Abstract

Title of Dissertation: Efficient Algorithms for VLSI Placement and Routing Problems

Shih-Chuan Hung, Doctor of Philosophy, 1996

Dissertation directed by: Professor Ronald I. Greenberg
Department of Electrical Engineering

This thesis is primarily concerned with routing problems in VLSI physical layout design. For both single-layer and over-the-cell wiring models, efficient channel routing algorithms are developed to minimize standard cost measures. In addition, we provide an empirical comparison of placement and global routing algorithms for FPGAs.

In the area of single-layer channel routing, this thesis considers three specific problems, namely the minimum separation, offset range, and optimal offset problems, in parallel computing models. For the minimum separation problem, we obtain $O(\lg N)$ time on a CREW PRAM or $O(\frac{\lg N}{\lg \lg N})$ time on a (common) CRCW PRAM, both with optimal work (processor-time product) of $O(N)$, where N is the number of terminals. For the offset range problem, we obtain the same time and processor bounds as long as only one side of the channel contains single-sided nets. For the optimal offset problem with single-sided nets on one side

of the channel, we obtain time $O(\lg N \lg\lg N)$ on a CREW PRAM or $O(\frac{\lg N}{\lg\lg N})$ time on a CRCW PRAM with $O(N \lg\lg N)$ work. Not only does this improve on previous results for river routing, but we can obtain an even better time of $O((\lg\lg N)^2)$ on the CRCW PRAM in the river routing context. In addition, wherever our results allow a channel boundary to contain single-sided nets, the results also apply when that boundary is ragged and N incorporates the number of bendpoints.

In the over-the-cell context, this thesis presents an efficient algorithm to reduce the channel density of a given channel. By choosing critical nets or subnets to route over the cells, our algorithm achieves better or equal channel density reduction with fewer tracks in the over-the-cell regions than previous works. In addition, this algorithm can easily be parallelized due to the independent nature of cost function computations for each interval.

The last part of this thesis shows the results of an empirical comparison of placement and global routing algorithms for FPGAs. We consider recently described algorithms that appear promising but that have not been compared on the same FPGA architecture and with the same benchmarks. We focus on a simultaneous placement and global routing scheme of Togawa et al. and what appears to be the leading scheme for sequential placement and routing embodied in the Vpr system of Betz. We compare the algorithms on two different FPGA architectures, using several benchmarks in each case.

Efficient Algorithms for VLSI Placement and Routing Problems

by

Shih-Chuan Hung

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1996

Advisory Committee:

Professor Ronald I. Greenberg, Chairman/Advisor
Professor Joseph JáJá
Professor Robert W. Newcomb
Professor Charles B. Silio
Professor Clyde P. Kruskal

Dedication

To my parents

Acknowledgements

I would like to express my sincere appreciation to my advisor, Ronald I. Greenberg, for his enlightening guidance and professional advice throughout this work. I would also like to thank Professors Joseph JáJá, Robert Newcomb, Charles Silioon and Clyde Kruskal on my thesis committee for offering helpful comments and suggestions.

I have benefited from the Electrical Engineering Department for the resources and stimulating research environment. All my friends also deserve my thanks for all the help they offered and for making my stay at Maryland memorable and pleasant.

Finally, I owe special thanks to my family for their unconditional support and constant encouragement during my Ph. D. study. Without their understanding and patience, this would not be possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In VLSI (Very Large Scale Integration) circuit design, one of the most crucial problems encountered is how to assign locations to circuit modules, predesigned pieces of the circuit, and to route the connections among them. These problems are part of the overall design process, throughout which it is necessary to satisfy technology constraints, such as minimum wire widths and separations, while attempting to optimize performance measures, such as area and delay. In general, this problem is difficult, and it is typical to break the overall problem into consecutive subtasks to help manage the complexity. But even most of the subtasks remain combinatorially intractable and this has led researchers to develop heuristics for these subproblems. Under some design styles, however, there may arise some subproblems that do have efficient exact solutions.

After decomposition, the problem of assigning locations to individual modules is commonly referred to as the *placement* problem. The *routing* problem then involves routing the connections of each net among terminals on the prepositioned modules. During the routing stage, a *global* or *loose routing* is performed first to break down each interconnection into segments and assign them to channels. Then, a *detailed* router is used to complete point-to-point connections of a net within or among channels. Routing is completed with a specified number of layers using *vias* to connect wires in adjacent layers. Although the primary objective in circuit design is to minimize the total area of the chip, it is also desirable to attend to performance by considering measures such as total wire length, number of vias, critical path constraints or more general delay measures.

This thesis is primarily focused on the routing problem with some attention also to the placement issue. We consider various design styles leading to specific forms of the placement and routing problems that allow some reduction of complexity, and we give some attention to both sequential and parallel computing models. The following section provides basic problem definitions and VLSI models used in this thesis.

1.1 Background

1.1.1 The Channel Routing Problem

The channel routing problem was introduced by Hashimoto and Stevens [33] in 1971, and has since been given much attention in a variety of technologies. This problem is the special case of wire routing in which interconnections are to be performed within a rectangular strip containing no obstructions, between terminals located on two opposite sides of the rectangle. Channel routing arises in many different methodologies of physical layout, e.g., gate arrays and standard cells.

In this section, we first give the formal definition of the channel routing problem and a few basic terminologies used in this thesis.

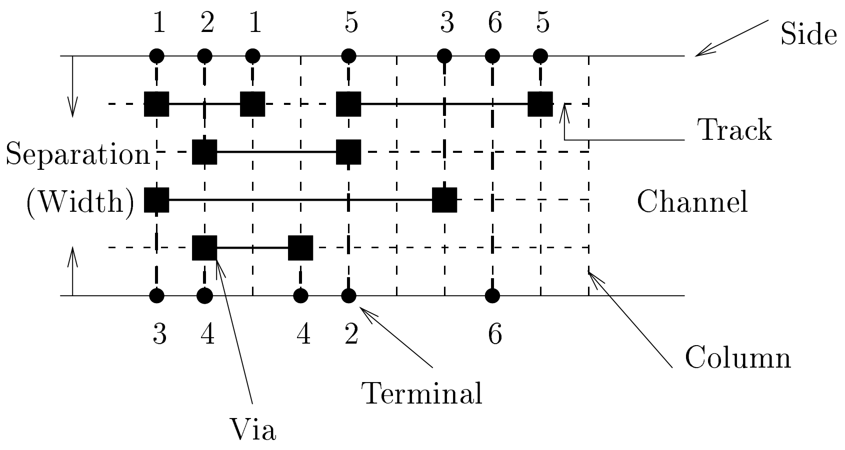


Figure 1.1: A channel routing instance. In this circuit, the *span* of the channel is 8 and the *span* of net 5 is 4. The width of the channel is 5.

A *channel* is a rectangular area bounded by two parallel lines, which are referred as *sides* of the channel. (For easy presentation, we draw these lines horizontally.) The labeled points along the sides of the channel are *terminals*, and terminals with the same label constitute a *net*. The channel routing problem is to connect all terminals of each net within the given channel in such a way that no two nets cross each other on the same layer. Often circuits are built up in an iterative or hierarchical fashion, whereby the placement is not quite fully specified before channel routing, i.e. the distance between the two sides of the channel is not specified. Given an instance, the prime objective is to find the *separation* or *width*, which is the minimum distance between the two sides of the channel required for a complete routing. (See Figure 1.1.)

For simplicity, we only consider a restricted model with rectilinear wires and all terminals residing at grid points. On this model, the horizontal grid lines between the two sides are called *rows* or *tracks*, and each vertical grid line is

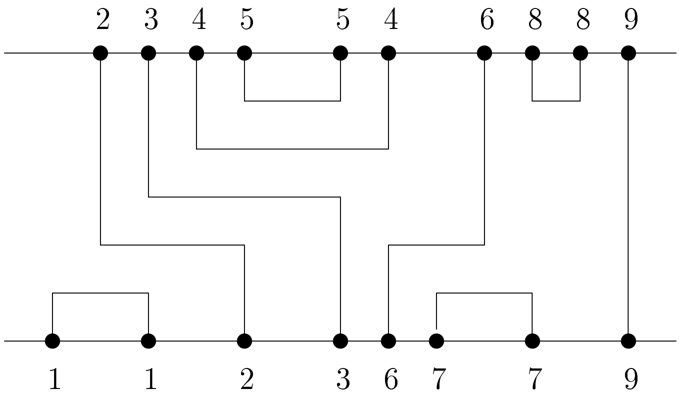


Figure 1.2: An example of a routed single-layer routing instance.

referred as a *column*. (Hence, the width of the channel is one more than the number of tracks.) The *span* of a channel is the closed interval between the rightmost and the leftmost terminals appearing on the two sides. Similarly, the span of a net is the interval between its leftmost and rightmost terminals.

In the literature, a variety of wiring models have been considered for different VLSI technologies and design styles. Each model lays out its own special rules on how wires can be realized on each layer and it plays a significant role in determining the key considerations and the algorithms to be employed. The primary criterion to classify wiring models is the number of conducting layers that are used, and the second concern deals with the rules of interconnecting and overlapping the layers.

We now introduce a number of wiring models to which results of this thesis are relevant.

- **Single-Layer Model**

  There is only one layer allowed in this model, which implies wires should not cross each other in the single-layer routing environment. Given an in-

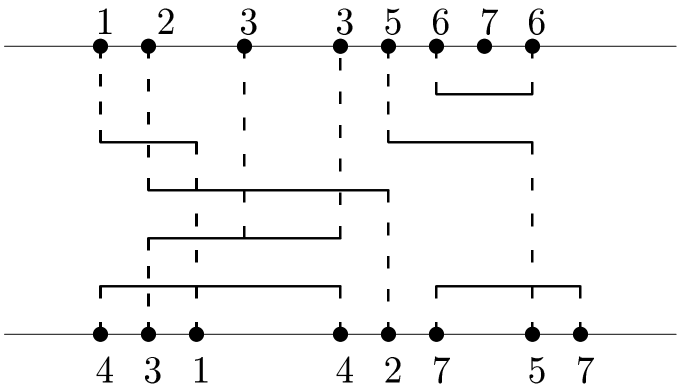


Figure 1.3: An example of a routed two-layer channel routing layout in the Manhattan wiring model. One layer is reserved for horizontal wires, and the second layer is reserved for vertical wires.

stance, this model can be applied only when the interconnection pattern is planar. A special case of this model, which has attracted a lot of attention, is called the *river routing* problem. It is essentially a single layer channel routing problem for two terminal nets, such that each net has one terminal on each side. Figure 1.2 shows a single-layer routing example.

- **The Manhattan Model**

  This is by far the most popular and extensively studied two-layer wiring model. In this model, one layer is preassigned to the horizontal tracks with the other layer reserved for the vertical tracks. If a wire has to change direction, a *via* or *contact* is used at the turning point. Thus paths may cross over each other, but can not share segments or turn at the same grid points. Figure 1.3 presents an example of this model.

- **Multi-Layer Model**

With the advance in VLSI technology, utilization of more than two layers for routing has become feasible. Ultimately, it may be possible to move towards a truly three-dimensional medium [26, 43, 48], but for now it is most reasonable to think of just several layers, each of which has a relatively high wire density in two dimensions. Often, some layers are reserved for only horizontal wires or vertical wires. Although the additional layers provide more degrees of freedom for the interconnection, it is difficult to use this freedom effectively. An effective strategy may be to divide a multi-layer problem into subproblems each dedicated to a subset of the wiring layers, some of which may be single-layer subproblems [25].

- **Over-the-Cell Model**

  In this model, we make use of the area outside the channel for routing, and generate a more compact layout. Over the years, this routing model has been successfully used in standard cell design to achieve considerable reduction in the channel width. Usually there is one layer reserved for the routing in the over-the-cell region, but recently studies have also proposed to use more than one layer in the over-the-cell area [41, 63].

Subject to all the design rules, there is a minimum width required for a complete routing in any given channel. There are a number of lower bounds, such as density, longest path in the vertical constraint graph and flux, that serve as predictors for the area requirements. The bounds are relative to a selected wiring model, and none of them is sufficient by itself, i.e. a different measure may dominate the others for any given channel.

Probably, the most generally useful lower bound is *density*.

Given a channel routing problem, the local density $d(i)$ of column i is the number of nets whose span contains i. The only exception is *trivial* nets (nets with just two terminals, both on the same column), since they don't contribute to the density of the columns where they reside. Define the channel density d as the maximum local density of all columns in the channel, e.g. $d = \max_i d(i)$.

With no wire overlap permitted in the Manhattan wiring model and the single layer wiring model, at least d tracks are needed to realize a successful layout. Therefore, $d+1$ is the obvious lower bound for computing the separation of the channel. On the other hand, some modification on this argument is needed when overlap between wire segments are allowed. For example, in the multiple layer wiring model, the channel width is at least $(d/k)+1$, where k is the number of layers on which horizontal routing is permitted.

Since most variations of the channel routing problem are not tractable, optimal algorithms are only obtainable for some restricted cases such as the river routing problem. Thus, the main focus is on efficient heuristics. A survey by LaPaugh and Printer [46] provides in-depth discussion for the channel routing problem.

