NON-RECTANGULAR EMBEDDED
PROGRAMMABLE LOGIC CORES

by

Tony Yau-Wai Wong

B.A.Sc, University of British Columbia, 1998

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Applied Science

in

The Faculty of Graduate Studies

Department of Electrical and Computer Engineering

We accept this thesis as conforming to the
required standard:

The University of British Columbia

May 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of _Electrical and Computer Engineering_

The University of British Columbia
Vancouver, Canada

Date _May 31, 2002_

ABSTRACT

NON-RECTANGULAR EMBEDDED PROGRAMMABLE LOGIC CORES

As System-on-a-Chip (SoC) design enters into mainstream usage, the ability to make post-fabrication changes will become more and more attractive. This ability can be realized using programmable logic cores. These cores are like any other *intellectual property* (IP) in the SoC design methodology, except that their function can be changed after fabrication. In many cases, non-rectangular programmable logic cores are required, either to better mesh with the other IP cores, or because of I/O constraints. However, most CAD algorithm and programmable logic architecture research targets stand-alone field programmable gate arrays (FPGA's), which are invariably square or rectangular. In this thesis, we enable researchers to evaluate non-rectangular programmable logic cores by a novel specification method and an enhanced CAD tool. We also show that existing placement and routing algorithms do not work well when targeting non-rectangular programmable logic cores, and we present enhancements to existing placement and routing algorithms that allow the algorithms to better target these cores. It is shown that the new algorithms lead to a 12% critical path improvement for "U"-shaped cores, and a 4% critical path improvement for "O"-shaped cores. The density and speed penalty for using these non-rectangular cores is significant, compared to square cores, however, we show that the penalty would be significantly larger if the original algorithms were used.

TABLE OF CONTENTS

iv

LIST OF FIGURES AND TABLES

ACKNOWLEDGMENTS

Chapter 1

OVERVIEW AND INTRODUCTION

1.1   Motivation

Recent years have seen impressive improvements in the achievable density of integrated circuits. In order to maintain this rate of improvement, designers need new techniques to handle the increased complexity inherent in these large chips. One such emerging technique is the System-on-a-Chip (SoC) design methodology. In this methodology, pre-designed and pre-verified blocks, often called *cores* or *intellectual property (IP)* are obtained from internal sources or third parties, and combined onto a single chip. These cores may include embedded processors, memory blocks, or circuits that handle specific processing functions. The SoC designer, who would have only limited knowledge of the structure of these cores, could then combine them onto a chip to implement complex functions.

No matter how seamless the SoC design flow is made, and no matter how careful an SoC designer is, there will always be some chips that are designed, manufactured, and then deemed unsuitable. This may be due to design errors not detected by simulation or it may be due to a change in requirements. This problem is not unique to chips designed using the SoC methodology. However, the SoC methodology provides an elegant solution to the problem: one or more embedded programmable logic cores

(EPLC's) can be incorporated into the SoC. The embedded programmable logic core is a flexible logic fabric that can be customized to implement *any* digital circuit *after fabrication*. Before fabrication, the designer embeds a programmable fabric (consisting of many uncommitted gates and programmable interconnects between the gates). After the fabrication, the designer can then program these gates and the connections between them. Several companies, including Actel, Adaptive Silicon, Atmel, eASIC, Lucent and QuickLogic already provide EPLC's [44,45,46,47,48,49].

Figure 1 shows two hypothetical examples of where EPLC's may be beneficial. In Figure 1(a), an EPLC is shown that implements interface logic between the other ASIC cores inside the chip and the peripherals outside the chip. As standards change, it is clearly beneficial if this interface logic is flexible. Figure 1(b) shows another use of EPLC's; in this case, the EPLC's are used as test logic controllers in SoC design [1]. This allows a test engineer to implement new test stimulus and/or test analysis circuits on the programmable core after the chip is fabricated. As testing proceeds, if errors are found, new tests can be devised and the new on-chip test circuitry can be implemented in the EPLC.
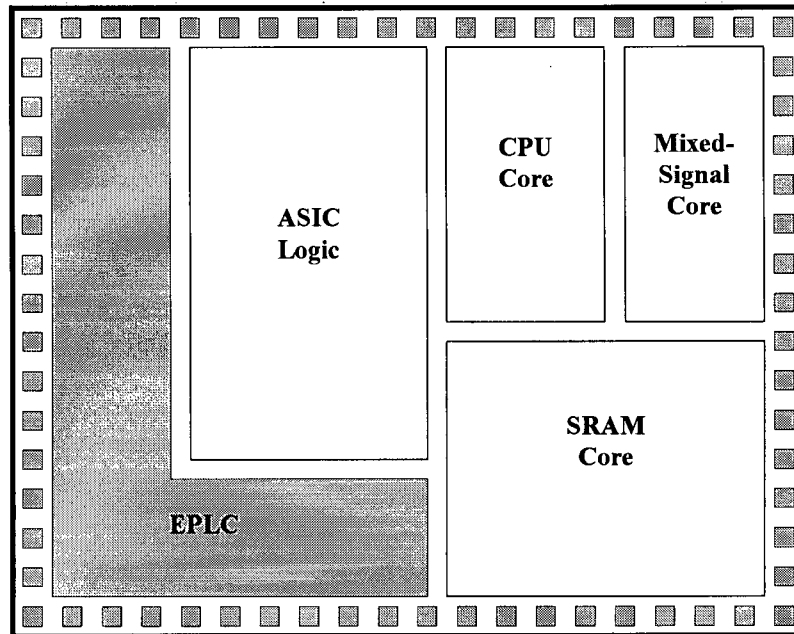
Figure 1.1(a) An "L"-shaped core used for interface logic
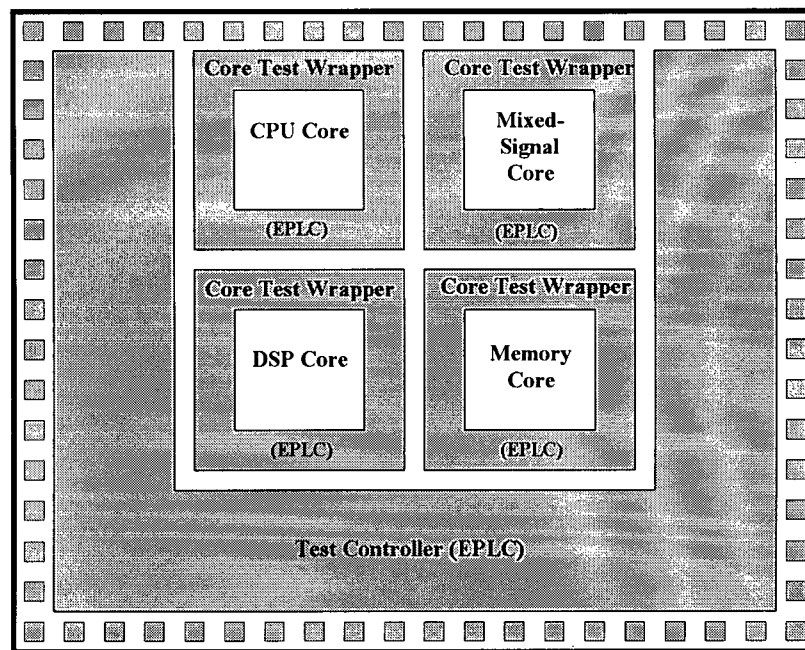
Figure 1.1(b) One "U"-shaped and four "O"-shaped cores used for test wrappers and control

3

These examples show two cases in which embedded programmable logic in a SoC would be advantageous. In general, there are a number of reasons to do this:

1. An EPLC enables communication chip designers to proceed with chip development before standards have been finalized. This is important, since time-to-market is critical in industry today. As an example, a network chip can be built in which the network protocol is implemented in programmable logic and the remainder is designed using fixed ASIC logic or fixed-function cores.

2. It is possible to make a single chip design that will be used by several customers. In this case, an EPLC would incorporate the customer-specific portion of the chip. Also, it could reduce the cost of developing the ASIC over several products. Figure 1.1a illustrates such chip design, where the processing functions and memory, which are common among all customers, are implemented in fixed ASIC logic and fixed-function cores, whereas the interface to the processing is implemented in programmable logic.

3. Some CAD tool vendors offer platform-based design. Using this design methodology, the CAD tool vendor provides basic hardware components for particular application needs. With EPLC's, the CAD tool vendor can now encapsulate the different requirements from customers into the programmable logic, and leave the

4

unchanged components implemented in fixed logic. This would greatly reduce the amount of work needed by the CAD tool vendors to customize the platform for future customers.

In order to use EPLC's effectively in SoC design, there are a number of essential issues that have to be addressed. For example, most CAD algorithm and programmable logic architecture research targets stand-alone field programmable gate arrays (FPGA's) [2, 41] which are invariably square or rectangular. In SoC design, however, it may be desirable to use an EPLC of a different shape. As shown in Figure 1.1(a), for example, the EPLC is "L"-shaped; not only may this better mesh with the other cores, but an L-shape may be very suitable if the I/O associated with this block spans more than one edge of the chip. In Figure 1.1(b), both "O"-shaped and "U"-shaped EPLC's can be seen. "O"-shaped EPLC's can be used as a test "wrapper" around other fixed-function cores, in this case, to map test signals to the cores. "U"-shaped EPLC's may also be used when a complete wrapping is not required. In any case, it is clear that EPLC's should be able to take on shapes other than rectangular. Therefore, new research has to be conducted for non-rectangular EPLC's if we want to use them in SoC design.

In addition, researchers must determine how best to integrate the FPGA CAD flow into the existing ASIC design flow in order to make it possible for chip designers to use EPLC's. Another problem is to how to verify the programmable portion in an ASIC chip in the pre-tape-out stage when the circuit implemented in the EPLC is unknown.

1.2 Research Goals

In this work, we focus on one of the important issues: CAD algorithms and architectures for non-rectangular EPLC's. Although there have been considerable research on CAD algorithms and architectures for programmable logic, all of them assume the shape of programmable logic is square or rectangular, which is true for stand-alone FPGA's. However, EPLC's can take on a variety of shapes and aspect ratios such as shown in Figures 1.1(a) and 1.1(b), and it seems likely that the existing algorithms used for stand-alone FPGA's may not perform well for non-rectangular EPLC's. In addition, it is unclear how to efficiently use non-rectangular cores in SoC design rather than rectangular cores to optimize area and delay. Therefore, an evaluation for the algorithms and architectures that target non-rectangular EPLC's is required.

In this paper, we focus on three aspects:

1. Design of a new specification method for describing a non-rectangular EPLC and providing an evaluation CAD tool to support the new specification format.

2. Improvement of the existing placement and routing algorithms on the evaluation CAD tool that better targets to non-rectangular EPLC's.

3. Using the enhanced CAD tool to evaluate the architectures of "L"-, "U"- and "O"-shaped EPLC's for area and delay efficiency.

The first goal is to create a simple specification method to enable users to enter device information such as the shape of an EPLC architecture to an evaluation CAD tool.

The second goal is to optimize the existing placement and routing algorithms used in a popular public FPGA CAD tool, VPR [6] for non-rectangular EPLC architectures. It is unknown how efficiently the existing placement and routing algorithms can map a user circuit into a non-rectangular core. We believe the placer and router can be improved by careful examination of both algorithms for non-rectangular cores.

The third goal of this research is to use the enhanced evaluation CAD tool to study the area and speed performances of non-rectangular EPLC architectures. Three EPLC's are studied: "L"-shaped, "U"-shaped and "O"-shaped cores. Using the enhanced placer and router, we quantify the area and delay results of "L"-, "U"- and "O"-shaped cores and hence determine the penalty of using these non-rectangular cores compared to square cores.

1.3 Organization of This Thesis

This thesis is organized as follows: Chapter 2 contains an introduction to EPLC architectures as well as to CAD algorithms that are used to map a user circuit onto an EPLC. Chapter 3 introduces the novel specification method that enables users to describe a non-rectangular EPLC in an architecture file being used by an evaluation CAD tool. Chapter 4 focuses on the algorithmic issues in the evaluation CAD tool targeting non-rectangular cores. Specifically, we propose enhancements to the placement and routing algorithms, and present experimental results that show that the

7

enhanced placer and router lead to higher quality results and runtime savings. In Chapter 5, we measure the delay and area performances of "L"-, "U"- and "O"-shaped cores by the enhanced placer and router, and evaluate the feasibility of using these three types of cores by comparing them to square cores. Finally, we conclude the thesis with a summary of the work presented and future work, and also summarize the contributions of this work.

Chapter 2

BACKGROUND AND PREVIOUS WORK

In this chapter, we present an overview of the architecture of EPLC's. In addition, we also cover the CAD flow used for mapping user circuits into EPLC's, along with a detailed discussion of the placement and routing algorithms.

2.1   EPLC Architecture

The architecture of commercially available EPLC's varies from vendor to vendor [44,45,46,47,48,49]. However, the architecture of EPLC's inherits much from the architecture of stand-alone FPGA's. In these devices, configurable logic blocks are placed in a grid, separated by horizontal and vertical wiring channels containing fixed metal tracks. These metal tracks are connected to each other and to the logic blocks using programmable switches as shown in Figure 2.1. This is often termed an island-style architecture. In this study, we will exclusively investigate island-style EPLC's.

logic blocks

wiring
channels
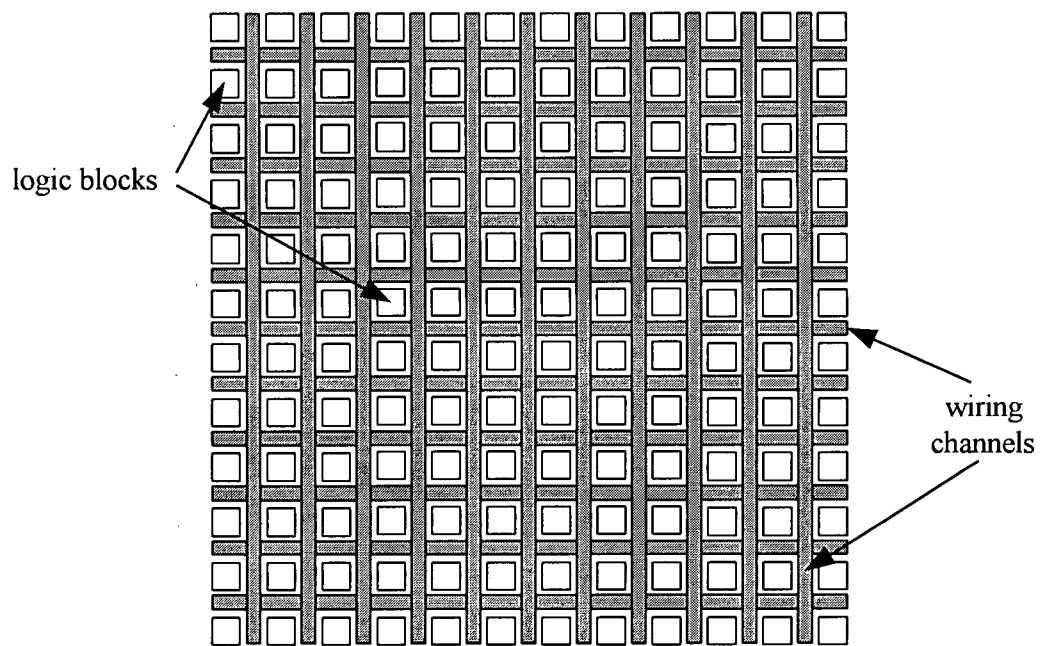
Figure 2.1 Island-style EPLC architecture

Constructing a non-rectangular core is straightforward. The island-style architecture in

most EPLC's provides a natural way to implement "L"-, "U"-, and "O"-shaped cores;

one or more logic blocks and the routing fabric around these logic blocks can be

removed from a rectangular core to form an "L"-, "U"-, or "O"-shaped core. An

example is shown in Figure 2.2; in this case, the logic blocks in the shaded region can

be removed to form a "U"-shaped core.

Shaded region is removed to
form a "U"-shaped core   logic blocks
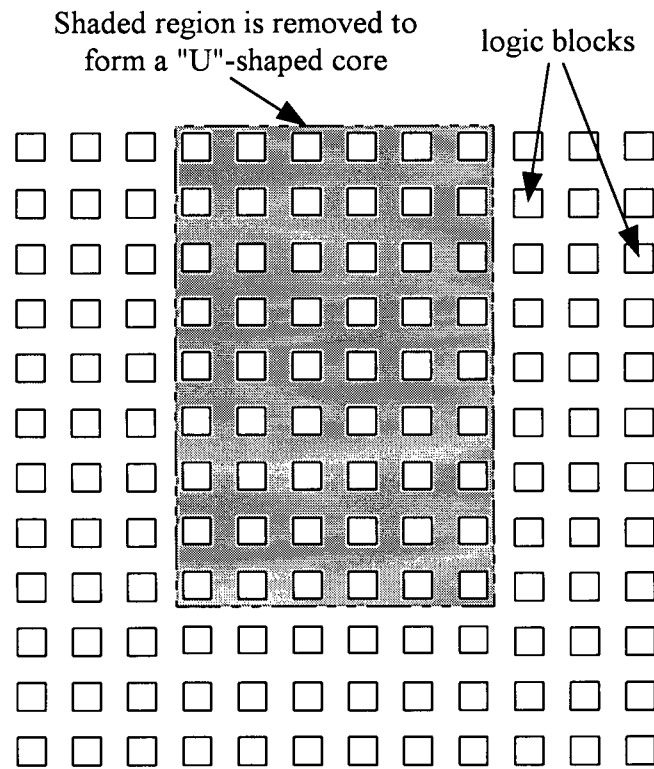
Figure 2.2 Constructing a "U"-shaped core from a square core

All EPLC architectures are comprised of logic resources and routing resources. In Subsections 2.1.1 and 2.1.2, we will discuss the architecture of logic resources and routing resources for the island-style EPLC architecture respectively.

