Field-Programmable Gate Array Architectures and Algorithms Optimized for Implementing Datapath Circuits

Andy Gean Ye

November 2004

Field-Programmable Gate Array Architectures and Algorithms Optimized for Implementing Datapath Circuits

by

Andy Gean Ye

A thesis submitted in conformity with

the requirements for the degree of

Doctor of Philosophy

November 2004

The Edward S. Rogers Sr. Department of

Electrical and Computer Engineering

University of Toronto

Toronto, Ontario, Canada

“We are pattern-seeking animals, the descendents of hominids who were especially dexterous at making causal links between events in nature. The associations were real often enough that the ability became engrained in our neural architecture. Unfortunately, the belief engine sputters occasionally, identifying false patterns as real . . .

The solution is science, our preeminent pattern-discriminating method and our best hope for detecting a genuine signal within the noise of nature’s cacophony.”

– “Codified Claptrap,” Michael Shermer, Scientific American, June 2003

Abstract

Field-Programmable Gate Arrays (FPGAs) are user-programmable digital devices that provide efficient, yet flexible, implementations of digital circuits. Over the years, the logic capacity of FPGAs has been dramatically increased; and currently they are being used to implement large arithmetic-intensive applications, which contain a greater portion of datapath circuits. Each circuit, constructed out of multiple identical building blocks called bit-slices, has highly regular structures. These regular structures have been routinely exploited to increase speed and area-efficiency in designing custom Application Specific Integrated Circuits (ASIC).

Previous research suggests that the implementation area of datapath circuits on FPGAs can also be significantly reduced by exploiting datapath regularity through an architectural feature called configuration memory sharing (CMS), which takes advantage of datapath regularity by sharing configuration memory bits across, normally independently controlled, reconfigurable FPGA resources. The results of these studies suggest that CMS can reduce the total area required to implement a datapath circuit on FPGA by as much as 50%. They, however, did not take into account detailed implementation issues such as transistor sizing, utilizable regularity in actual datapath circuits, and Computer-Aided Design (CAD) tool efficiencies.

This study is the first major in-depth study on CMS. The study found that when detailed implementation issues are taken into account, the actual achievable area savings can be significant less than the previous estimations — the CMS architecture investigated in this study is only about 10% more area efficient than a comparable conventional and widely studied FPGA architecture for implementing datapath circuits. Furthermore, this increase in area efficiency has a potential speed penalty of around 10%.

To conduct the study, a new area-efficient FPGA architecture is designed along with its supporting CAD tools. The architecture, called Multi-Bit FPGA (MB-FPGA), is the first completely specified FPGA architecture that employs CMS routing resources. This sharing significantly reduces the number of configuration memory bits and consequently increases its area efficiency.

The use of the CMS resources, however, imposes new demands on the traditional FPGA CAD algorithms. As a result, a complete set of CAD tools supporting FPGAs containing CMS resources are proposed and implemented. These tools are designed to extract and utilize datapath regularity for the CMS resources. It is shown that these tools yield excellent results for implementing a set of realistic datapath circuits on the MB-FPGA architecture.

Acknowledgements

I would like to take this opportunity to express my sincere thanks and appreciation to my academic supervisors. Professor Jonathan S. Rose and Professor David M. Lewis have provided continual source of guidance, support, advice, and friendship through out my graduate studies. It has been my privilege to work with these two experienced academics and excellent engineers. They have made my doctoral studies a truly rewarding and unforgettable experience. I would especially like to thank Professor Jonathan S. Rose for taking the extra mile to point out the big pictures in my research and my academic career. I would also like to thank Professor David M. Lewis for all his extremely detailed and insightful technical advice.

My father and mother have always been a constant support throughout my studies and my personal life. Their courage, kindness, hard-working ethics, and constant striving for goodness, have been a great inspiration to me. I am especially inspired by their courage in overcoming almost insurmountable difficulties in immigrating and establishing themselves in Canada. This thesis is as much an achievement of theirs as it is mine.

