in Figure 4.3.

```
Cluster(){
  Randomly assign the building blocks into a number of clusters;
  /* the number of clusters and the capacity of a cluster are
     specified by the designer */
  While (stopping criterion is not satisfied)
    Rearrange the clustering result according to the principles of
    simulated annealing;
    /* the movement set includes moving a building block to a new cluster
       and swapping the ownerships of two blocks; the cost function is
       defined by the inter-cluster coherency, the intra-cluster coherency,
       and the block area in the clusters */
}
```

Figure 4.3 The algorithm for clustering a placement problem.

The clustering algorithm accepts two parameters from the designer: the number of desired clusters and the maximum capacity of a cluster. The capacity of a cluster can also be determined automatically as $\lceil n/r \rceil$, where n is the number of blocks in a placement problem and r is the number of desirable clusters. Two types of movements are used to modify a clustering result to generate a new state in the simulated annealing process:

1. moving a building block from its original cluster to another non-full cluster; and
2. swapping the ownership of two building blocks among two clusters.

The objective of simulated annealing in this clustering process is to maximize the intra-cluster coherency or to minimize the inter-cluster coherency, while maintaining the balance between the block area in different clusters. This objective is based on the following considerations.

1. It is desirable that the coherency between the clusters become null. If this is possible, an ideal clustering result is produced. The building blocks of the circuit are mapped into independent subcircuits which can be annealed independently. The parallel placement method does not affect the placement cost at all. This strategy is implemented by including a term,

   $$\sum_{i,j} h_{ij}, \quad \text{for all blocks } i \text{ and } j \text{ not in the same cluster,}$$

   in the cost function of the building block clustering process, where h_{ij} is the coherency between blocks i and j.

2. Restating condition No. 1, the aim is to maximize the intra-cluster coherency of all the clusters under the constraint that each cluster has an upper limit on the number of blocks that it can hold. The sum of all the elements in a coherency matrix is a constant since the circuit connectivity is fixed. The maximization of the intra-cluster coherencies is equivalent to the minimization of the inter-cluster coherency. The constraint on the capacity of a cluster is necessary since the intra-cluster coherency will definitely be maximized when all the blocks are put into a single region. This term can be defined as

$$\frac{1}{\sum_{i=1}^{|G|} \sum_{j,k} h_{jk}}, \text{ for all blocks } j \text{ and } k \text{ in cluster } i,$$

where G is the set of all clusters, and $|G|$ is the number of desired clusters.

3. The block areas in all the clusters should be approximately equal. This will facilitate the application of the previously described FS model, which requires that the blocks to be placed have approximately equal sizes. This consideration is implemented as a term in the cost function, defined as

$$\sum_{i=1}^{|G|} |A_i - A_{ave}|,$$

where A_i is the total block area of cluster i, and A_{ave} is the average block area of all the non-empty clusters.

The reasoning behind this clustering scheme is based on the ultimate goal of a placement process, which states that the blocks strongly connected should be placed close to each other. The clustering scheme maximizes the intra-cluster connectivities and hence the blocks clustered together as neighbors in the final configuration are strongly connected.

Once the building blocks have been clustered into groups, the next step is to assign these clusters to different regions of the chip. This assignment must be done in a manner such that the relative locations of the clusters in the parallel placement resemble the sequential placement result. A simplified version of the FS placement algorithm is applied to establish the relative locations of the clusters on a chip by considering the clusters as superblocks and thus place them into different regions of the chip.

It is desirable that the maximum number of building blocks in a region be small enough so that the assumption of neighborhood displacements is satisfied. For large placement problems, the number of clusters generated by the clustering scheme may be too large for the FS placement algorithm to operate upon efficiently. A hierarchical clustering method is applied to solve this problem, the concept of which is illustrated in Figure 4.4. The clustering scheme can be applied hierarchically and repeatedly by considering the clusters generated at one level as superblocks for the clustering process at next higher level. Only the inter-cluster connections are considered in the clustering operation for the next higher level. A tree structure is then created to guide the mapping process. The root indicates the final chip and the leaves of the tree represent the building blocks. The number of levels in the mapping hierarchy is determined by working from the leaves toward the root until the number of clusters in one level is appropriate for the FS placement algorithm.

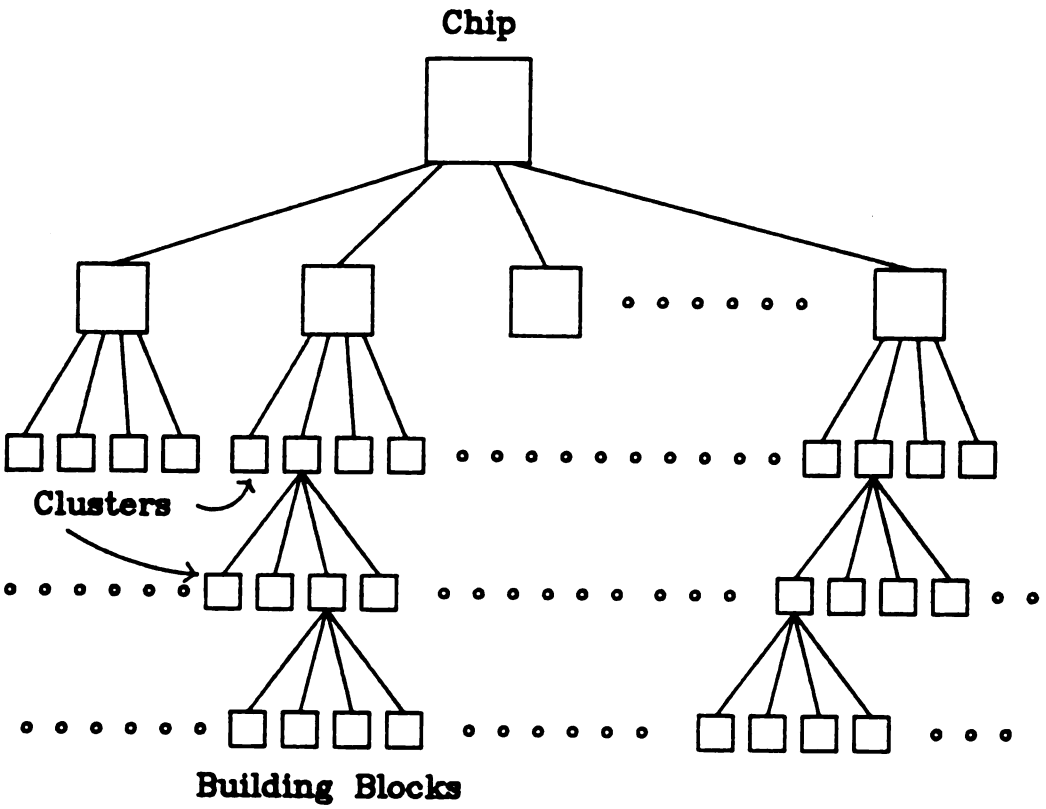


Figure 4.4 A hierarchy for the clustering and mapping processes.

A top-down approach using the hierarchy generated by the clustering process is used to map the building blocks to different regions of the chip. For each level in the mapping hierarchy, the inter-cluster interconnections are used to determine the relative locations of the clusters, which are used along with the intra-cluster interconnections to perform the mapping operation at next lower level. Notice that the intra-cluster interconnections at one level are considered as inter-cluster interconnections at the next lower level.

The relative locations of the clusters in one level of the hierarchy are established using a simplified version of the FS placement algorithm. The FS placement algorithm requires knowledge of the sizes of the clusters and the inter-cluster terminal locations. The cluster sizes are used to define the slot array while the inter-cluster terminal locations are necessary for considering different orientations of the clusters. The members within the clusters have yet to be placed at this stage and hence neither the cluster sizes nor the terminal locations for inter-cluster connections are known. If the chip is assumed to have unlimited area, one possible approach is to define the slot size arbitrarily and apply the FS placement algorithm on this array. However, in a more practical case, having limited chip area, an underestimated slot size will create a configuration which requires an area larger than the chip. This result can be seen by considering that the number of slots in one column (row) of the array is obtained by dividing the length (width) of the chip by the length of a slot. The largest total block area in a cluster is used to give a reasonable estimation of the slot size. All the inter-cluster interconnections are temporarily considered to emerge from the center of the slots which hold their respective clusters.

An FS placement algorithm with a reduced set of possible movements is then applied to this level of mapping in the hierarchy. Since the wiring length is calculated by using the slot centers as starting and terminating points, the changes in cluster orientations will not affect the wiring distance and thus are not considered in the FS placement algorithm. When the FS algorithm converges, the relative locations of the clusters on a chip are determined. The cluster orientations will be considered by the mapping performed at the next lower level.

After the relative locations of the clusters at one level are determined, the process continues to perform the mappings of the members in these clusters. The chip region allocated for the members of a cluster is determined by the slot size defined at the next higher level.

Recall that it was impossible to determine the best orientations of the clusters in the previous mapping process since all the calculations of wiring length are done with respect to the cluster centers. When the mapping of the members of a cluster is performed, the information about the relative locations of the clusters is available and is utilized to consider the orientations of the clusters. The first step is to define the terminal locations on the region assigned for a cluster for the inter-cluster interconnections. The relative locations of the clusters indicate approximately the direction in which an inter-cluster connection will enter or leave. The members in a cluster can be mapped to locations so that the inter-cluster connections are minimized in their respective directions and thus their costs are reduced. The consideration of the inter-cluster connections in the intra-cluster mapping takes the orientations of the clusters into account. The inter-cluster terminals are defined, using the information of the relative cluster locations, according to the following rules. Some examples of defining inter-cluster terminals are provided in Figures 4.5a and 4.5b.

1. If an inter-cluster interconnection only involves with two clusters, then a terminal is set up for this connection at the intersection of the region boundary and the line connecting the centers of both clusters.

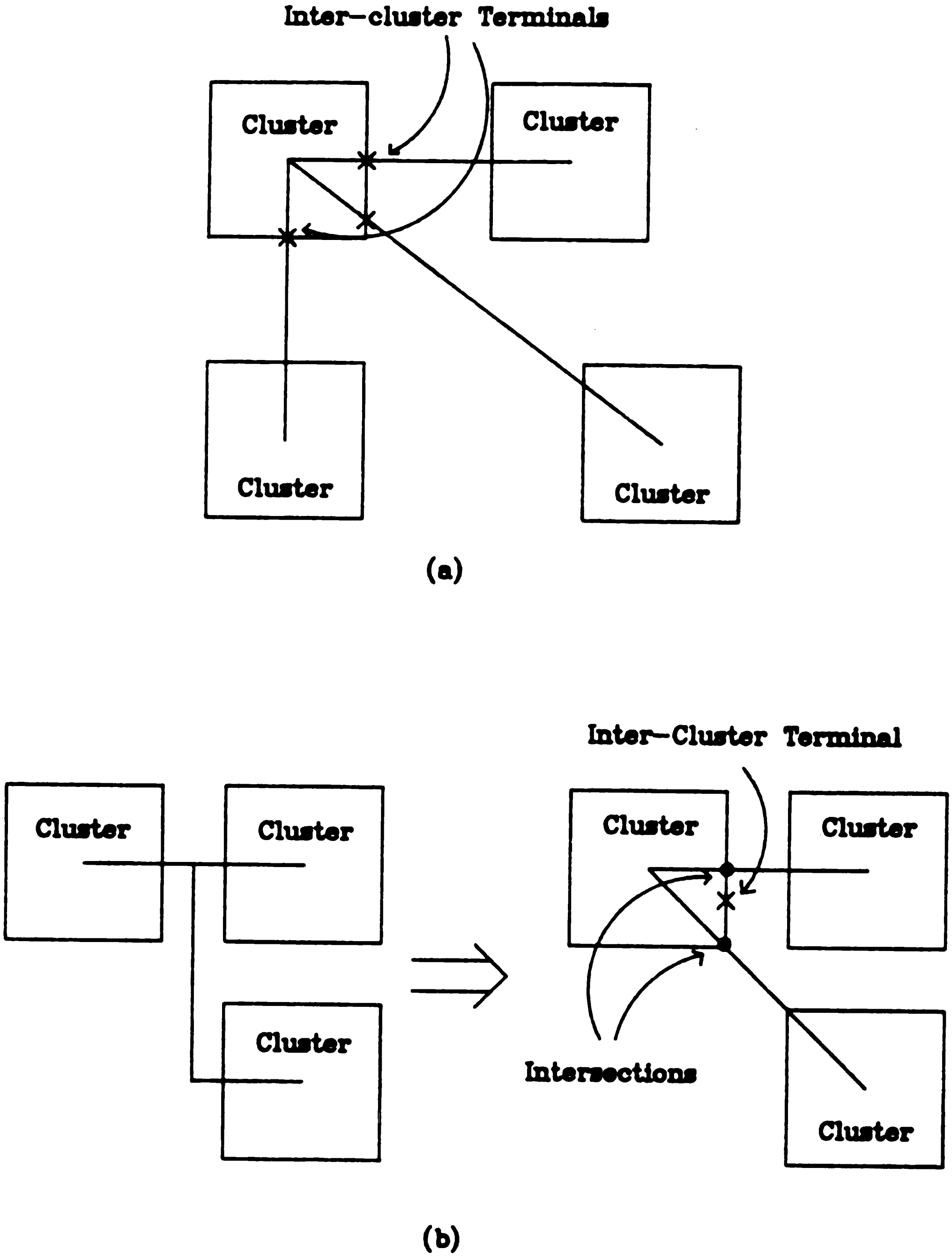


Figure 4.5 Some examples of defining inter-cluster terminals, (a) an inter-cluster interconnection involving only two clusters, and (b) an inter-cluster interconnection involving more than two clusters.