2.1.1 Logic Resources

A logic block (depicted in Figure 2.3) is the building block of an island-style EPLC architecture. Each logic block implements a small part of the logic required by a user circuit. By connecting logic blocks together properly, an EPLC can implement the entire user circuit.

11
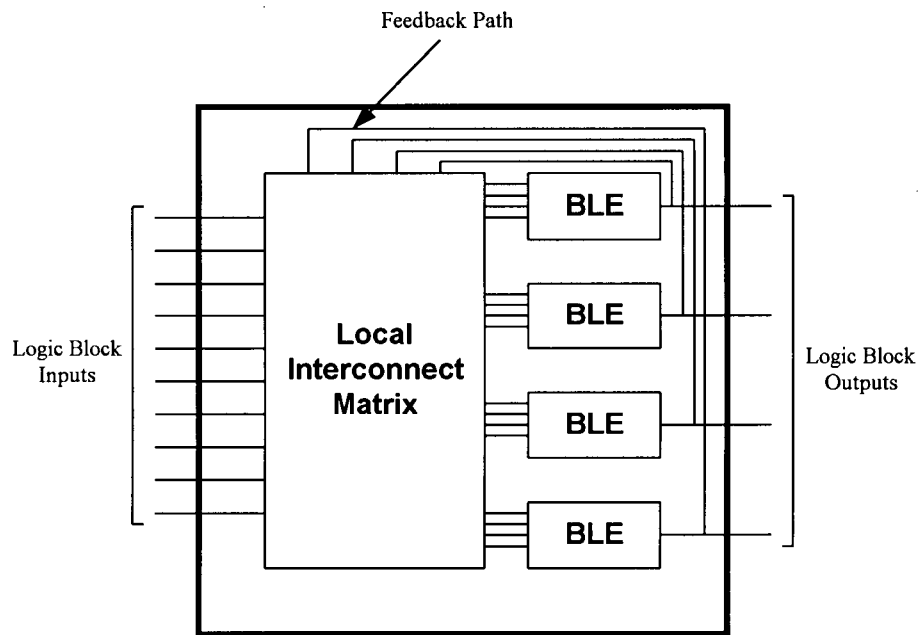
Figure 2.3 Logic block architecture

Inside a logic block, there are one or more basic logic elements (BLEs). Each BLE contains a lookup-table (LUT) and a register. The register is used for implementing a sequential circuit, and can be enabled or disabled by using the 2:1 multiplexer and a user-programmable SRAM cell. Figure 2.4 shows the internal structure of a basic logic element.
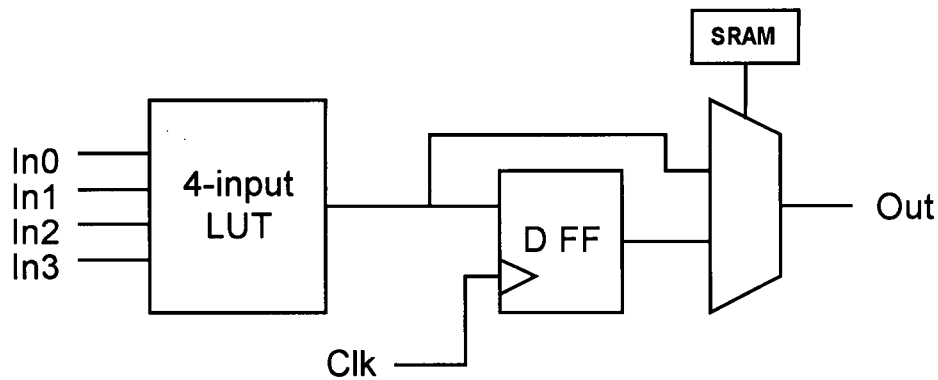
Figure 2.4 Simplified model of BLE

Lookup-tables can be used to implement logic very efficiently because a k-input LUT can realize any function of k-inputs. Rose *et al.* [12] have shown that a 4-input LUT produces the highest area-efficiency, and the result is widely accepted by industry. We will use 4-input LUT for the rest of this study. Figure 2.5 shows a 4-input LUT, implemented by sixteen 1-bit SRAM cells and five 4:1 multiplexers.
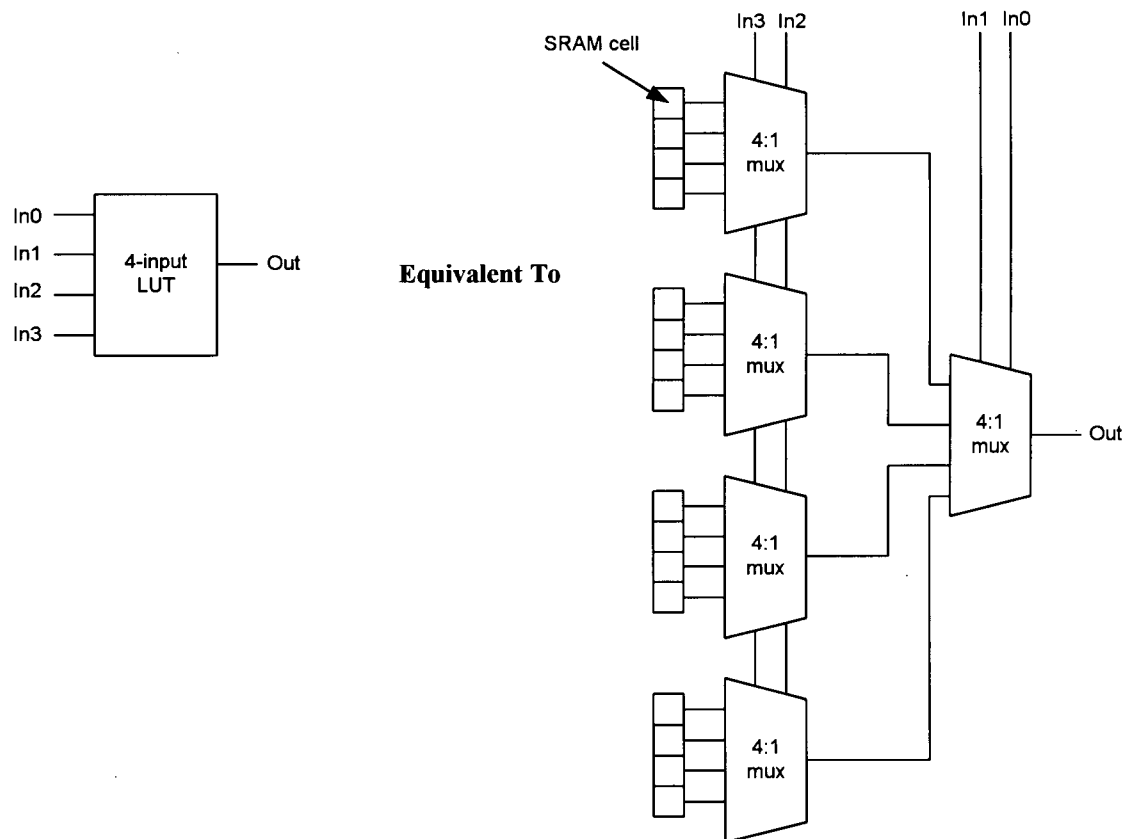
Figure 2.5 4-input lookup-table

The previous research [43] has shown that is preferable to include more than one BLE in each logic block. The user (via a CAD tool) can then cluster tightly connected BLEs together by the programmable local interconnect matrix, thereby reducing the

necessary global routing resources. This decreases the routing area and delay. However, if the number of BLEs in a logic block grows too large, the area and delay savings in global routing will be outweighed by the area and delay penalties imposed from the local interconnect within the logic blocks. Ahmed *et al.* [4] found that the number of BLEs in a logic block for the best area-delay efficiency ranges from four to ten. In our study, a logic block size of four is chosen, as illustrated in Figure 2.3.

2.1.2 Routing Resources

The routing resources enable the interconnect between logic blocks as well as between BLE's within a logic block. They are also used to connect the off-chip signals to the logic blocks through I/O pads. Routing resources are crucial to the overall area and speed because they account for a significant amount of chip space and critical path delay [17,36]. The routing resources in an EPLC can be categorized as:

1. Routing between logic blocks

2. Routing inside logic blocks

Figure 2.6 shows only the routing between logic blocks in a single tile. Most EPLC's are created by replication of such a tile (a tile contains one logic block and its associated routing fabric).
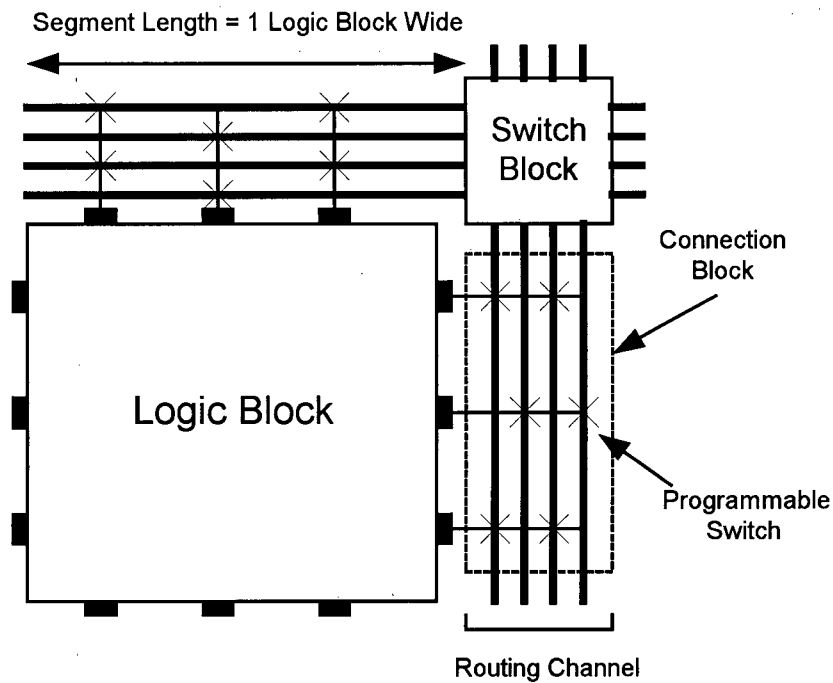
Figure 2.6 Routing architecture in an island-style EPLC

Routing Between Logic Blocks

The routing between logic blocks contains three components:

1. Routing channels

2. Switch blocks

3. Connection blocks

Each routing channel consists of a number of fixed metal tracks that run either horizontally or vertically. In most EPLC's, each track in a channel is not a single long wire spanning the entire length of an EPLC. Instead, the wires are broken into a number of smaller wire segments, and each wire segment spans one or more logic

15

blocks. In this work, we focus on island-style EPLC architectures in which each channel contains the same number of tracks and each metal track spans one logic block as shown in Figure 2.6.

A switch block occurs at each intersection between horizontal and vertical routing channels, as shown in Figure 2.6 determines all possible connections between these channels. Four different topologies for switch blocks have been proposed in the previous work: the disjoint switch block [52], the universal switch block [13], the Wilton switch block [8] and the Imran switch block [14]. In this work, the Wilton switch block is used for all the EPLC architectures because it has the best area-efficiency when all the routing wires span one logic block [5].

A connection block shown in Figure 2.6 connects the logic block pins to the routing channel and vice versa. The number of tracks in each routing channel to which each logic input and output pin can connect is called the connection block flexibility. Betz [5] has showed that the Wilton switch block works best when each logic block input and output pin connects to one quarter of the tracks in the neighboring routing channel. This is also adopted in this work.

The connections in the connection blocks and the switch blocks are made by programmable switches. Each programmable switch contains either a pass transistor or a tri-state buffer, controlled by a SRAM cell shown in Figure 2.7. In this study, we will exclusively focus on SRAM-based EPLC's.

Pass Transistor   Tri-state Buffer
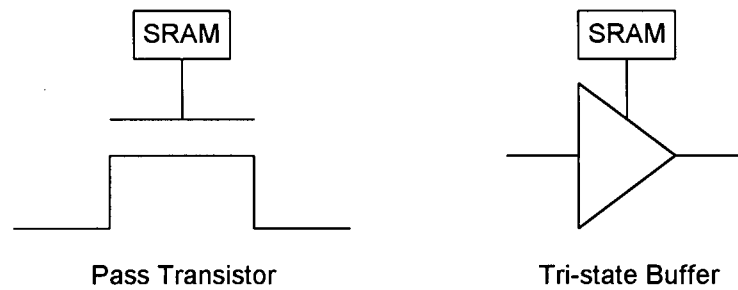
Figure 2.7 Two types of SRAM-based programmable switch

Routing Inside Logic Blocks

Routing within a logic block is performed using a local interconnect matrix as shown in Figure 2.3. The interconnect matrix allows connections between the logic block inputs and the BLE inputs in the logic block, and also enables the feedback paths from the BLE outputs to the BLE inputs in the logic block. If any of the logic block inputs, and any of the BLE outputs can be connected to any of the BLE inputs, the interconnect matrix is called fully connected. If only some of the logic inputs, and the feedbacks can be connected to any of the BLE inputs, the interconnect matrix is called depopulated. In this study, we use a fully connected interconnect matrix because it is most widely used in industry [51,52]. Further studies on the architecture of the interconnect matrix can be found in [15,16,17].

2.2   CAD for EPLC's

The main function of an EPLC CAD tool is to convert a high-level circuit description (that a human can understand) into a programming file (unreadable format) setting the state of every programmable switch in an EPLC, in order to realize the user circuit.

17

The conversion is carried out through a number of sequential steps, as depicted in Figure 2.8.

High-Level Circuit Description

High-Level Synthesis

Technology-Independent Logic Optimization

Technology Mapping

Packing into Logic Blocks

Placement

Routing
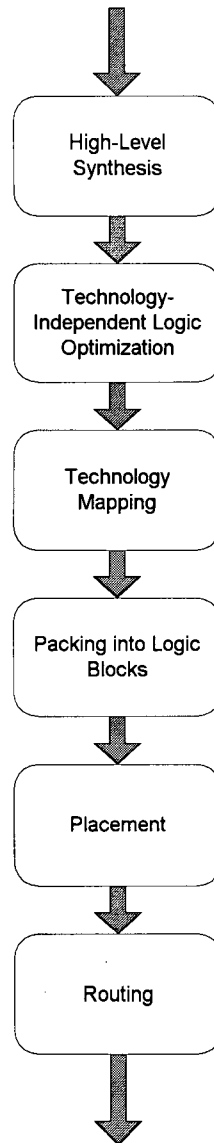
EPLC Programming File

Figure 2.8 A typical EPLC CAD flow

First, users describe their circuit in high-level description language (such as VHDL or Verilog) or schematic, and then synthesize their description into a netlist of basic gates. This step is called high-level synthesis [18,19]. Then the netlist of gates is processed by a technology-independent logic optimization algorithm [19] in which the redundant logic is removed. Next, it is mapped into lookup tables by a technology mapping algorithm [10]. Next, the netlist of lookup tables is packed into logic blocks [7]. Then, each logic block is assigned a physical location in an EPLC using a placement algorithm [6]. Next, the connections among logic blocks will be routed on the wire segments on an EPLC by a routing algorithm [6]. Finally, all the connections are made among the logic blocks and the I/O pads.

All steps prior to placement can operate independently of the shape of the EPLC. Placement and routing, however, are inherently geometrical in nature; thus, we would expect these algorithms to be strongly influenced by the shape of the target core. In Chapter 4, we will show that this intuition is correct. Therefore, in the following discussion (and the rest of the thesis) we focus on non-rectangular placement and routing.

2.2.1 Placement

Placement algorithm determines the exact location of user logic blocks on an EPLC. Typically, placement has three optimization goals:

1. Minimize Wire-Length

2. Minimize Congestion

3. Minimize Critical Path Delay

The first goal is to reduce the amount of wire needed to make all required connections between the logic blocks. The reduction in wiring directly results in a denser and usually faster EPLC implementation for a circuit. One way to achieve this goal is to try to position tightly connected logic blocks close to each other.

The second goal is to balance the wiring usage across the EPLC to avoid congestion of routing resources. Since most of EPLC's have evenly distributed routing channels, the circuit could be unroutable if an area in the EPLC is so congested that the required channel width exceeds the channel width offered in EPLC.

The third goal is to minimize the circuit delay on an EPLC. In order to achieve this objective, placement should be able to recognize which net is more timing-critical, and then place the logic blocks which are connected by more timing-critical nets closer together to minimize the path delay. Further details on these goals will be discussed in Chapter 4.

There are a number of well-known algorithms to solve the placement problem, such as min-cut (partitioning-based) [20], analytic [22,23], and simulated annealing [24,25,26]. In this study, we focus on simulated annealing algorithm because it was employed in the popular public FPGA CAD tool called VPR [6]; this is the CAD tool upon which our algorithms will be built.

The simulated annealing algorithm originates from the concept of the industrial annealing process used to gradually cool molten metal [26]. The pseudo-code for a generic simulated annealing-based placer is depicted in Figure 2.9. Initially, a simulated annealing-based placer randomly places logic blocks into physical locations in an EPLC. Then the placement is iteratively improved by randomly swapping logic blocks and measuring the placement cost resulting from every swap by a cost function. A cost function is used to evaluate the quality of any placement of logic blocks, such as wire-length cost, congestion cost and path delay cost. If a swap results in a reduction in the placement cost, the move is always accepted. If the placement cost increases, there is still a chance of the move being accepted even though it makes the placement worse. The probability of acceptance is modeled by a formula $e^{-\Delta C/T}$, where ΔC is the positive change in cost function the move causes, and T is the "temperature". At first, T is very high and almost all moves are accepted; however, it is gradually decreased when the placement is refined so that near the end of the anneal, probability of accepting a move that makes the placement worse is very low. The purpose of accepting the moves that make a placement worse is to prevent the simulated annealing-based placer from becoming trapped in a local minimum in the cost function.