1.1.2 FPGAs

Field-Programmable Gate Arrays (FPGAs) provide a new approach to ASIC implementation that features both large scale intergration and user programmability. An FPGA consists of a regular array of logic blocks that implement combinational and sequential logic functions and a user-programmable routing network that provides interconnections between the elements. In a conventional ASIC design, such as Mask Programmable Gate Arrays (MPGAs) and standard cells, the connections between logic blocks are implemented at a fabrication facility. In FPGAs, the connections are implemented in the field using a user-programmable routing network. Thus, the manufacturing turn-around times are reduced from weeks to minutes and the prototype costs are decreased from tens of thousands of dollars to hundreds of dollars [60]. The key advantage of FPGAs is reprogrammability, which allows in-circuit flexibility when the specifications are likely to change in the final application. In FPGAs, a data channel is provided for easy transfer of the new logic function and reprogramming the FPGA.

However, user-programmable routing devices have drawbacks: density and performance. Consisting of metal wire segments connected by programmable switches, the programmable connections take up greater area and incur greater delay than simple metal wires. To overcome the density drawback, FPGA architectures employ highly functional logic blocks that reduce the total number of logic blocks and hence the number of programmable connections necessary to implement a given design. These complex logic blocks also reduce the performance penalty by decreasing the number of logic blocks and programmable connections on the critical paths in the circuit.

Given a logic circuit, the layout is generated in the FPGA by partitioning the logic into individual logic blocks and then interconnecting these blocks as required via the programmable switches. There are several ways of designing an FPGA, which usually involve tradeoffs in the complexity and flexibility of both the logic blocks and the interconnection resources. Thus, the design of the logic blocks, coupled with that of the interconnection resources should be versatile, and facilitate the implementation of a wide variety of digital circuits.

In FPGAs, a logic block has a fixed number of inputs and outputs, and a

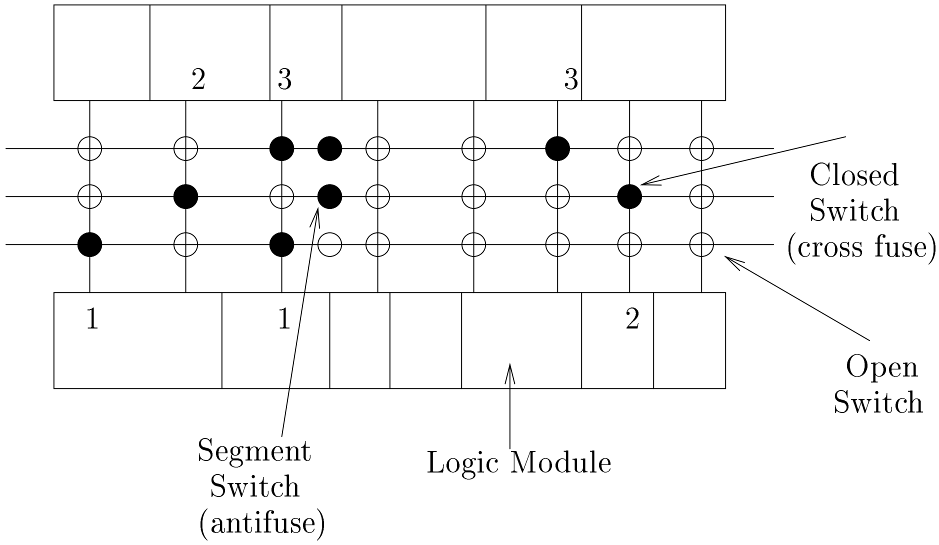


Figure 1.4: A segemented channel routing model for FPGAs.

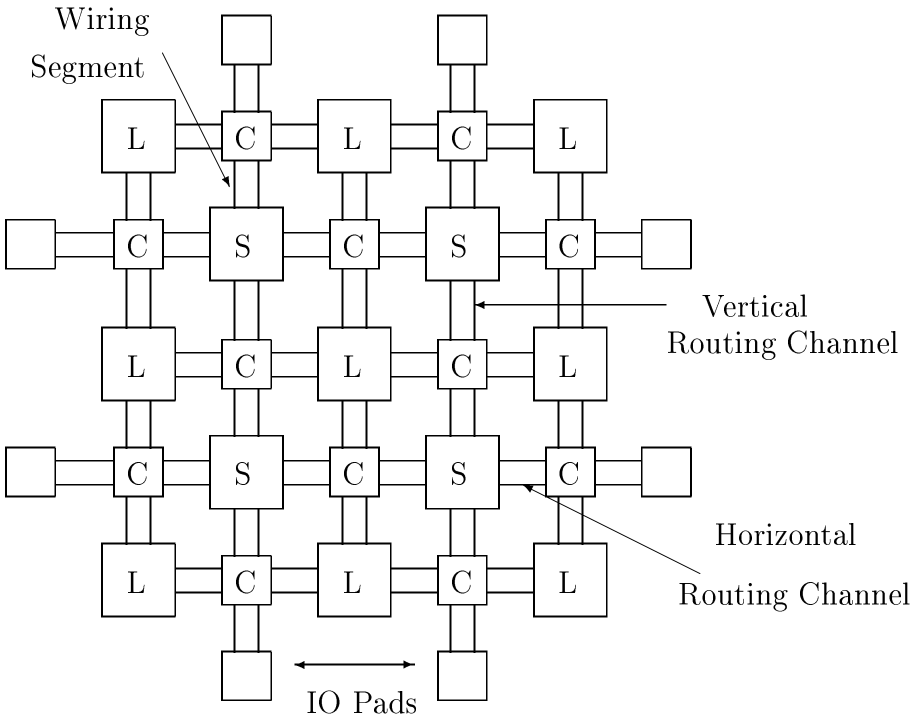


Figure 1.5: Non-segmented routing model for FPGAs. Here, L, S and C blocks represent logic, switch and connection blocks respectively.

wide range of functions can be implemented by a single logic block. There are two types of logic blocks. The first type is based on look-up tables, in which a function is implemented by simply loading its look-up table into the logic block at power up. This type is also called the *Configurable Logic Block* (CLB). The second type is multiplexer-based FPGAs. in which the programming is achieved by routing different inputs into the logic blocks.

Also, there are two models of routing network: the segmented and non-segmented models.

Figure 1.4 shows the segmented routing model in which tracks in the channels contain predefined wiring segments of same or different lengths with other wiring segments passing through the channel vertically. Connections between two horizontal segments is provided through an *antifuse*, whereas the connection between a vertical segment and a horizontal segment is provided by a *cross fuse* [30].

A typical non-segmented routing model is shown in Figure 1.5. In this model, the routing channels consist of two kinds of blocks: Connection blocks (C) and Switch blocks (S). The C blocks hold routing switches that serve to connect the logic block pins to the wire segments, while the S blocks contain switches that allow one wire segment to be connected to another [9].

Regardless of the selected model, the design cycle of FPGAs can be classified as partitioning (also referred to as technology mapping), placement and routing. To realize a circuit in an FPGA, the entire circuit is first partitioned to map into individual logic blocks, and then in the placement phase, all logic blocks are placed onto physical locations on the FPGA. In the routing step, the objective is to achieve 100% connections and at the same time optimize the timing characteristics.

1.1.3 Parallel Computation

Considerable amount of efforts have been directed toward developing parallel algorithms to handle different subproblems involved in the circuit design process. A popular model is the parallel *random-access machine* (PRAM) model. In addition to each processor using its own local memory, we assume that processors communicate by exchanging data through a shared memory unit in this PRAM model. Also, each access of the shared memory is in unit time. While some bridging must be done to move to a concrete parallel machine, much of the desired strategy for parallelization can be exposed by considering the PRAM model.

In the PRAM model, there is the possibility of read and write conflicts, in which more than one processor tries to read from or write into the same shared memory cell concurrently. Several different variations of the model exist based on distinctions in how to handle these conflicts. The weakest one is the *exclusive read exclusive write* (EREW) PRAM model, in which concurrent reads from the same memory location and concurrent writes into the same memory location are disallowed. If only concurrent reads are allowed, then it becomes the *concurrent read exclusive write* (CREW) PRAM model. If both types of access are permitted, then we obtain *concurrent read concurrent write* (CRCW) PRAM model. The principal varieties of CRCW PRAMs are based on how concurrent write conflicts are handled. The *common* CRCW PRAM model only allows concurrent writes when all processors attempting to write to the same location are writing the same value. In the *arbitrary* CRCW PRAM model, any competing processor may be the one to succeed. The *priority* CRCW PRAM model assumes that the indices of the processors are linearly ordered, and allows the processor with the minimum index to succeed.

Many studies of algorithms and complexity in the PRAM model focus on the trade-off between the time for the computation and the number of processors required [39]. In this thesis, we measure the performance of a given parallel algorithm in terms of worst-case analysis by using the time-processor product as the work of the parallel algorithm. A commonly accepted framework is to refer to a parallel algorithm as *efficient* if it runs in time polylogarithmic in the input size with the work exceeding the number of steps in an optimal sequential algorithm by at most a polylog factor. On the other hand, an *optimal* parallel algorithm runs in polylog time and its work is within a constant factor of the number of steps in an optimal (fastest possible) sequential algorithm.

1.2 Thesis Overview

In the next chapter, we consider three specific problems related to the single-layer channel routing problem in the PRAM models. The *minimum separation* problem involves finding the minimum vertical separation between two rows of terminals that is required for routability (given that the horizontal positions of the terminals are completely fixed). In the other two problems, we are allowed to offset the upper row of terminals as a block to the left or the right, though the individual terminals do not shift position relative to one another. In particular, the *optimal offset problem* involves finding the offset that minimizes the amount of separation necessary to route the channel. The *offset range problem* involves finding all offsets that give enough room to route at a given separation.

For single-layer routing, we provide efficient parallel algorithms for the minimum separation, offset range, and optimal offset problems. We consider all the variations of these problems that have linear-time sequential solutions rather than limiting attention to the "river-routing" context, where single-sided connections are disallowed. For the minimum separation problem, we obtain $O(\lg N)$ time on a CREW PRAM or $O(\frac{\lg N}{\lg \lg N})$ time on a (common) CRCW PRAM, both with optimal work (processor-time product) of $O(N)$, where N is the number of terminals. For the offset range problem, we obtain the same time and processor bounds as long as only one side of the channel contains single-sided nets. For the optimal offset problem with single-sided nets on one side of the channel, we obtain time $O(\lg N \lg \lg N)$ on a CREW PRAM or $O(\frac{\lg N}{\lg \lg N})$ time on a CRCW PRAM with $O(N \lg \lg N)$ work. Not only does this improve on previous results for river routing, but we can obtain an even better time of $O((\lg \lg N)^2)$ on the CRCW PRAM in the river routing context. In addition, wherever our results allow a channel boundary to contain single-sided nets, the results also apply when that boundary is ragged and N incorporates the number of bendpoints.

With advanced technologies, the demand for smaller and faster layout in VLSI design has encouraged designers to consider alternative ways of achieving further improvement in routing. Also, with the growing capability to have more layers of interconnect in the semiconductor fabrication industry, several researchers have proposed a scheme called *Over-the-Cell* wiring model, which uses extra layers to route a subset of nets or subnets (intervals) over the cells (i.e. outside the channel) to reduce the number of tracks needed inside the channel.

In Chapter 3, we provide an efficient algorithm to reduce the channel density of a given circuit in the over-the-cell wiring model. By choosing critical nets or subnets to route over the cells, our algorithm achieves better or equal channel density reduction with fewer tracks in the over-the-cell regions than previous research works. Experimental results using several benchmark circuits from the literature show that the proposed scheme obtains high quality results with much faster running time than existing heuristics. In addition, our algorithm can easily be parallelized due to the independent nature of cost function computations for each interval.

The final part of this thesis, in Chapter 4, shows the results of an empirical comparison of placement and global routing algorithms for FPGAs. We consider recently described algorithms that appear promising but that have not been compared on the same FPGA architecture and with the same benchmarks. We focus on a simultaneous placement and global routing scheme of Togawa et al. [65] and what appears to be the leading scheme for sequential placement and routing embodied in the Vpr system of Betz [5]. We compare the algorithms on two different FPGA architectures, using several benchmarks in each case.

In summary, this thesis provides the following contributions:

- the first efficient parallel algorithms for the following single-layer channel routing problems:

  — minimum separation with single-sided nets

  — offset range with single-sided nets on one side

  — optimal offset with single-sided nets on one side

  and

  — an improved parallel algorithm for optimal offset in the river routing context

- an algorithm for over-the-cell channel routing that yields faster and higher quality results than previous work in practice.
- a fair comparison of placement and global routing heuristics for FPGAs indicating which is best to use in practice.

Chapter 2

Parallel Algorithms for Single-Layer Channel Routing

In the single-layer context, this chapter considers three specific problems, namely the minimum separation, offset range, and optimal offset problems, in PRAM computing models. We take into account all the variations of these problems that have linear-time sequential solutions rather than limiting attention to the "river-routing" context, where single-sided connections are disallowed. In addition, wherever these results allow a channel boundary to contain single-sided nets, the results also apply when the boundary is ragged. This chapter is mostly derived from a joint paper with Ronald Greenberg of University of Maryland and with Jau-Der Shih of National Pingtung Teachers College in Taiwan [24].

Much attention has been given to single-layer routing for VLSI. Most popular has been river routing [21], the connection of two (horizontal) rows of corresponding terminals using the channel region between the rows of terminals; see also [53] and the references therein. More general arrangements of modules and nets have been considered for testing routability of terminals in fixed positions, but it is also desirable to answer more sophisticated questions. For example, the *minimum separation* problem involves finding the minimum vertical separation between two rows of terminals that is required for routability (given that the horizontal positions of the terminals are completely fixed). In other problems, we are allowed to offset the upper row of terminals as a block to the left or the right, though the individual terminals do not shift position relative to one another. In particular, the *optimal offset problem* involves finding the offset that minimizes the amount of separation necessary to route the channel. The *offset range problem* involves finding all offsets that give enough room to route at a given separation.