2. If an inter-cluster interconnection involves more than two clusters, the determination of the location at which it enters or leaves the cluster is non-trivial. The concept of a center of mass is applied to determine the approximate locations of the inter-cluster terminals. The inter-cluster connection is first decomposed into several interconnections, each of which connects the center of the cluster under consideration to that of another cluster. Imaginary unit masses are placed at the intersections of these connections with the region boundary. An inter-cluster terminal is set up at the center of mass for these intersections. Assume that there are k intersections $p_i(x_i, y_i)$, $i = 1,\ldots, k$, where x_i and y_i are the x- and y-coordinates of intersection p_i. The location of the inter-cluster terminal defined for this connection is calculated as

   $$x_t = \frac{\sum_{i=1}^{k} x_i}{k} \tag{4.5a}$$

   and

   $$y_t = \frac{\sum_{i=1}^{k} y_i}{k}, \tag{4.5b}$$

   where x_t and y_t are the coordinates of the inter-cluster terminal.

The FS placement algorithm is then utilized to map the members of a cluster into its region. In addition to the consideration of the intra-cluster connections between the members, the wiring distances to and from the inter-cluster terminals are also included in the cost evaluation.

The mapping of the building blocks in a placement problem to their respective chip regions is now complete. The remaining task of the placement process is to apply simulated annealing to the chip regions to determine the locations of the building blocks. The terminals on the boundary of a region are defined for inter-regional connections using the same rules described above for the inter-cluster connections. Simulated annealing processes are then performed on all the regions in parallel to determine the final configuration. If the number of available processing units is less than the number of regions defined on the chip, the multiple displacement simulated annealing processes can be applied sequentially to different parts of the chip. This concept is illustrated in Figure 4.6.

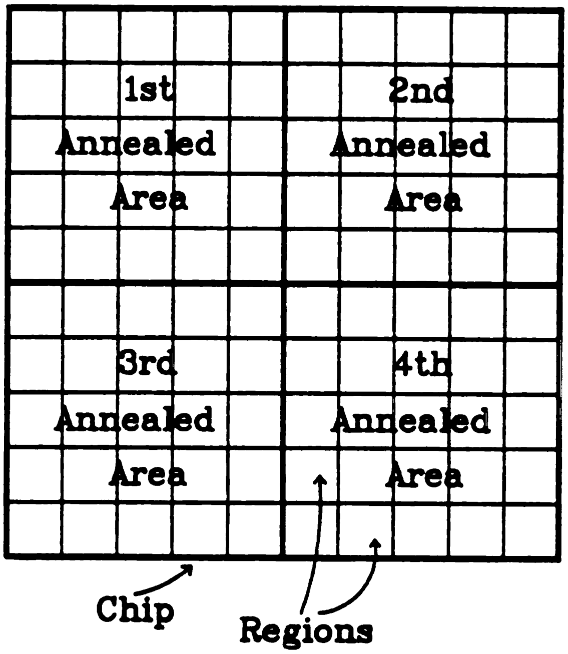


Figure 4.6 The application of simulated annealing to different parts of a chip.

4.3 Degree of Parallelism

One important criterion in the evaluation of a parallel algorithm is a consideration of the degree of parallelism within the algorithm. The degree of parallelism in an algorithm determines the amount of speedup in computation time obtainable in a parallel processing environment.

The parallel placement method can be described as a mapping process, which includes block clustering, the mapping of the clusters into different regions of the chip, and the concurrent simulated annealing processes. The clustering has to be done hierarchically and sequentially; however, experiments show that the computation time required by the clustering operation is insignificant in comparison to the total placement time. The mapping operations at the same level can be performed independently and in parallel. Whenever the relative locations of the clusters at one level of the hierarchy are established, the construction of the inter-cluster terminals for the cluster area can also be done in parallel for all the clusters. This implies that this approach is especially suitable for parallel processing.

To further investigate the potential of this approach in a parallel environment, the placement method is simulated and compared against a sequential placement method. The two parameters of interest in this comparison are the placement acceleration and the rate of processor utilization. The results of these comparisons are graphed in Figures 4.7 and 4.8, respectively.

Assume the placement algorithm has a computational complexity of $O(n)$, where n is the number of blocks in the placement problem. This is the lower bound of the computation complexity of any placement algorithm. Let the time required for the placement of n building blocks be n time units. In the case of the parallel placement

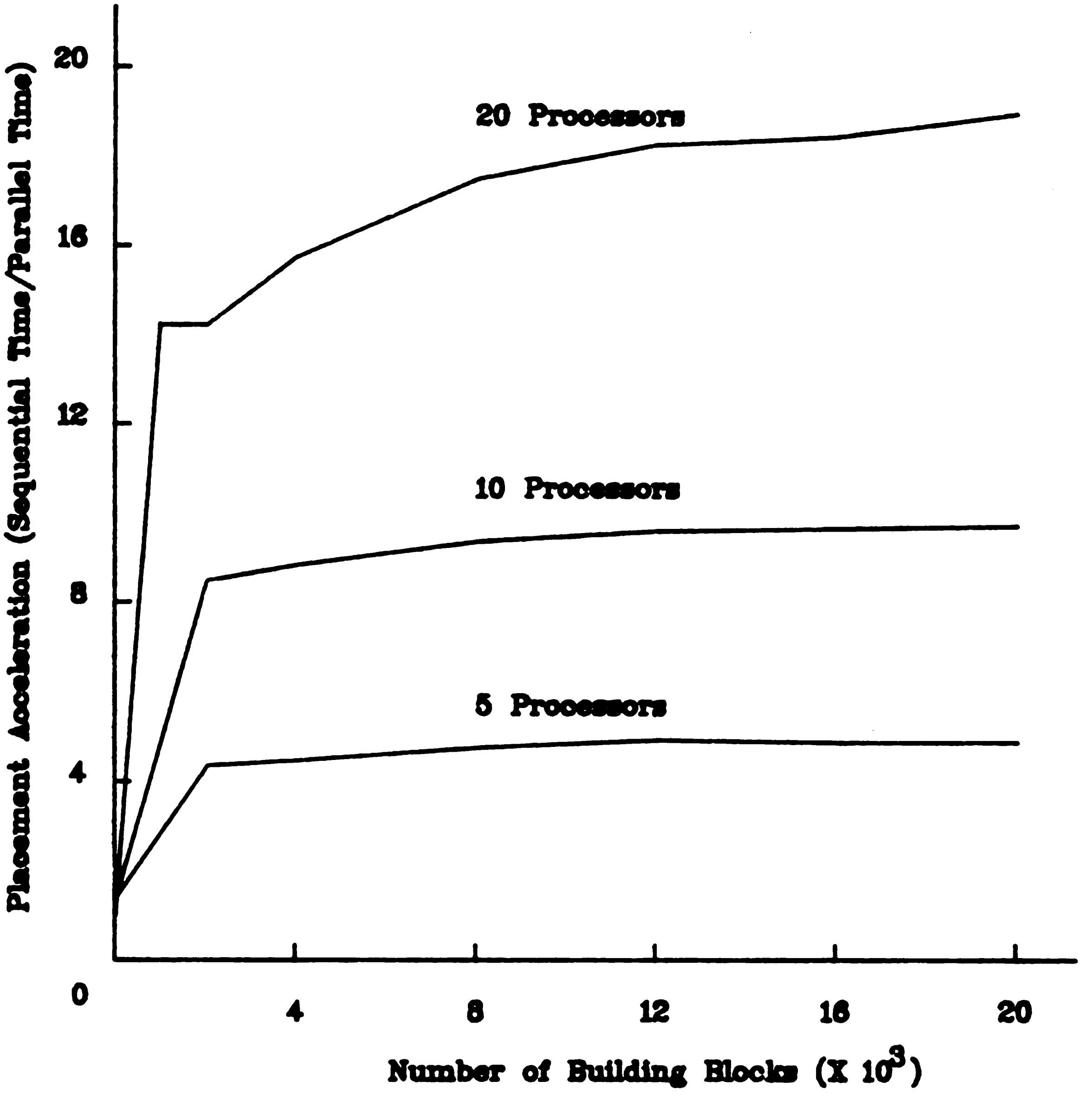


Figure 4.7 Acceleration obtained by the parallel placement method.