I would like to thank my academic supervisors, the Natural Sciences and Engineering Research Council, the Ontario Government, Communications and Information Technology Ontario, and Micronet for their financial support.

Finally, I would like to thank all my friends for endless hours of play, insightful discussions, rejuvenating lunch outings, friendship, support, and encouragement. Thank you all!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Introduction to Field-Programmable Gate Arrays

Field-Programmable Gate Arrays (FPGAs) are user programmable digital devices that provide efficient, yet flexible, implementations of digital circuits. An FPGA consists of an array of programmable logic blocks interconnected by programmable routing resources. The flexibility of FPGAs allows them to be used for a variety of digital applications from small finite state machines to large complex systems. The research reported in this thesis is focused on reducing the implementation area of large, arithmetic-intensive, systems on FPGAs through architectural innovations. We also present new and innovative Computer-Aided Design (CAD) algorithms which are designed to support the new architecture.

Since their invention in 1984 [Cart86], FPGAs have become one of the most widely used platforms for digital applications. Comparing to alternative technologies, which directly fabricate hardware on silicon, FPGAs have the advantage of instant manufacturability and infinite re-programmability. They also incur lower cost for low to medium volume production of digital devices. Unlike full fabrication of integrated circuits, which require highly specialized manufacturing facilities and cost hundreds of thousands of dollars to prototype, FPGAs can be programmed on the desks of their designers. This makes the verification of hardware designs much faster — once a mistake is found, unlike full fabrication, which has to rebuild masks, corrections on FPGAs only take the reprogramming of a few configuration memory bits. This also allows multiple design iterations to be done quickly and at a much lower cost. FPGA based applications also can be updated after they are delivered to their customers allowing incremental hardware improvements and adaptation of old designs to new protocols and specifications. Furthermore, FPGA CAD tools are much cheaper to acquire than comparable CAD tools that support full fabrication.

These advantages allow FPGAs to compete head on with full fabrication technologies, such as the Application Specific Integrated Circuit (ASIC) technology, for market share. The user-programmability of FPGAs, however, also has its shortcomings: FPGAs are more expensive in high volume production; circuits implemented on FPGAs are usually many times bigger and slower than comparable ASICs. In order for FPGAs to overtake full fabrication technologies, FPGA researchers need to find new and innovative ways of improving the performance and logic density of FPGAs.

1.2 Thesis Motivation

Over the years, the capacity of FPGAs has increased dramatically. Current state-of-the-art devices can contain near 100,000 logic elements (where a logic element is typically a 4-input look-up table, a flip-flop, and 1-bit worth of arithmetic carry logic) [Alte02] [Xili02] with a projected logic capacity of several million logic gates [Xili02]. In comparison, the first FPGA [Cart86] contains only 64 logic blocks with a projected capacity of between 1000 and 1600 gates. Since the logic capacity has grown significantly, the application domain of FPGAs has been greatly expanded. Modern FPGAs are often used to implement large arithmetic-intensive applications, including CPUs, digital signal processors, graphics accelerators and internet routers.

Arithmetic-intensive applications often contain significant quantities of regular structures called *datapaths*. These datapaths are constructed out of multiple identical building blocks called *bit-slices*. They are used to perform mathematical or logical operations on multiple-bits of data. It is our hypothesis that *greater area efficiency can be achieved in FPGAs by incorporating datapath specific features*. One such feature is the *configuration memory shar-*

ing (CMS) routing resources proposed by Cherepacha and Lewis in [Cher96], which takes the advantage of the regularity of datapath circuits by sharing configuration memory bits across normally independent routing resources. This reduces the number of programming bits needed to control these resources and consequently reduces FPGA area.

The primary focus of this thesis is to explore in-depth methods of increasing FPGA logic density for arithmetic circuits using multi-bit logic and CMS routing structures under a highly automated modern design environment. The goal of the study is to determine the most appropriate amount of CMS routing resources in order to achieve the best logic density improvement for real circuits using real automated CAD tools. Since routing area typically consists of a significant percentage of the total FPGA area, its reduction is particularly important to reduce the overall FPGA area. This research is a continuation of the DP-FPGA work [Cher96]. It is also closely related in methodology to several previous FPGA research projects [Betz99a].