We consider these problems in all contexts for which linear-time sequential algorithms are known instead of considering only river routing, where each net is restricted to have exactly one terminal on each side of the channel. The input we assume is two arrays of terminals sorted by x-coordinate. The top terminals (and, in arithmetic contexts, their x-coordinates) are denoted $t_1, t_2, \ldots, t_m$, and the bottom terminals are denoted $b_1, b_2, \ldots, b_n$. We use N to denote $m + n$. Associated with each terminal is a net number. Terminals belonging to the same net are to be connected together. We also assume that each terminal has a pointer to the next terminal of the same net in a clockwise ordering of the terminals. (Such an assumption is implicit in [11], albeit for two-terminal nets. The assumption can actually be eliminated as long as there is a constant number of terminals per net, since a simple $O(1)$ time and $O(N)$ work algorithm on the arbitrary CRCW PRAM allows every terminal to find all other terminals of the same net, and general results for EREW and common CRCW simulations yield the needed bounds [38]. Even when the number of terminals per net is not constant, it is possible to eliminate the assumption by allowing randomization or a modest increase in time or work, using results on sorting of small integers (e.g, [32, 51]).) For simplicity, we use a rectilinear, grid-based model in which terminals lie on gridpoints and wires are disjoint paths through grid edges. Also, for convenience, we allow routing on channel boundaries.

We henceforth assume that each net has two terminals. Multiterminal nets can be handled by a transformation described in [27], that might be considered "folklore". Then a *single-sided net* has its two terminals on the same side of the channel, whereas a *two-sided net* is the type of net allowed in river routing.

We also assume henceforth that the channel is routable in one layer, i.e., no two nets are topologically forced to cross. (This condition can be verified without increasing the running time of our parallel algorithm by viewing the first and last terminals of each net as a left parenthesis and right parenthesis, respectively, and testing for proper nesting. The test for proper nesting requires only a prefix sum (discussed further in Section 2.1) of -1 and $+1$ values for left and right parentheses, respectively.)

The results obtained in this chapter are summarized in Table 2.1 and Table 2.2, where the river routing model is as described above, the general model includes any single-layer channel routing problem, and the intermediate model is one in which all single-sided nets are on one side of the channel. Henceforth, CRCW always refers to the *common* CRCW, in which all concurrent writes to the same memory location must be of the same value.

Most prior work on single-layer routing has been limited to sequential models of computation; linear-time sequential algorithms for the problems considered in this paper can be found in [27], [28], and [53]. For the river routing model only, parallel results for the problems in this paper are given by Aggarwal and

problem	model	CREW time	CREW work
min. sep.	general	$O(\lg N)$	$O(N)$
offset range	intermediate	$O(\lg N)$	$O(N)$
optimal offset	intermediate	$O(\lg N \lg\lg N)$	$O(N \lg\lg N)$
optimal offset	river	$O(\lg N \lg\lg N)$	$O(N \lg\lg N)$

Table 2.1: Running time and work (processor-time product) in CREW model for the algorithms presented in this chapter.

problem	model	CRCW time	CRCW work
min. sep.	general	$O(\lg N / \lg\lg N)$	$O(N)$
offset range	intermediate	$O(\lg N / \lg\lg N)$	$O(N)$
optimal offset	intermediate	$O(\lg N / \lg\lg N)$	$O(N \lg\lg N)$
optimal offset	river	$O((\lg\lg N)^2)$	$O(N \lg\lg N)$

Table 2.2: Running time and work (processor-time product) in CRCW model for the algorithm presented in this chapter.

Park [1]. For optimal offset, their results are $O(\lg^2 N)$ time on the CREW and $O(\lg N \lg\lg N)$ time on the CRCW, both with $O(N \lg N)$ work. We obtain superior bounds even for the more general problem labeled as the "intermediate" model in Table 2.1 and Table 2.2. We improve these bounds even further for river routing on the CRCW. Chang, JáJá, and Ryu [11], independently obtain the same bounds as Aggarwal and Park for minimum separation in the river routing model (matching our CREW bounds for the general model and improving the CRCW bounds of the general model) and also give an optimal (work) algorithm for routability testing for switchboxes.

The remainder of this paper is organized as follows. In Section 2.1, we explain the parallel operations used in this paper. We also indicate how to conveniently express the routability conditions for *single-layer channel routing*. These conditions are then used in Section 2.2 to solve the minimum separation, offset range and optimal offset problems. Section 2.3 gives some concluding remarks, including a brief explanation of why our results can be readily extended to cases in which channel boundaries containing single-sided nets are ragged (i.e., arbitrary, horizontally monotone, rectilinear boundaries, rather than straight lines.)

2.1 Preliminaries

2.1.1 Basic Parallel Operations

Given a sequence of N elements $\{x_1, x_2, \ldots, x_N\}$ with a binary associative operator $*$, the *prefix sums* are all the partial sums defined by:

$$p_i = x_1 * x_2 * \ldots * x_i \qquad 1 \leq i \leq N$$

For a given array $A(i)$ with $1 \le i \le N$, the *range maxima problem* is to find the element with maximum value between two given positions i and j. A query can be answered in $O(1)$ time after the preprocessing described in [38].

The range maxima preprocessing can be implemented in $O(\lg N)$ (respectively $O(\frac{\lg N}{\lg \lg N})$) time with $O(N)$ work on a CREW (CRCW) PRAM [3]. The prefix sums computation can be performed with the same time and processor bounds on the CREW [45], and on the CRCW as long as the input elements are integers in the interval $[1, N]$ [15].

Another useful operation is that of finding all nearest smaller values (ANSV). The form of the problem that we will use is as follows. Given a sequence of values $a_1, a_2, \ldots, a_N$, find for each i, the least $j > i$ such that $a_j < a_i$. This problem can be solved with linear work in $O(\lg \lg N)$ time on the CRCW and $O(\lg N)$ time on the CREW [2].

Note also that finding the maximum of N numbers can be done with the same resource bounds as ANSV [38].

2.1.2 Cut Conditions

We need a few definitions in order to use a general theory of single-layer routing developed by Maley. Define a *critical cut* to be a line segment that connects a top and bottom terminal or runs from a terminal straight across to the opposite side of the channel. Define a *pivotal cut* to be a line segment that connects a top and bottom terminal or runs at 45° from a terminal to the opposite side of the channel. Also let the *flow* across a cut χ be the number of nets that must cross χ, namely those nets having terminals on both sides of χ and those having an endpoint of χ as a terminal. (For example, in Figure 2.1, $flow(\overline{ab})$ equals 5

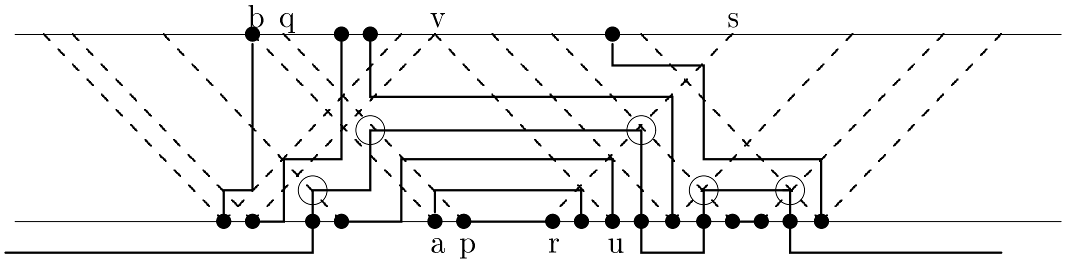


Figure 2.1: With all single-sided nets on the bottom, only those 45° cuts from bottom terminals shown with dashed lines need be checked. The hollow circles mark the convex corners of the contour of single-sided nets.

and $flow(\overline{uv})$ is 3.) The *capacity* of χ is one greater than the maximum of the horizontal and vertical separations of its endpoints; if χ is the line segment from (x_1, y_1) to (x_2, y_2), then

$$capacity(\chi) = max\{|x_1 - x_2|, |y_1 - y_2|\} + 1 \ .$$

(We allow routing on the channel boundaries.) The cut χ is *safe* if $flow(\chi) \leq capacity(\chi)$, which means that there is enough space along χ for the wires to get through.

Lemma 1 *A channel is routable if and only if every critical cut or every pivotal cut is safe.* ■

This lemma follows from the corresponding results in [50, §2.1,2.3,2.6.5]. (Our slightly different definitions of flow and capacity allow Maley's formulation in terms of cuts emanating from "feature" endpoints to correspond to cuts emanating from terminals. Since we allow routing on the channel boundaries, the only "features" are terminals and two routing obstacles (horizontal lines) located one unit outside of what we have been referring to as the channel boundaries.)

We can further strengthen the result for critical cuts as follows. Define the *span* of a cut χ to be the horizontal distance between its endpoints. For example, the span of cut $\overline{pb}$ is 7. Call χ *sparse* if χ is not vertical and $flow(\chi) \leq span(\chi)+1$, and *dense* otherwise. A sparse cut is safe regardless of the separation, but a dense cut χ is safe if and only if the separation is at least $flow(\chi) - 1$.

Lemma 2 *The minimum channel separation is the maximum of $flow(\chi) - 1$ over dense critical cuts χ.* ∎

When all single-sided nets are on the bottom, we can strengthen the result for pivotal cuts, but first we must review results regarding *contours* of single-sided nets. The *contour* of the single-sided nets is the routing boundary that the two-sided nets must stay unit distance away from when the single-sided nets are routed as tightly as possible against the bottom of the channel. (See Figure 2.1.)

The following Lemma from [11] shows that a contour of single-sided nets can be found efficiently.

Lemma 3 *The bendpoints in the contour of a set of N single-sided nets can be found in $O(\lg N)$ (respectively $O(\frac{\lg N}{\lg \lg N})$) time using $O(\frac{N}{\lg N})$ (respectively $O(\frac{N \lg \lg N}{\lg N})$) processors on a CREW (CRCW) PRAM.* ∎

Now, we are ready to state a result of [28] relating to pivotal cuts:

Lemma 4 *A channel with all single-sided nets on the bottom is routable if and only if all 45° cuts from bottom terminals of two-sided nets and all 45° cuts crossing the contour of single-sided nets at a convex corner are safe.* ∎

2.2 The Algorithms

2.2.1 The Minimum Separation Problem

Our algorithm for this problem is based on Lemma 2. To ensure that vertical cuts are captured, first add a dummy terminal across from each real terminal. Then we find the minimum separation that makes all dense (critical) cuts emanating from bottom terminals safe. To find all dense cuts emanating from b_j, we search for the two farthest dense cuts, one going to the right and one going to the left from b_j; these two cuts form a "cone" such that cuts emanating from b_j are dense if and only if they lie inside the cone.

We now provide some further definitions and notations used in this subsection. First, we say that a terminal is covered by a single-sided net on its side of the channel if it lies in the closed interval defined by the endpoints of the net; define $S(\tau)$ to be the number of single-sided nets covering τ. Now, two-sided nets are said to lie to the left or right of a terminal on the top (or bottom) according to the location of the net's top (respectively bottom) terminal. Also, a two-sided net is a right net if its top terminal is to the right of its bottom terminal; it is a left net if its top terminal is to the left of its bottom terminal. Define $R(\tau)$ to be the number of right nets to the left of terminal τ, and $L(\tau)$ to be the number of left nets to the left of τ. Also define $\mathrm{IL}(\tau)$ (and $\mathrm{IR}(\tau)$) to be 1 if τ is a terminal of a left (respectively right) net, and zero otherwise.

The heart of this algorithm is to form the cone for each terminal b_j. According to the definition, a nonvertical cut $\overline{t_i b_j}$ is dense if $|t_i - b_j| + 1 < flow(\overline{t_i b_j})$. We now show how to find the farthest cuts emanating from each terminal b_j on the bottom that are dense. Note that for any dense cut $\overline{t_i b_j}$, $R(t_i) \leq R(b_j)$, and $L(t_i) \geq L(b_j)$.

Thus, for dense cuts, $flow(\overline{t_i b_j}) = L(t_i) - L(b_j) + \mathrm{IL}(t_i) + R(b_j) - R(t_i) + \mathrm{IR}(b_j) + S(t_i) + S(b_j)$. Defining, $\mathrm{bl}(j)$ to be $b_j + L(b_j) - R(b_j) - S(b_j) - \mathrm{IR}(b_j) + 1$ and $\mathrm{tl}(i)$ to be $t_i + L(t_i) - R(t_i) + S(t_i) + \mathrm{IL}(t_i)$, we need to find the smallest t_i such that $\mathrm{bl}(j) < \mathrm{tl}(i)$; for similar definitions of $\mathrm{tr}(i)$ and $\mathrm{br}(j)$, we also find the largest t_i such that $\mathrm{tr}(i) < \mathrm{br}(j)$. It can be shown that the four functions bl, tl, br, and tr are non-decreasing. Now, we can give an algorithm for forming the cone for each bottom terminal.

procedure FIND-CONES

1. Compute $\mathrm{tl}(i), \mathrm{bl}(j), \mathrm{tr}(i)$ and $\mathrm{br}(j)$ for $1 \le i \le m$ and $1 \le j \le n$.

2. Merge $\mathrm{tl}(i)$ with $\mathrm{bl}(j)$ and $\mathrm{tr}(i)$ with $\mathrm{br}(j)$ in order of nondecreasing values. If a tie occurs, put $\mathrm{br}(j)$ before $\mathrm{tr}(i)$ and put $\mathrm{bl}(j)$ after $\mathrm{tl}(i)$.