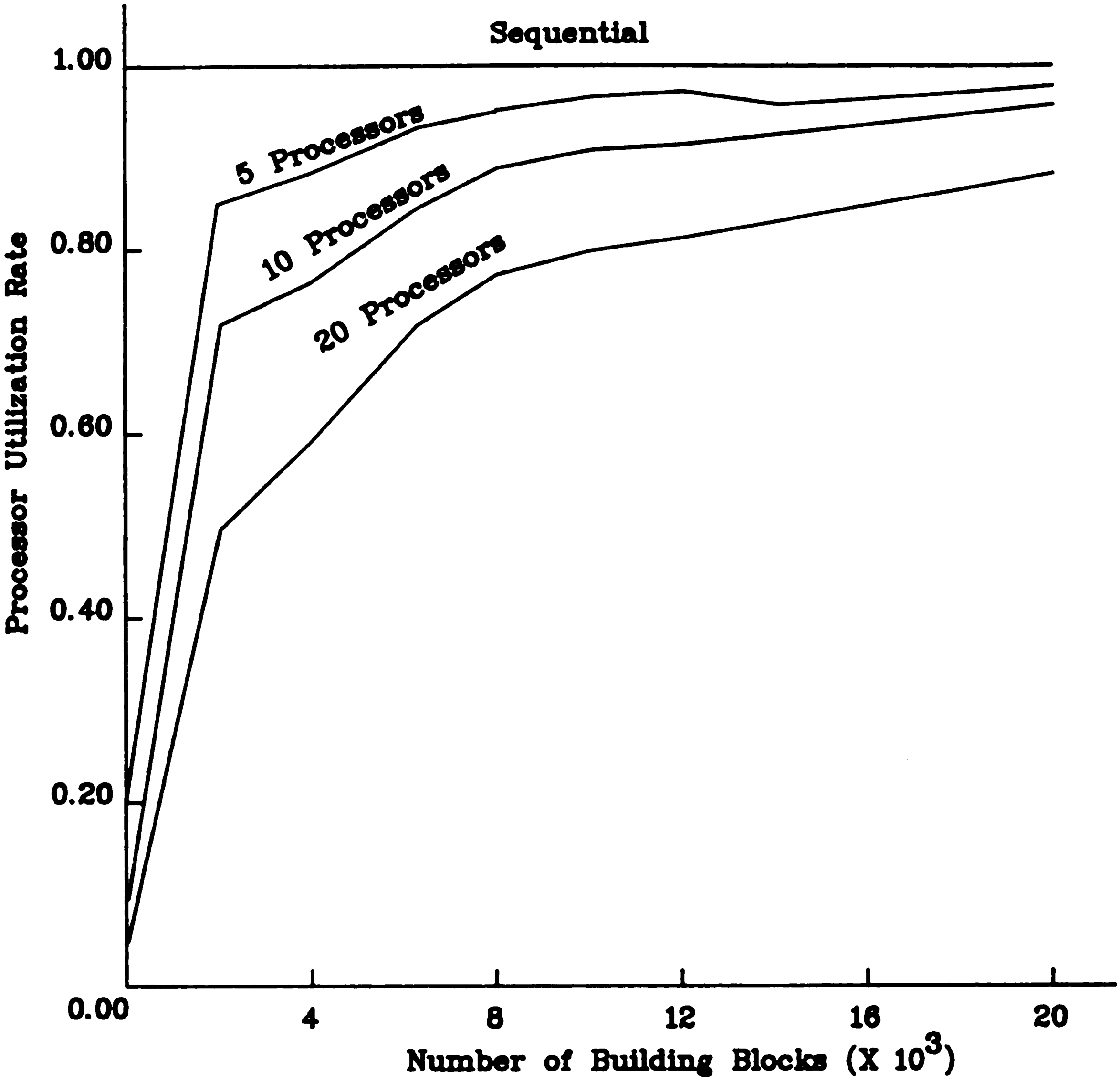


Figure 4.8 Processor utilization rate for the parallel placement method.

method, the capacity of a cluster is set as $M = 50$, while different numbers of processing units, $P = 5$, 10, and 20, are considered in the comparisons. The computation times for the parallel placement method are generated by performing the mappings or placements at the same level of the hierarchy in parallel. For a placement problem with n building blocks, the number of levels in the hierarchy is

$$L = \lceil \log_M n \rceil. \tag{4.6}$$

The time required for completion of the parallel placement is given by

$$T = \sum_{i=1}^{L} T_i, \tag{4.7}$$

where T_i is the time required to finish one level of mapping or placement. The time consumed in one level of the hierarchy depends on the number of jobs at this level, which is defined iteratively by

$$J_i = \frac{J_{i+1}}{M},\ i = 1,\ldots, L, \quad \text{and } J_{L+1} = n, \tag{4.8}$$

where J_i is the number of jobs performed at level i. After the number of jobs has been determined, the time required by one level of the hierarchy can be expressed as

$$T_i = \begin{cases} M, & \text{if } J_i \le P, \\ M\lceil J_i/P \rceil, & \text{if } J_i > P. \end{cases} \tag{4.9}$$

In Figure 4.7, the acceleration of the parallel placement method is calculated by dividing the time required by the sequential method by that of the parallel method. The curves for processor utilization shown in Figure 4.8 are generated in the following manner. At each level of the hierarchy, the processor utilization is defined as the total working time of the processors divided by the time spent at that level. The processor utilization for the entire parallel placement process is defined as

$$U = \frac{\sum_{i=1}^{L} W_i}{T}, \tag{4.10}$$

where W_i is the working time of the processors in level i and is defined as

$$W_i = \frac{MJ_i}{P}. \tag{4.11}$$

The comparisons shown in Figures 4.7 and 4.8 demonstrate the significant speedup and the high processor utilization rate when the parallel placement method is implemented in a parallel processing environment. The larger the number of building blocks in a placement problem, the greater the speedup that can be achieved from this placement method with multiple displacement simulated annealing. The utilization rate of the processors also approaches 100% when the number of building blocks greatly exceeds the number of processors.

4.4 Hardware Accelerator

The parallel placement method and its associated mapping scheme have been described. After the building blocks are clustered hierarchically, the placement process describes levels of mappings or placements. At each level of the hierarchy, the operation includes the generation of inter-cluster terminals for each cluster and the mappings of their members. Both of these operations can be done for all the clusters in parallel. Best of all, operations in one cluster are independent of those in other clusters. The number of available processors can be increased to reduce the number of blocks in one region for a large placement problem. For generality, the discussion in the previous section assumed a multiple level hierarchy. The increment of processors will, of course, affect their utilization rate, especially at the upper levels of the hierarchy when the number of clusters is less. But, from a practical point of view, if the placement algorithm is designed to handle 50 building blocks at a time, only a two-level hierarchy is needed to guide the placement of 2,500 building blocks. The blocks are grouped into 50 clusters, each of which contains 50 blocks, and the FS placement algorithm is used to map these clusters into different regions for concurrent simulated annealing.

A general-purpose multiprocessor can be used to implement this parallel placement method. This type of computing facility is, however, very expensive at present. The current trend of VLSI design automation is to have all the design steps handled by a workstation, which is relatively inexpensive. It is not economical to dedicate a general purpose multiprocessor system for the placement problem. Furthermore, the computing resources in a general purpose multiprocessor could be under-utilized for this placement method. Multiprocessors generally provide data paths with associated control mechanisms for inter-processor communication. This is not necessary for the parallel placement method since all the placements performed in different regions are independent. In addition, the architecture of the processing units in a general purpose multiprocessor is designed to be universal and not customized for the specific operations of the placement problem. Finally, it is not easy to increase the number of processing units in a general purpose multiprocessor. These observations justify the proposal of a special purpose multiprocessor for the hierarchical placement problem.

4.4.1 Processor Array

The structure of a processor array for the placement process is proposed in Figure 4.9, which is designed according to the requirements of placement with multiple displacement simulated annealing. It consists of a two-dimensional array of processing units. Each processing unit possesses the basic computation capability to perform the simulated annealing process. In contrast to the structure introduced in Figure 4.1, there is no direct communication path provided between the node processors. A controlling microprocessor is connected to all the node processors through a global bus. This is provided to execute the block clustering, which has to be done sequentially, and to assign the building blocks to their respective host processors. The controlling microprocessor also acts as an interface between the processor array and the host CAE workstation.

If there are only two levels of hierarchy in the placement problem, the operation of the processor array is very simple. After the mapping scheme is completed, the building blocks are assigned to the node processors handling their respective chip regions. The simulated annealing process executed within the node processors determines their locations within those regions. Finally, the locations and orientations of the building blocks are reported to the CAE workstation through the global bus under the control of the microprocessor. On the other hand, if there are more than two levels in the placement hierarchy, the microprocessor has the responsibility of assigning jobs to the node processors to fully utilize them for each level of mapping in the hierarchy.

The processor array can be implemented in a VLSI-based structure. If a larger array is needed, it can be constructed easily by combining several chips on a board to form a bigger accelerator so that a large chip design can be handled in one run. This board level hardware accelerator is shown in Figure 4.10.

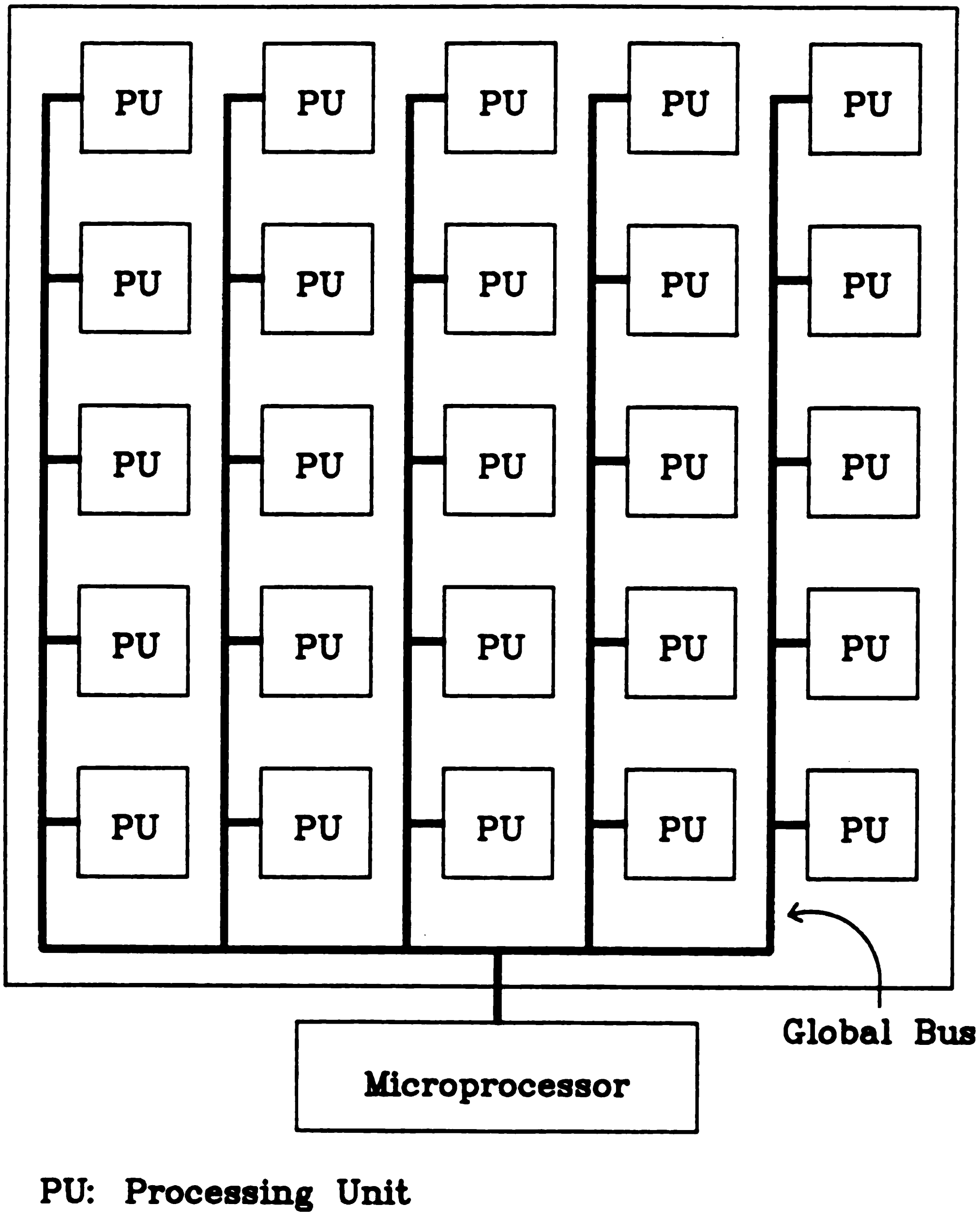


Figure 4.9 The structure of a processor array for the parallel placement process.