1.3 Research Approach

Datapath-oriented FPGA architectures are studied in this thesis using an experimental approach. A parameterized FPGA architecture, called Multi-Bit FPGA (MB-FPGA), with bus-based CMS routing resources has been proposed. A complete CAD flow for the architecture has also been implemented. The experiments consist of varying the amount of CMS routing resources and measuring the effects on the implementation area of datapath circuits. The results of the experiments provide insight to the amount of CMS routing resources that are needed to achieve area savings for real datapath applications using real CAD tools.

1.4 Thesis Contributions

To the best knowledge of the author, the MB-FPGA architecture is the first completely specified special-purpose FPGA architecture targeting datapaths. It is also the first FPGA architecture containing CMS routing resources supported by a complete set of CAD tools. Furthermore, the architectural study presented here represents the first in-depth empirical study on the effectiveness of CMS routing resources in translating datapath regularity into area savings. Previous studies [Cher96] [Leij03] on the subject are all analytical in nature. As a result, none of them takes the detailed transistor-sizing issues, the actual benchmark regularity, and the area efficiency of the CAD algorithms into account. As it will be shown by the results of this study, these previous studies are much less accurate and tends to overestimate the benefits of the CMS resources.

1.5 Thesis Organization

This thesis is organized as follows. Chapter 2 reviews the background information relevant to this work, including a review of various CAD tools available for transforming high-level descriptions of digital circuits into FPGA programming information. The review includes a brief description of representative tools from each major class of CAD tools. The chapter also describes the work of two previous architectural studies that significantly influenced the work presented in this thesis.

Chapter 3 presents a new, highly parameterized, datapath-oriented FPGA architecture called the Multi-Bit FPGA (MB-FPGA). The architecture is unique in that it uses a mixture of conventional routing resources and CMS routing resources. The combination allows a homogenous architecture for the efficient implementation of large datapath circuits as well as small non-datapath circuits. The architecture is the basis from which the CAD flow presented in

Chapter 4, 5, and 6 are designed and the experiments presented in Chapter 7 and 8 are conducted.

Chapter 4, 5, and 6 presents a new datapath-oriented CAD flow. The flow includes several new algorithms covering the entire process of transforming and optimizing high-level circuit descriptions into FPGA programming information. These algorithms are unique in that they effectively preserve and utilize datapath regularity on CMS routing resources. In particular, Chapter 4 discusses datapath-oriented synthesis; Chapter 5 presents a datapath-oriented packing algorithm; and Chapter 6 discusses datapath-oriented placement and routing.

Using the synthesis and packing tools presented in Chapter 4 and 5, Chapter 7 characterizes and quantifies the amount of regularity presented in a typical datapath circuit. Analytically, this regularity information is used to determine good values for several important MB-FPGA architectural parameters, including the degree of configuration memory sharing (called *granularity*) and the proportion of CMS routing resources.

The MB-FPGA is directly explored in Chapter 8 using an experimental approach. The CAD flow presented in Chapter 4, 5, and 6 is used to implement a set of datapath circuits on the MB-FPGA architecture. For each circuit the best area is evaluated by varying a range of architectural parameters. The experiments measure the effectiveness of CMS routing on improving the area efficiency of datapath circuit implementations and the effect of these routing resources on performance.

Finally, Chapter 9 provides concluding remarks and directions for future research.

2 Background

2.1 Introduction

This chapter reviews the two main fields of research, FPGA CAD tools and FPGA architectures, that are studied in this thesis. Section 2.2 provides some necessary background information on FPGA CAD that is assumed in various discussions, particularly in Chapter 4, 5, and 6, which discuss CAD design for the MB-FPGA architecture. Section 2.3 describes several previous FPGA architectures to provide a point of reference for the MB-FPGA architecture presented in Chapter 3 and the FPGA modeling methodology that is used throughout this work.