3. For each $\mathrm{bl}(j)$, find the nearest $\mathrm{tl}(i)$ to the right in the merged sequence. If we do not find such a $\mathrm{tl}(i)$ corresponding to a t_i with lesser x-coordinate than b_j, then the farthest dense cut to the left from b_j is vertical. Similarly, for each $\mathrm{br}(j)$, find the nearest $\mathrm{tr}(i)$ to the left in the merged sequence, and select a vertical cut if necessary.

The merging can be done using the approach of Kruskal [44] in $O(\lg \lg N)$ time with $O(N)$ work on a CREW PRAM. Also steps 1 and 3 can be implemented by using prefix-sums. Therefore, algorithm FIND-CONES can be implemented with optimal work in $O(\lg N)$ (respectively $O(\frac{\lg N}{\lg \lg N})$) time on a CREW (CRCW) PRAM.

Once we find the cone for each bottom terminal, we can use the information to find the minimum separation for the single-layer channel routing problem:

procedure MINIMUM-SEPARATION

1. Apply algorithm FIND-CONES to find the farthest dense cuts to form a cone for every terminal on the bottom.

2. Find the maximum flow $F(b_j)$ among the cuts inside the cone for every terminal b_j.

3. The minimum separation is $-1 + \max\{F(b_1), F(b_2), \ldots, F(b_n)\}$.

Theorem 5 *Algorithm* MINIMUM-SEPARATION *finds minimum separation for single-layer channel routing with $O(N)$ work in time $O(\lg N)$ on a CREW PRAM and in time $O(\frac{\lg N}{\lg \lg N})$ on a CRCW PRAM.*

Proof. We have already explained how step 1 can be performed within the specified time and processor bounds, and step 3 simply involves a minimum that can be computed in the same bounds. To find the maximum flow in each cone in step 2, we use the range maxima technique. Since, the flow for a dense cut $\overline{t_i b_j}$ is $L(t_i) - L(b_j) + \mathrm{IL}(t_i) + R(b_j) - R(t_i) + \mathrm{IR}(b_j) + S(t_i) + S(b_j)$, and the terms dependent on j are fixed for any given cone, the task is to find the maximum of $\mathrm{tl}(i) - t_i$ over each cone. The preprocessing for range maxima and the single query per b_j can also be implemented within the stated bounds. ∎

2.2.2 The Offset Range Problem

In this subsection, we consider the offset range problem for single-layer channel routing with single-sided nets on one side. Without loss of generality, assume that all single-sided nets are on the bottom. Henceforth, we use s to represent the separation and d for the offset (the positive or negative distance by which the upper block of terminals is moved right from its original position).

According to Lemma 4, we only need to ensure that all 45° cuts from bottom terminals of two-sided nets and all 45° cuts crossing the contour of single-sided nets at a convex corner are safe. The most direct approach to obtain the desired result for offset range is to simply check *all* 45° cuts from bottom terminals, but we use a slightly less direct approach that will help us to solve the optimal offset problem as well. For this approach, we need to separate the bottom terminals of two-sided nets from those of single-sided nets. Let $a_1, a_2, \ldots, a_m$ be the x-coordinates of the bottom terminals of two-sided nets and $c_1, c_2, \ldots, c_h$ the x-coordinates of the bendpoints of the contour of single-sided nets.

Using Lemma 4, it is shown in [28] that we need only check $4m$ cuts. To obtain this formulation, we define T_i to be the number of two-sided nets whose bottom terminals are to the left of c_i. Also, let E_i be the extension of the single-sided contour at c_i, the nonnegative distance that the contour rises above its baseline at that column. Finally, define c_{r_i} to be the nearest bendpoint of the single-sided contour to the right of a_i such that $T_{r_i} - i + E_{r_i} > s$ and c_{l_i} to be the nearest bendpoint to the left of a_i such that $i - T_{l_i} + E_{l_i} > s + 1$.

Adapting the result of [28] to the notation in this chapter, we have that the pair (s, d) is feasible if and only if $l(s) < d < u(s)$, where

$$l(s) = \max_{1 \le i \le m} \{a_{i-s-1}, c_{l_i} - E_{l_i}\} + s - t_i \tag{2.1}$$

and

$$u(s) = \min_{1 \le i \le m} \{a_{i+s+1}, c_{r_i} + E_{r_i}\} - s - t_i \ . \tag{2.2}$$

Here we define $a_j = -\infty$ if $j \le 0$ and $a_j = \infty$ if $j > m$; also c_{l_i} (c_{r_i}) is defined to be $-\infty$ (∞) if there is no bendpoint satisfying the necessary conditions.

We can now prove the following theorem:

Theorem 6 *The offset range for single-layer channel routing with single-sided nets on one side can be found with $O(N)$ work in $O(\lg N)$ (respectively $O(\frac{\lg N}{\lg \lg N})$) time on a CREW (CRCW) PRAM.*

Proof. The contour of single-sided nets can be found within the requisite resource bounds by Lemma 3. The l_i and r_i values can be found by solving an ANSV problem, and then we need only compute a maximum and minimum of $m \leq N$ values; these operations can be performed with $O(m)$ work in $O(\lg m)$ time on the CREW or $O(\lg \lg m)$ time on the CRCW as indicated in Section 2.1. ∎

2.2.3 The Optimal Offset Problem

This subsection considers the optimal offset problem for river routing and channels with single-sided nets on one side. In both cases, the optimal offset problem is solved by using an algorithm for offset range as a subroutine. For channels with single-sided nets on one side, we use the results of Section 2.2.2 to obtain optimal offset results better than those of Aggarwal and Park, even though their results apply only to river routing. For river routing on the CRCW, we improve their optimal offset bounds even further.

Our algorithm adapts Mirzaian's halving technique [53] for relating optimal offset to offset range. We actually focus here on finding optsep(P), the minimum separation attainable with an optimal offset for the arrangement of terminals and nets comprising problem P; once optsep(P) is determined, the solution of the feasible offset problem can be used to determine the optimal offsets. From the original problem P, we create a simpler problem P^e that has about half the separation of P. The basic idea is to halve the extensions of the contour of single-sided nets, remove every other two sided net, and compact the channel horizontally to eliminate the freed space. More precisely, we perform the transformation specified as follows:

$$t_i^e = t_{2i} - i\ , \quad a_i^e = a_{2i} - i\ , \quad r_i^e = r_{2i}\ , \quad \text{and} \quad l_i^e = l_{2i} \qquad 1 \le i \le \lfloor m/2 \rfloor$$

and

$$c_j^e = c_j - \lceil T_j/2 \rceil\ , \quad \text{and} \quad E_j^e = \lfloor E_j/2 \rfloor \qquad j \in \{\, r_i^e, l_i^e \mid 1 \le i \le \lfloor m/2 \rfloor \,\}\ .$$

The following lemma from [28] states the relationship between $\mathrm{optsep}(P)$ and $\mathrm{optsep}(P^e)$:

Lemma 7 *Let* $s = \mathrm{optsep}(P)$ *and* $s^e = \mathrm{optsep}(P^e)$. *Then* $2s^e \le s \le 2s^e + 3$.

■

Let P_0 be the original problem, and define P_k to be P_{k-1}^e, for $1 \le k \le p$. Also, let $s_k = \mathrm{optsep}(P_k)$. From the above lemma, once we know s_{k+1}, then s_k can be solved by checking only 4 possible separations to achieve the optimal offset. Now, s_{k-1} can also be solved by checking all the possible separations derived from the 4 separations. (Again each possible separation of s_k induces 4 possible separations for s_{k-1}.) By considering the union of these separations, we only have to check 10 possible separations to solve s_{k-1}. Continuing this line of reasoning, we have the following corollary:

Corollary 8 *If* s_k *is known, then* s_{k-l} *can be solved by checking only* $3 \cdot 2^l - 2$ *possible separations.* ■

procedure OPTIMAL-SEPARATION

1 **for** $i \leftarrow \lg p$ **to** 1 **do**

2 find $P_{p/2^i}$

3 **endfor**

4 **for** $i \leftarrow 1$ **to** $\lg p$ **do**

5 find $s_{p/2^i}$

6 **endfor**

7 find s_0

Figure 2.2: This algorithm finds the minimum separation for the optimal offset problem.

Figure 2.2 shows our algorithm to solve the optimal offset problem. For ease of presentation, we assume the number of two-sided nets m is 2^p for some integer p. The algorithm finds the optimal separations $s_{p/2^i}$, for $1 \leq i \leq \lg p$. Each separation is determined from previously computed separations by using Corollary 8.

We first derive the resource bounds of this algorithm when applied to river routing, and then we extend the analysis to channels with single-sided nets on one side.

Theorem 9 *The optimal offset for river routing can be found with $O(N \lg\lg N)$ work in $O(\lg N \lg\lg N)$ (respectively $O((\lg\lg N)^2)$) time on a CREW (CRCW) PRAM.*

Proof. First we explain the CREW result. Lines 1–3 can be executed in time $O(\lg m)$ time with $O(m)$ work. (To see this, it suffices to consider repeated halving of the number of two-sided nets in constant time, with the work at each stage decreasing geometrically to a constant as the number of nets decreases.) In lines 4–6, each $s_{p/2^i}$ can be found in $O(p) = O(\lg m)$ time as follows. First, $s_{p/2}$ has at most $2^{p/2}$ possible values because there are only $2^{p/2}$ nets in $P_{p/2}$. The feasibility of each separation can be checked in $O(p)$ time with $O(2^{p/2})$ work by the proof of Theorem 6 or [1]. So all the possible separations can be checked simultaneously with $O(2^p)$ work. The minimum separation among the feasible separations is the optimal separation. Now, suppose $s_{p/2^i}$ is known; then at most $O(2^{p/2^{i+1}})$ possible values need to be checked for $s_{p/2^{i+1}}$ by Corollary 8. By a similar argument as before, $s_{p/2^{i+1}}$ can be found in $O(p)$ time with $O(2^p)$ work. Finally, Line 7 is again $O(p)$ time and $O(2^p)$ work. The total time and work, including all the passes through the loop in lines 4–6, are $O(p \lg p)$ and $O(2^p \lg p)$, respectively.

On the CRCW, we can do each pass through the loop in lines 1–3 in $O(\lg\lg m)$ time with $O(m)$ work, and there are $O(\lg\lg m)$ passes. The remaining analysis is the same as before except with the $O(\lg\lg m)$ time offset range result for river routing on the CRCW that comes from Aggarwal and Park [1] or the proof of Theorem 6. This yields $O(\lg p)$ time and $O(2^p)$ work to compute each $s_{p/2^i}$ (and s_0). ■

When there are single-sided nets on one side of the channel, we need modify our argument only slightly. The analysis above remains valid, except that we must do one computation up front of the contours and the l_i and r_i values. Adding in the extra resources required, we obtain the following:

Corollary 10 *The optimal offset problem for channels with single-sided nets on one side can be solved with $O(N \lg\lg N)$ work in $O(\lg N \lg\lg N)$ (respectively*

$O(\frac{\lg N}{\lg \lg N}))$ time on a CREW (CRCW) PRAM. ■

2.3 Remarks

This chapter has provided efficient parallel algorithms for (1) the minimum separation problem for general single-layer channels and (2) offset problems for single-layer channels in which only one side of the channel has multiple connections to a single net. In addition, we have improved previous results for optimal offset in river routing problems, where each net has exactly one terminal on each side of the channel.

The above results also apply whenever a channel boundary that is allowed to have multiple connections to a single net is also allowed to be ragged. (We must first add dummy terminals at all the bendpoints of the ragged boundary, increasing N accordingly.) For the minimum separation problem, we need only incorporate the extension of the boundary at t_i into the range maxima computation in the proof of Theorem 5 and then the extension at b_j into the maximization over the dense cuts emanating from b_j. For the offset problems, we simply incorporate boundary extension terms into each of the a and c terms in Equations 2.1 and Equations 2.2.

An obvious open question is whether any of the bounds on time or work can be improved. In particular, the algorithms for optimal offset use $N \lg \lg N$ work rather than the $O(N)$ work that can be achieved sequentially. An additional open question is whether offset problems can be efficiently solved in parallel when both sides of the channel contain single-sided nets; for sequential computation, this problem is considered in [29].

Chapter 3

An Efficient Algorithm for Reducing Channel Density by Over-the-Cell Routing

As more and more layers are made available for routing, the over-the-cell routing style has become both practical and important in layout design. This chapter presents an efficient algorithm to reduce the channel density of a given channel. By choosing critical nets or subnets to route over the cells, the proposed algorithm achieves better or equal channel density reduction with fewer tracks in the over-the-cell region than previous results published in the literature. An earlier version of this work appears as [36].

The traditional channel routing problem involves minimizing the separation between two rows of terminals using the channel region in between to connect terminals belonging to the same net. Over the years, researchers have developed many algorithms to obtain competent performance; related work includes [7, 14, 18, 20, 22, 25, 57, 58, 59, 67].