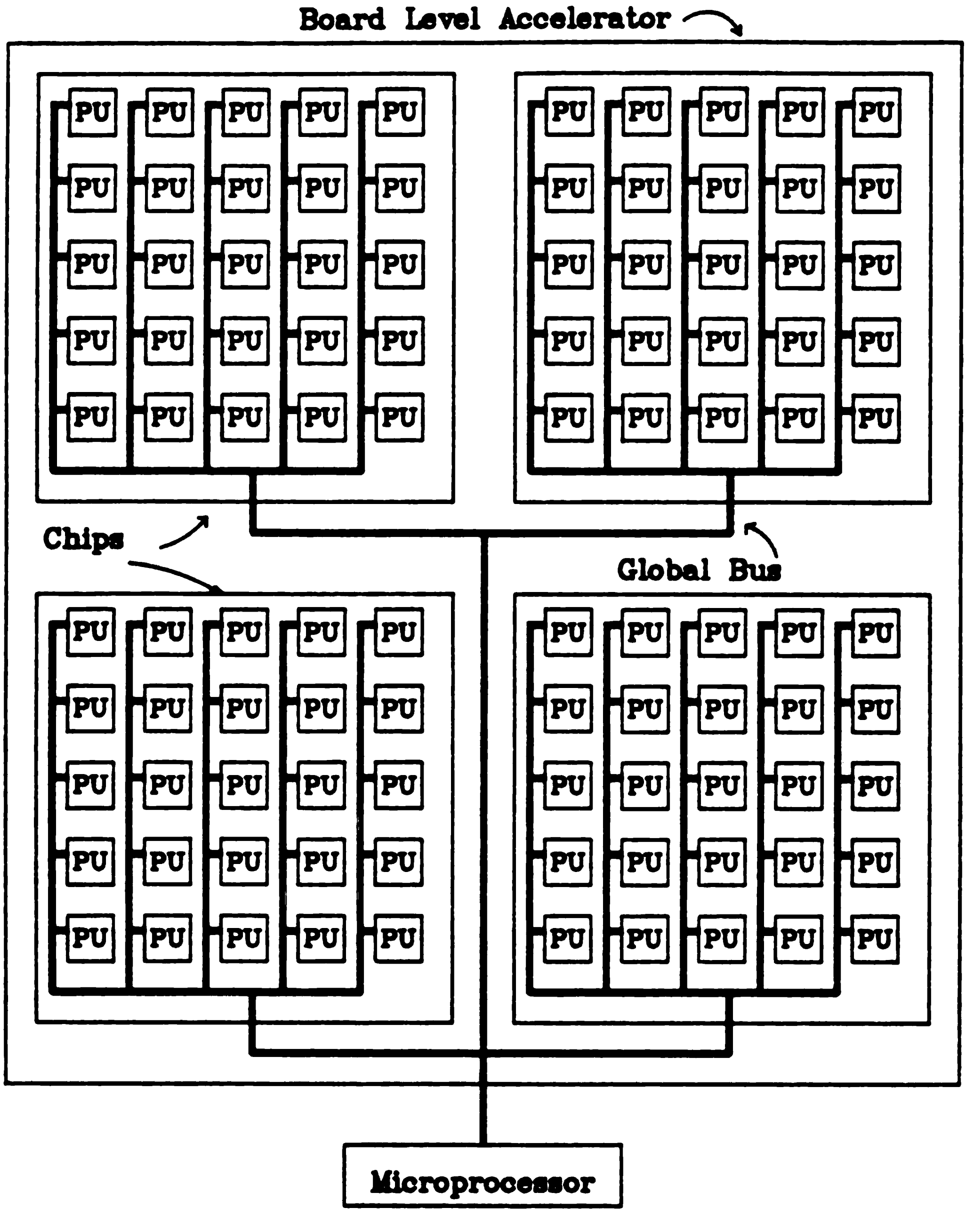


Figure 4.10 A board level hardware accelerator for the placement problem.

4.4.2 Node Processor (PU) Details

All the node processors (PUs) in the array are the same. The entire array can be built by duplication of the PUs on a chip. In order to define an architecture for a node processor, the required operations of the simulated annealing process are considered. In the process described in Section 2.4, many operations, such as the selection of block displacement and the decision of accepting or rejecting a configuration, require the generation of random numbers. A hardware pseudo random number generator is therefore necessary. Random numbers can be generated by means of a simple circuit called an autonomous linear feedback shift register. This is a series connection of D flip-flops with no external inputs and with all feedback provided by means of exclusive-OR gates (XORs) [40].

Most of the computation time is consumed in the parallel simulated annealing of the chip regions. To further accelerate the simulated annealing process executed in a node processor, the inherent parallelism in the process is utilized. As in the TimberWolf package, the orientation change of a building block is only considered after its movement is rejected. The orientation change of a building block can only slightly change the cost and thus the process should concentrate on the movement of blocks, especially at early stages of the placement process. At the later stages of the annealing process, the locations of the building blocks are almost fixed. The process then should concentrate on the orientation changes of the blocks. The method of rotating blocks only after movement is rejected automatically takes these considerations into account. This was also followed in this research. Further study revealed that the rotation of a block is totally independent of any previous movement. The rotation of a block is based on the original block information, it is not considered at all if the block has been moved to a new location.

In the later stages of the process, the rejection of block movements becomes quite common since the layout is approaching its near-optimal configuration. Figure 4.11 lists the steps used in both block movement and rotation. Both operations require almost the same amount of computation and thus consume approximately the same time. This observation indicates that an iteration in which a block movement is rejected will take a computation time twice as long as an iteration in which a block movement is accepted. The annealing process concentrates on the iterations with rejected block movement when the pseudo temperature approaches 0. This suggests that the block movement and rotation can be performed concurrently within a node processor. A block diagram of a node processor implementing these concurrent operations is shown in Figure 4.12. The rotation result is immediately available after a block movement is rejected. If the block movement is accepted, the rotation result is simply discarded. The concurrent generations of block movement and rotation will speed up the simulated annealing process, especially in its later stages. When the node processors are used for the mapping of clusters, the rotation generation mechanism is turned off since the rotation of clusters is not considered.

Local memory is provided in a node processor to store the necessary database. The most convenient format for representing the location and orientation of a building block is to use the transformation matrix

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \end{bmatrix}, \tag{4.12}$$

where the orientation of the block is indicated by t_{11}, t_{12}, t_{21}, and t_{22}, and its location is indicated by t_{13}, and t_{23}. The present coordinates of any point on a building block can be calculated by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{4.13}$$

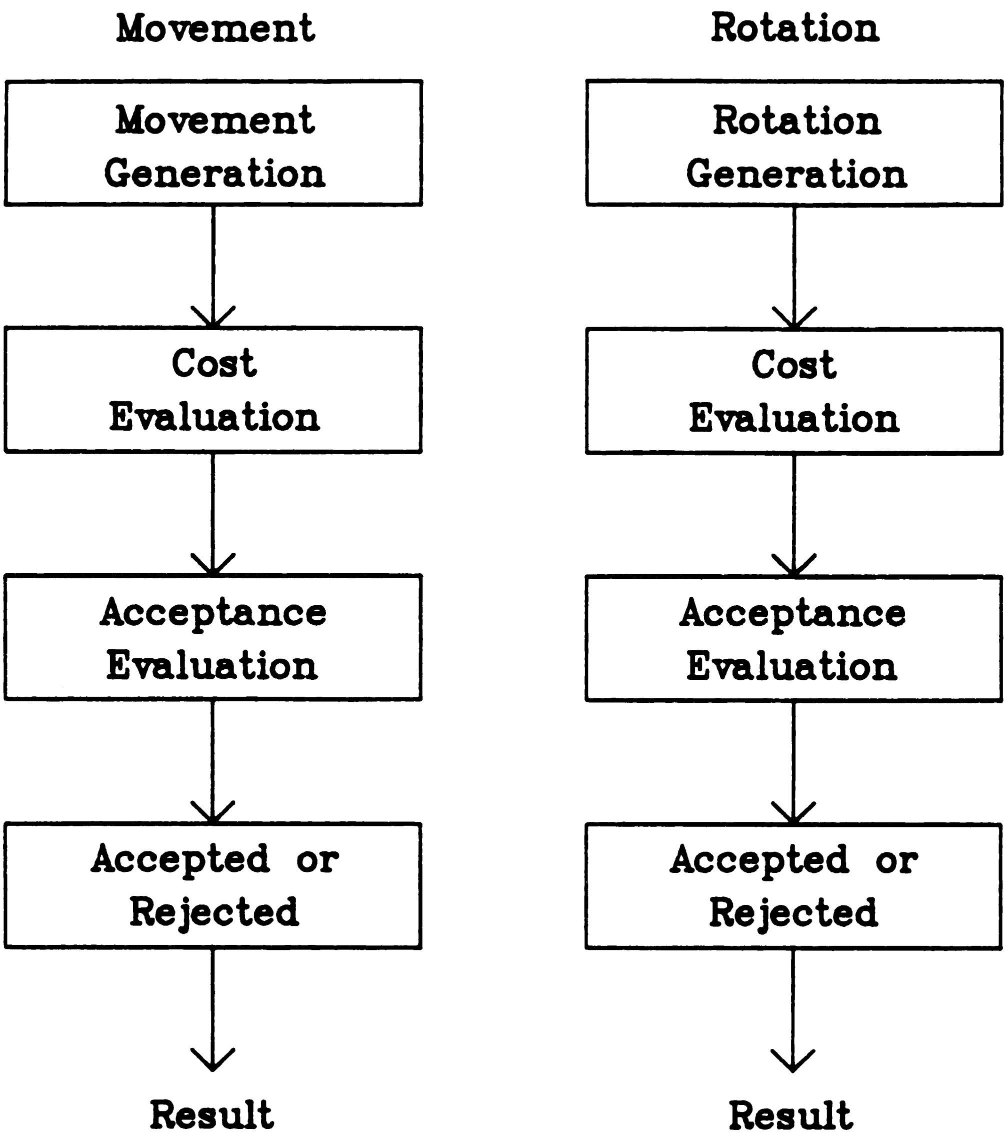


Figure 4.11 The operations of block movement and rotation.

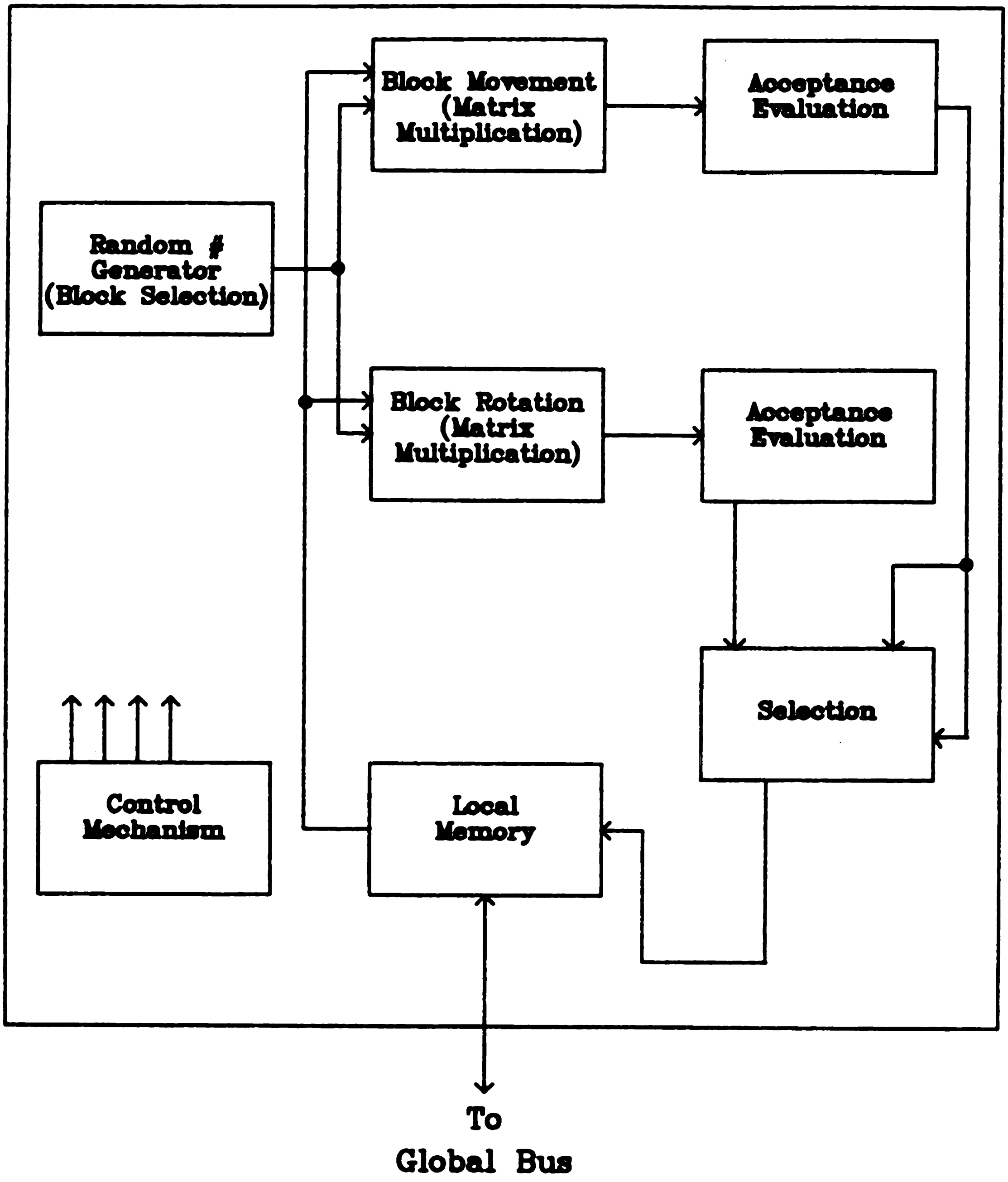

Block Movement (Matrix Multiplication)
Acceptance Evaluation
Random # Generator (Block Selection)
Block Rotation (Matrix Multiplication)
Acceptance Evaluation
Selection
Control Mechanism
Local Memory
To Global Bus

Figure 4.12 The block diagram for a node processor.

where x' and y' are the new coordinates of a point, designated originally by x and y. The calculation of the movement and rotation of a block generally involves the multiplication of matrices. It is therefore necessary to include a suitable architecture to handle matrix multiplication rapidly in a node processor. Several such architecture are available in the literature [41, 42].