2.2 FPGA CAD Flow

Since the focus of this thesis is the design of a datapath-oriented FPGA architecture supported by a highly automated modern design environment, this chapter begins with an overview of the modern CAD tools that are commonly used to implement circuits on FPGAs. A typical CAD flow for FPGAs consists of a series of interconnected CAD tools as illustrated in Figure 2.1. The input to the flow usually is a high-level description of the hardware, expressed in high-level hardware description languages such as Verilog or VHDL.

The description is read by a synthesis program [Call98] [Cora96] [Koch96a] [Koch96b] [Kutz00a] [Kutz00b] [Nase94] [Nase98] [Syno99] [Synp03], which maps the description language into a network of Boolean equations, flip-flops, and pre-defined modules. During the synthesis process, the Boolean equations are optimized with respect to estimated implementation area and delay. The optimizations performed at this stage are limited to those that can benefit circuit implementations on any medium, not just FPGAs. Some synthesis algorithms, including [Call98] [Cora96] [Koch96a] [Koch96b] [Kutz00a] [Kutz00b] [Nase94] [Nase98],

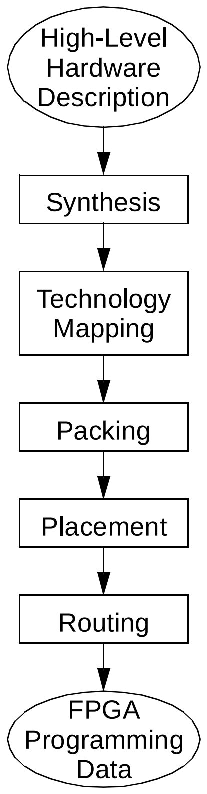


Figure 2.1: FPGA CAD Flow

also attempt to preserve the regularity of datapath circuits by maintaining a hierarchy that clearly delineates the boundary of bit-slices. These algorithms are often called the *datapath-oriented synthesis algorithms*.

The Boolean equations are then first mapped into a circuit of FPGA Look-Up Tables (LUTs) through the technology mapping process [Syno99]. Then the packing process [Betz97a] [Betz99a] [Marq99] [Bozo01] groups LUTs and flip-flops into logic blocks, each of which usually contains several LUTs and flip-flops. During the technology mapping and the packing process, the circuit is again optimized with respect to estimated implementation area and delay. This time the optimizations are targeted towards specific implementation technologies. Area is typically optimized by minimizing the number of LUTs or logic blocks that are required to implement the circuit; and delay is often optimized by minimizing the number of LUTs or logic blocks that are on the estimated timing-critical paths of the circuit.

The specific location of each logic block on the target FPGA is determined during the placement process [Betz99a] [Kirk83] [Marq00a] [Sech85] [Sech86] [Sech87] [Sun95] [Swar95]. A placement program assigns each logic block to an unique location to optimize delay and minimize wiring demand.

Finally, during the routing process [Betz99a] [Brow92a] [Brow92b] [Chan00] [Ebel95] [Lee61] [Swar98], a routing program is used to connect logic blocks together by determining the configuration of the programmable routing resources. The main task of all routing programs is to successfully establish all connections in a circuit using the limited amount of physical resources available on the target FPGA. The other task of the routing programs is to minimize delay by allocating fast physical connections to timing-critical paths.

Together the synthesis, technology mapping, and packing process are commonly called the front end of the FPGA CAD flow; and the placement and routing steps are commonly called the back end of the FPGA CAD flow. The remainder of this section reviews previous work on each stage of the FPGA CAD flow. In particular, several tools discussed below, including the Synopsys FPGA compiler [Syno99] for synthesis and technology mapping, the T-VPACK packer [Marq99] [Betz99a] for packing, and the VPR (Versatile Placer and Router) [Betz99a] tools for placement and routing, serve as the framework from which the CAD work described in Chapter 4, Chapter 5, and Chapter 6 is developed.