The rate at which temperature is decreased, the exit condition for stopping the annealing, the number of moves tried at each temperature (InnerLoopCriterion), and how each potential move is generated, are specified by the annealing schedule. A good annealing schedule is very important to achieve good results in a reasonable runtime.

```
P = RandomPlacement();
T = InitialTemperature();

while (ExitCriterion() == false) {
   while (InnerLoopCriterion() == false) {
      P_new = GenerateNewMove(P);
      ΔC = Cost(P_new) – Cost(P);
      r = random(0,1);
      If (r < e^(-ΔC/T)) {
         P = P_new;
      }
   }
   T = UpdateTemperature();
}
```

Figure 2.9 Pseudo-code of a generic simulated annealing-based placement algorithm

2.2.2 Routing

The routing algorithm assigns the routing resources to all the nets in the user circuit.

Generally, there are two optimization goals for a router:

1. Complete routing of all nets

2. Delay Minimization

The first goal is to route all required nets using the routing resources on an EPLC

without any resource contention. This objective is hard to achieve in EPLCs because

the routing resources are fixed. However, if there is contention unresolved by a router,

the user circuit cannot be implemented in an EPLC. Therefore, this is the most

important goal.

22

The second goal is to minimize the circuit delay by using short paths and fast routing resources for nets which are on or near the critical path. However, this second goal is always competing with the first goal. A router has to simultaneously solve these two problems using the limited routing resources.

In general, there are two types of routers. The first type consists of combined global-detailed routers [9,27,28,29,31] in which a complete routing path is determined in one step. The second type consists of two-step routers which first perform global routing [32] to determine which logic block pins and channel each net would use, and then perform detailed routing [33,34] to determine the track(s) each net would use within the specified channel. Since the result quality of detailed routing is highly constrained by the choices the global router makes, combined global-detailed routers often have better routing results. Therefore, throughout this study, we use combined global-detailed routing which is also supported by VPR.

By far the most common technique to this routing problem is Dijkstra's algorithm [35] which finds the shortest path between a net source pin and a net sink pin. However, this approach alone often results in an unroutable circuit, where contention for routing resources by nets is still unresolved. Rip-up and retry techniques [37] are often used to resolve competition for routing resources. There are also routers [29] using rip-up and retry approach to tackle the congestion as well as timing problems. One such algorithm is the Pathfinder algorithm [9]. This algorithm is employed by the VPR CAD tool.

The Pathfinder algorithm is shown in Figure 2.10. It uses a breadth-first search through the *routing resource graph*[1] to connect net terminals. The key idea of Pathfinder is to allow physical tracks to be shared by multiple nets during the routing process. Once all nets have been routed successfully, the cost of two nets sharing a track will be increased slightly. Each net is then ripped-up and re-routed. This is repeated for several iterations; each time the cost of sharing becomes slightly higher. When, at the end of an iteration, no track is shared by more than one net, a legal routing has been found, and the algorithm terminates.

During routing, the fitness of each potential segment n that might be added to the net is evaluated using the following cost function:

$$\text{Cost}(n) = \text{Criticality} \times \text{Delay}(n) + [1 - \text{Criticality}] \times b(n) \times h(n) \times p(n) \tag{2.1}$$

where Delay(n) is the Elmore delay of the segment n, and b(n), h(n), and p(n) are the base cost, the historical congestion cost, and the present congestion cost of using segment n, and the Criticality is a measure of how close to the critical path the currently routed net is. The first term in the cost function represents the delay of the currently routed net, while the second term represents the congestion cost of the current segment. Nets with a higher value of Criticality are thus routed primarily for speed, while other nets are routed primarily for congestion. This ensures that, as routing progresses, nets which are not critical are moved away from congested regions.

---

[1] A directed graph, in which each node represents either a wiring segment or a logic block pin and each edge represents a programmable switch, is a very general way to describe an EPLC [5].

24

```
R(i) be the set of nodes, n, in the current routing of net(i).

Criticality(i, j) = 1 for all nets i and sinks j;

while (OverusedResource() == true) {
    for (each net, i) {
        rip-up routing tree R(i) and update affected p(n);
        R(i) = NetSource(i);

        for (each sink, j of net(i) in decreasing Criticality(i,j) order) {
            PriorityQueue = R(i) at PathCost(n) = Criticality(i,j) × Delay(n) for each node n in R(i);

            while (sink(i,j) not found) {
                remove lowest cost node, m, from PriorityQueue;

                for (all fanout nodes n of node m) {
                    add n to PriorityQueue at Pathcost(n) =  Cost(n) + PathCost(m);
                }
            }

            for (all nodes, n, in path from R(i) to sink(i,j)) {    /* Backtracing */
                update p(n);
                add n to R(i);
            }
        }
    }

    Update h(n) for all nodes n;
    perform timing analysis and update Criticality(i,j) for all nets i and sinks j;
}   /* End of a routing iteration */
```

Figure 2.10 Pseudo-code of the Pathfinder routing algorithm

2.3   Focus and Contributions of This Thesis

In order to use a non-rectangular core, placement and routing tools that map a user

circuit onto an EPLC are required. Such place and route tools have been well-studied

for stand-alone FPGAs; however, as described in Section 2.1, EPLC's may differ in that

they may not be square or rectangular. We need new place and route tools that enable

25

users to efficiently specify a non-rectangular EPLC and generate the correct architecture for evaluations. In Chapter 3, we propose a new specification method and incorporate it into the public-domain VPR place and route tool that is representative of industry tools developed by FPGA vendors.

Since place and route tools use algorithms that are inherently geometrical, it seems likely that the same algorithms may not perform well with non-rectangular EPLC's. In Chapter 4, we propose enhancements to existing placement and routing algorithms in VPR that allow the algorithms to better target the non-rectangular cores. In Chapter 5, we investigate the area and delay performances of "L"-, "U"- and "O"-shaped EPLC's in various relative aspect ratios.

The contributions of this thesis are summarized as follows:

1. A novel specification method for describing non-rectangular EPLC's

2. Enhancements to existing placement and routing algorithms that optimize for non-rectangular EPLC's

3. Experimental evaluation of "L"-, "U"- and "O"-shaped EPLC's for the area and delay efficiencies

Chapter 3

A SPECIFICATION METHOD FOR NON-RECTANGULAR EPLC'S

In this chapter, we present a novel method of specifying a non-rectangular EPLC. This specification is used to provide device information to a CAD tool, as shown in Figure 3.1 [38]. This specification method we have developed is based on that used by the VPR place and route tool; the primary difference is that our specification method allows for the description of cores which are "L"-shaped, "U"-shaped, and "O"-shaped. This chapter first reviews the specification method used by VPR, and then presents the enhancements needed in order to specify these non-rectangular shapes.

3.1   Motivation

A simple, flexible method of specifying programmable logic architectures is important for two reasons:

1. Computer-Aided Design tools that map user circuits to the programmable logic devices or cores must be provided with information regarding the targeted architecture. Vendors of stand-alone FPGA devices typically have a single CAD tool that maps circuits to an entire family of devices (or several families). These CAD tools read device files to obtain information regarding the

target architectures; the FPGA vendor provides a separate device file for each member of the FPGA family. This concept of device files allows a single CAD tool to target several devices. In addition, as FPGA vendors introduce new parts, they only need produce new device files; the CAD algorithms themselves need not change.

2. In the EPLC architecture experiments presented later in this thesis, we experimentally map user circuits to a large number of potential EPLC architectures. In order to do this, a single CAD tool must be able to support a wide range of architectures; as in the commercial tools described above, the CAD tool needs to be able to read in device information describing the architecture to be targeted.

For both of these reasons, a simple way to specify the characteristics of an EPLC is required. The place and route tool VPR [6] provides a clean and simple way to specify such architectures. However, since VPR was originally written to target stand-alone FPGAs, it only supports architectures which are square or rectangular. Thus, this chapter will describe how we have extended the VPR specification method to allow for the specification of non-rectangular cores.

3.2 Original Specification Scheme

As described above, our specification method is based on the existing method employed by the VPR CAD tool [6]. In this section, we describe this existing method; in the next, we describe our enhancements.

```
# 4 I/O pads per row or column
io_rat 4

#############################
### Relative Channel Width ###
#############################
# Define uniform X/Y channel ratio for each region
chan_width_io 1.0
chan_width_x uniform 1.0
chan_width_y uniform 1.0

##################
### Logic Block ###
##################
# Class 0 is LUT inputs, class 1 is the output, class 2 is the clock.
# 10 logic inputs.
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
outpin class: 1 top
outpin class: 1 right
outpin class: 1 bottom
outpin class: 1 left
inpin class: 2 global top        # Clock, assume routed on global resource.

# Cluster of 4 4-LUTs
subblocks_per_clb 4
subblock_lut_size 4

###############################
### Detailed Routing Architecture ###
###############################
switch_block_type wilton
Fc_type fractional
Fc_output .25
```

```
Fc_input .25
Fc_pad 1

# Metal 3, min W (0.28 um), min space (0.28 um)
# The following values are assuming a tile length of 116.00 um

segment frequency: 1.0 length: 1 wire_switch: 2 opin_switch: 2 Frac_cb: 1 \
Frac_sb: 1 Rmetal:  32.360 Cmetal:  3.946e-14

switch 0 buffered: no  R: 456.500  Cin: 3.7500e-15 Cout: 3.7500e-15 Tdel: 0

# switch_1 width is approximatley  10.0 times minimum width
switch 1 buffered: yes R: 913.000  Cin: 1.6200e-15 Cout: 3.7500e-15 Tdel: 4.2600e-11

# switch_2 width is approximatley   5.0 times minimum width
switch 2 buffered: yes R: 1826.000  Cin: 1.6200e-15 Cout: 1.8750e-15 Tdel: 4.0700e-11

# 0.7 um width. Multiplied by 0.693 to agree with Rswitch values.

R_minW_nmos 4565
R_minW_pmos 8674   # 1.9x R of an nmos


######################
### Timing Analysis ###
######################
C_ipin_cblock 1.62e-15
T_ipin_cblock 3.7700e-10
T_ipad 242e-12    #Clk_to_Q + 2:1 mux
T_opad 4.5e-11
T_sblk_opin_to_sblk_ipin 3.0100e-10
T_clb_ipin_to_sblk_ipin 3.0100e-10
T_sblk_opin_to_clb_opin 0

T_subblock T_comb: 4.01e-10  T_seq_in: 2.95e-10  T_seq_out: 2.42e-10
T_subblock T_comb: 4.01e-10  T_seq_in: 2.95e-10  T_seq_out: 2.42e-10
T_subblock T_comb: 4.01e-10  T_seq_in: 2.95e-10  T_seq_out: 2.42e-10
T_subblock T_comb: 4.01e-10  T_seq_in: 2.95e-10  T_seq_out: 2.42e-10
```

Figure 3.1 Example of an original VPR-format architecture file

Architecture specifications can be supplied to VPR through the use of an *architecture file*.

Figure 3.1 shows an example architecture file. With the exception of the parameter

io_rat (which represents the number of I/O pads that fit into one row or one column

of the FPGA), all the architecture parameters belong to one of the following four

groups:

30

1. Relative Channel Widths,

2. Logic Block,

3. Detailed Routing Architecture,

4. Timing Analysis.

The group of parameters relating to the relative channel widths has three lines. The first sets the ratio of the width of the channels between the I/O pads and the widest core. The other two specify the distribution of tracks for the vertical and horizontal channels respectively. Separating the vertical and horizontal channel distributions allows the user to specify directionally biased routing architecture [2,50,51], or non-uniform routing architectures [40].

There are four parameters for the logic block description group. One specifies the number of inputs to each lookup-table. Another sets the number of lookup-tables in each logic block. These two parameters have been studied extensively for their optimal values [4,12]. The remaining two parameters define the location (which side of the logic block) and the type (local/global resource) of each logic block's input and output pins. Betz [5] showed that pin placement had different impacts on the results of non-biased and biased routing architectures.

The parameters in the detailed routing architecture group are required if and only if combined global and detailed routing is to be performed. The results from combined global and detailed routing are more accurate and complete than those from global

routing because there are nine more parameters being considered in the CAD tool. The nine parameters contain the following information: switch block topology; Fc values for logic block input pin, output pin or I/O pad pin; segmentation distribution; switch block internal population; connection block internal population; switch type used to connect each routing wire to other wires; switch type definition; metal width and spacing of the various routing wires [38]. All this information is important to determining the connectivity and the electrical properties of the FPGA routing architecture.

The last group of parameters involves timing analysis. There are eight parameters that describe the different parts of the delay of a signal coming from a routing track, through both the logic block's input pin and the logic block itself, and to the logic block's output pins. These parameters are required only if timing analysis is to be performed on the placed and routed circuit. This information can be used to find a good approximation for calculating the critical path delay of the routed circuit.

Note that these parameters do not *completely* describe an FPGA architecture. As an example, although the length of each wiring segment can be specified in the detailed routing architecture group, the actual start and end points of each wire in each channel cannot be specified. The VPR program contains algorithms which read in an architecture file, and create a model of an FPGA using the specified parameters, making intelligent decisions about the details not specified in the architecture file. This frees the designer from specifying every detail, meaning a wider range of architectural parameters can be considered during experimentation. The "construction" of an

32

FPGA using the architecture file parameters is an interesting problem in itself, and has been well studied in [39].

The constructed architecture is represented within the CAD tool as a routing resource graph. This is a generic graph data structure in which each node is a routing track or logic block pin, and each edge is a programmable connection between logic block pins. The CAD tools then use this routing resource graph during placement and routing.

3.3   Enhanced Architecture Specification Method

From the architecture file specifications, there is no attention directed to the overall shape of the FPGA architecture. In the command-line of VPR, users can specify the aspect ratio of the FPGA so that a rectangular or square FPGA can be formed. However, we need shape-specific parameters in order to describe a non-rectangular EPLC. Therefore, we propose an extension to the specification format, which consists of three new parameters as follows:

Shape of an EPLC

1. **size** {*fixed* | *aspect_ratio*}

2. **region** <id> bottom_left: <x_coord> <y_coord> top_right: <x_coord> <y_coord>

3. **cregion** bottom_left: <x_coord> <y_coord> top_right: <x_coord> <y_coord> top: <region_id> bottom: <region_id> left: <region_id> right: <region_id>

The first parameter, **size,** has only two valid values. One is *fixed,* which means the specifications of the parameters **region** and **cregion** are treated as fixed dimensions of the EPLC. This feature is very useful for evaluating the performance of a fixed-size EPLC. The second valid value is *aspect_ratio,* replacing the original VPR's command-line option aspect_ratio. If *aspect_ratio* is assigned to parameter **size,** the specifications of the parameters **region** and **cregion** are treated as the aspect ratio of an EPLC. In other words, the specified shape of the core will be kept unchanged, but not its size. This is very helpful for running benchmark experiments when each benchmark circuit is of a different size.

The second parameter, **region,** specifies a square or rectangular area that can be defined as part or all of an EPLC. A region is defined by its integer identifier, and then by the x and y coordinates at both bottom left and top right corners of the region. The coordinate (0,0) starts at the bottom left corner of this coordinate system. As a few rectangular or square regions are defined, a non-rectangular EPLC can be formed. In Figure 3.2, two regular regions and one connection region form an "L"-shaped EPLC.

The third parameter, **cregion,** is called the connection region. We posit the restriction that every two regular regions must be connected through the connection region. The only difference between a connection region and a regular region is that the widths of all horizontal and vertical channels of a connection region depend totally on the channel widths of the two neighboring regular regions, respectively. This is depicted in Figure 3.2. This parameter helps distinguish between a connection region and a regular

region, because of the extra constraint the channel widths of a connection region put on a directionally biased routing architecture. A connection region is defined by the x and y coordinates at both bottom left and top right corners of the connection region. For each side of a connection region, a region identifier must be entered to show which regular region it makes contact with. If no regular region has contact with that side of a connection region, -1 should be entered. For this parameter, we give a limit of only two orthogonal regular regions which can make contact with one connection region.
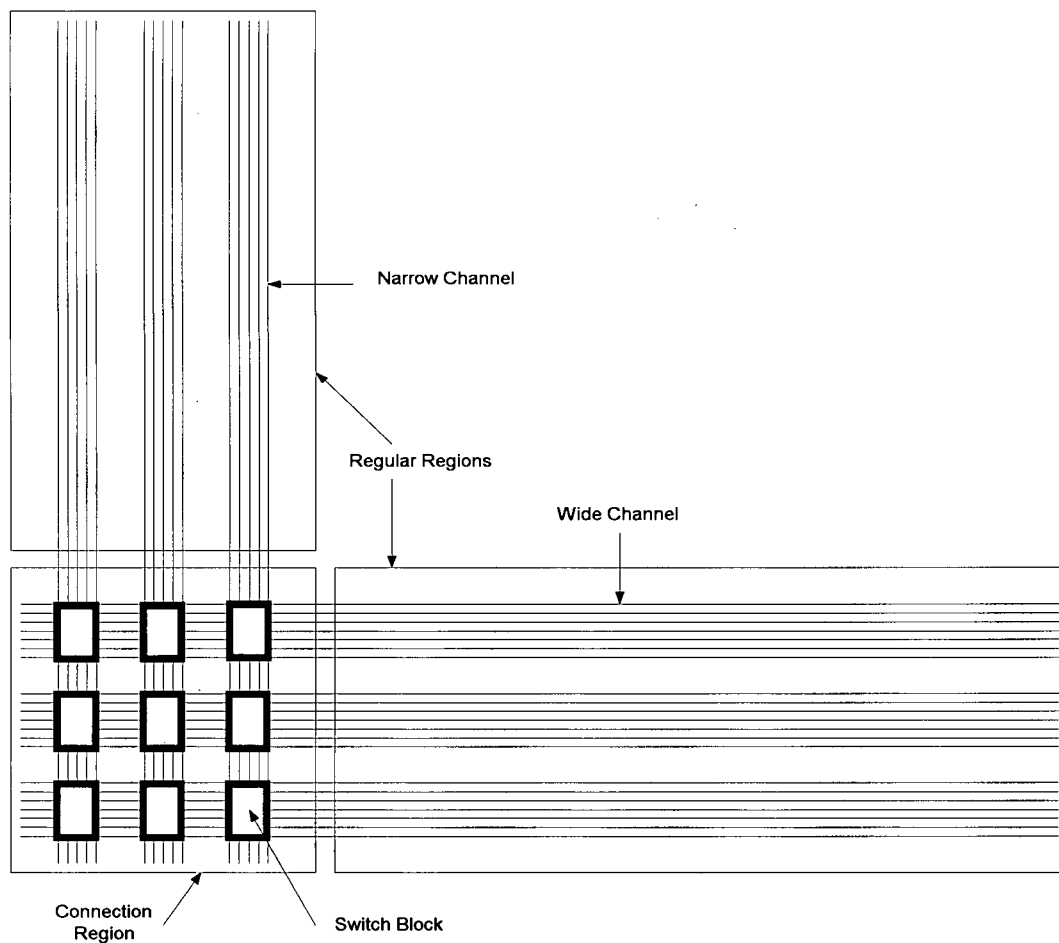
Figure 3.2 An "L"-shaped directionally biased routing architecture

The VPR place and route tool was enhanced to support this extended specification description. This involved changes to both the architecture file parser and the code that generates the routing resource graph from the architecture description. Our current implementation is limited to segment length of 1; although most commercial EPLC architectures typically have longer segments, we can still use this tool to gather information about the effectiveness of various placement and routing algorithms that target the non-rectangular cores.