CHAPTER V

TEST RESULTS AND CONCLUSIONS

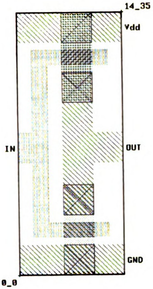
The partitioning and acceleration approaches developed for improving the building block placement process in VLSI design have been presented. In this chapter, the comparison of these approaches for practical design cases is presented and discussed. In order to evaluate the performance of these approaches, a benchmark is established using the concepts of the TimberWolf package [8]. As described in Chapter II, TimberWolf is an integrated set of placement and routing programs. The placement and global routing cost estimation algorithms are incorporated in the benchmark. The results obtained from the approaches developed in this research are compared against the benchmark. These results verify numerically that a significant amount of acceleration of the building block placement process using simulated annealing has been obtained at a very modest increase in cost. Finally, future trends in the VLSI design automation related to this research are discussed.

5.1 Design Cases

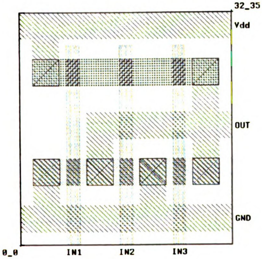
The new placement problem model has been tested on five design cases with varying numbers of building blocks and connectivities. The building blocks involved in these design cases are functional blocks with different sizes as typically used in a VLSI custom design. The complexity of these building blocks ranges from a logic inverter to an exclusive-OR (XOR) gate. All the building blocks in the library have been created on a Sun-3 workstation using Magic, the graphic layout tool developed at University of California at Berkeley [16]. SCMOS (Scalable Complimentary Metal Oxide Semiconductor) technology with double metal layer was used.

Two examples of the building blocks in this library, an inverter and a three-input NOR gate, are provided in Figure 5.1. After the internal structure of a functional circuit is laid out, the smallest rectangle that encloses the circuit is defined as the building block and the locations of its input/output terminals are indicated on its boundary. The rectangles enclosing the building blocks are also shown in Figure 5.1. These rectangles are treated as black boxes with fixed input/output terminals in the placement process.

The first and second design cases used in this research are of the same function, a 1-bit full adder, but they are constructed using different components. Figure 5.2 shows the schematic logic diagrams of these two full adder implementations [43]. The third design case chosen is a 4-bit BCLA (Block Carry Lookahead) unit with block carry generate/propagation functions [44]. Its schematic logic circuit is provided in Figure 5.3. The fourth design case is a 4-bit ALU (Arithmetic Logic Unit) with carry lookahead as shown in Figure 5.4 [45]. In order to test the performance of the proposed approaches working on a design case with a larger number of building blocks, an artificial test case is created by expanding the ALU circuit (Design Case No. 4). The last design case is provided by duplicating the ALU circuit 5 times to generate a circuit with a few hundred building blocks and interconnections. The parameters of these design cases, consisting of the number of building blocks and interconnecting nets, are listed in Table 5.1.



(a)



(b)

Figure 5.1 The layouts of (a) an inverter, and (b) a 3-input NOR.

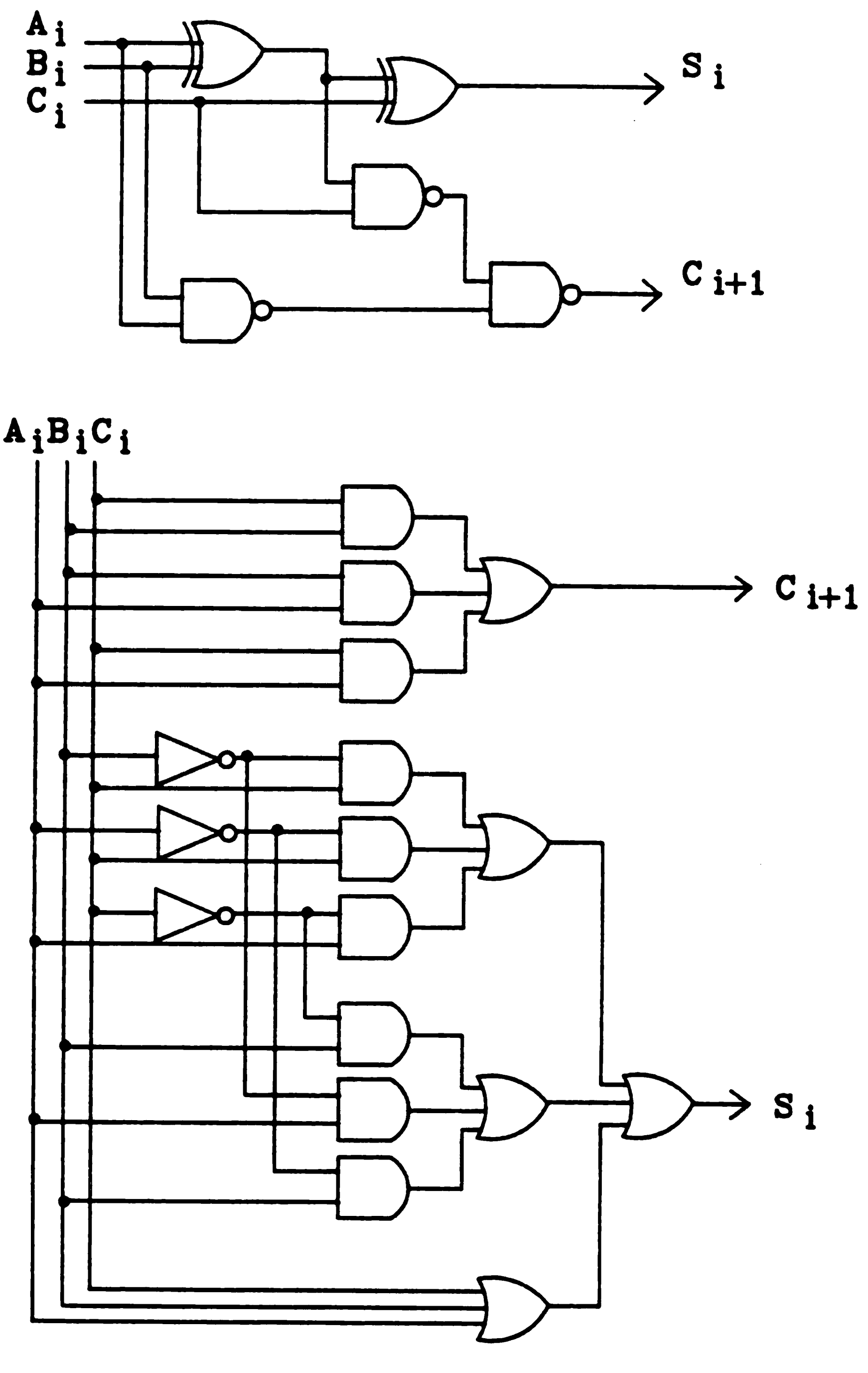


Figure 5.2 The schematic diagrams of two implementations of a 1-bit full adder [43].

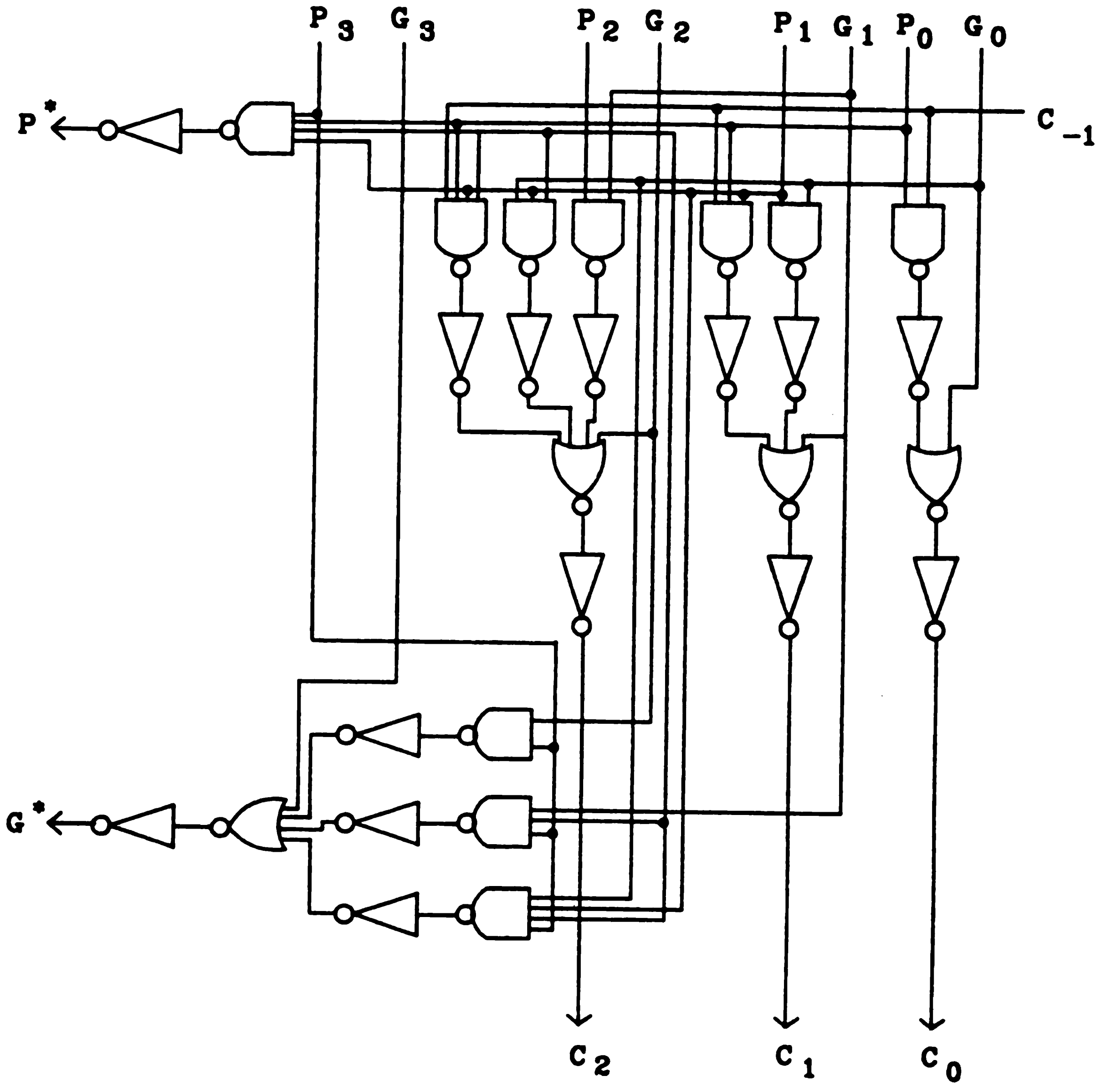


Figure 5.3 The schematic diagram of a 4-bit BCLA [44].