2.2.1 Synthesis and Technology Mapping

There are several commercially available synthesis tools for FPGAs, including the Synopsys *FPGA Compiler* [Syno99], Synplicity's *Synplify* [Synp03], and Altera *Quartus II* [Quar03]. In general, these tools perform both the task of synthesis and technology mapping;

however, none of these tools preserves the regularity of datapath circuits since they usually optimize across the boundaries of bit-slices. These cross-boundary optimizations often destroy the regularity of datapath circuits. This section first describes the various features of the Synopsys FPGA Compiler [Syno99], which is used as a part of a datapath-oriented synthesis flow built for the MB-FPGA architecture. Then previous research on datapath-oriented synthesis is reviewed in detail.

2.2.1.1 Synopsys FPGA Compiler

The Synopsys FPGA Compiler performs a combination of synthesis and technology mapping. The input to the compiler consists of three files including a circuit description file, an architectural description file, and a compiler script file. The circuit description file describes the behavior of the circuit that is to be synthesized. The format of the file can be in either Verilog, VHDL, or several other high-level or low-level hardware description languages.

The architectural description file describes the properties of two fundamental FPGA building blocks that the input circuit is to be mapped into, the LUTs and the flip-flops. The description includes parameters describing various delay and area properties of each building block. The LUTs are combinational circuit elements each with several inputs and one output. A LUT can be used to implement any single output Boolean function that has the same number of inputs as the LUT. The flip-flops, on the other hand, are used to implement sequential circuit elements.

The compiler script file gives specific compile-time instructions to the FPGA compiler. It can be used to set up various synthesis boundaries in the input circuit so that circuit elements will not be merged across these boundaries during the synthesis and the technology mapping process. In this research, this feature is used to preserve datapath regularity; and it is described in more detail in Chapter 4.

The final output of the Synopsys FPGA compiler is a network of LUTs and flip-flops that implements the exact functionality of the input circuit. The compiler can output the final result in a variety of file formats including the Verilog and the VHDL formats.

2.2.1.2 Datapath-Oriented Synthesis

Datapath-oriented synthesis techniques can be roughly classified into four categories including hard-boundary hierarchical synthesis, template mapping [Call98] [Cora96] [Nase94] [Nase98], module compaction [Koch96a] [Koch96b], and the regularity preserving logic transformation algorithm [Kutz00a] [Kutz00b]. Note that most of these algorithms were primarily developed to speed up the development cycle (tool runtime) of their applications; and they often pay little attention to area optimization.

Hard-boundary hierarchical synthesis is the simplest form of regularity preserving synthesis. It preserves datapath regularity by performing optimizations strictly within the boundaries of user-defined bit-slices. However, as will be shown in Chapter 4, this method suffers from the problem of high area inflation when compared to conventional synthesis algorithms that do not preserve datapath regularity.

Template mapping [Call98] [Cora96] [Nase94] [Nase98] attempts to reduce the area inflation of the hard-boundary hierarchical synthesis by mapping the input datapath onto a set of predefined templates. These templates are datapath circuits that have been designed to be very area efficient. In theory, if one can define an arbitrarily large datapath template library and has an unlimited amount of time to reconstruct the input datapath circuits out of these templates, one can achieve excellent area efficiency. However, in real life, limited by a reasonably sized datapath template library and limited computing time, the template mapping algorithm also performs poorly in terms of area efficiency and can have over 48% area inflation [Cora96].

Module compaction [Koch96a] [Koch96b] takes one step further. It merges some of the user-defined bit-slices into larger bit-slices while still preserving the regularity of datapath circuits. This algorithm is modified in Chapter 4 into an very area efficient datapath-oriented synthesis algorithm when complemented with several extra optimization steps. Without these optimization steps, however, the area efficiency of the module compaction algorithm as proposed in [Koch96a] [Koch96b] is still quite poor. For example, the algorithm discussed in [Koch96b] has an area inflation of on the order of 17%.