3.4   Examples of Non-Rectangular EPLC's

In this section, we show examples of how to describe "L"-shaped, "U"-shaped and "O"-shaped EPLC's in the architecture files, and display routing results using the enhanced VPR. Figures 3.3, 3.4 and 3.5 show only the lines of Shape group on the architecture files for an "L"-shaped core, a "U"-shaped core and an "O"-shaped core respectively. The rest of the lines on their architecture files are the same as those depicted in Figure 3.1, because all of them have the same settings for their relative channel widths, logic block, detailed routing architecture, and timing. Moreover, they all choose *aspect_ratio* for parameter **size**, meaning the enhanced VPR must find the minimum possible size for implementing a circuit, while the shape and the aspect ratio of a core remain as specified. In Figure 3.3, two rectangular regions and one square connection region are used to form an "L"-shaped core. In Figure 3.4, three rectangular regions and two square connection regions are used to form a "U"-shaped core. In Figure 3.5, four rectangular regions and four square connection regions create an "O"-shaped core.

```
# Size of the EPLC
size aspect_ratio

# Define region(s) by assigning bottom left and top right (x y) coordinates.

region 0 bottom_left: 1 0 top_right: 3 1
region 1 bottom_left: 0 1 top_right: 1 3

# Define connection region(s) by assigning bottom left and top right
# (x y) coordinates and also specifying the region index where
# the side of the cregion contacts with.

cregion bottom_left: 0 0 top_right: 1 1 top: 1 bottom: -1 left: -1 right: 0
```

Figure 3.3 Only Shape group is shown in the architecture file for an "L"-shaped core.

```
# Size of the EPLC
size aspect_ratio

# Define region(s) by assigning bottom left and top right
# (x y) coordinates.

region 0 bottom_left: 0 2 top_right: 1 6
region 1 bottom_left: 1 0 top_right: 5 2
region 2 bottom_left: 5 2 top_right: 6 6

# Define connection region(s) by assigning bottom left and top right
# (x y) coordinates and also specifying the region index where
# the side of the cregion contacts with.

cregion bottom_left: 0 0 top_right: 1 2 top:  0 bottom: -1 left: -1 right:  1
cregion bottom_left: 5 0 top_right: 6 2 top:  2 bottom: -1 left:  1 right: -1
```

Figure 3.4 Only Shape group is shown in the architecture file for a "U"-shaped core

```
# Size of the EPLC
size aspect_ratio

# Define region(s) by assigning bottom left and top right
# (x y) coordinates.

region 0 bottom_left: 0 1 top_right: 1 5
region 1 bottom_left: 1 0 top_right: 5 1
region 2 bottom_left: 5 1 top_right: 6 5
region 3 bottom_left: 1 5 top_right: 5 6

# Define connection region(s) by assigning bottom left and top right
# (x y) coordinates and also specifying the region index where
# the side of the cregion contacts with.

cregion bottom_left: 0 0 top_right: 1 1 top:  0 bottom: -1 left: -1 right:  1
cregion bottom_left: 5 0 top_right: 6 1 top:  2 bottom: -1 left:  1 right: -1
cregion bottom_left: 5 5 top_right: 6 6 top: -1 bottom:  2 left:  3 right: -1
cregion bottom_left: 0 5 top_right: 1 6 top: -1 bottom:  0 left: -1 right:  3
```

Figure 3.5 Only Shape group is shown in the architecture file for an "O"-shaped core.

An MCNC[2] benchmark circuit named C6288 is used to demonstrate that the enhanced VPR can place and route a circuit on these core architectures as specified by the architecture files in Figures 3.3, 3.4 and 3.5. Figures 3.6, 3.7 and 3.8 display the pictures of the final routing results for "L"-shaped, "U"-shaped and "O"-shaped cores. In Figure 3.6, the minimum channel width that can route the circuit for an "L"-shaped core is 20 tracks, whereas the minimum channel width for "U"-shaped and "O"-shaped cores is much larger, 28 and 24, as depicted in Figures 3.7 and 3.8. An in-depth study of the architecture of non-rectangular cores is presented in Chapter 5.

---

[2] Microelectronics Corporation of North Carolina
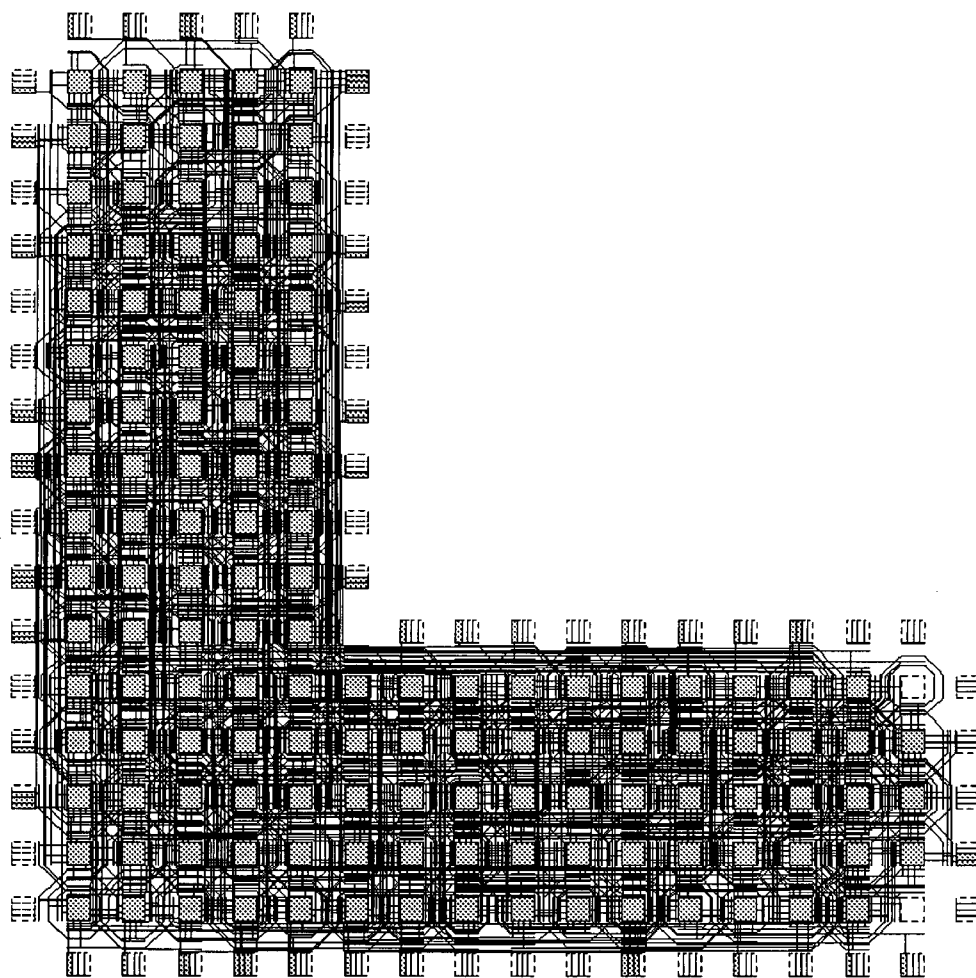
Routing succeeded with a channel width factor of 20.

Figure 3.6 Final routing result of an "L"-shaped EPLC

Routing succeeded with a channel width factor of **28**.
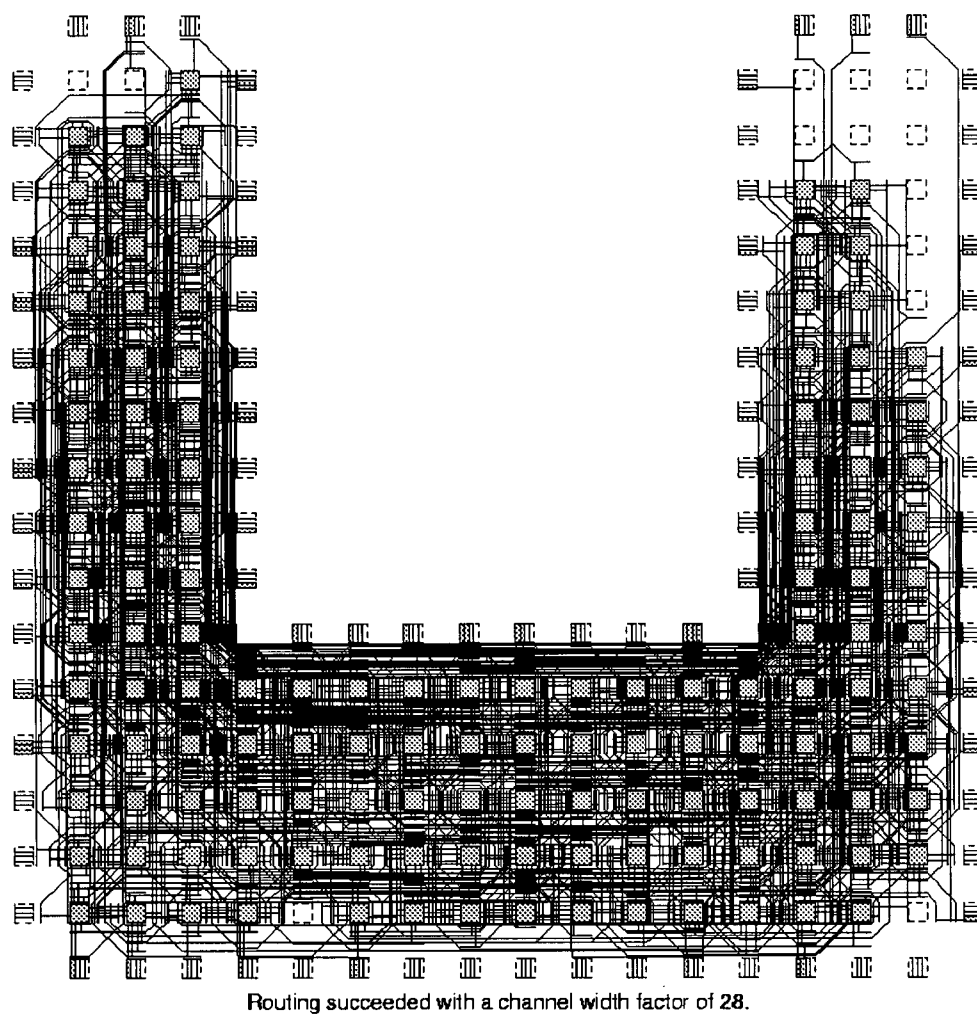
Figure 3.7 Final routing result of a "U"-shaped EPLC

40
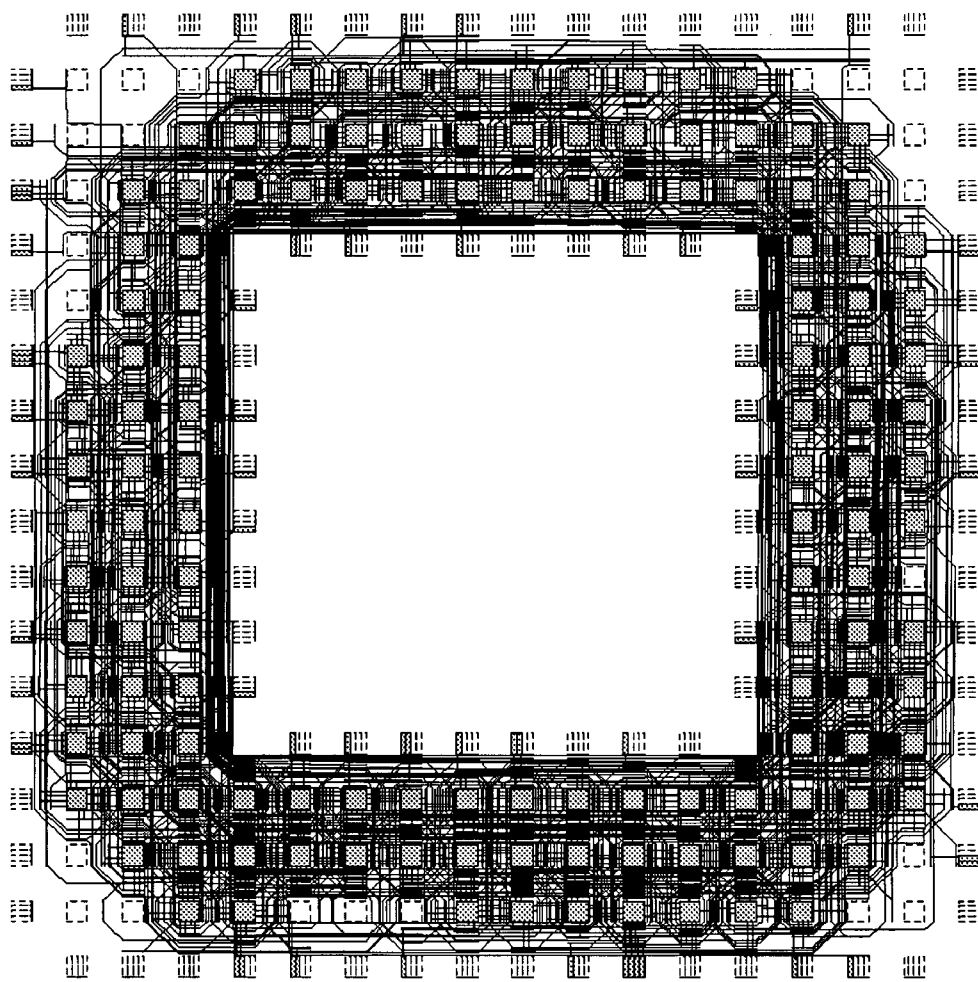
Routing succeeded with a channel width factor of 24.

Figure 3.8 Final routing result of an "O"-shaped EPLC

41

3.5   Summary

An efficient yet simple specification method is crucial to the design of EPLC architectures because it allows researchers to experiment with their core designs by running many benchmark circuits on the CAD tool. However, the current FPGA CAD tools do not support one of the important design criteria of an EPLC in a SoC design, which is to allow non-rectangular cores. In this chapter, we have presented a new specification method that describes a non-rectangular core on an architecture file used by the FPGA CAD tool VPR. Furthermore, we have briefly explained how to implement these new specifications on the enhanced VPR. Examples of "L"-shaped, "U"-shaped and "O"-shaped EPLC's are pictured, taken from routing results generated by the enhanced VPR.

Chapter 4

PLACEMENT AND ROUTING ALGORITHMS

The density and the speed of a programmable logic core depend not only on its architecture, but also on how well a CAD tool maps a circuit into the programmable device. In the previous chapter, we introduced the new CAD tool, built on the existing VPR program, which can place and route a circuit on non-rectangular programmable core architectures using existing placement and routing algorithms in VPR. However, these placement and routing algorithms are not optimized for non-rectangular core architectures. It is not clear how efficiently both the placement and routing algorithms in VPR map a circuit into a non-rectangular core. Therefore, an examination of these placement and routing algorithms is required.

In this chapter, we focus on the algorithmic issues of non-rectangular programmable logic core architectures. In Section 4.1, we first describe the existing placement and routing algorithms in VPR that targets stand-alone FPGA's. Section 4.2 then shows how these algorithms can be modified to better support "L"-, "U"-, and "O"-shaped EPLC's. In Section 4.3, we will experimentally investigate how the enhanced algorithms perform compared to the original algorithms. Finally, a summary is presented in Section 4.4.

4.1 Placement and Routing for Stand-Alone FPGA's

We have based our algorithms on the existing VPR place and route tool [6]. VPR is representative of industrial tools developed by FPGA vendors. The following subsections describe the relevant details of the algorithms; a more complete description is in [3,5]. Section 4.2 will show how these algorithms were enhanced to better target non-rectangular cores.

4.1.1 Placement

The main task of placement is to assign user logic blocks to physical logic block locations in a programmable logic core. In this study, we refer to the timing-driven placement algorithm used in VPR as T-VPlace. T-VPlace takes the netlist of a user circuit and intelligently maps it onto the available sites (logic blocks or I/O pads) in a programmable logic core. T-VPlace is a simulated annealing based algorithm [26] as described in Section 2.2.1. For more details of simulated annealing algorithm, please refer to Section 2.2.1.