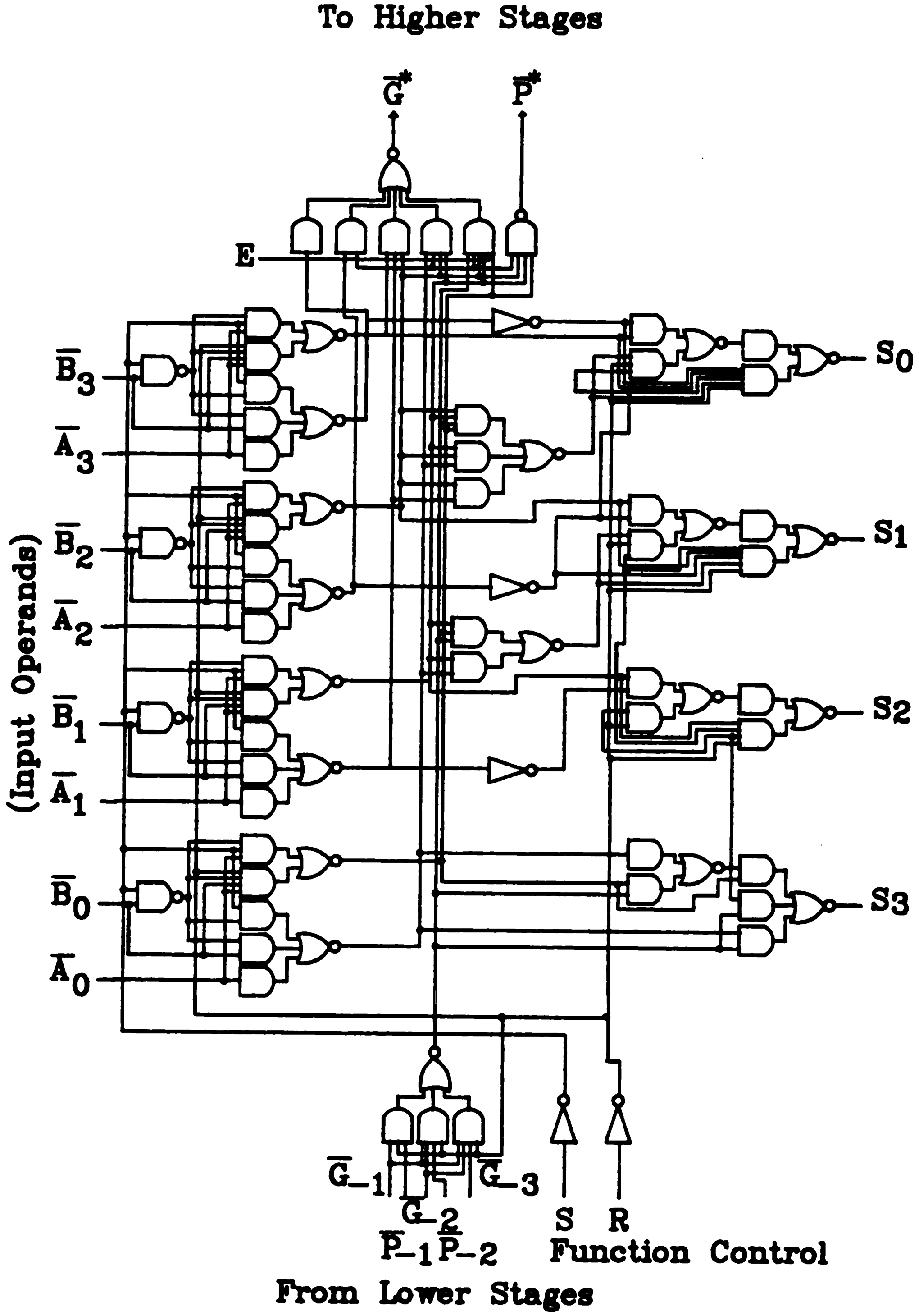


Figure 5.4 The schematic diagram of a 4-bit ALU [45].

Table 5.1. Parameters of the design cases tested in this research.

Case No.	Circuit Name	No. of Building Blocks	No. of Nets
1	Full Adder 1	5	6
2	Full Adder 2	17	18
3	BCLA	28	31
4	ALU	80	87
5	ALU X 5	400	435

5.2 Implementation of Test Programs

Test programs were developed to implement the approaches presented in the previous chapters. They include the FS and FSWPB algorithms utilizing the new placement problem model (Section 3.2), and the parallel placement method (Section 4.2). In this section the implementation description of the placement process in these programs and the benchmark is discussed. The specific details are described in later sections along with their results. All test programs and the benchmark are coded in the C language and the results were obtained on a VAX 8600 operating under UNIX. The data structure and details of the test programs are designed to be as close as possible to the benchmark program to make the comparison meaningful.

5.2.1 New Configuration Generation

The programs begin with a random initial placement configuration. A new configuration is generated by introducing small random modifications to the present configuration. Both the benchmark and the test programs implement the same set of possible modifications. Permitted movements in the problem space consist of moving a block to a new location, exchanging the locations of two blocks, rotating a block by multiples of 90^o, and mirror imaging a block. This set of movements is rich enough to reach all possible solutions.

The building blocks (including the submodules in the case of the FSWPB algorithm) are numbered from 1 to *n*, where *n* is the number of building blocks. The generation of a new configuration is determined by two random numbers. An initial random number, between 1 and *n* is generated. The block with that number is then picked for movement. The TimberWolf package chose 5 to be the ratio of single block displacements to block interchanges [8]. This ratio is also implemented in this research to make the comparison meaningful. The generation of the second random number is weighted to produce the desired ratio. This occurs by generating a random number between 1 and *m*, where *m* is *n* multiplied by the desired ratio. If the two numbers selected both represent blocks, then the pair of blocks are interchanged to generate a new configuration. Otherwise, if the second number selected does not represent a block, the block indicated by the first number is moved to a new location. In the benchmark program, the new location is randomly picked by generating two numbers within the coordinate system of the chip, while in the programs of this research, an empty slot is chosen randomly to accommodate the block to be moved. If this new configuration is rejected, then an orientation change of the first block chosen is attempted. Five orientation changes of a building block are considered in the programs: rotating a block (by 90^o, 180^o, and 270^o), and mirror imaging a block (left-right and upside-down). A random number between 0 and 4 is generated to select the type of orientation change.

In the later stages of the simulated annealing process, when the value of the pseudo-temperature approaches zero, the displacement of a block has very little chance of being accepted unless the displacement is very local. Similarly, an exchange of distant blocks has a slim chance of being accepted. Hence, it is more efficient to employ a range limiter, which limits the range of the displacement of a block during the later stages of the simulated annealing process.

5.2.2 Cost Function

The cost of a configuration is defined as the total estimated wire length, approximated by summing the estimated lengths of all interconnecting nets. The wire length of an interconnecting net is estimated by one half of the perimeter of the smallest rectangle that encloses all the terminals involved in the net. This cost function can be expressed as

$$C = \sum_{i=1}^{k} L_i, \tag{5.1}$$

where C is the cost of a configuration, k is the number of interconnecting nets, and L_i is the estimated wiring length of net i. Following TimberWolf, the cost function in the benchmark program includes a second portion consisting of a total sum of overlap penalties to deal with the block overlap [8]. This cost function can be expressed as

$$C_b = \sum_{i=1}^{k} L_i + \frac{A^2}{T}, \tag{5.2}$$

where C_b is the cost of a configuration, k is the number of interconnecting nets, L_i is the estimated wiring length of net i, A is the total overlap area, and T is the pseudo-temperature. According to this cost function, the penalty on the overlap area becomes more severe as the pseudo-temperature decreases.

5.2.3 Pseudo-Temperature

In the current implementation of the test programs and the benchmark, the pseudo-temperature is decreased according to equation (2.2). The best results have been obtained by introducing a variable α. The assignment of α is summarized in the following equation.

$$\alpha = \begin{cases} 0.85 & \text{if } T \geq 100, \\ 0.95 & \text{if } 100 > T \geq 40, \\ 0.8 & \text{if } T < 40. \end{cases} \tag{5.3}$$

For the annealing process implemented in this research, the pseudo-temperature is assigned to be $T = 1000$ at the beginning of the process.

5.2.4 Inner Loop Criterion

The inner loop criterion is implemented by the specification of the number of new configurations generated for each stage of the annealing process. This number is specified as a multiple of the number of building blocks for the placement problem. For the approaches developed in this research, 20 new configurations per block are generated at each stage. The benchmark program has many more degrees of freedom (movement to any location on the chip), and hence 100 new configurations are generated per block at each stage.

5.2.5 Stopping Criterion

The stopping criterion is implemented by recording the cost function's value at the end of each stage of the annealing process. The stopping criterion is satisfied when the cost function's value has not changed for 5 consecutive stages. In the FSWPB algorithm utilizing the new placement problem model, the stopping criterion for the process is temperorarily ignored prior to a user-specified switching temperature, at which the large blocks are partitioned and the chip slots are redefined.

5.3 New Problem Model Result

The FS and FSWPB algorithms applying the new placement problem model to the simulated annealing process have been tested. Partitioning of the building blocks is not used in the first approach (FS) and the chip is divided into an array of slots, each of which is large enough to hold the largest building block. This approach is especially suited to applications for which the blocks are approximately the same size. The second approach (FSWPB), more appropriate for the building blocks with varied sizes, starts the placement process as in the first approach with a slot size determined by the largest building block. The stopping criterion, however, is temporarily ignored until a user specified switching temperature is reached. At the switching temperature, the building blocks are partitioned into submodules with pseudo interconnections and the configuration at that temperature is used as the initial placement for the remaining stages.

The programs have been tested on the design cases discussed above in Table 5.1. The test results of these two approaches, including the placement cost and CPU time, are listed in Tables 5.2 and 5.3. As explained in the previous sections, the new placement model reduces the solution space of the problem. Using the new model, the programs need to generate only 20 new configurations per block at each temperature stage to produce final results comparable to the benchmark program that generates 100 new configurations per block at each temperature. For the test results obtained with partitioned blocks, the switching temperature is set at $T = 20$. The cost of the pseudo-interconnections is calculated by C^2/T, where C is the estimated wiring length due to the pseudo-interconnections and T is the temperature. The time and cost comparison ratios are calculated by dividing the computation times and costs, respectively, by those obtained from the benchmark.

In all the design cases presented and other cases that have been tested, the placement costs, defined by the total estimated wiring distance, are comparable to the costs obtained in the benchmark; however, the computation time has been significantly reduced by applying this new model. This is expected due to the reduction of the solution space. In addition, the computations of overlap area and the penalty term are eliminated. The fact that a reduced number of new configurations per building block are generated allows the algorithm to converge to a near optimal solution much faster.

It has been found that the switching temperature plays an important role in determining the performance. Experiments show that too high a switching temperature, i.e., partitioning the blocks at the beginning of the annealing process, produces inferior results. A small number of submodules may not be able to cluster into neighbor locations. This problem can be explained by the fact that the method of simulated

Table 5.2. The test results of the FS algorithm.

Design Case	FS		Benchmark		Comparison Ratio	
	Time (mins.)	Cost	Time (mins.)	Cost	Time	Cost
1	0.08	335	0.97	266	0.082	1.33
2	0.75	1470	19.91	1259	0.038	1.17
3	0.77	3058	81.27	2088	0.009	1.46
4	5.24	13640	149.69	10971	0.035	1.24
5	70.21	85182	2033.23	63859	0.035	1.33

annealing is developed with statistics. In the FSWPB, this problem is solved by the establishment of a switching temperature. The placement is divided into two phases, the first phase develops the relative locations of the building blocks and the second phase shakes the configuration to further improve the placement cost and reduce the chip area.