Recently, the demand for smaller and faster layout in VLSI design has encour-

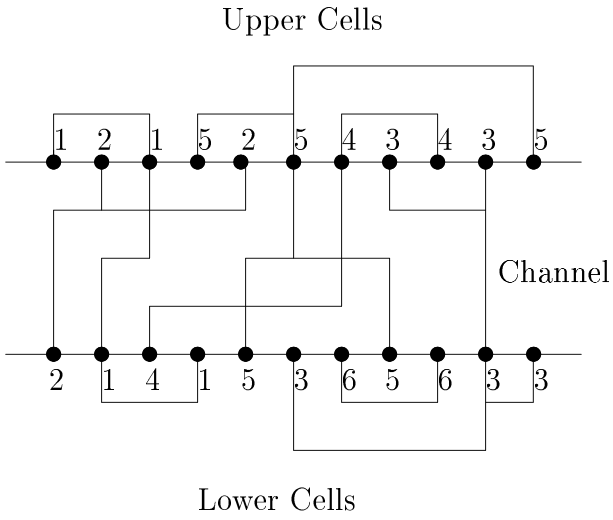


Figure 3.1: An example of over-the-cell routing. Without over-the-cell routing, greater channel width would be required. Here we are considering routing in two layers within the channel, one for horizontal wires, and one for vertical wires.

aged designers to consider alternative ways of achieving further improvement in routing. Also, with the growing capability to have more layers of interconnect in the semiconductor fabrication industry, several researchers have proposed a scheme called over-the-cell routing which uses an extra layer to route a subset of nets or subnets outside the channel to reduce the number of tracks needed inside the channel; the remaining nets or subnets are routed within the channel using traditional channel routers. We will henceforth use the term *channel width* to refer to the number of tracks in the channel (the number of horizontal grid lines between the rows of terminals that are used for wires). Figure 3.1, adopted from [17], shows how over-the-cell routing improves upon traditional channel routing in Figure 3.2.

In [17], Cong and Liu assumed that the more nets or subnets routed over

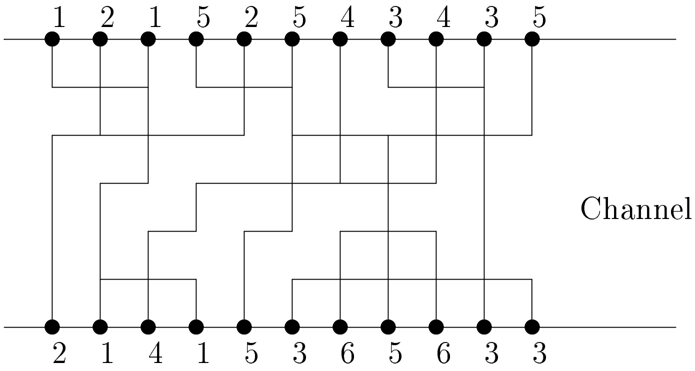


Figure 3.2: The channel routing instance in Figure 3.1 requires 5 tracks without routing over-the-cell.

the cells, the smaller the area for the final layout could be. However, some works [19, 49] showed that this assumption is not necessarily true. While several factors affect the number of tracks required to route the nets remaining in the channel, the number of tracks is usually close to the channel density (for routing in two layers). (Channel density is the maximum of the column densities, where the density of a column is the number of nets that must cross it.) We therefore focus here on removing nets or subnets crossing critical columns (columns with maximum density in the current circuit), which, if routed over the cells, reduce the congestion of the current circuit. In Figure 3.3, only nets or subnets contributing to channel density reduction are routed over the cells. The final layout and channel density are the same as in Figure 3.1; however, there are fewer tracks used in the over-the-cell regions. In Figure 3.3, the channel density within the channel is reduced from 4 to 3, and the channel width for the final layout within the channel is reduced from 5 to 3.

Lin et al. formulated over-the-cell routing problems as linear programming problems [49]; although they obtained good solutions, the running time of their

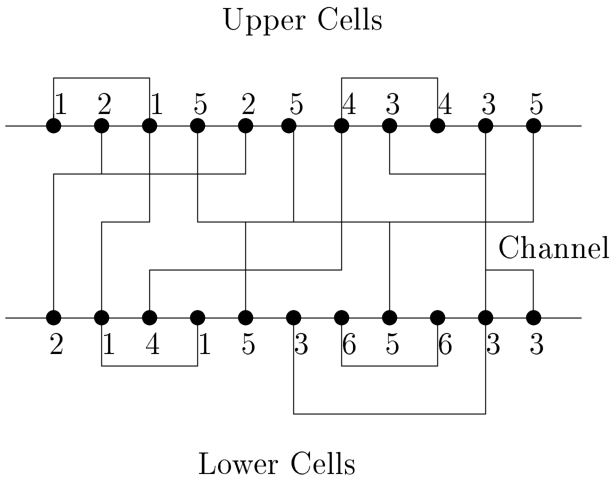


Figure 3.3: An example of routing nets or subnets crossing the most congested columns over the cells.

algorithm increases dramatically as the size of the input problem grows. In this chapter, we present an efficient heuristics to reduce the channel density of a given circuit by over-the-cell routing; our algorithm obtains high quality results and also overcomes the difficulty encountered in [49].

The remainder of this chapter is organized as follows. In the following section, we review the prior work done on over-the-cell routing. Then the basic strategy of our heuristics and the algorithm itself are discussed in Section 3.2. Section 3.3 shows experimental results of our study and comparisons to previous work. Then, we explain potential parallel implementation of the proposed algorithm in Section 3.4. Section 3.5 gives concluding remarks.

3.1 Previous Work

In contrast to traditional channel routing, over-the-cell channel routing makes use of the over-the-cell regions to reduce the area needed within the channel. The most commonly considered model has used two routing layers in the channel and one over the cells. The algorithm presented in this chapter can be applied to any situation in which one layer is used for the over-the-cell routing. Reducing density of the remining nets is generally a good strategy regardless of the number of layers to be used within the channel. It is well known that ordinary channel routing problems are NP-complete [47, 55, 56, 62], and Gudmundsson and Ntafos [31] indicate that over-the-cell routing problems are as well.

For over-the-cell routing problems, the common approach [17, 19] can be divided into three phases: routing over the cells, choosing nets or subnets within the channel, and routing within the channel. In the first phase, nets or subnets to be routed over the cells are chosen. Then, the connectivity of nets is preserved in the second phase by selecting a subset of each net's terminals to be connected within the channel. Finally, the nets or subnets within the channel are routed using a traditional channel router. Assuming that the fewer nets or subnets there are inside the channel, the smaller the final layout would be, Cong and Liu [17] selected a maximum independent set of a circle graph to represent the portions of nets for over-the-cell routing, and then they routed the remaining nets or subnets inside the channel with traditional channel routers. Differing from Cong and Liu's approach, Das et al. [19] iteratively diluted the portion of the channel having the maximum congestion and span, and then subsequently forced the smallest set of other nets to be routed inside the channel. Furthermore, their heuristics selects over-the-cell candidates on both sides of the channel simultaneously. Also, Lin et al. [49] proposed to improve routing layout by reducing the channel density using over-the-cell routing; their approach removes only critical nets or subnets which contribute to the reduction of the channel density. This heuristic formulates a given problem as a linear programming problem, and then solves it using existing software packages.

WISER developed by Holmes et al. [34] is based on the standard cell design technology and uses "vacant" terminals for over-the-cell routing. We don't emulate that approach here, since our algorithm is not designed for standard cell layouts, and the percentage of "vacant" terminals in the channel is usually not high. Then, Hou and Chen [35] took advantage of the permutation property (the interchange of gates and terminals does not affect the functions of the cells) in standard cell design to develop a dynamic programming based algorithm for over-the-cell routing.

Katsadas and Kinnen [41] developed a new methodology using over-the-cell routing for multi-layer (4 layers in this case) routing problems. This two-phase approach uses the first two layers to route a group of nets within the channel in the first phase, then, in the second phase, the remaining nets are routed over the entire routing area (which also includes the over-the-cell regions) with the other two layers. Later, Terai et al. [63] proposed a new model for over-the-cell routing, which differs from the general approach by using two layers in the over-the-cell area and three layers for the area inside the channel.

3.2 The Heuristics

3.2.1 The Basic Concepts

Our strategy is to iteratively choose nets or subnets in the more congested columns of a given circuit to route over the cells, which differs from the approach in [19] by choosing a candidate for over-the-cell routing without testing all combinations of nets or subnets.

First, nets with multiple terminals on the same side of the channel are partitioned into two-terminal intervals (with two adjacent terminals as endpoints) or subnets as shown in Figure 3.4. Also in order to ensure the connectivity of two-sided nets, we adopt the idea of classifying each interval as *red* or *blue* from [49]. An interval is *blue* if, on the opposite side of the channel, there is at least one terminal from the same net located inside its region; otherwise, it is *red*. For example, in Figure 3.5, both intervals of net '5' are *blue* and both intervals of net '3' are *red*.

Then, for convenience, we classify nets into different groups: *Single, Twosided, Vertical, Notfree*, and *Red*. *Single* is the group of nets with all terminals on the same side of the channel (e.g. net '1' in Figure 3.5), and group *Vertical* contains two-sided nets with two terminals on the same column and at least three terminals in total (e.g. net '2' in Figure 3.5). *Notfree* consists of two-terminal two-sided nets (e.g. net '4' in Figure. 3.5) and also nets with one terminal in the channel (e.g. net '6' in Figure 3.5). (We allow such one-terminal nets, because nets may come with a specification that they should be routed out the left and/or right side of the channel.) The *Red* group includes the remining two-sided nets that have no blue intervals (e.g. net '3' in Figure 3.5). Then, *Twosided* is the group of two-sided nets which do not satisfy the conditions mentioned above (e.g. net '5' in Figure 3.5).

It is easy to understand that the two-sided nets in group *Notfree* can not be routed over the cells, so they do not contribute to further channel density reduction. To preserve the connectivity of a net in the group *Twosided*, a vertical wire can be run from the track occupied by a blue interval in the channel to a terminal for that net on the opposite side of the channel. It is therefore clear that, for net *N*, as long as at least one *blue* interval remains to be routed inside the channel, the connectivity of *N* is preserved. The connectivity in group *Vertical* is preserved through the vertical connection of the two terminals located on the same column. For nets in group *Red*, the connection of two nearest terminals on opposite sides of the channel (e.g. for net '3' in Figure 3.5, the connection between the first terminal of net '3' on the top and the second one on the bottom) serves the same purpose. Hence, nets in groups *Vertical* and *Red* can be handled in the same fashion as nets in group *Single* with respect to selection of intervals to route over the cells, without destroying the connectivity.

We now provide notations used in this chapter. For a given circuit, let t be the number of terminals and $cost_{[a,b]}$ be the cost of interval $[a, b]$. Define d_{init} to be the channel density of the initial circuit. Then, define d_{max} and d_{second} to be the maximum and the second largest column density of the updated circuit at the current iteration. Also, define $d(j)$ to be the density of column j and d_j the value contributed by column j to $cost_{[a,b]}$ at the current iteration.

In order to judge the condition of congestion in a circuit, we use column densities as the parameter for computing cost functions in the proposed algorithm. Also, since a column without any terminal on it does not provide any new

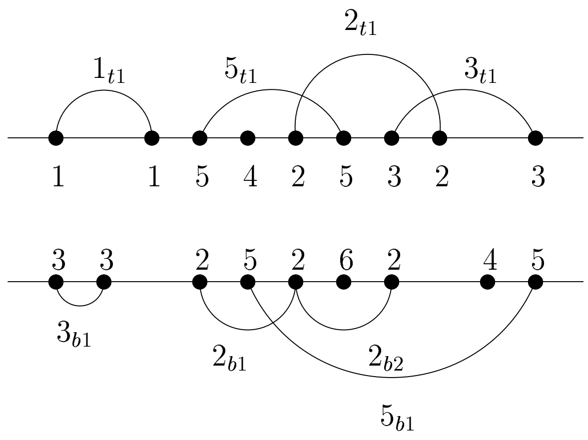


Figure 3.4: The partitioning of multi-terminal nets into two-terminal intervals. For example, net '3' is divided into two intervals 3_{b1} and 3_{t1}.

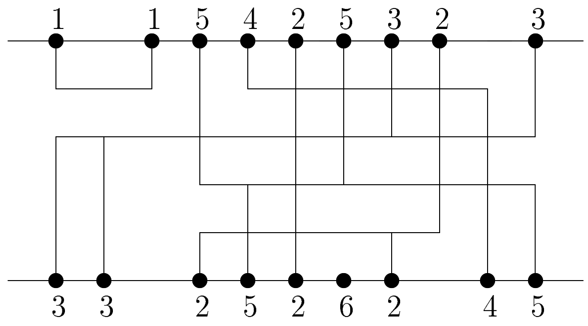


Figure 3.5: The classification of nets in this chapter: *Single (e.g. net '1'), Vertical (e.g. net '2'), Red (e.g. net '3'), Notfree (e.g. net '4' and net '6'),* and *Twosided (e.g. net '5').*

information, we only take columns with terminals on them into consideration. After experimentation with various alternatives, the cost function decoded upon was $cost_{[a,b]} = \sum_{a \leq j \leq b} d_j$. Here, $d_j = d(j)$ if (i) $d(j)$ is the maximum or second maximum in the current circuit, and (ii) if interval $[a, b]$ is routed over the cells, then $d(j)$ is reduced by one. (Since there may be more than one interval of the same net crossing column j, routing one over the cells does not necessarily reduce $d(j)$.) Otherwise, $d(j) = 0$. Also, in case of a tie, we choose an arbitrary winner.

3.2.2 The Over-the-Cell Algorithm

Our approach for this problem is based on the fact that selecting intervals crossing more congested columns for routing over the cells results in a better layout [19, 49]. As shown in Figure 3.6, we repeatedly search for a new interval $[a, b]$ to route over the cells by evaluating cost functions for all intervals crossing a column with the maximum density. Also, our approach has the capability either to route over the cells on both sides of the channel simultaneously or to handle the two sides separately. However, our study shows that there is no significant difference between the results of these two schemes. For preserving connectivity of net N, if all *blue* intervals of net N are routed over the cells, then we preserve the connectivity by connecting an endpoint of the last *blue* interval $[a, b]$ routed over the cells to the nearest terminal inside interval $[a, b]$ of the same net on the opposite side of the channel; we choose the connection with endpoint a if the cost of this connection is less than the cost of the connection with endpoint b.