T-VPlace tries to minimize two cost functions:

1. Wiring Cost,

2. Timing Cost.

Wiring cost is the estimated amount of interconnect needed to route a circuit with the current placement. The total wiring cost of a placement is the summation of every net's bounding box half perimeter. Figure 4.1 depicts a bounding box half perimeter for an eight terminal net. If a net has more than three terminals, a factor q will be

44

multiplied to the net's half of bounding box half perimeter to compensate for wire-length underestimation [42]. In general, the lower the total wiring cost is, the higher the area density of a circuit will become.
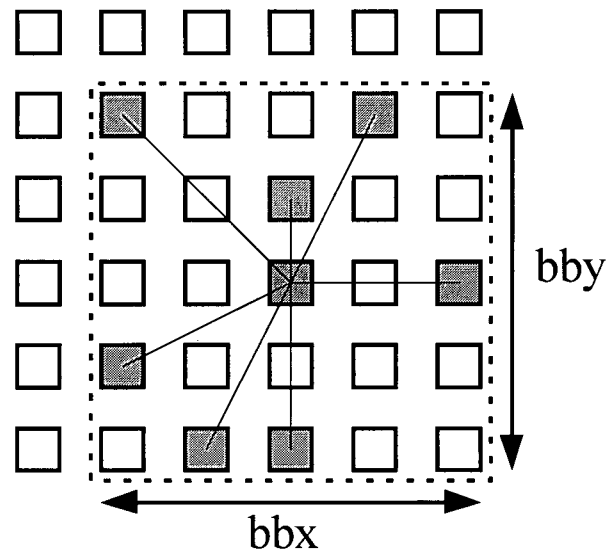
Figure 4.1 Example of half of a bounding box perimeter (bbx + bby) of an eight terminal net

Timing on the critical path is another important issue that a placer should consider. In T-VPlace, the timing cost of a net is defined as follows:

$$\text{Timing Cost} = \text{Delay} \times \text{Criticality}^{\text{Criticality\_Exponent}} \tag{4.1}$$

The delay of a connection clearly depends on the placement; however, calculation of this delay during placement is difficult. T-VPlace uses a pre-computed matrix that contains the delay of each potential connection (the shortest route of each connection is performed, and Elmore delay is used to estimate the delay). Note that it is not necessary to compute the delay between every pair of logic blocks; in most standalone

FPGA's, the delay between location (x,y) and (x+Δx, y+Δy) can be approximated as being independent of the values of x and y. This is shown Figure 4.2; the delay of a net connecting a and b will be approximately the same as the delay of a net connecting c and d. Thus, a single array indexed on the span of each wire is enough. For each potential placement, the x and y span of each wire can be found, and the matrix can be used to quickly find the delay of each connection. For more details, see [3].
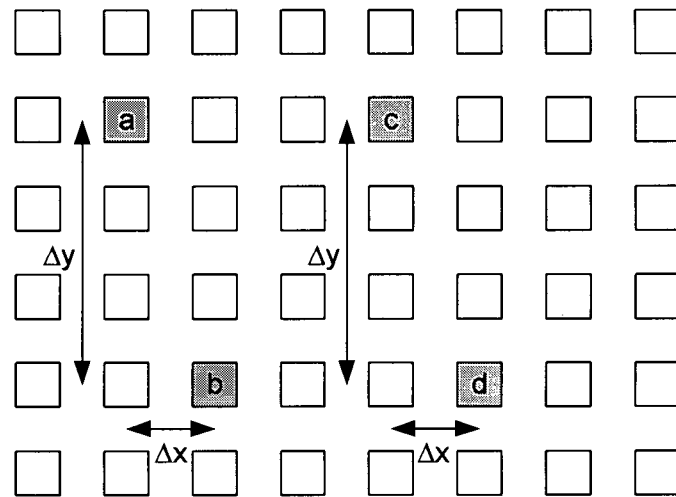
Figure 4.2 Net connecting a to b will have approximately the same delay as net connecting c to d

In order to focus on minimizing the delay on the critical path and let the delay of the non-critical paths to be increased, the terms Criticality and Criticality_Exponent are introduced to give more weight on the connections that are critical and less weight on the connections that are non-critical. Criticality is defined as follows:

$$Criticality = 1 - Slack/Max\_Delay \tag{4.2}$$

where Max_Delay is the critical path delay and Slack is the amount of delay that can be added to a connection without increasing the critical path delay. By default,

Criticality_Exponent starts as 1 and then is gradually increased the value to 8, a schedule recommended in [3]. The total timing cost of a placement is the sum of every net's timing cost.

4.1.2 Routing

Routing is the process that determines the connections among all the logic block input pins and output pins required by the user circuit. It is the final step of the EPLC CAD flow depicted in Figure 2.8. For a timing-driven router, there are two goals to achieve:

1. Make a circuit routable on a given programmable logic core,

2. Make the critical path fast.

Since high circuit speeds are very important, timing-driven routing is much more desirable than the purely routability-driven routing that focuses on the first goal only [30]. In VPR, the routing algorithm is timing-aware so it takes care of both routability and circuit speed at the same time.

In this study, we refer to the timing-driven routing algorithm used within VPR as T-VRoute. T-VRoute takes two input files: the netlist of a circuit to be mapped, and a placement file generated by a placement tool. The goal of T-VRoute is to successfully route all the connections required between the logic block input and output pins where their locations are specified by the placement file, and optimize the speed of the circuit at the same time. Since routing is a NP-complete problem, no optimal solution is guaranteed by any routing algorithm. However, a heuristic solution can be obtained.

47

T-VRoute performs a number of routing iterations. Each routing iteration comprises the following two steps:

1. Timing analysis,

2. Rip-up and reroute of each net.

In order to optimize the delay of a circuit, timing analysis is required. There are two parts in the process of timing analysis. The first part involves computing the delay of all of the paths in a circuit. T-VRoute models pass transistors as linear resistors, wires as an RC pi-network, and buffers as resistors plus constant delay. Therefore, a net's routing can be modeled as an RC-tree [21]. Then the Elmore delay model is employed to estimate the delay of the RC-tree; this gives high fidelity of delay estimation of all of the paths in a circuit [5]. The second part is to calculate the amount of delay that may be added to each connection before it becomes critical, which is called the slack of that connection. The value of slack will then be used to guide T-VRoute to preferentially route delay-critical connections to make these connections as fast as possible.

The second step involves ripping up the previously routed net and rerouting it. The purpose of this step is to resolve competition for routing resources, and at the same time improve circuit speed. T-VRoute achieves this goal by employing the Pathfinder negotiated congestion-delay algorithm [9]. The Pathfinder algorithm is a modified maze router allowing overuse of routing resources and utilizing a cost function to resolve the contention for routing resources gradually, and also directly optimize the

48

timing of a circuit. As a result, this algorithm produces good quality routed circuits. Further information about Pathfinder algorithm is in Section 2.2.2.

When a routing iteration is finished for all nets (but some nets could still have contention for routing resources), the process of timing analysis will be repeated to update the delay of the newly routed nets, and then the ripping-up and rerouting will be repeated. This continues until all congestion is resolved. By default, T-VRoute tries 30 iterations to route a circuit. If T-VRoute fails, it will stop and deem the circuit unroutable.

Since standard maze routing can be a very slow process for a large FPGA, making it run fast is also an important goal. T-VRoute uses two methods to speed the execution:

1. Directed search,

2. Net bounding box.

Directed search is based on the knowledge of the expected remaining distance or cost from the current position to the destination in order to speed the search process. To take this into account, the cost function in Equation 2.1 is modified into:

$$\text{Total Cost(n)} = \text{Cost(n)} + \text{Expected Cost(n)}, \tag{4.3}$$

where the Expected Cost term is computed using the Manhattan distance between the current routing segment and the sink of the connection. Figure 4.3 shows this graphically; the Expected Cost term for the segment would be x+y. In this way, most

49

connections are found very quickly; the algorithm reverts to a standard maze router if there is significant congestion.
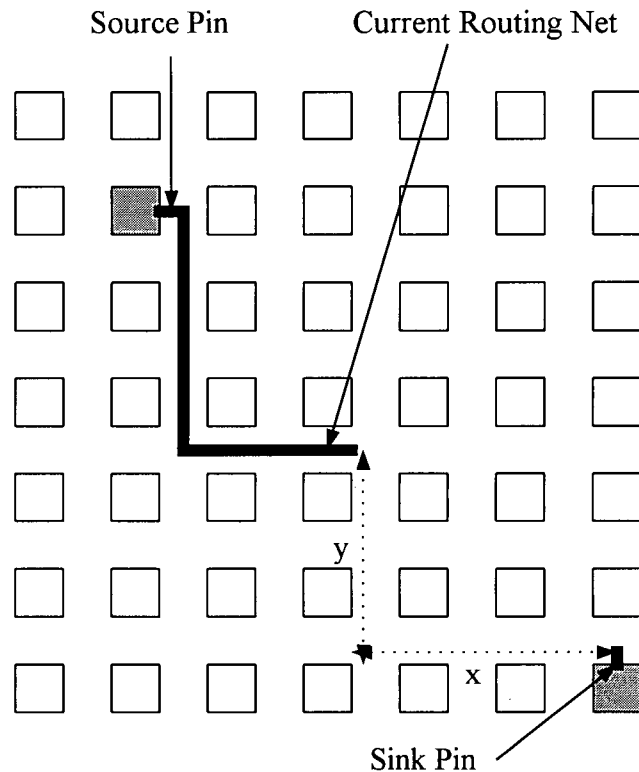
Figure 4.3 Calculating the expected cost of a net during routing

The second method to speedup the search process is to prevent the route from expanding routing resources that lie more than a preset distance outside the net's bounding box. By default, T-VRoute only considers routing resources which are either within a net's bounding box or no more than 2 tiles away from the boundary box as shown in Figure 4.4. This net bounding box search helps reduce CPU time with a small degradation on the quality of routing [5].
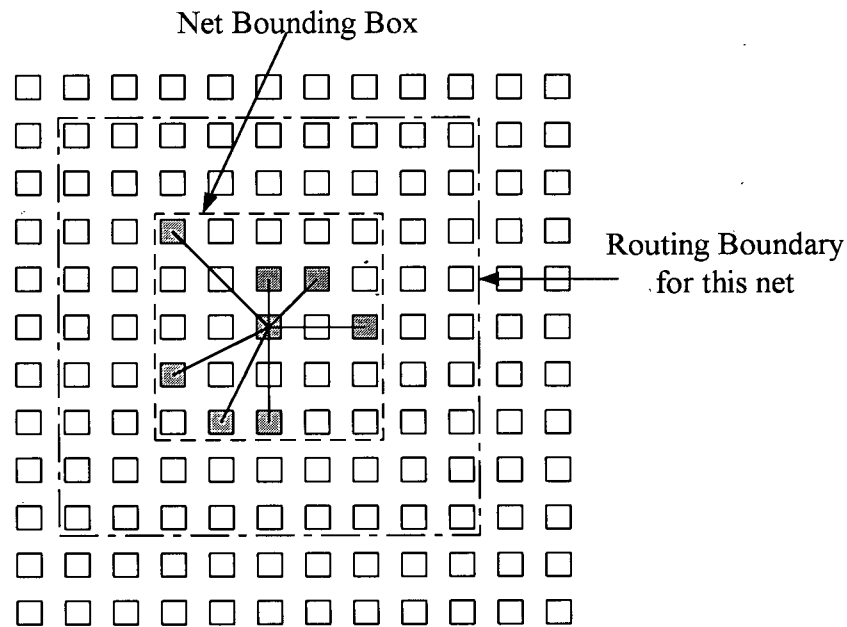
Figure 4.4 Example of routing boundary for an eight terminal net

4.2 Placement and Routing for Non-Rectangular EPLC's

The VPR placement and routing algorithms described in Section 4.1 are both geometry-based. In the following two subsections, we will show why these algorithms are not suitable for non-rectangular cores, and will show enhancements to better support these cores.

4.2.1 Placement

The current placement tool does not produce good solutions on "O"- and "U"-shaped cores. Figure 4.5 shows the problem. As described in Section 4.1.1, the delay between a pair of logic blocks is found using a pre-computed delay lookup table, indexed by the x and y span of the net. As shown in Figure 4.5, the Manhattan distance between two blocks may not correctly represent the shortest path distance between the two nodes in "U"- and "O"-shaped cores. Therefore, the delay value stored in the pre-computed

51

delay lookup table gives a very poor estimate of the actual delay of this connection. This poor delay estimation also affects the Criticality of this connection. If the connection shown in Figure 4.5 is on or near the critical path, the placer would not pay as much attention as it should to minimizing the delay of this connection and eventually could result in a slower circuit.
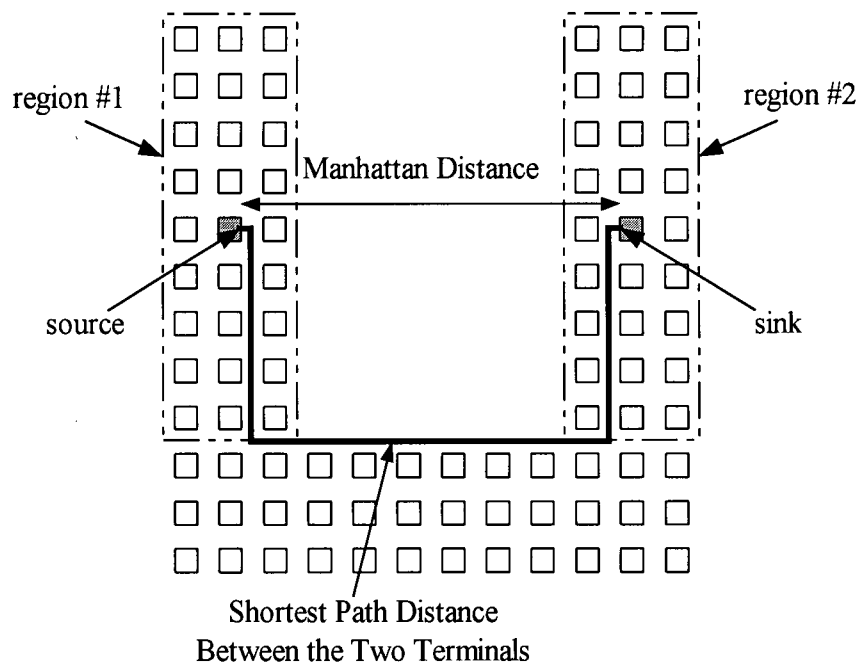
Figure 4.5 Difference between the Manhattan distance and the shortest path distance in a "U"-shaped core

In a "U"- or an "O"-shaped core, the same delay table cannot be used for each block; different (x,y) locations should have different delay tables. One possible solution is to compute a separate delay table for each block location (x,y). However, this is very non-scalable, and could make the placer run very slow and require too much memory. Our solution is to compute separate delay tables for all blocks in Regions 1 and 2 in Figure

4.5; this delay table is used for all nets that span the two regions and the original delay table is used for all other nets. A similar technique can be used for "O"-shaped cores. The pseudo-code is depicted in Figure 4.6.

```
D = Estimated delay between the source at (x1, y1) and the sink at (x2, y2).

if (CoreShape = "U" || CoreShape = "O") {        /* U-shaped or O-shaped core*/
    if (OppositeRegions(x1, y1, x2, y2)) {       /* source and sink at the two opposite */
                                                 /* regions */
        D = Source_to_Sink_DelayTable(x1, y1, x2, y2);
    }
    else {                                       /* both source and sink are not at the */
                                                 /* two opposite regions */
        delta_x = abs(x2 − x1);
        delta_y = abs(y2 − y1);
        D = OriginalDelayTable(delta_x, delta_y);   /* then use original delay table */
    }
}
else {                                           /* L-shaped or rectangular or square core */
    delta_x = abs(x2 − x1);
    delta_y = abs(y2 − y1);
    D = OriginalDelayTable(delta_x, delta_y);    /* use original delay table */
}
```

Figure 4.6 Pseudo-Code of the correct delay estimation for "U"- and "O"-shaped cores

4.2.2   Routing

The current routing algorithm does not produce good solutions on "O"- and "U"-shaped cores. Figure 4.7 shows the first problem. As described in Section 4.1.2, the router finds a bounding box around each net, expands it by two logic blocks in each direction; the router then does not explore routes outside this bounding box. As shown in Figure 4.7, this can cause a problem in "U"- and "O"-shaped cores. In the figure, the net will not be successfully routed, because all potential routes must pass

outside the bounding box. The solution to this problem is straightforward; we simply remove the bounding box constraint, and allow the router to explore the entire graph. Although this slows down the algorithm somewhat, experiments have shown later in Section 4.3.2 that the impact on run-time is very small.
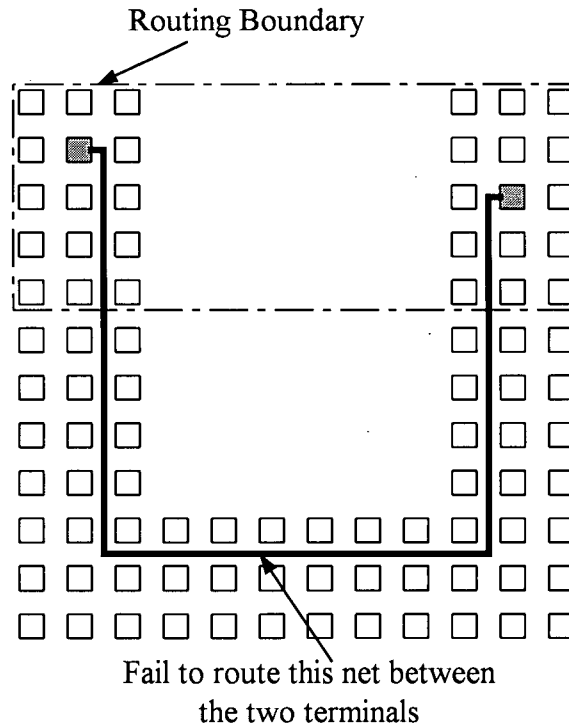
Figure 4.7 Failure of routing in a "U"-shaped core due to the net bounding box

The second problem is that the expected cost term in the direct search may be inaccurate. As described in Subsection 4.1.2, the expected cost is computed as the Manhattan distance between the current routing segment and the sink of the connection. Figure 4.5 shows an example where this estimation of the expected cost is incorrect. In this case, the preferred route is to leave the source in a downward direction. However, using the current cost function, the upward direction appears

54

equally attractive. Although the correct route will still be found eventually, the portion of the routing fabric that will be explored is large, especially considering that the bounding box constraint has been removed, as described above. This leads to long run times.