The increased number of blocks and larger connectivities caused by the partitioning of building blocks into submodules has an adverse effect on the computation time. In the design cases tested, the size of the placement problem is enlarged about five times after the partitioning. The computation times still favor this model and thus this effect can be compensated for by the significant gain in computation speed.

Table 5.3. The test results of the FSWPB algorithm.

Design Case	FSWPB		Benchmark		Comparison Ratio	
	Time (mins.)	Cost	Time (mins.)	Cost	Time	Cost
1	0.21	276	0.97	266	0.216	1.037
2	1.14	1365	19.91	1259	0.057	1.084
3	0.92	2185	81.27	2088	0.011	1.046
4	7.73	11671	149.69	10971	0.052	1.064
5	166.76	68521	2033.23	63859	0.082	1.073

5.4 Parallel Placement Result

The parallel placement method and the associated mapping scheme are tested on Design Cases 1 to 4. As mentioned in Section 4.2, the generation of the coherency matrix has to follow the definition of the cost function to ensure the reservation of the spatial localities in the configuration. Since the placement cost is determined by the estimated total wiring length, the coherency between building blocks is calculated by their interconnections. If an exclusive connection exists between two blocks, the coherency assigned to this connections is one unit. Otherwise, the connection involves more than two blocks. The coherency between any two blocks in the group defined by this connection is the reciprocal of the number of blocks connected.

The FS placement results generated in Section 5.3 for Design Cases 1 to 4 are presented in Figures 5.5 to 5.8. The clusters of building blocks constructed by the mapping scheme are shown in these figures by different shadings. As demonstrated by the mapping results, the spatial localities of a placement configurations are retained in the partitioning scheme. The blocks clustered as a group are neighbors in the configuration generated sequentially. Thus, the undesired effect of partitioning the placement problem is minimized.

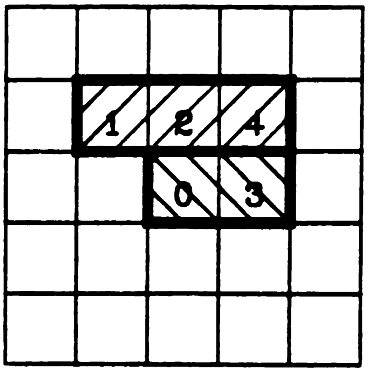


Figure 5.5 The mapping result of Design Case No. 1 on its FS placement.

After the mapping operation is completed, the parallel placement method described in Section 4.2 is applied to find the relative locations of the clusters and the locations of the blocks within their respective clusters. The inter-cluster connections are used to determine the relative locations of the clusters, which are used along with the intra-cluster connections to place the blocks within a cluster. Tables 5.4 provides the test results in all 4 design cases.

As indicated in the test results, the mapping scheme has the capability of discovering and retaining the spatial localities in the placement problem. Finally, the

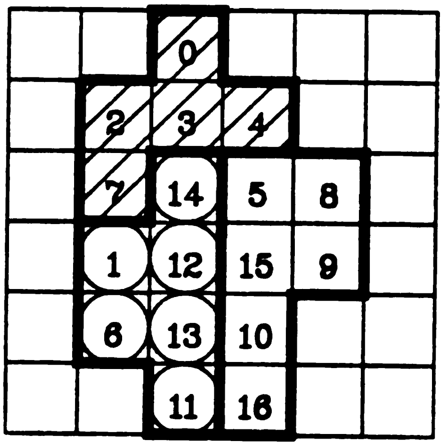


Figure 5.6 The mapping result of Design Case No. 2 on its FS placement.

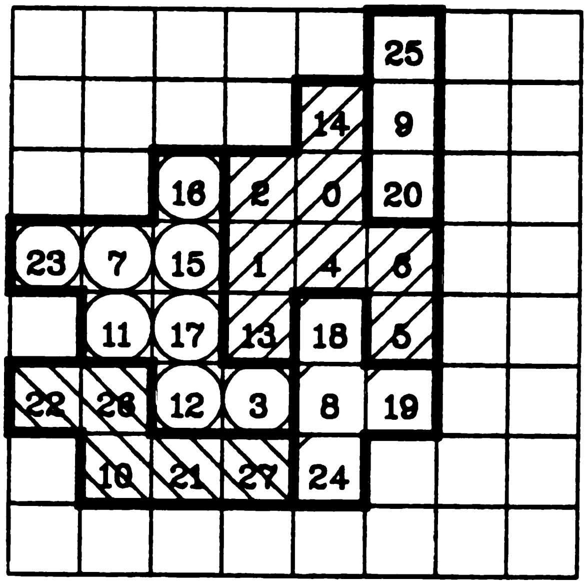
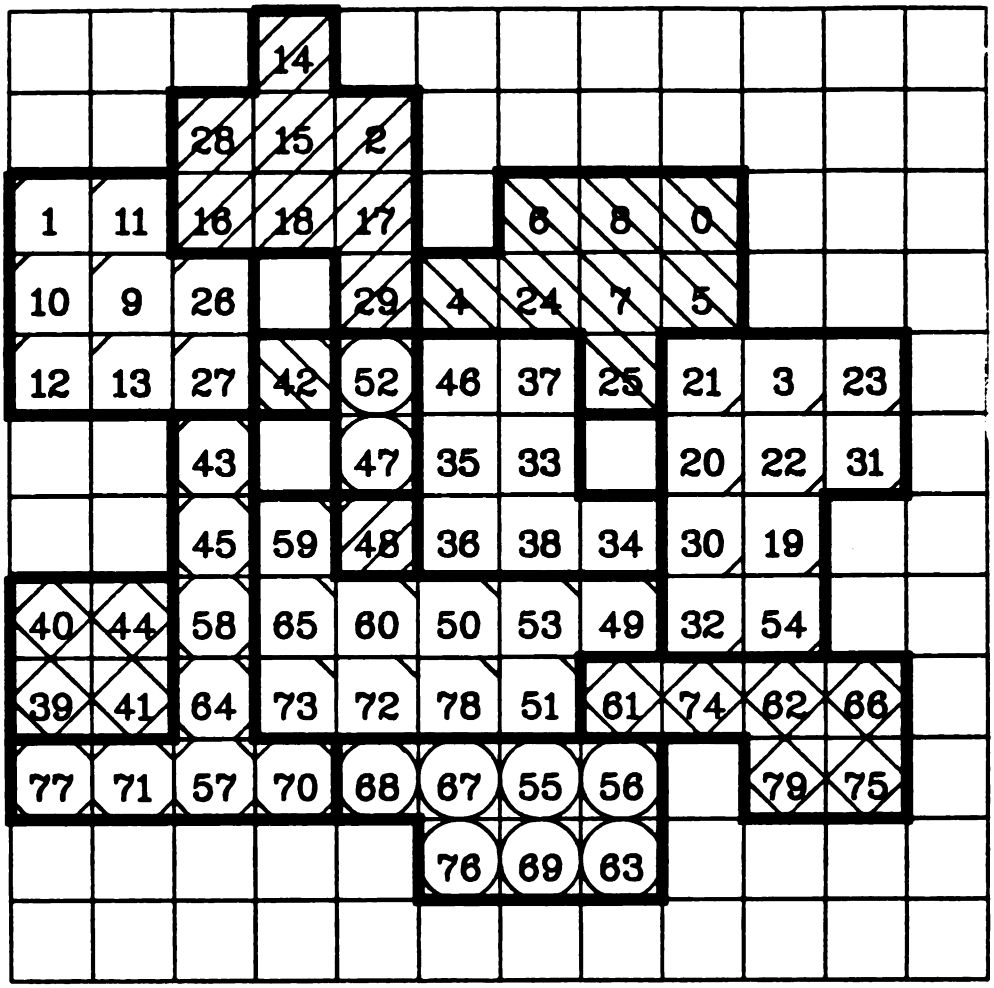


Figure 5.7 The mapping result of Design Case No. 3 on its FS placement.

hierarchical placement method generates configurations for the design cases, the costs of which are comparable to those obtained sequentially but with a significantly reduced computation time.



			14								
		28	15	2							
1	11	16	18	17		6	8	0			
10	9	26		29	4	24	7	5			
12	13	27	42	52	46	37	25	21	3	23	
		43		47	35	33		20	22	31	
		45	59	48	36	38	34	30	19		
40	44	58	65	60	50	53	49	32	54		
39	41	64	73	72	78	51	61	74	62	66	
77	71	57	70	68	67	55	56		79	75	
					76	69	63				

Figure 5.8 The mapping result of Design Case No. 4 on its FS placement.

Table 5.4. The results of the parallel placement method.

Design Case	Parallel			Benchmark		Comparison Ratio	
	No. of Processors	Time (mins.)	Cost	Time (mins.)	Cost	Time	Cost
1	2	0.048	392	0.97	266	0.049	1.47
2	3	0.19	1366	19.91	1259	0.010	1.08
3	5	0.20	2805	81.27	2088	0.002	1.34
4	9	1.6	14132	149.69	10971	0.011	1.29

5.5 Contributions

The building block placement process using simulated annealing has been accelerated and improved in a number of ways in this research. A new model for representing the placement problem in a CAE system was presented in Section 3.2. The critical contribution of this model is that the size of the problem space in a simulated annealing placement process was significantly reduced so that the process converges to a near-optimal solution more rapidly. In addition, the drawbacks of the conventional models are eliminated without sacrificing their advantages. Several goals have been achieved by this model. The sizes and the shapes of the building blocks and their terminal locations are considered. By taking this information into consideration, the placement process can produce a more practical configuration with increased routability. Moreover, this model saves the overhead required to eliminate overlap between building blocks during or after the placement process. Finally, it is applicable to any hierarchical VLSI design methodology.

Two improved building block placement algorithms using simulated annealing, which implement this placement problem model, are developed in this research. The performance, including the placement quality and the computation time, of these algorithms was tested and evaluated. Test results on different examples show that the placement results are comparable to those obtained by the benchmark program and the computation time is reduced by more than an order of magnitude.

An explicit mapping scheme for the building block placement problem was developed. This mapping scheme locates the clusters of neighbor building blocks on a placement configuration produced sequentially. By means of anticipating the building block clusters in the layout, the spatial localities of building blocks are thus retained and the effect of partitioning the placement problem is minimized. The mapping scheme maps the building blocks into different regions of a chip so that concurrent simulated annealing process can be applied to the placement problem. A parallel simulated annealing placement method was also developed and tested. The placement results obtained for the test cases are degraded, in the worse case, by a factor of about 50%. But, the computation time of the placement process is shown to be reduced by as much as 2 orders-of-magnitude. Even though this represents testing on only several cases and is not rigorously proven, the evidence once again indicates a balance of modest placement degradation contrasted to orders-of-magnitude time savings.

Finally, the architecture of a VLSI placement engine which applies the power of parallel processing to building block placement problem is proposed and discussed in Section 4.4. The placement engine requires only a few different, simple processing elements so that its design and test can be simplified. The design is also modular and extensible, so one can create a large processor by combining the designs of smaller cells and/or chips. The placement engine can be utilized as a coprocessor in a CAE system for any VLSI design technology.

The approaches developed in this dissertation can easily be adapted for use in other placement algorithms. However, due to the near optimum characteristic of the simulated annealing process, they are especially suitable for use in this class of algorithms.

5.6 Trends in VLSI Design Automation and Future Study

There is a strong relationship between VLSI technology and CAE tools. As the density of components on a VLSI-based structure grows, the design process, especially the physical design stage, becomes heavily dependent on automated design tools. Better CAE tools will facilitate the improvement of VLSI technology, and vice versa. Several approaches for the improvement of the building block placement process in VLSI design have been studied in this dissertation. In order to reduce the overhead of a general purpose computer, the architecture of a dedicated VLSI placement engine is proposed in this work. The implementation of this placement engine onto an ASIC and its evaluation are thus an essential next step of this research.