Now, we discuss the complexity of this algorithm. For the worst case analysis, there are number of $O(t * d_{init})$ computations at each iteration to find an interval

```
    procedure OVER-THE-CELL
1   repeat
2       Find the maximum density d_max and
          the second maximum density d_second
          of the updated circuit
3       Find all columns j such that d(j) = d_max,
          and there is at least one terminal
          on column j
4       For all columns found in step 3, compute
          the cost for all intervals crossing these
          columns
5       Find the interval with the maximum cost
6       Update circuit
7   until there is no interval for OTC routing
```

Figure 3.6: This algorithm reduces the channel density by repeatedly choosing an interval to route over the cells.

Example	ex1	ex2	ex3	ex4	1	3a	3b	3c	4b	5	De
# col.	417	421	421	421	35	62	61	79	119	119	174
# nets	235	282	291	270	21	45	47	54	57	62	72
Density	16	15	11	19	12	15	17	18	17	20	19
cpu time (Sec.)	.08	.1	.13	.1	.03	.04	.04	.05	.08	.07	.13

Table 3.1: Sizes and running time of benchmarks tested in this chapter.

to route over the cells, and it takes $O(t)$ iterations to reduce the current channel density by one. Here, t is the number of terminals. Also, there are $O(d_{init})$ steps of channel density reduction; hence, the worst case complexity is $O(t^2 d_{init}^2)$. However, in our experiments, it usually takes less than 10 iterations for this algorithm to terminate.

3.3 Experimental Results

We have implemented this algorithm on a SUN SPARCstation 2 in C, and sizes of benchmarks tested in this chapter are listed in Table 3.1 along with the running time of our experiments. Examples ex1 to ex4 are benchmarks used in [25], and the benchmarks other than ex1 to ex4 are the same as in [67].

[1] The density of 4b is given as 20 in [17] and [49], but if the correct example 4b from [67] was used, the density is, in fact, 17.

[2] Exhaustive checking of all combinations of subsets to route over the cells has led to the conclusion that the density actually cannot be reduced below 12 and that there is an error in [49].

Ex.	Original density	Density over the upper cells			Density over the lower cells			Density remaining within the channel		
		[17]	[49]	ours	[17]	[49]	ours	[17]	[49]	ours
1	12	4	2	1	3	3	2	9	9	9
3a	15	3	3	4	6	3	2	12	12	12
3b	17	2	2	3	5	4	3	13	13	11
3c	18	3	4	4	4	4	4	14	13	12
4b	17[1]	5	2	3	4	3	3	16	13	13
5	20	4	6	5	3	4	2	14	11[2]	12

Table 3.2: Results for the algorithm presented in this chapter and comparisons with previous work.

Table 3.2 shows the comparison of our results to those of [17] and [49], which indicate that they use the same examples as in [67].

Lin et al. [49] indicate that the drawback of their approach is the fast growth of running time as the problem size increases. We overcome this difficulty with the proposed algorithm and still obtain high quality results. As shown in Table 3.1, the running time of our algorithm is no more than 0.13 seconds for all circuits that we tested, which is several thousand times faster than the running times shown in [49]. Although the speed of our experiments benefits from the state-of-the-art of computer technology, it still has the advantage of having small slow-down ratio as input sizes increase.

3.4 Potential Parallel Implementation

The independent nature of computations in our heuristics makes it more suitable for parallel implementations. As pointed out in Sec. 3.2, our algorithm can search for intervals to route over the cells on both sides of the channel simultaneously. Also, for parallel implementations, the searching steps for an over-the-cell interval can be implemented in constant time on parallel machines like Connection machines CM2. However, since the sizes of our benchmarks generally are not large enough to yield speedup on parallel machines, we only tested out the possibility of searching for intervals to route on the top and on the bottom simultaneously in our experiments. Our study shows that this strategy works as well as searching for one interval only.

3.5 Remarks

In this chapter, we provide an efficient algorithm for channel density reduction in over-the-cell routing. The quality of our results is at least as good as previous works and the running time is faster. It would be interesting to further test parallel implementation of this algorithm in future research.

Chapter 4

An Empirical Comparison of Placement and Global Routing Algorithms for FPGAs

This chapter presents the results of an empirical comparison of placement and global routing algorithms for field-programmable gate arrays (FPGAs). We focus on a simultaneous placement and global routing scheme of Togawa et al. [65] and what appears to be the leading scheme for sequential placement and routing embodied in the Vpr system of Betz [5]. This thesis compares the algorithms on two different FPGA architectures, using several benchmarks in each case. This chapter is drawn from a joint paper with Ronald Greenberg of University of Maryland [37].

In recent years, FPGAs have become an increasingly popular medium for rapid, low-cost implementation of VLSI circuits. An FPGA consists of an array of logic blocks embedded in an interconnect structure and surrounded by IO blocks. For an FPGA, the interconnect contains segments of wire, where the segments may be of various lengths and may not be uniformly distributed. Programmable switches in the interconnect serve to connect the logic blocks to the wire segments, or one wire segment to another. In FPGAs, a circuit is implemented by partitioning it into portions that can be implemented by individual logic blocks, and the blocks are interconnected as required via switches.

A typical design flow for a circuit using FPGAs involves logic optimization, technology mapping, placement and routing. The overall goal is generally to optimize area or delay or some combination of these measures. Several logic synthesis systems are available, e.g., MIS [8], SIS [61] and BOLD [6]. After logic optimization, the circuit is partitioned into FPGA logic blocks by a technology mapping program, e.g., [16, 23, 40]. Having mapped the circuit into logic blocks, a placement program then assigns each logic block to a specific location on an FPGA array. In the FPGA environment, the placement problem is quite similar to the case of VLSI circuits using standard cell implementations. In the final stage, the routing program establishes connections required among logic blocks by choosing wire segments and enabling programmable switches. Again, this problem is similar to the same case in the standard cell environment, but it is more complicated due to the limited routing resources on the FPGA. The routing program has to ensure that all connections are formed; otherwise, the circuit can not be realized on the FPGA. Usually, this process contains two steps: global routing and detailed routing. Good routing techniques such as [10, 12, 30, 52] have been published in the literature.

In order to improve performance or speed the design process, some researchers have combined together two or three stages in the design flow. Such works include [13, 54, 64, 65].

The two architectures [10] and [65] studied in this chapter are shown in Fig-

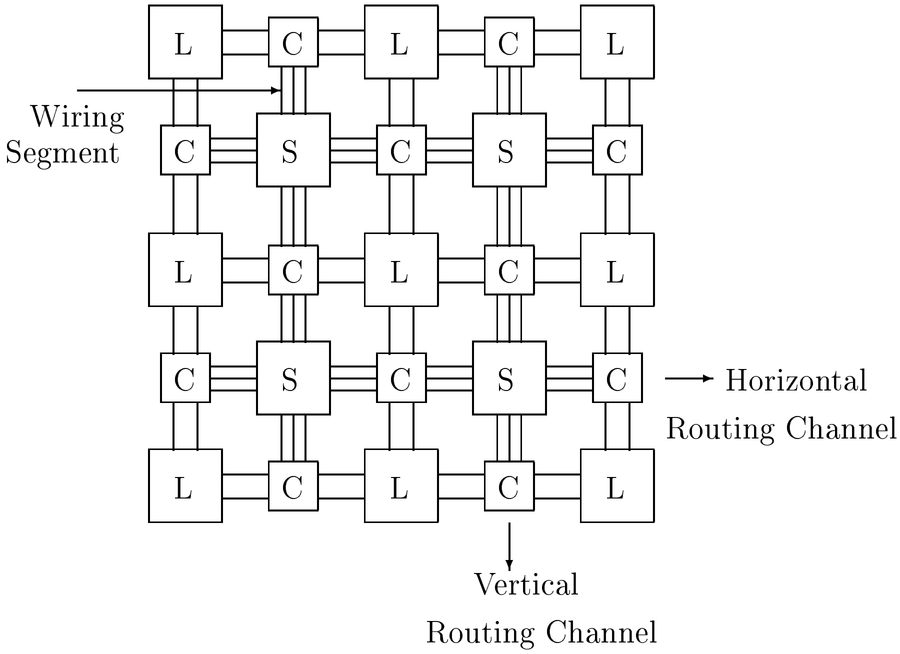


Figure 4.1: The FPGA model from [10].

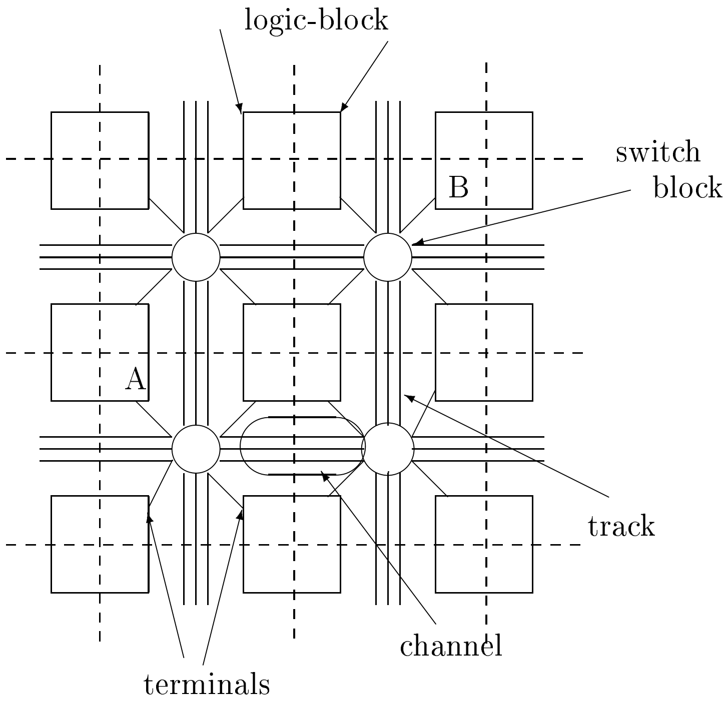


Figure 4.2: The FPGA layout model in [65].

ure 4.1 and Figure 4.2 respectively. The main difference between these two architectures is that the model in [10] uses connection blocks to implement routing between logic blocks and switch blocks while the other model has fixed connections between logic blocks and switch blocks. In this paper, we implement the simultaneous placement and global routing algorithm in [65] on the architecture of [10] as well as on the architecture for which it was originally designed; we also test traditional approaches which do placement and global routing sequentially. For each of the two FPGA models, we do an empirical comparison of the simultaneous placement and global routing with sequential approaches.

The remainder of this chapter is organized as follows. In the following section, we review the two architectures studied in this experiment. In section 4.2, we give an overview of the heuristics proposed in [65] and the sequential placement and global routing program Vpr of Betz [5]. (Betz's program performs better than other sequential placement and global routing schemes presented in the literature.) Then, we present experimental results for the two FPGA architectures in the following section. Finally, we give conclusions in the last section.

4.1 FPGA Models

This section reviews the two FPGA models from [10] and [65] studied in this chapter. First, we begin this section by explaining a few terms shown in Figure 4.1 and Figure 4.2.

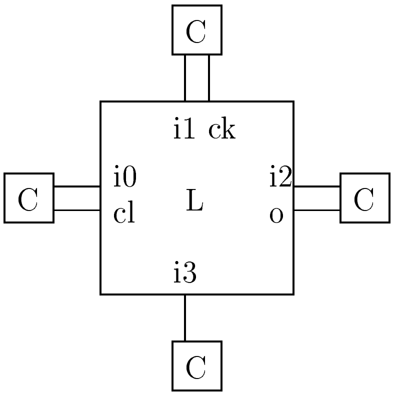
A *wiring segment* is a straight section of wire that is used to form a portion of a connection. A *routing channel* is the rectangular area that lies between two rows or two columns of logic blocks, and each routing channel contains a number of tracks. A *track* is a horizontal (vertical) line spanning an entire horizontal (vertical) routing channel, that may be composed of a number of wire segments of various lengths. Also, the term *terminals* refers to the pins on any logic block (L), switch block (S) or connection block (C).

4.1.1 The First Model

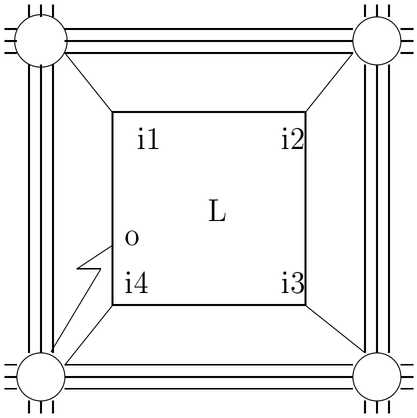
Figure 4.1 shows the FPGA model of [10], a two-dimensional array of logic cells interconnected by vertical and horizontal routing channels.

There are three parts in this model: the L blocks, C blocks and S blocks. The IO blocks which serve for primary inputs and outputs appear as L blocks on the periphery of the chip. The L blocks are programmable and house the combinational and sequential logic which form the functionality of a circuit. Figure 4.3(a) shows the terminal positions of an L block defined in this experiment. Each L block consists of seven pins: four inputs, one clock signal, one control signal and one output terminal.