Our solution is to explicitly add terms to better estimate the distance from the current routing segment to the sink in "U"- and "O"-shaped cores. This is shown in Figure 4.8.

```
D = Estimated distance between the current location (x1, y1) and the destination location
(x2,y2).
H = Horizontal part of the estimated distance D.
V = Vertical part of the estimated distance D.

y1a is the y-coordinate where the region containing (x1, y1) touches the connection region
above it. The same applies for y2a.
y1b is the y-coordinate where the region containing (x1, y1) touches the connection region
below it. The same applies for y2b.


if (CoreShape = "U") {                       /* U-shaped core */
    V = abs (y1 - y1b) + abs (y2 - y2b);     /* use lower path only */
}
else if (CoreShape = "O") {                  /* O-shaped core */
    V1 = abs (y1 - y1a) + abs (y2 - y2a);    /* use upper path */
    V2 = abs (y1 - y1b) + abs (y2 - y2b);    /* use lower path */
    V = min (V1, V2);                        /* choose the shortest path */
}
else {                                       /* L-shaped or rectangular or square core */
    V = abs (y1 - y2);                       /* then use Manhattan distance */
}

H = abs (x1 - x2);
D = H + V;
```

Figure 4.8 Pseudo-Code of the correct shortest path distance calculation for "U"- and "O"-shaped cores

Note that none of these enhancements to the placement and routing algorithms are needed for "L"-shaped cores. Circuits can be placed and routed on "L"-shaped cores using the same tools as for square and rectangular cores.

4.3   Algorithm Evaluation

In this study, we use an empirical method to evaluate the proposed algorithmic enhancements, and compare the enhanced algorithms to VPR's existing algorithms. This involves technology-mapping, packing, placing and routing benchmark circuits into non-rectangular EPLC architectures. The area and delay results of each circuit implementation are then computed, and from this we are able to compare the performance of the enhanced algorithms to that of the original algorithms.

4.3.1   Experimental Methodology

To evaluate the proposed enhancements, we experimentally mapped sixteen large MCNC benchmark circuits onto a model EPLC. We assumed an island-style EPLC, where each logic block contains four 4-input lookup tables and four flip-flops. It was assumed that each fixed wiring track spans one logic block, and a Wilton switch block [8] is employed. We assumed a 0.18 µm CMOS process available from TSMC.
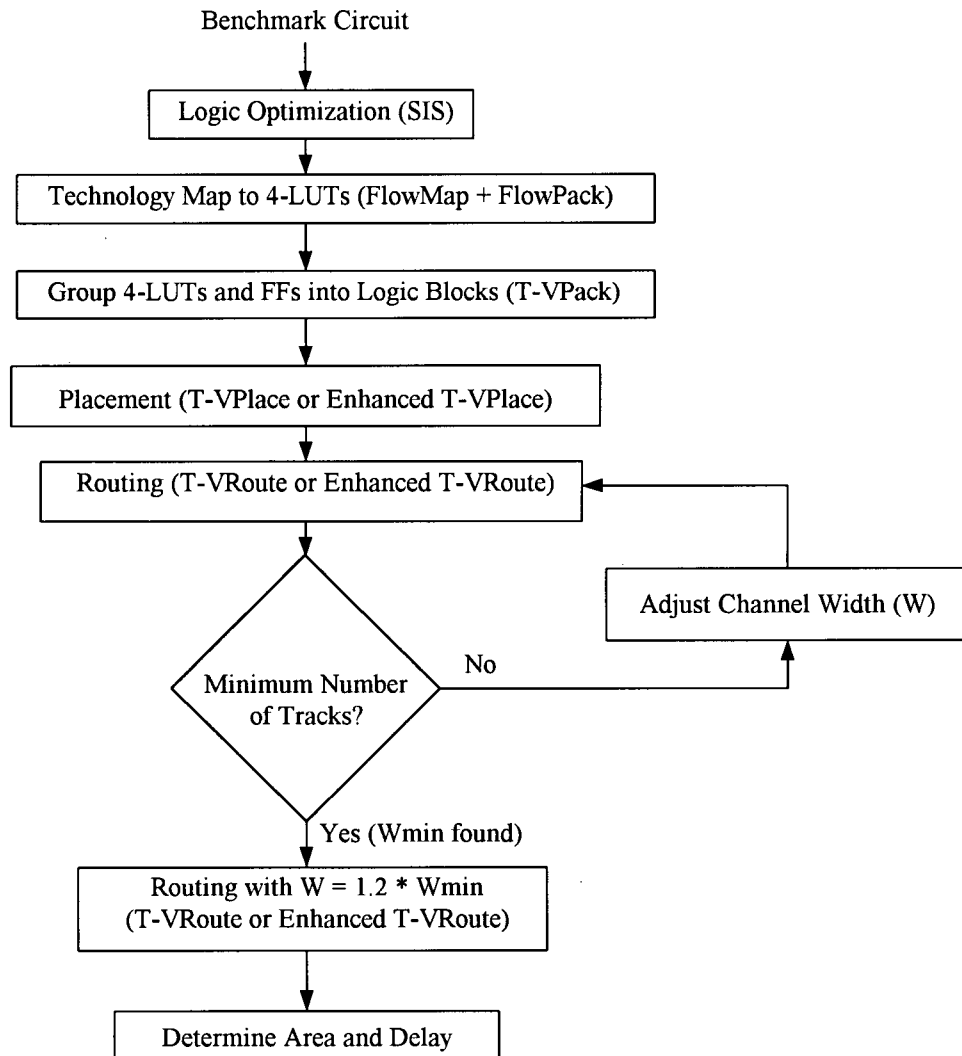
Figure 4.9 Algorithm evaluation CAD flow

The CAD flow that we use is depicted in Figure 4.9. First, each benchmark circuit is optimized by SIS [19] and technology mapped into 4-input lookup tables and flip-flops using Flowmap/Flowpack [10]. Then the timing-driven packing algorithm T-VPack [7] is employed to pack the lookup tables and flip-flops into logic blocks. The logic blocks were then placed and routed on an appropriated sized EPLC using both the original and enhanced algorithms. For each circuit, we sized the EPLC to be the smallest shape

that that meets the relative ratios show in Figure 4.10. Only "U"- and "O"-shaped core results are described in this subsection, since as described above, existing place and route tools work well with "L"-shaped cores. Routing was performed twice; the first route was used to find the minimum number of routing tracks needed for 100% routability. This number was then increased by 20%, and the routing repeated. This "low-stress" routing is representative of the routing performed in real industrial designs [5]. After that, we apply our delay model to estimate the delay of the circuit critical path, and our area model to estimate the total transistor area needed to lay out all the routing[3] in this core architecture. Finally, we use the area and delay results to compare the performance between the enhanced and original algorithms on "U"- and "O"-shaped cores. All the experiments were run on a 400MHz UltraSparc workstation.
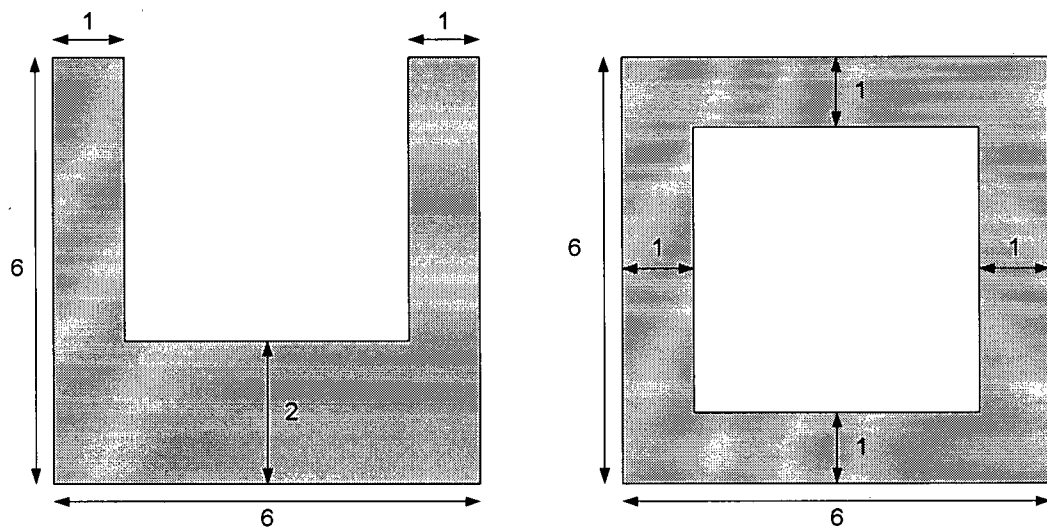
Figure 4.10 Relative aspect ratios of a "U"-shaped core and an "O"-shaped core used in algorithm evaluation

---

[3] All the results give only the routing area of the EPLC because the logic block is held constant throughout all the experiments. To allow averaging results from benchmark circuits of different sizes, we are only interested in routing area per tile (logic block).

4.3.2 Experimental Results

Tables 4.1(a) and 4.1(b) show the routing area per tile (in terms of the number of Minimum Transistor Equivalents[4]), critical path delay, and algorithm runtime for all sixteen circuits implemented on a "U"-shaped EPLC. In Tables 4.1(a) and 4.1(b), columns 2 and 3 show results for the original VPR placement and routing tool, columns 4 and 5 show the results for the original VPR placement algorithm and the enhanced routing algorithm, and columns 6 and 7 show the results for the enhanced placement and routing algorithms. As the Tables 4.1(a) and 4.1(b) show, the enhanced router produces similar results to the original router, but with a 62% faster run-time. When the enhanced placer is used, the runtime is increased somewhat, but the critical path delay is reduced by 12%.

Tables 4.2(a) and 4.2(b) show the results for an "O"-shaped core. Again, the improvement in runtime of the enhanced router is significant (40%). The improvement in critical path when the enhanced placer is used is 4%.

---

[4] A minimum transistor equivalent area is the layout occupied by the smallest transistor that can be contacted in a process, plus the minimum spacing to another transistor above it and to its right [5].

Circuit	Original Placer and Router		Original Placer, Enhanced Router		Enhanced Placer and Router	
	Area (MTE)	Critical Path (ns)	Area (MTE)	Critical Path (ns)	Area (MTE)	Critical Path (ns)
C6288	6883	44.9	5561	39.1	6016	40.9
alu4	10133	32.8	10510	36.5	11159	29.8
Apex2	11601	32.8	11601	39.7	11191	35.3
Apex4	13120	38.6	14808	32.4	13120	29.4
bigkey	11427	22.6	12248	25.3	11427	17.1
Dsip	10302	16.0	10914	14.7	10085	12.9
ex1010	10651	59.2	10469	57.5	10469	54.3
ex5p	11634	29.5	12292	29.9	12292	28.5
frisc	14552	33.2	12913	35.2	14082	35.1
iir16	6205	34.2	6318	32.0	6544	32.5
misex3	11541	32.6	11367	32.3	11541	26.1
misex3c	7264	20.9	7799	22.4	8262	18.3
S298	8447	40.6	8447	40.9	7665	36.8
seq	12181	34.0	11364	32.3	11364	27.5
sort8	11191	78.8	11043	79.4	11191	61.1
tseng	9940	17.4	9570	15.7	9940	16.1
Geo. Av.	10185	32.7	10149	32.5	10137	28.9
Diff %	--	--	-0.4%	-0.6%	-0.5%	-11.6%

Table 4.1(a) Area and delay results for "U"-shaped EPLC's

Circuit	Original Placer and Router		Original Placer, Enhanced Router		Enhanced Placer and Router	
	Placement Runtime (s)	Routing Runtime (s)	Placement Runtime (s)	Routing Runtime (s)	Placement Runtime (s)	Routing Runtime (s)
C6288	31	29	31	13	37	11
alu4	146	617	146	234	170	231
apex2	281	387	281	210	323	187
apex4	130	255	130	75	167	77
bigkey	284	452	284	256	446	228
dsip	314	300	314	127	351	144
ex1010	1755	2233	1755	704	1679	953
ex5p	125	195	125	105	138	92
frisc	1049	2030	1049	844	1022	1033
iir16	869	481	869	79	836	311
misex3	136	192	136	53	172	92
misex3c	30	53	30	30	33	22
S298	214	222	214	95	246	141
seq	254	412	254	176	315	226
sort8	245	423	245	126	291	141
tseng	138	134	138	45	173	84
Geo. Av.	216	304	216	117	250	142
Diff %	--	--	0%	-62%	16%	-53%

Table 4.1(b) Runtime results for "U"-shaped EPLC's

Circuit	Original Placer and Router		Original Placer, Enhanced Router		Enhanced Placer and Router	
	Area (MTE)	Critical Path (ns)	Area (MTE)	Critical Path (ns)	Area (MTE)	Critical Path (ns)
C6288	5254	41.3	5254	38.8	5445	40.9
alu4	9606	33.9	9606	30.9	9328	29.8
Apex2	11454	35.8	11630	34.5	11630	31.6
Apex4	12211	33.0	11964	30.9	11964	32.0
bigkey	9484	19.3	9850	20.2	9208	17.1
Dsip	9850	15.8	9484	17.1	9208	17.4
ex1010	9820	55.6	9820	59.7	9820	50.6
ex5p	12205	31.1	12632	31.6	13117	29.8
frisc	13133	44.2	13302	46.7	13507	41.1
iir16	7587	33.0	7587	33.6	8361	38.6
misex3	10955	31.2	10955	30.0	10559	27.2
misex3c	6787	20.9	6787	17.3	6787	17.5
S298	7844	44.1	7844	42.4	7568	43.6
seq	11103	29.3	11299	29.3	10644	30.0
sort8	11300	84.1	11105	88.8	10645	73.4
tseng	8254	15.7	8884	16.0	8440	16.8
Geo. Av.	9546	32.3	9615	32.0	9516	30.9
Diff %	--	--	0.7%	-1%	-0.3%	-4.3%

Table 4.2(a) Area and delay results for "O"-shaped EPLC's

Circuit	Original Placer and Router		Original Placer, Enhanced Router		Enhanced Placer and Router	
	Placement Runtime (s)	Routing Runtime (s)	Placement Runtime (s)	Routing Runtime (s)	Placement Runtime (s)	Routing Runtime (s)
C6288	29	12	29	11	38	11
alu4	131	220	131	125	182	117
apex2	253	553	253	241	346	267
apex4	117	279	117	194	176	117
bigkey	338	283	338	171	442	241
dsip	282	251	282	171	379	212
ex1010	1307	2608	1307	1171	1749	1540
ex5p	124	228	124	168	155	205
frisc	851	2276	851	1509	1155	1395
iir16	660	269	660	155	952	436
misex3	126	308	126	123	192	148
misex3c	26	24	26	13	35	15
S298	170	275	170	195	258	195
seq	216	365	216	229	306	262
sort8	221	406	221	198	289	243
tseng	137	62	137	46	175	59
Geo. Av.	192	247	192	148	264	173
Diff %	--	--	0%	-40%	37%	-30%

Table 4.2(b) Runtime results for "O"-shaped EPLC's

4.4 Summary

In this chapter, we have presented enhanced placement and routing algorithms for "U"-shaped and "O"-shaped embedded programmable logic cores. For a typical "U"-shaped core as shown in Figure 4.10, the algorithms give a 12% reduction in the critical path of the resulting circuit, compared to algorithms optimized for square and rectangular cores. A critical path reduction of 4% for an "O"-shaped core was obtained. For both "U"- and "O"-shaped cores, the enhanced router runs much faster than the original router but the enhanced placer runs slower than the original placer because calculation for additional delay tables is required. Overall, the run-time of the two algorithms together remains roughly the same for an "O"-shaped core, and is reduced by 25% for the "U"-shaped core.

Chapter 5

ARCHITECTURE STUDY FOR NON-RECTANGULAR EPLC'S

In the previous chapter, we developed the enhanced VPR place and route tool that can better map a user circuit into non-rectangular EPLC's than the original VPR. In this chapter, we will use an experimental approach to investigate the area and delay efficiency of three non-rectangular EPLC architectures: "L"-shaped, "U"-shaped and "O"-shaped cores. The potential uses of these three non-rectangular EPLC architectures in SoC design have been illustrated in Chapter 1. In this chapter, we will focus on two aspects related to these three non-rectangular cores:

1. In [2], it was suggested that a thinner rectangular core results in a lower area density and circuit speed than a thicker rectangular core. In this work, we will quantify how the thinness of each of the three non-rectangular cores affects its area density and circuit speed. This is important for chip designers to understand how flexible each of the three non-rectangular cores can be for reasonable performance tradeoffs.

2. We quantify the density and delay penalties on all three non-rectangular cores compared to square cores.

These two aspects will be studied by an experimental approach that is described in Section 5.1. Section 5.2 discusses the effect of thinness on the non-rectangular cores and shows the relative dimensions of "L"-, "U"- and "O"-shaped cores that we investigate in this work. In Section 5.3, we will experimentally quantify the area and delay efficiency for using the three non-rectangular cores, and measure the penalty compared to square cores. Lastly, a summary will be given in Section 5.4.

5.1 Experimental Methodology

In this chapter, we are investigating how shape and thinness affect the speed and area efficiency of different core architectures while we hold the other architectural parameters constant. Each non-rectangular core uses an island-style architecture, and each channel contains the same number of tracks and has the same segment length of one (one logic block wide). Each core architecture contains a number of logic blocks, each containing 4 basic logic elements (BLE's) with 10 inputs. The input and output pins are evenly distributed around the perimeter of the logic block. Each of the logic block inputs and outputs can be connected to one-quarter of the tracks in a neighboring channel. Each BLE consists of a 4-input LUT and a flip-flop. At the intersection of each horizontal and vertical channel is the Wilton switch block [8], and each programmable connection within the switch block is buffered. A 0.18 μm CMOS process from TSMC and eighteen MCNC benchmark circuits are used for this investigation.