Further improvement can be made to the placement algorithms provided in this research. Due to the wasted area in the slots defined in the new model of this research, the wiring distance is generally higher than that of an optimal result. After a configuration is established, a good compaction scheme can be used following the placement to reduce the wasted chip area between the building blocks. The compaction of a configuration, which is also an NP-complete combinatorial optimization problem, is at least as hard as the placement process itself. The traditional heuristic compaction schemes suffer from the local minimum problem and often destroy the optimal relationships between the blocks on the chip generated by the placement process. The simulated annealing approach should be a good means to attack this problem, and should be further examined.

It is impossible to produce a good placement configuration without taking the later routing process into consideration. This observation is quite easy to understand since a placement process does not complete the design problem until the router can successfully finish the connections between the building modules. The traditional research in VLSI design automation studies the placement and routing problems separately, which is quite impractical. A future research direction in this area is to combine the placement and routing into one process to produce a more practical configuration.

One major problem arises in the routing process when the router determines that the space, or channel, reserved for the routing in a certain area is not sufficient to complete all the necessary interconnections. When this situation occurs, the router has several choices. The router can quit and inform the designer that the routing cannot proceed further. The designer can try to complete the routing manually. The router can also rip up some of the connections and reroute the wires in the neighborhood. It is very likely that no matter how hard the designer or the router tries, the routing will fail eventually. The configuration must then be rearranged to provide more space in the congested area for the interconnections. In a combined placement and routing process, the placement process can accurately anticipate the routability of the configuration so that it produces and guarantees 100% routability. The compaction and expansion of a configuration will play an important role in this process. When it is determined that a certain area is too congested for routing, the building blocks in that area can be moved apart (i.e., expansion) to generate a wider channel. On the other hand, if it is found that the space reserved in a certain chip area is too dispersed, the building blocks in the neighborhood can be moved closer (i.e., compaction) to reduce the size of the chip. The compaction and expansion of a placement configuration must be, of course, done in a manner that does not destroy the relationships between the moved building blocks and the other blocks.

Present ASIC design technology is dominated by the gate array approach. Since only the metallization step in the fabrication is missing, the mass production of identical gate array chips is possible. The design of a gate array chip can be completed in a rather short time by personalization. Design costs are low; however, the performance of a chip designed by this approach is sub-optimal due to the fixed sizes and locations of the components on the chip. The trend in ASIC design technology is shifting from the gate array approach to the standard cell approach. Even though the standard cells still have a constraint of identical heights, the design of a cell is more flexible than those in the gate array approach. Better performance is thus possible with good cell design and careful layout. This trend can be projected to anticipate that future ASIC design technology will involve the utilization of mega-cells. Two reasons can be given to justify this anticipation. After all the constraints on the design of a library cell are removed, the cell can be readily optimized, which will eventually result in a better ASIC performance. Furthermore, the design methodology of mega-cells will be more similar to the traditional circuit design using off-the-shelf components. Due to the typically varied height and width of mega-cells, their placement and routing are much more difficult and the compaction and expansion operations discussed above become extremely important for a good layout.

Instead of generating the entire circuit from scratch, the trend of silicon compilation is to generate the layout using predesigned library cells. Since the library cells have to be universal in the sense that they can be fetched and used in any future design, the parameters of the cells themselves, such as I/O driving capability, are not optimized for a certain application. This inflexibility reduces the performance of the chip produced. One trivial solution to this problem is to prepare and store cells of the same function but with different parameters. This will greatly increase the requirement on the library memory and create problems in database management. A possible future research direction in silicon compilation is to store a basic model of a cell in the library and optimize it on-the-fly according to a specific application when it is called up.

In conclusion, current trends project that design automation shows great promise in VLSI circuit design. As the computer-assisted design tools are improved, the performance gap between the chips produced by a silicon compiler and full custom design will eventually be removed. These new techniques will allow any suitable algorithm or architecture to be implemented inexpensively on silicon chip in a relatively short turn-around time without concern to design complexity or chip performance.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Niessen, C., "Hierarchical Design Methodologies and Tools for VLSI Chips," *Proc. IEEE,* Vol. 71, No. 1 (January 1983), pp. 66-75.

2. Wallich, P., "The One-Month Chip: Design," *IEEE Spectrum,* Vol. 21, No. 9 (September 1984), pp. 30-34.

3. Vai, M.-K. and Shanblatt M. A., *Performance-Design Tradeoff of Hierarchical VLSI Design Entry Points,* Tech. Report MSU-ENGR-85-020, Michigan State University, East Lansing, Michigan, (August 1985).

4. Donath, W. E., "Complexity Theory and Design Automation," *Proc. 17th Design Automation Conference,* (June 1980), pp. 412-419.

5. Preas, B. T. and Karger, P. G., "Automatic Placement A Review of Current Techniques," *Proc. 23rd Design Automation Conference,* (June 1986), pp. 622-629.

6. Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P., "Optimization by Simulated Annealing," *Science,* Vol. 220, No. 4598 (May 13, 1983), pp. 671-680.

7. Romeo, F., Sangiovanni-Vincentelli, A., "Research on Simulated Annealing at Berkeley," *Proc. 1984 International Conference on Computer Design,* (October 1984), pp. 652-680.

8. Sechen, C. and Sangiovanni-Vincentelli, A., "The Timberwolf Placement and Routing Package," *Proc. 1984 Custom Intgrated Circuits Conference,* (May 1984), pp. 522-527.

9. Lauther, U., "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Proc. 16th Design Automation Conference,* (June 1979), pp. 1-10.

10. Kravitz, S. A. and Rutenber, R. A., "Multiprocessor-Based Placement by Simulated Annealing," *Proc. 23rd Design Automation Conference,* (June 1986), pp. 157-162.

11. Rutenber R. A. and Kravitz, S. A., "Layout by Annealing in a Parallel Environment," *Proc. 1986 International Conference on Computer Design,* (October 1986), pp. 434-437.

12. Casotto, A., Romeo, F. and Sangiovanni-Vincentelli, A., "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *Tech. Digest 1986 International Conference on Computer-Aided Design,* (September 1986), pp. 30-33.

13. Ueda, K., Komatsubara, T. and Hosaka, T., "A Parallel Processing Approach for Logic Module Placement," *IEEE Trans. on Computer-Aided Design of IC and Systems,* Vol. CAD-2, No. 1 (January 1983), pp. 39-47.

14. Losleben, P., "Computer Aided Design for VLSI," *Very Large Scale Integration VLSI: Fundermentals and Applications,* edited by Barbe, D. F., Springer-Verlag, Berlin, (1982), pp. 89-125.

15. The MOSIS Crew, "Chips and Boards Through MOSIS," *Digest Compcon Spring 85,* (February 1985), pp. 184-186.

16. Ousterhout, J. K., Hamachi, G. T., Mayo, R. N., Scott, W. S. and Taylor, G. S., "The Magic VLSI Layout System," *IEEE Design and Test,* Vol. 2 No. 1 (February, 1985), pp. 19-30.

17. Waller, L., "Can Distributors Stake out a Claim in the ASIC Business?" *Electronics,* Vol. 59, No. 65 (August 7, 1986), pp. 124-125.

18. Newton, A. R., "A Survey of Computer Aids for VLSI Layout," *Tech. Digest 1982 Symposium on VLSI Technology,* (September 1982), pp. 66-75.

19. Newton, A. R. and Sangiovanni-Vincentelli, A. L., "Computer-Aided Design for VLSI Circuits," *IEEE Computer,* Vol. 19, No. 4 (April 1986), pp. 38-60.

20. Wallich, P. "A Review of Engineering Workstations," *IEEE Spectrum,* Vol. 21, No. 10 (October 1984), pp. 48-53.

21. Daniel, M. E. and Gwyn, C. W., "CAD Systems for IC Design," *IEEE Trans. on Computer-Aided Design of IC and Systems,* Vol. CAD-1, No. 1 (January 1982), pp. 2-12.

22. Mead, C. and Conway, L., *Introduction to VLSI Systems,* Addison-Wesley Pub. Co., Reading, Massachusetts (1980), pp. 115-127.

23. Crawford, J. F., "EDIF: A mechanism for the Exchange of Design Information," *IEEE Design & Test,* Vol. 2, No. 1 (February 1985), pp. 63-69.

24. German, S. M. and Lieberherr, K. J., "Zeus: A Language for Expressing Algorithms in Hardware," *IEEE Computer,* Vol. 18, No. 2 (February 1985), pp. 55-65.

25. Lieberherr, K. J., "Toward a Standard Hardware Description Language," *IEEE Design & Test,* Vol. 2, No. 1 (February 1985), pp. 55-62.

26. Shahdad, M., Lipsett, R., Marschner, E., Sheehan, K., Cohen, H., Waxman, R. and Ackley, D., "VHSIC Hardware Description Language," *IEEE Computer,* Vol. 18, No. 7 (July 1985), pp. 50-52.

27. Mead, C. and Conway, L., *Introduction to VLSI Systems,* Addison-Wesley Pub. Co., Reading, Massachusetts (1980), pp. 47-51.

28. Weste, N. and Eshraghian, K., *Principles of CMOS VLSI Design A System Perspective,*" Addison-Wesley Pub. Co., Reading, Massachusetts (1985), pp. 98-113.

29. Hwang, K., *Computer Arithmetic Principles, Architecture, and Design,* John Wiley & Sons, New York, (1979), p. 184.

30. Akers, S. B., "On the Use of the Linear Assignment Algorithm in Module Placement," *Proc. 18th Design Automation Conference,* (June 1981), pp. 137-144.

31. Rivest, R. L., "The "PI" (Placement and Interconnect) System," *Proc. 19th Design Automation Conference,* (June 1982), pp. 125-127.

32. Dunlop, A. E. and Kernighan, B. W., "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on Computer-Aided Design of IC and Systems,* Vol. CAD-4, No. 1 (January 1985), pp. 92-98.

33. Ueda, K., Komatsubara, T. and Hosaka, T., "A Parallel Processing Approach for Logic Module Placement," *IEEE Trans. on Computer-Aided Design of IC and Systems,* Vol. CAD-4, No. 1 (January 1985), pp. 12-22.

34. Sha, L. and Dutton, R. W., "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks," *Proc. 22nd Design Automation Conference,* (June 1985), pp. 602-608.

35. Wipfler, G. J., Wiesel, M. and Mlynski, D. A., "A Combined Force and Cut Algorithm for Hierarchical VLSI Layout," *Proc. 19th Design Automation Conference,* (June 1982), pp. 671-677.

36. Quinn, N. R., Jr. and Breuer, M. A., "A Force Directed Component Placement Procedure for Printed Circuit Boards," *IEEE Trans. on Circuits and Systems,* Vol. 32, No. 6 (June 1979), pp. 377-388.

37. Ross, S. M., *Introduction to Probability Models,* Academic Press Inc., London, England, (1985), pp. 132-187.

38. Wah, B. W., Li, G.-J., and Yu, C. F., "Multiprocessing of Combinatorial Search Problems," *IEEE Computer,* Vol. 18, No. 6 (June 1985), pp. 93-108.

39. Hwang, K., "Multiprocessor Supercomputers for Scientific/Engineering Applications," *IEEE Computer,* Vol. 18, No. 6 (June 1985), pp. 57-73.

40. Mccluskey, E. J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits,* Prentice-Hall, Englewood Cliffs, New Jersey, (1986), pp. 457-462.

41. Kung, H. T., "The Structure of Parallel Algorithms," *Advances in Computers,* Vol. 19, Academic Press, New York (1980), pp. 65-112.

42. Hwang, K. and Cheng, Y. H., *VLSI Arithmetic Arrays and Modular Networks for Solving Large-Scale Linear System of Equations,* Tech. Report No. TR-EE 80-4, Purdue University, W. Lafayette, Indiana, (March, 1980).

43. Hwang, K., *Computer Arithmetic Principles, Architecture, and Design,* John Wiley & Sons, New York, (1979), p. 42.

44. Hwang, K., *Computer Arithmetic Principles, Architecture, and Design,* John Wiley & Sons, New York, (1979), p. 87.

45. Hwang, K., *Computer Arithmetic Principles, Architecture, and Design,* John Wiley & Sons, New York, (1979), p. 115.