In this model, C blocks are rectangular switch boxes with connection points on all four sides, and they are used to connect terminals on an L block to the routing channel via programmable switches. While S blocks are also rectangular switch boxes, the purpose of them is to connect wiring segments in one channel to those in another. Since the physical architectures of S and C blocks have a strong influence on the routability of an FPGA, Figure 4.4 shows the general structures of S and C blocks. The flexibility of S blocks is defined by F_s, which is the total number of other connections that a wire segment is allowed to connect to when this segment enters an S block. Similarly, the term F_c reflects the flexibility of C blocks, which indicates the number of tracks that each terminal



(a) (i0, i1, i2, i3: input terminals; o:output terminal; ck: clock signal; cl: control signal)



(b) (i1, i2, i3, i4: input terminals; o: output terminal)

Figure 4.3: (a) The pin positions on an L block in [10]. (b) The terminal positions on an L block in [65].

on an L block can connect to.

4.1.2 The Second Model

Figure 4.2 shows the FPGA model defined in [65], which is based on the architecture proposed in [42]. Compared to the model in Figure 4.1, the main difference is the direct connections of pins on L blocks to S blocks as shown in Figure 4.3(b). Therefore, S blocks are connected to each other by vertical or horizontal tracks on the routing channels without going through any C blocks. (There is no C block in this architecture.)

Each L block in this model has five pins (four input signals and one output signal) as shown in Figure 4.3(b).

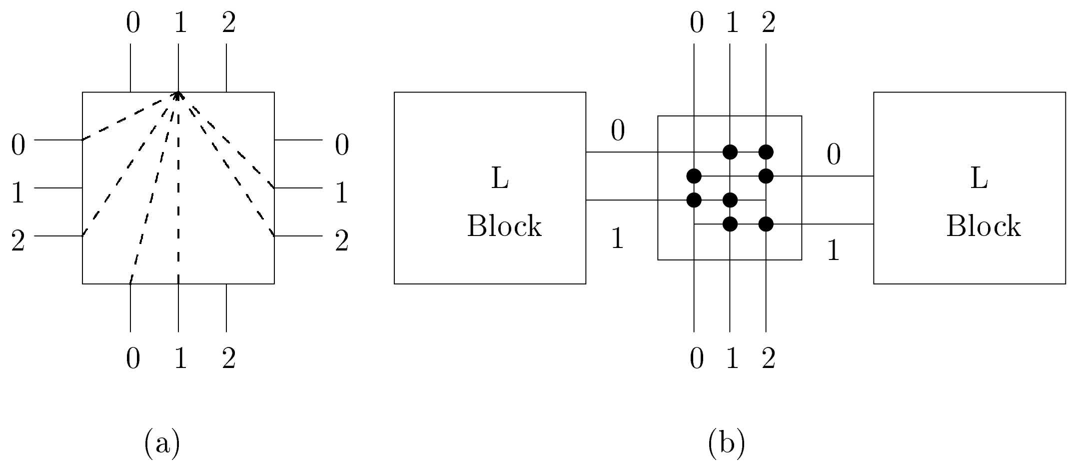


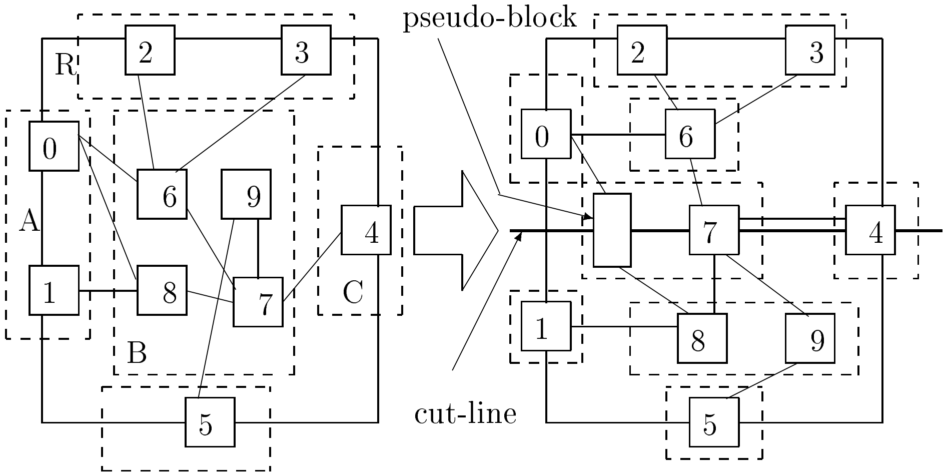
Figure 4.4: (a) An S block with $F_s = 6$, showing the possible connections from the top center terminal. (b) A C block with $F_c = 2$.

4.2 The Heuristics

In this section, we review the heuristics proposed in [52] and [65].

4.2.1 The Simultaneous Scheme

The algorithm in [65] may be loosely viewed as a recursive process of "bi-partitioning" a given layout region and a set of logic blocks. Cut-lines alternate between horizontal and vertical from one level of the recursion to the next and are drawn along grid lines (dashed in Figure 4.2). In actuality, each horizontal cut-line partitions (in order) the three sets of blocks that have been assigned to the left boundary of the subregion, the right boundary of the subregion and the interior of the subregion into three sets each: those to be assigned above the cut-line, those assigned below the cut-line, and those on the cut-line. The situation is similar for vertical cut-lines. (Partitioning the blocks on a subregion



(a) Before partitioning. (b) After Partitioning.

Figure 4.5: Partitioning a subregion during an iteration.

boundary may amount to simply adopting a partition previously produced from partitioning an adjacent subregion or may involve considering an augmented region comprised of the two subregions bordering the boundary.)

As a starting point for the recursive process, the IO blocks are randomly distributed around the periphery of the original region, while the remaining (logic) blocks are assigned to the interior. Whenever a net crosses a cut-line in a partitioning stage, a *pseudo-block* is generated on this cut-line to indicate this connection. Upon termination of the entire process, the global route for a net N is determined by the logic blocks and pseudo-blocks of N. Figure 4.5 shows the process of partitioning a subregion during an iteration, and the simultaneous placement and global routing approach is outlined in Figure 4.6.

Each step of partitioning a block set for a subregion involves generating an acyclic directed graph G_R whose vertices correspond to the logic blocks and pseudo-blocks inside and on the boundary of the subregion (or the augmented

Place IO blocks on the four boundaries of the layout region with a reasonable balance.

Put the layout region as a subregion into a queue Q.

repeat

  Retrieve a subregion R from Q.

  Partition R into two subregions R_1 and R_2 by a cut-line. The cut-line will partition the longer side of the region to make the subregions close to squares.

  Corresponding to the partition of R, assign the block sets of R to R_1, R_2 and the cut-line.

  For the logic blocks on the cut-line, assign nets to pins so as to avoid unnecessary crossings of the cut-line.

  Generate a pseudo-block on the cut-line for each net crossing it.

  Put R_1 and/or R_2 into Q if they can be further partitioned.

until Q is empty

Figure 4.6: The algorithm of Togawa et al. for simultaneous placement and global routing.

region obtained by concatenating the two subregions bordering a boundary block set). G_R is obtained by doing breadth-first search from the primary inputs based on the connections between blocks, placing an edge in G_R corresponding to each pair of connected blocks and directing the edge from the block searched earliest to the block searched later. Vertices corresponding to blocks on the boundaries of the subregion that have been assigned above, below, or on the cut-line (in the case of a horizontal cut) are assigned labels of 1, 0, or 0.5, respectively. For any other vertex v, once labels have been assigned to all vertices from which G_R contains an edge directed into v, the label of v is set to the average of those labels. Then the labeling process is repeated with all edges of G_R reversed. The final label of each vertex is the average of the two labels obtained for that vertex in the "forward" labeling and the "backward" labeling. Finally, based on the sorted order of the blocks by their corresponding labels, the blocks are assigned above, below, and on the cut-line in the same proportion as the number of available block locations in those regions of the FPGA.

4.2.2 The Vpr implementation of Placement and Global Routing

The Vpr system [5] follows the traditional approach of performing placement first and then performing global routing. The placement task within Vpr is performed using simulated annealing, which has become popular for this phase. This is also the stronger of the two placement techniques used by Togawa et al. for comparison to their system. Since simulated annealing has become a widely familiar technique, further description is omitted here; more information about the methodology of Vpr may be found in the work of Betz and Rose [4]. But the global routing portion of Vpr is more novel; it uses the PathFinder global router of McMurchie and Ebeling [52], which we overview here.

PathFinder actually incorporates the goal of minimizing delay of critical paths in addition to finding a way to route without congestion. (There is also a version of the approach of Togawa et al. that incorporates path length constraints with little change to area requirements [66].) The basic approach is referred to as the Negotiated Congestion (NC) algorithm. Initially nets are routed along minimum delay (least cost) paths regardless of the congestion incurred. Then successive iterations are performed of ripping up and rerouting all the nets, one at a time. At each stage of routing a net, the least cost path is again chosen, but the costs are updated from iteration to iteration. A penalty term is used to gradually increase the cost of using routing resources that are used by other signals. In addition, another cost function term is used to penalize repeated selection of the same route for a net; this is helpful in the situation where a net passing through an uncongested area must be moved to allow a net passing through a congested area to find an uncongested route. Finally, to balance delays, the cost function is actually a weighted average of the cost as described above and the cost based simply on delay; for routing resources that belong to paths of longer delay, the cost is more heavily weighted towards actual delay. The routing process terminates when all nets have been routed without congestion.

4.3 Experimental Results

We have implemented the algorithm of [65] in C, and this program and Vpr [5] are tested on a SUN SPARC IPX. We perform comparisons using each of the two

FPGA models described in Section 4.1, for fairness, in case we would find that each algorithm is better suited to one architecture or the other. The approach of Togawa et al. was originally designed with the architecture of [65] in mind, whereas other placement and routing techniques have most often been applied to an architecture as in [10]. We use different sets of benchmark circuits for the two architectures; in each of the two cases, however, we are performing a fair comparison of different algorithms presented with the same placement and routing tasks.

4.3.1 Experimental Results on the First FPGA architecture

In this section, we report on experiments based on the FPGA model from [10]. The simultaneous placement and global routing scheme and Vpr are applied to the set of circuits (for which technology mapping has already been performed) used in [10]; we also compare with the data reported in [12], the strongest previously published results for these benchmarks. (Results also appear in [10] but the authors focused more on detailed routing there and didn't optimize the performance on the placement and global routing stages.)

We list all the data in Table 4.1. In this table, W refers to the number of tracks per routing channel required to complete the global routing.

Circuit	No. Connect.	No. Blocks	No. Nets	W required by method of [65]	W for Vpr	W listed in [12]
BUSC	392	109	151	12	4	9
DMA	771	224	213	17	5	8
BNRE	1257	362	352	19	5	11
DESM	1422	401	420	17	5	9
Z03	2135	586	608	24	6	11

Table 4.1: Comparison of three placement and global routing approaches on the architecture of [10]. In this section, W refers to the number of tracks per routing channel required to complete the global routing.

4.3.2 Experimental Results on the Second FPGA architecture

In this section, we report on experiments based on the FPGA model from [65]. The simultaneous placement and global routing scheme and Vpr are applied to the set of MCNC benchmark circuits used in [65]. In this case, however, the benchmarks were not available to us after technology mapping; rather we performed technology mapping on our own using SIS [61]. In most cases, the mapping resulted in fewer blocks and nets than in the mapped circuits of [65]. This tends to lead to a more compact array of placed blocks and a larger required channel width to complete the global routing; thus the results for placement and global routing reported here cannot be directly compared to those reported in [65]. Our results are listed in Table 4.2.

Circuit	No. Blocks	No. Nets	W required by Vpr	W required by method of [65]
con1	15	13	3	3
rd53	15	12	4	4
misex1	34	27	4	5
z4ml	21	17	3	3
f51m	50	42	4	6
misex2	92	74	5	8
bw	96	68	5	9
vg2	467	459	5	17
duke2	307	278	6	21

Table 4.2: Comparison of placement and global routing approaches on the architecture of [10].

4.4 Remarks

Our experiments point to Vpr as the best existing algorithm for placement and global routing on FPGAs. It excels not only over other approaches using traditional methods but also over the novel simultaneous placement and global routing scheme of Togawa et al. While Togawa et al. obtained good results in comparison to traditional placement and global routing schemes at hand at the time, Vpr represents a significant advance over those techniques. Given our head to head comparisons of these algorithms on the same architectures and same benchmarks, we can conclude that the approach of Vpr is generally superior, rather than the comparison being very dependent on the architecture and set of circuits.

It should be noted, however, that Vpr does require more time than the method of Togawa et al. On the SUN SPARC IPX, the user CPU time for our implementation of the algorithm of Togawa et al. ranged from .09 seconds to 23 seconds; for Vpr, the times ranged from 8 seconds to 3049 seconds. An interesting area of further investigation would be to see if Vpr can continue to obtain superior results with modifications that limit its running time to something more comparable to the time required by the other algorithm.

Another interesting area for future investigation is to see if some of the ideas of by Togawa et al. can be used to develop a simultaneous placement and global routing scheme that produces higher quality results as well as good speed.

Chapter 5

Conclusions and Open Questions

This thesis has provided efficient algorithms for channel routing problems in the single-layer wiring model and the over-the-cell wiring model. In addition, we do an empirical comparison of placement and global routing algorithms for FPGAs. This section summaries the key findings in this thesis and give suggestions for further investigations.

In the single-layer context, we consider three problems in the PRAM computational models. An efficient algorithm is developed to solve the minimum separation problem for general channels. For the offset problems, we consider the restricted model, in which only one side of the channel is allowed to have multiple connections to a single net. A further improvement for the optimal offset problem is obtained in the river routing model. Furthermore, we give suggestions to incorporate a ragged boundary into the channel, where a single net is allowed to have multiple terminals. Our results can apply to this new type of channels with simple modifications.