The CAD flow we use to evaluate the three non-rectangular core architectures is identical to that of Chapter 4. Each circuit was optimized and technology-mapped

using SIS [19] and Flowmap/Flowpack [10]. The logic elements were then packed to logic blocks using T-VPack [7], and timing-driven placement and routing were performed using the enhanced VPR place and route tool as described in Chapter 4. For each circuit and architecture, the minimum number of tracks required for 100% routability was found; the number of tracks in each channel was then increased by 20% in order to perform "low-stress" routing. After the "low-stress" routing, the area and delay results are computed by our area and delay models. Finally, we use these quantities to compare both the area and delay efficiency among these core architectures.

5.2   Effect of Thinness on Non-Rectangular Core Architectures

In this work, we are investigating the effect of thinness on "L"-shaped, "U"-shaped and "O"-shaped core architectures. In [2], it was suggested a more rectangular EPLC would result in a circuit with a lower density and speed. It is unclear whether "L"-shaped, "U"-shaped and "O"-shaped core architectures follow the same trend as a rectangular core when their shapes become thinner. Also, if there are density and speed disadvantages from thinner core architectures, we want to quantify the area and delay penalties imposed on them so that chip designers can make a reasonably good decision on what the shape of an EPLC should look like.
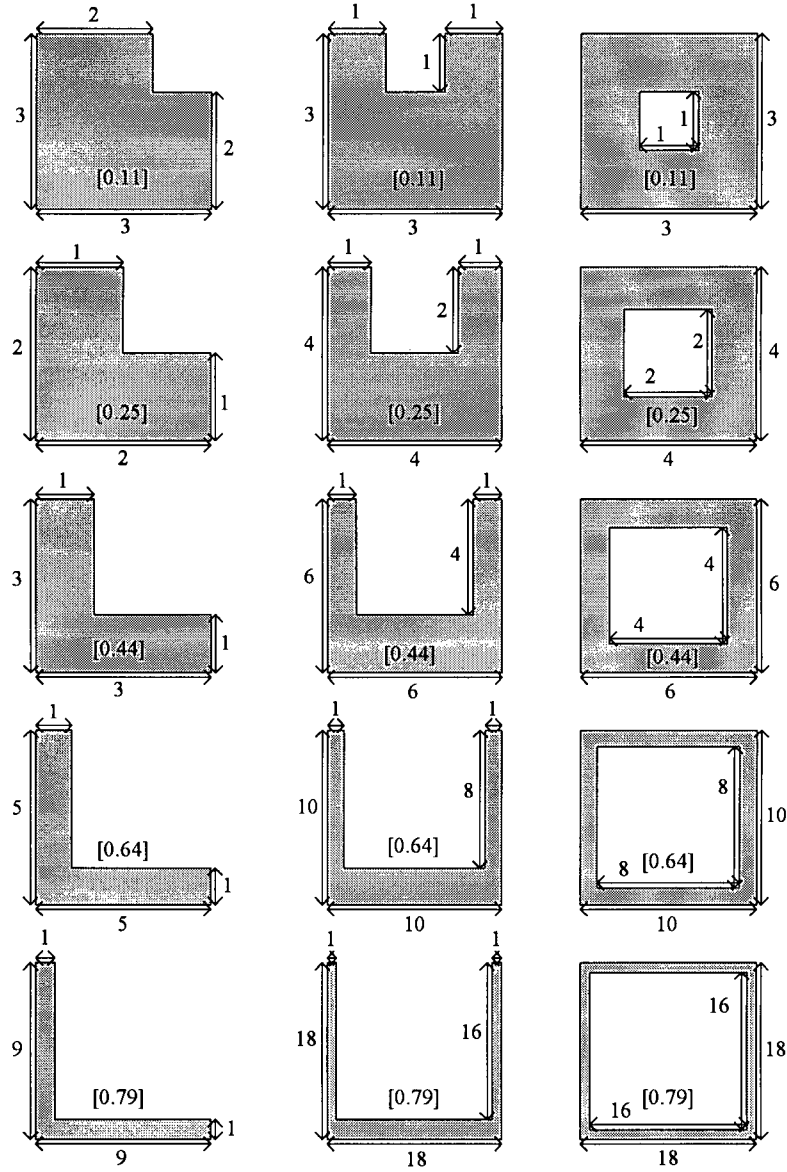
Figure 5.1 Relative aspect ratios for "L"-shaped, "U"-shaped and "O"-shaped cores under investigation (the number inside a bracket is its thinness value)

In this experiment, we assumed "L"-shaped, "U"-shaped and "O"-shaped cores employing the same architecture framework described in Section 5.2. In order to compare those three non-rectangular cores fairly, we introduce a metric *thinness* to measure how thin they are. The value of thinness ranges from zero to less than one,

which represents the proportion of a square core that is removed to create the non-rectangular core. If the value of thinness equals zero, that would describe a square core. The purpose of defining this metric is to provide an indicator how non-square these non-rectangular cores are as we will compare the area and delay efficiency between the three non-rectangular cores and square cores. In Figure 5.1, "L"-shaped, "U"-shaped and "O"-shaped cores in each row have the same value of thinness. For example, thinness of 0.11 means 11% of a square core area has been removed in order to create an "L"-shaped, a "U"-shaped or an "O"-shaped core shown in the first row in Figure 5.1. In this paper, we are going to investigate "L"-shaped, "U"-shaped and "O"-shaped cores in various thinness between 0.11 (a very thick core) and 0.79 (a very thin core).

The relative aspect ratios of each "L"-shaped, "U" –shaped or "O"-shaped cores under investigation are illustrated in Figure 5.1. Notice that these are not the exact dimensions of what a target platform would be implemented on, but rather relative aspect ratios used for specifying their core architectures for the CAD tool. The enhanced VPR tool described in Chapter 4 can find the smallest possible size for a core of interest to realize a target circuit and at the same time keep its relative aspect ratios.

5.3   Experimental Results

Figure 5.2 shows the routing area comparisons for "L"-shaped, "U"-shaped and "O"-shaped cores as a function of thinness. The vertical axis is the number of minimum-width transistor areas per tile in the routing fabric of an EPLC, geometrically averaged over all 18 benchmark circuits. Since all non-rectangular cores use the same logic

blocks, the area occupied by a logic block is the same throughout all cores. Therefore, we are only interested in their routing area.

Routing Area vs. Thinness for L-shaped, U-shaped and O-shaped Cores
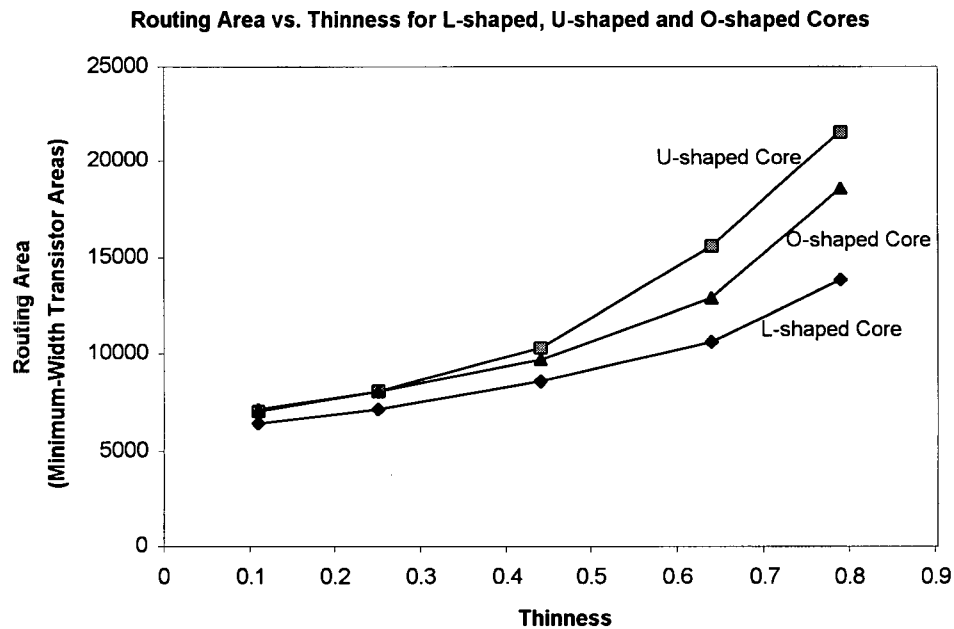

Figure 5.2 Routing area results

A general trend shown in Figure 5.2 is that, for all three cores, as the thinness of the core increases, the routing area increases. Over the entire range of thinness, an "L"-shaped core has the best area-efficiency among all three cores. An "O"-shaped core performs better than a "U"-shaped core when thinness equals 0.44 or above. When the thinness is below 0.44, both "O"-shaped and "U"-shaped cores have very similar results.

Figure 5.3 shows the delay comparisons for the three non-rectangular cores. The vertical axis is the critical path delay of each circuit, geometrically averaged over all benchmarks.

68

**Delay vs. Thinness for L-shaped, U-shaped and O-shaped Cores**
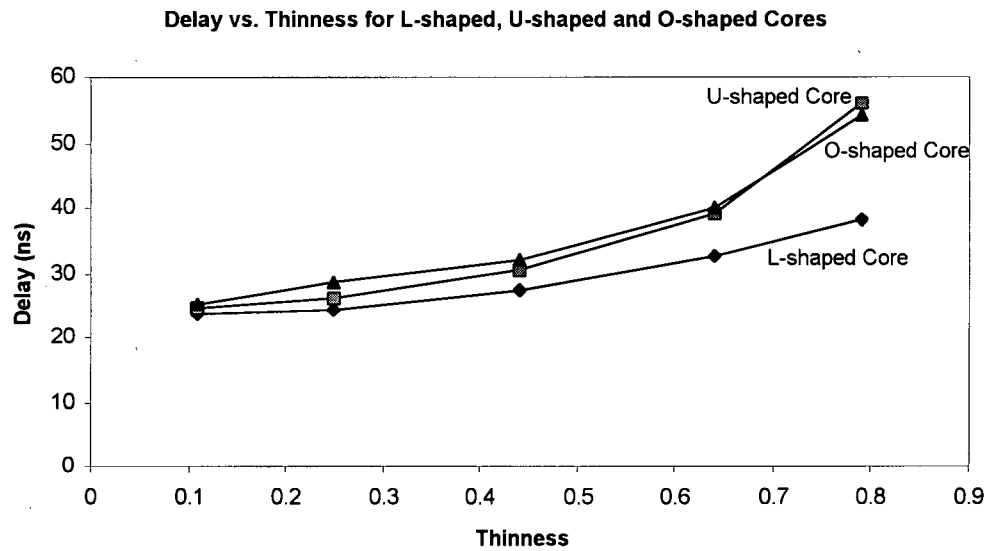
Figure 5.3 Delay results

Figure 5.3 shows that when the shape of the core becomes narrower, the critical path delay increases especially for "U"-shaped and "O"-shaped cores. An "L"-shaped core results in the fastest circuit speed among all three cores over the entire range of thinness. On the other hand, a "U"-shaped core has shorter delay than an "O"-shaped core except for thinness of 0.79, in which a "U"-shaped core runs slightly slower than an "O"-shaped core.

Figure 5.4 shows the comparisons of the minimum channel width required for the three non-rectangular cores. These results are based on "high-stress" routing where a benchmark circuit is just barely routable, whereas the results shown in Figure 5.2 and 5.3 are based on "low-stress" routing where 20% more routing resources than the minimum required are given to route a given circuit.

Minimum Channel Width vs. Thinness for L-shaped, U-shaped and O-shaped Cores
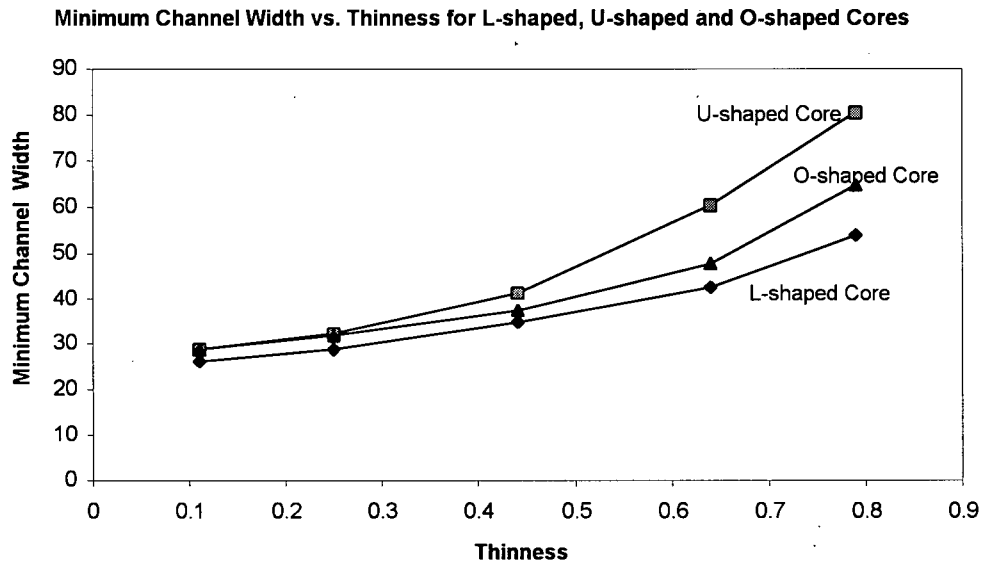

Figure 5.4 Minimum channel width results

As shown in Figure 5.4, an "L"-shaped core requires the least number of tracks per channel to route a circuit, thus has the highest routing flexibility among all three cores. An "O"-shaped core needs fewer tracks per channel to route a circuit than a "U"-shaped core when thinness is above 0.11.

Figure 5.5 and 5.6 show the comparisons of both routing area and delay penalties associated with using the three non-rectangular cores over a square core respectively. A square core is chosen for comparison because it is the best shape for area-efficiency and speed [2]. The routing area and delay penalties are shown as percentages.

Routing Area Penalty Over Square Core vs. Thinness for L, U and O-shaped Cores
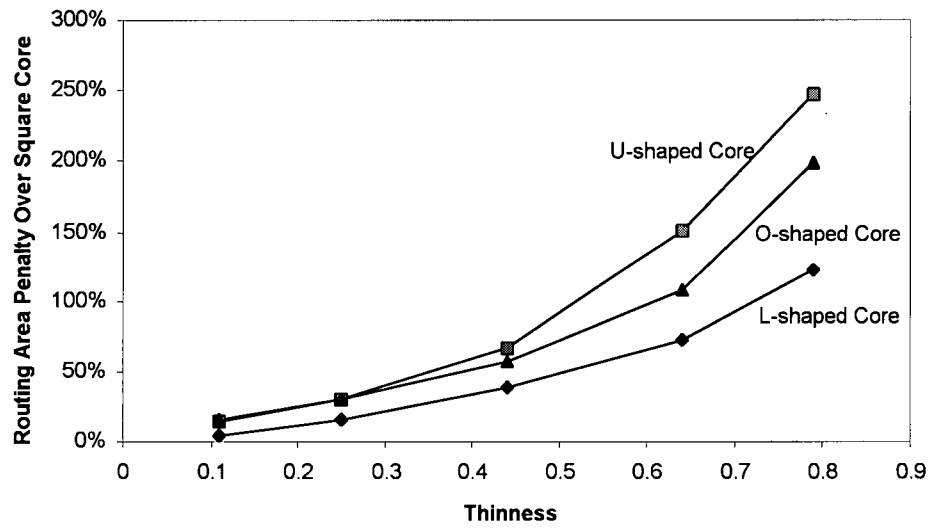

Figure 5.5 Routing area penalty over a square core

Delay Penalty Over Square Core vs. Thinness for L, U and O-shaped Cores
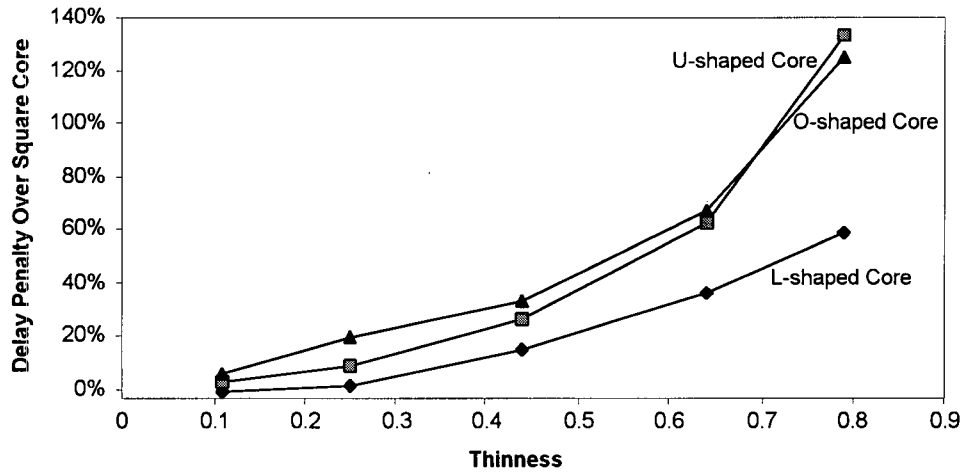
Figure 5.6 Delay penalty over a square core