With more recent developments in parallel computing, an obvious open question is whether any of the bounds on time or work can be improved (though the work bounds are clearly optimal for the minimum separation and offset range problems.) Particularly promising may be the possibility of improving the running time by using randomized or approximate solution schemes for certain tasks.

In Chapter 3, this thesis provides an efficient algorithm for channel density reduction in over-the-cell routing. Research work in this wiring model has focused mostly on reducing the channel density of the given circuits by making use of the area outside the channels. However, for high performance circuits, this improvement is not enough to overcome the increase in the total wire length and the length of critical paths. Hence, more research is needed in this area to reduce the channel width as well as to improve performance. In addition, research can be directed toward developing new cell models and also to take advantage of recently developed techniques in parallel computing for further improvements.

The last part of this thesis shows the results of an empirical comparison of placement and global routing for FPGAs. Given a global routing, there is no guarantee to obtain a successful detailed routing. In FPGAs, this problem becomes even harder due to the limited routing resources. In order to achieve a complete layout, a significant amount of researches has to be done in order to fully utilize the potential of FPGAs. However, most works reported in the literature are rather architecture dependent, being designed in particular for row-based or symmetric FPGAs. It would be a great breakthrough to develop good algorithms that are universal to all FPGA architectures.

Bibliography

[1] A. Aggarwal and J. K. Park. Parallel searching in multidimensional monotone arrays. *Journal of Algorithms.* To appear. Earlier versions appear as IBM Research Report RC 14826 and in *29th Annual Symposium on Foundations of Computer Science.*

[2] O. Berkman, Baruch Schieber, and U. Vishkin. Some doubly logarithmic optimal parallel algorithms based on finding all nearest smaller values. Technical Report UMIACS-TR-88-79, University of Maryland Institute for Advanced Computer Studies, October 1988. To appear in *J. Algorithms.*

[3] O. Berkman and U. Vishkin. Recursive star-tree parallel data-structure. Technical Report UMIACS-TR-90-40, University of Maryland Institute for Advanced Computer Studies, March 1990. Earlier version in *30th Annual Symposium on Foundations of Computer Science.*

[4] V. Betz and J. Rose. On biased and non-uniform global routing architectures and CAD tools for FPGAs. Technical report, University of Toronto, 1996.

[5] Vaughn Betz. Vpr user's manual. University of Toronto, 1996.

[6] D. Bostick, G. D. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrrison, and D. Ravenscroft. The BOULDER optimal logic de-

sign system. In *IEEE International Conference on Computer-Aided Design (ICCAD-94)*, pages 62–65. IEEE Computer Society Press, 1994.

[7] Douglas Braun, Jeffrey L. Burns, Fabio Romeo, Alberto Sangiovanni-Vincentelli, Kartikeya Mayaram, Srinivas Devadas, and Hi-Keung Tony Ma. Techniques for multi-layer channel routing. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 7(6):698–712, June 1988.

[8] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: a multiple-level logic optimization system. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 6(6):1062–1081, November 1987.

[9] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *Field Programmable Gate Arrays.* Kluwer Acedemic, 1992.

[10] Stephen Brown, J. Rose, and Z. Vranesic. A detailed router for field programmable gate arrays. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 11(5), 1992.

[11] Shing-Chong Chang, Joseph JáJá, and Kwan Woo Ryu. Optimal parallel algorithms for one-layer routing. Technical Report UMIACS-TR-89-46, University of Maryland Institute for Advanced Computer Studies, April 1989.

[12] Yao-Wen Chang, S. Thakur, Kai Zhu, and D.F. Wong. A new global routing algorithm for FPGAs. In *IEEE International Conference on Computer-Aided Design (ICCAD-94)*, pages 356–361. IEEE Computer Society Press, 1994.

[13] C. S. Chen, Y. W. Tsay, T. T. Hwang, Allen Wu, and Y. L. Lin. Combining technology mapping and placement for delay-optimization in FPGA designs. In *IEEE International Conference on Computer-Aided Design (ICCAD-93)*, pages 123–127. IEEE Computer Society Press, 1993.

[14] Yun Kang Chen and Mei Lun Liu. Three-layer channel routing. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, CAD-3(2):156–163, April 1984.

[15] R. Cole and U. Vishkin. Faster optimal prefix sums and list ranking. *Information and Control*, 81:334–352, 1989.

[16] Jason Cong and Yuzheng Ding. An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. In *IEEE International Conference on Computer-Aided Design (ICCAD-92)*, pages 48–53. IEEE, 1992.

[17] Jingsheng Cong and C. L. Liu. Over-the-cell channel routing. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 9(4):408–418, April 1990.

[18] Jingsheng Cong, D. F. Wong, and C. L. Liu. A new approach to three- or four-layer channel routing. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 7(10):1094–1104, October 1988.

[19] Sandip Das, Subhas C. Nandy, and Bhargab B. Bhattacharya. An improved heuristic algorithm for over-the-cell channel routing. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 3106–3109, 1991.

[20] D. Deutsch. A 'dogleg' channel router. In *Proceedings of the 13th ACM/IEEE Design Automation Conference*, pages 425–433. IEEE Computer Society Press, 1976.

[21] Danny Dolev, Kevin Karplus, Alan Siegel, Alex Strong, and Jeffrey D. Ullman. Optimal algorithms for structural assembly. *VLSI Design*, pages 38–43, 1982. Earlier version in *Proceedings of the 13th ACM Symposium on Theory of Computing.*

[22] R. J. Enbody and H. C. Du. Near-optimal n-layer channel routing. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 708–714. IEEE Computer Society Press, 1986.

[23] R. J. Francis, J. Rose, and Z. G. Vranesic. Technology mapping for delay optimization of lookup table-based FPGAs. In *MCNC Logic Synthesis Workshop*, 1991.

[24] Ronald I. Greenberg, Shih-Chuan Hung, and Jau-Der Shih. Parallel algorithms for single-layer channel routing. In *Algorithms and Computation: 4th International Symposium, ISAAC '93 Proceedings*, pages 456–465. Springer-Verlag, 1993. Revised version submitted to *Information Processing Letters.*

[25] Ronald I. Greenberg, Alexander T. Ishii, and Alberto L. Sangiovanni-Vincentelli. MulCh: A multi-layer channel router using one, two, and three layer partitions. In *IEEE International Conference on Computer-Aided Design (ICCAD-88)*, pages 88–91. IEEE Computer Society Press, 1988.

[26] Ronald I. Greenberg and Charles E. Leiserson. A compact layout for the three-dimensional tree of meshes. *Applied Mathematics Letters*, 1(2):171–176, 1988. Also see erratum in vol. 1, no. 3, p. 315.

[27] Ronald I. Greenberg and F. Miller Maley. Minimum separation for single-layer channel routing. *Information Processing Letters*, 43(4):201–205, September 1992.

[28] Ronald I. Greenberg and Jau-Der Shih. Single-layer channel routing and placement with single-sided nets. *Computers & Mathematics With Applications*, 1993. To appear.

[29] Ronald I. Greenberg and Jau-Der Shih. Feasible offset and optimal offset for general single-layer channel routing. *SIAM Journal on Discrete Mathematics*, 8(4):543–554, November 1995.

[30] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal. Segmented channel routing. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 567–572. ACM/IEEE, 1990.

[31] G. Gudmundsson and S. Ntafos. Channel routing with superterminals. In *Proceedings of the 25th Annual Allerton Conference on Communication, Control, and Computing*, pages 375–376, 1987.

[32] Torben Hagerup. Constant-time parallel integer sorting. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 299–306. ACM Press, 1991.

[33] Akihiro Hashimoto and James Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the 8th ACM/IEEE*

Design Automation Conference, pages 155–169. IEEE Computer Society Press, 1971.

[34] Nancy D. Holmes, Naveed A. Sherwani, and Majid Sarrafzadeh. New algorithm for over-the-cell channel routing using vacant terminals. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 126–131. IEEE Computer Society Press, 1991.

[35] Cliff Y. Hou and C. Y. Roger Chen. A pin permutation algorithm for improving over-the-cell channel routing. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 594–599. IEEE Computer Society Press, 1992.

[36] Shih-Chuan Hung. An efficient algorithm for reducing channel density by over-the-cell routing. In *Proceedings of International Computer Symposium*, 1994.

[37] Shih-Chuan Hung and Ronald I. Greenberg. An empirical comparison of placement and global routing algorithms for FPGAs. Technical Report EE-TR-96-26, University of Maryland, 1996.

[38] Joseph JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[39] R. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machine. Technical Report UCB/CSD 88/408, University of California, Berkeley, CA, 1988.

[40] K. Karplus. Xmap: A technology mapper for table-lookup field programmable gate arrays. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 240–243. ACM/IEEE, 1991.

[41] Evagelos Katsadas and Edwin Kinnen. A multi-layer router utilizing over-cell areas. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 704–708. IEEE Computer Society Press, 1990.

[42] K. Kawana, H. Keida, M. Sakamoto, K. Shibata, and I. Moriyama. An efficient logic block interconnect architecture for user-reprogrammable gate array. In *Proceedings of IEEE 1990 Custom Integrated Circuits Conference*, pages 31.3.1–31.3.4, 1990.

[43] Thomas F. Knight, Jr. Technologies for low latency interconnection switches. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 351–358. Association for Computing Machinery, 1989.

[44] C. Kruskal. Searching, merging and sorting in parallel computation. *IEEE Trans. Computers*, C-32(10):942–946, October 1983.

[45] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[46] Andrea S. LaPaugh and Ron Y. Pinter. Channel routing for integrated circuits. *Annual Review of Computer Science*, 4:307–363, 1989.

[47] Andrea Suzanne LaPaugh. *Algorithms for Integrated Circuit Layout: An Analytic Approach.* PhD thesis, Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology, November 1980. MIT/LCS/TR-248.

[48] Frank Thomson Leighton and Arnold L. Rosenberg. Three-dimensional circuit layouts. *SIAM Journal on Computing*, 15(3):793–813, August 1986.

[49] Min-Siang Lin, Hourng-Wern Perng, Chi-Yi Hwang, and Youn-Long Lin. Channel density reduction by routing over the cells. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 120–124. IEEE Computer Society Press, 1991.

[50] F. Miller Maley. *Single-Layer Wire Routing and Compaction*. MIT Press, 1990.

[51] Yossi Matias and Uzi Vishkin. Converting high probability into nearly-constant time — with applications to parallel hashing. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 307–316. ACM Press, 1991.

[52] Larry McMurchie and Carl Ebeling. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 111–117, 1995.

[53] Andranik Mirzaian. River routing in VLSI. *Journal of Computer and System Sciences*, 34:43–54, 1987.

[54] Sudip K. Nag and Rob A. Ruteenbar. Performance-driven simultaneous place and route for row-based FPGAs. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pages 301–307. IEEE Computer Society Press, 1994.

[55] Rajat K. Pal, Sudebkumar P. Pal, Ajit Pal, and Alak K. Dutta. Np-completeness of multi-layer no-dogleg channel routing and an efficient

heuristic. In *Proceedings of the 6th International Conference on VLSI Design*, pages 80–83, Bombay, India, 1993.

[56] Rajat Kumar Pal, Sudebkumar Prasant Pal, and Ajit Pal. On the computational complexity of multi-layer channel routing. Technical Report TR/IIT/CSE/92/02, Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur, December 1992.

[57] James Reed, Alberto Sangiovanni-Vincentelli, and Mauro Santomauro. A new symbolic channel router: YACR2. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(3):208–219, July 1985.

[58] Ronald L. Rivest, Alan E. Baratz, and Gary Miller. Provably good channel routing algorithms. In H. T. Kung, Bob Sproull, and Guy Steele, editors, *VLSI Systems and Computations*, pages 153–159. Computer Science Press, 1981.

[59] Ronald L. Rivest and Charles M. Fiduccia. A "greedy" channel router. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 418–424. IEEE Computer Society Press, 1982.

[60] J. Rose, R. J. Francis, D. Lewis, and P. Chow. Architecture of programmable gate arrays: The effect of logic block functionality on area efficiency. *IEEE Journal of Solid-State Circuits*, 25(5):1217–1225, October 1990.

[61] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL Memorandum M89/49, U. of California at Berkeley, May 1992.

[62] Thomas G. Szymanski. Dogleg channel routing is NP-complete. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(1):31–41, January 1985.

[63] M. Terai, K. Takahashi, K. Nakajima, and K. Sato. A new model for over-the-cell channel routing with three layers. In *IEEE International Conference on Computer-Aided Design (ICCAD-91)*, pages 432–435. IEEE Computer Society Press, 1991.

[64] Nezomu Togawa, Masao Sato, and Tatsuo Ohtsuki. Maple: A simultaneous technology mapping, placement and global routing algorithm for field-programmable gate arrays. In *IEEE International Conference on Computer-Aided Design (ICCAD-94)*, pages 156–163. IEEE Computer Society Press, 1994.

[65] Nezomu Togawa, Masao Sato, and Tatsuo Ohtsuki. A simultaneous placement and global routing algorithm for FPGAs. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 483–486, 1994.

[66] Nezomu Togawa, Masao Sato, and Tatsuo Ohtsuki. A simultaneous placement and global routing algorithm for symmetric FPGAs. In *Proceedings FPGA'94*, 1994.

[67] Takeshi Yoshimura and Ernest S. Kuh. Efficient algorithms for channel routing. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, CAD-1(1):25–35, January 1982.