Delay Penalty Over Square Core vs. Thinness for U-shaped Cores
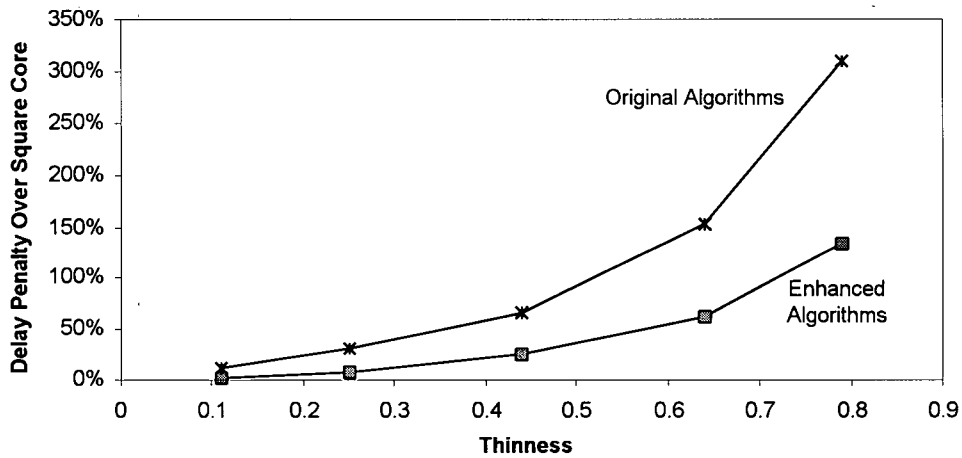
Figure 5.7 Delay penalty reduction by enhanced algorithm on "U"-shaped cores

Delay Penalty Over Square Core vs. Thinness for O-shaped Cores
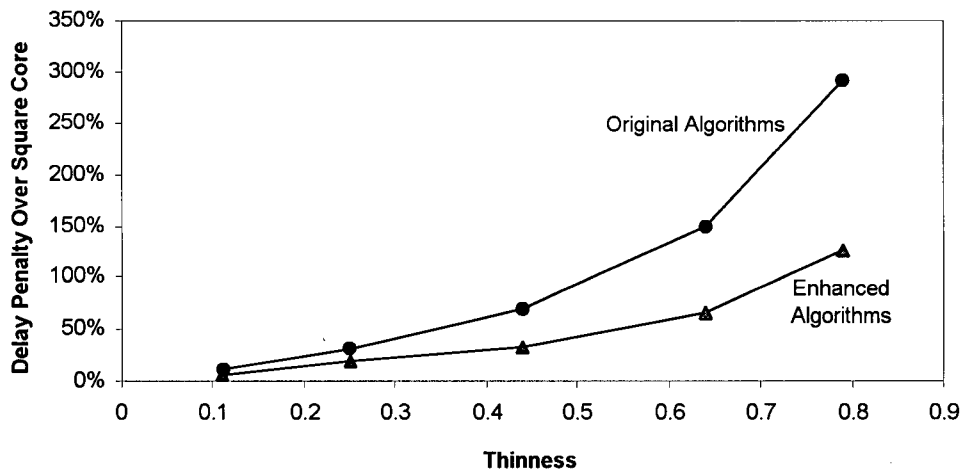
Figure 5.8 Delay penalty reduction by enhanced algorithm on "O"-shaped cores

It seems it is unwise to use any of the three non-rectangular EPLC's in chip design because of their area and delay penalties over a square core. However, in many applications, the fixed shapes and sizes of the other IP cores will dictate that a non-rectangular EPLC is required. For thick "L"-shaped cores (thinness of 0.25 or below), the area and delay penalties over a square core are small, at most 15% for area penalty

72

and 1% for delay penalty. For thick "U"-shaped and "O"-shaped cores (thinness of 0.25 or below), the area and delay penalties are less than 30%. As thinness increases, however, the penalties become much larger; for thinness of 0.79, the area and delay penalties of "L"-shaped cores are 122% and 58% respectively, the area and delay penalties of "O"-shaped cores are 199% and 125% respectively, the area and delay penalties of "U"-shaped cores are 247% and 133% respectively.

For the same thinness, the results show that an "L"-shaped core is the best shape among the three non-rectangular cores in terms of area efficiency, circuit speed and routing flexibility. This is due to two factors. First, the thickness of an "L"-shaped core is approximated twice as thick as that of a "U"-shaped core or an "O"-shaped core with the same thinness metric. Second, an L-shaped core requires fewer long connections to connect I/O pins together than "U"-shaped and "O"-shaped cores. For almost any value of thinness, an "O"-shaped core has better routing flexibility than a "U"-shaped core resulting in an "O"-shaped core having higher area density than a "U"-shaped core. In applications such as core test wrappers, it is advantageous to use an "O"-shaped core over a "U"-shaped core because such a design may require a very thin core and it is shown a thin "O"-shaped core has better area-efficiency and circuit speed than a thin "U"-shaped core. In addition, an "O"-shaped core has higher I/O pins per unit area than a "U"-shaped core, which is beneficial to high-bandwidth testing applications.

When we compare the area impact to the delay impact, clearly the delay impact is not as significant as the area impact. Figure 5.7 shows that this is directly a result of the

improved place and route algorithms described in Chapter 4. Using the enhanced algorithms, for a thinness of 0.64, the delay penalty of a "U"-shaped core is approximately 60%. If the original algorithms had been employed, the delay penalty would be almost 150%. The results for an "O"-shaped core shown in Figure 5.8 are similar.

5.4   Summary

In this chapter we have presented the experimental results for "L"-shaped, "U"-shaped and "O"-shaped EPLC's. We have shown that for the same thinness, "L"-shaped cores have the best in density and speed compared to "U"-shaped and "O"-shaped cores, while thinness is 0.44 or above, "O"-shaped cores have better area-efficiency than "U"-shaped cores. We have also shown that for thinness is 0.25 or below, the area (less than 30%) and delay (less than 20%) penalty of "L"-shaped, "U"-shaped and "O"-shaped cores are much smaller than the same shaped cores with higher thinness.

It is important to note that we are *not* suggesting that all programmable logic cores should be rectangular or square. Indeed, in many cases, a non-rectangular core will be required, either because of I/O constraints or because it is the only shape that will fit well with other cores in an SoC design. Instead, our results show that *if* such a core is used, the enhancements to the placement and routing algorithms are required, in order to reduce the delay and area penalty as much as possible.

Chapter 6

CONCLUSIONS

In this thesis, we investigated the CAD algorithm and architecture of non-rectangular embedded programmable logic cores. Specifically, we focused on "L"-shaped, "U"-shaped and "O"-shaped cores. First, we developed a new specification method to support non-rectangular cores, and incorporated it into the FPGA CAD tool VPR in order to place and route those cores. Next, we enhanced the existing routing and placement algorithms targeting non-rectangular cores. We also examined the area and delay performances of the three non-rectangular core architectures.

In order to adapt the SoC design style, non-rectangular EPLC's may be needed to better mesh with the other ASIC cores which dictate the shape of an EPLC. However, there is no published work discussing how to efficiently use a non-rectangular EPLC in an SoC design. There is also no freely available CAD tool that can be used to describe a non-rectangular core. In Chapter 3, we presented a simple and efficient specification method that describes a non-rectangular core on an architecture file used by the free industry-strength FPGA evaluation CAD tool VPR. We also modified VPR to correctly place and route a circuit on "L"-shaped, "U"-shaped and "O"-shaped cores. As a result, an evaluation for these cores can be done using this modified version of VPR.

Since the routing and placement algorithms in VPR were not optimized for non-rectangular cores, there may be potential improvement that can be made through careful examination on the algorithms. In Chapter 4, we proposed two enhancements for the routing algorithm, and showed that for typical "U"- and "O"-shaped cores shown in Figure 4.9, the enhanced routing algorithm has a runtime that is 2.1 times that of the original algorithm on a "U"-shaped core, and 1.4 times on an "O"-shaped core. We also proposed an enhancement for the placement algorithm. For the same "U"- and "O"-shaped cores, we demonstrated that the combined enhanced placement and routing algorithms outperforms the original VPR by producing not only denser implementation, but also on average 12% faster circuits on a "U"-shaped core, and 4% faster circuits on an "O"-shaped core.

In order to understand the tradeoffs between using a non-rectangular core and a rectangular core in an SoC design, it is important to evaluate the delay and area performances of a non-rectangular core. Chapter 5 used the enhanced VPR with the enhanced placer and router to examine "L"-shaped, "U"-shaped and "O"-shaped cores over a range of thinness. The experimental results showed that for the same thinness, "L"-shaped cores have the best in density and speed compared to "U"-shaped and "O"-shaped cores, while for a thinness of 0.44 or above, "O"-shaped cores are more area efficient than "U"-shaped cores. We have also shown that for a thinness of 0.25 or below, the area (less than 30%) and delay (less than 20%) penalty of the three non-rectangular cores are much smaller than the same shaped cores with higher thinness.

Even though the penalty for using non-rectangular cores is significant, in many cases they will be required, either to better mesh with the other ASIC cores, or because of I/O constraints. Our results show that if such a core is used, the enhancements to the placement and routing algorithms are required, in order to reduce the speed and area penalty as much as possible.

6.1　Future Work

Currently the new specification method lets users describe a non-rectangular core on the architecture file but does not allow the specification of a heterogeneous routing architecture, where the horizontal and vertical channels have different numbers of tracks. In [2], it was suggested that for a rectangular EPLC, channels in the long direction should have more tracks than channels in the narrow direction. An interesting project would remove this restriction and explore whether applying results from [2] into each rectangular region on a non-rectangular core, such as an "L"-shaped, a "U"-shaped or an "O"-shaped core, would improve area density and circuit speed.

Segmentation distribution has an important impact on the density and delay of EPLC's. In this paper, we restrict all wires to span one logic block length. It will be interesting to explore non-rectangular core architectures that have wires spanning more than one logic block, and determine the optimal segmentation length for each non-rectangular core. Also, it is essential to find good switch block topologies for use with segmented non-rectangular core architectures.

As we understand how various architectural components, such as horizontal and vertical channel width ratio, segmentation distribution and switch block topology, affect the speed and density of a non-rectangular EPLC, an "embedded programmable logic generator" can be created to provide an EPLC with the optimal architecture to a chip designer. Using this generator, the chip designer can request a core with certain properties (shape, size, number and type of interface pins etc.), and the generator would automatically create an EPLC that best meets the user's specifications. A project is currently under way at the University of British Columbia to implement such a generator.

6.2   Summary of Contributions

The contributions of our work are summarized as follows:

i.   We developed a novel specification method to describe a non-rectangular EPLC in an architecture file used by the modified FPGA CAD tool VPR.

ii.   We proposed enhancements to existing placement and routing algorithms and quantified the improvement on area, delay and runtime over the original algorithm when targeting non-rectangular cores.

iii.   We quantified and compared the area and delay performances of "L"-shaped, "U"-shaped and "O"-shaped cores over a variety of

thinness, and measured the penalty of using these three non-rectangular cores compared to square cores.

This work has enabled researchers or chip designers to evaluate non-rectangular EPLC's in the SoC design style through the new specification method and the enhanced CAD tool. In addition, new understandings of the area and delay efficiency of "L"-shaped, "U"-shaped and "O"-shaped cores are provided in this research.

REFERENCES

1. S.J.E. Wilton and R. Saleh, "Programmable Logic IP Cores in SoC Design: Opportunities and Challenges," IEEE Custom Integrated Circuits Conference, May 2001, pp. 63 – 66.

2. P. Hallschmid and S.J.E. Wilton, "Detailed Routing Architecture for Embedded Programmable Logic IP Cores," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 2001, pp. 69 – 74.

3. A. Marquardt, V. Betz and J. Rose, "Timing-Driven Placement for FPGA," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2000, pp. 203 – 213.

4. E. Ahmed and J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 2000, pp. 3 – 12.

5. V. Betz, J. Rose and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs," Kluwer Academic Publishers, 1999.

6. V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," International Workshop on Field-Programmable Logic and Applications, August 1997, pp. 213 – 222.

7. A. Marquardt, V. Betz and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 1999, pp. 37 – 46.

8. S.J.E. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory," Ph.D. thesis, University of Toronto, 1997.

9. C. Ebeling, L. McMurchie, S.A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," IEEE Trans. on VLSI, Vol. 3, No. 4, December 1995, pp. 473 – 482.

10. J. Cong and Y. Ding, "Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 1, January 1994, pp. 1-12.

11. S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," Technical Report, Microelectronics Center of North Carolina, 1991.

12. J. Rose, R. Francis, D. Lewis and P. Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," JSSC, October 1990, pp. 1217 – 1225.

13. Y.W. Chang, D. Wong and C. Wong, "Universal Switch modules for FPGA design," ACM Transactions on Design Automation of Electronic Systems, Vol. 1, January 1996, pp. 80 – 101.

14. M.I. Masud and S.J.E. Wilton, "A New Switch Block for Segmented FPGAs," in International Workshop on Field-Programmable Logic and Applications, Glasgow, U.K., September 1999, pp. 274 – 281.

15. G.G. Lemieux, P. Leventis and D. Lewis, "Generating Highly-Routable Sparse Crossbars for PLDs," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 2000, pp. 155 – 164.

16. G.G. Lemieux and D. Lewis, "Using Sparse Crossbars Within LUT Clusters," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 2001, pp. 59 – 68.

17. M.I. Masud, "FPGA Routing Architectures: A Novel Switch Block and Depopulated Interconnect Matrix Architectures," M.A.Sc Thesis, University of British Columbia, December 1999.

18. R.K. Brayton, G. D. Hachtel and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," in Proceedings of the IEEE, Vol. 78, No. 2, February 1990, pp. 264 – 300.

19. E.M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," ICCD, 1992, pp. 328 – 333.

20. D. Huang and A. Kahng, "Partitioning-Based Standard-Cell Global Placement with an Exact Objective," ACM Symposium on Physical Design, 1997, pp. 18 – 25.

21. M. Khellah, S. Brown and Z. Vranesic, "Modelling Routing Delays in SRAM-based FPGAs," Canadian Conference on VLSI, 1993, pp. 6B.13 – 6B.18.

22. C. Alpert, T. Chan, D. Huang, A. Kahng, I. Markov, P. Mulet and K. Yan, "Faster Minimization of Linear Wirelength for Global Placement," ACM Symposium on Physical Design, 1997, pp. 4 – 11.

23. B. Riess and G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement," IEEE International Symposium on Circuits and Systems, 1995, pp. 377 – 380.

24. W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," IEEE Transaction on CAD, March 1995, pp. 349 – 359.

25. W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits," DAC, 1995, pp. 211 – 215.

26. S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," Science, May 1983, pp. 671 – 680.

27. Y.L. Wu and M. Marek-Sadowska, "An Efficient Router for 2-D Field Programmable Gate Arrays," European Design Automation Conference, 1994, pp. 412 – 416.

28. M.J. Alexander, G. Robins, "New Performance-Driven FPGA Routing Algorithms," DAC, 1995, pp. 562 – 567.

29. Y.S. Lee, A. Wu, "A Performance and Routability Driven Router for FPGAs Considering Path Delays," DAC, 1995, pp. 557 – 561.

30. A. Yan, R. Cheng and S.J.E. Wilton, "On the Sensitivity of FPGA Architectural Conclusions to Experimental Assumptions, Tools and Techniques," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 2002, pp. 147 – 156.

31. M. Placzewski, "Plane Parallel A* Maze Router and Its Application to FPGAs," DAC, 1992, pp. 691 – 697.

32. Y. Chang, S. Thakur, K. Zhu and D. Wong, "A New Global Routing Algorithm for FPGAs," ICCAD, 1994, pp. 356 – 361.

33. S. Brown, J. Rose and Z.G. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays," IEEE Transaction on CAD, May 1992, pp. 620 – 628.

34. G. Lemieux and S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs," ACM/SIGDA Physical Design Workshop, 1993, pp. 215 – 226.

35. E. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerical Mathmatics, Vol. 1, 1959, pp. 269 – 271.

36. S. Oldridge, "A Novel FPGA Architecture Supporting Wide Shallow Memories," M.A.Sc Thesis, University of British Columbia, April 2002.

37. W. Dees and R. Smith, "Performance of Interconnection Rip-Up and Reroute Strategies," DAC, June 1981, pp. 382 – 390.

38. V. Betz, "VPR and T-VPack User's Manual version 4.30," March 2000.

39. V. Betz and J. Rose, "Automatic Generation of FPGA Routing Architectures from High-Level Descriptions," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 2000, pp. 175 – 184.

40. B.K. Britton, Y. T. Oh, W. Oswald, H. T. Nguyen, S. Singh, G. Lee, Wai-Bor Leung, C. Spivak, J. Steward and C. T. Chen, "Second Generation ORCA Architecture Utilizing 0.5um Process Enhances the Speed and Usable Gate Capacity of FPGAs," IEEE International ASIC Conference, September 1994, pp. 474 – 478.

41. S. Brown, R. Francis, J. Rose and Z. Vranesic, "Field Programmable Gate Arrays," Kluwer Academic Publishers, 1992.

42. C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," ICCAD, 1994, pp. 690 – 695.

43. S. Singh, J. Rose, D. Lewis, K. Chung and P. Chow, "Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed," IEEE Custom Integrated Circuits Conference, 1991, pp. 6.1/1 – 6.1/6.

44. C. Matsumoto, "Actel Plans to Produce FPGAs as ASIC Cores," Electrical Engineering Times, September 15, 2000.

45. C. Matsumoto, "LSI Logic ASICs to add Programmable Logic Cores," Electrical Engineering Times, August 29, 1999.

46. "Embedded FPGA Cores Enable Programmable ASICs, ASSPs," Electrical Engineering Times, September 20, 1999.

47. "Startup stakes out ground between FPGAs and ASICs," Electrical Engineering Times, July 9, 2001.

48. "Lucent Introduces ORCA Series 4 FPGA," Programmable Logic News and Views, pp. 7-11.

49. R. Merritt, "QuickLogic Steps up Merger of FPGA with IP Cores – DSP First Target," Electrical Engineering Times, August 9, 2000.

50. Actel Corporation, Data Book 2001, http://www.actel.com.

51. Altera Inc., Data Book 2001, http://www.altera.com.

52. Xilinx Inc., Data Book 2001, http://www.xilinx.com.