Finally, the regularity preserving logic transformation algorithm [Kutz00a] [Kutz00b] takes an entirely different approach to datapath-oriented synthesis. Instead of preserving user-defined regularity, it tries to extract regularity from flattened datapath logic. As a result, although it is effective in area optimization, its effectiveness, in preserving datapath regularity, is limited by the amount of regularity that can be discovered by the extraction process.

2.2.2 Packing

All existing packing algorithms place LUTs and flip-flops into FPGA logic blocks. Each logic block has a fixed capacity, which is determined by the number of LUTs and flip-flops that the logic block contains and the available number of unique logic block inputs and outputs. The VPACK algorithm [Betz97a] tries to maximize the number of LUTs that can be packed into a logic block by grouping highly connected LUTs together. The T-VPACK algorithm [Marq99] improves upon the VPACK algorithm by using the timing information on top of the connectivity information. Other packing algorithms, including RPACK and T-RPACK [Bozo01], further improve upon the VPACK and the T-VPACK algorithms by using routability information on top of the connectivity and timing information. Note that all four packing algorithms assume a fully connected logic cluster architecture, which is described in detail in Section 2.3.1.1. Furthermore, during the packing process each packing algorithm considers individual LUTs or DFFs in isolation. As a result, none of these algorithms preserves the regularity of the datapath circuits during the packing process.

2.2.3 Placement and Routing

This section gives an overview of the VPR placement and routing tools [Betz99a], which serve as the basis for the MB-FPGA placement and routing software and algorithms described in Chapter 6. The VPR placer is based on the *simulated annealing* algorithm [Kirk83] [Sech85], while the VPR router is a *negotiation-based* router [Ebel95]. Note that simulated annealing based algorithms [Betz99a] [Kirk83] [Marq00a] [Sech85] [Sech86] [Sech87] [Sun95] [Swar95] are one of the most widely used types of placement algorithms for FPGAs, while many FPGA routing algorithms are negotiation-based routers [Betz99a] [Chan00] [Ebel95] [Lee61] [Swar98]. None of the existing placement [Betz99a] [Kirk83] [Marq00a] [Sech85] [Sech86] [Sech87] [Sun95] [Swar95] and routing algorithms [Betz99a] [Brow92a] [Brow92b] [Chan00] [Ebel95] [Lee61] [Swar98] preserves the regularity of datapath circuits. For placement, regularity is destroyed by existing placers, which only incrementally improve the placement of individual logic blocks. Routers, on the other hand, also destroy the regularity information as they only route one net at a time.

2.2.3.1 VPR Placer and Router

The VPR placer and the VPR router are contained in a single computer program. The input to the program consists of two files, a circuit description file and an architectural description file. The circuit description file describes a network of logic blocks that is to be implemented on an FPGA. The architectural description file specifies the detailed architecture of the

FPGA. The architectural choices in the architectural description file are limited to the variants of the logic cluster based FPGA architecture described in Section 2.3.1.

The VPR Placer

The VPR placer performs placement using the simulated annealing algorithm [Kirk83] [Sech85]. It first places each logic block randomly onto an unoccupied location on the FPGA. It then moves two logic blocks by swapping their physical locations or moves a logic block into a location that is not occupied by any other logic blocks. After each move, the algorithm either keeps the move or discards the move by comparing the placement before the move with the placement after the move using a set of metrics. These metrics represent an estimation of how easily a particular placement can be routed and the achievable speed of the placement after routing. Usually the optimization strategy chooses a placement with a metric indicating easier routing or better speed. But occasionally, the algorithm chooses the opposite in the hope that a bad placement choice can lead to a very good one in subsequent moves.

A key metric in simulated annealing is called the *annealing temperature.* At the start of a placement process, the temperature is set at a very high value. Throughout the placement process, the temperature is gradually lowered to zero. At high temperatures, the optimization strategy will be more likely to choose a bad move; while at low temperatures, fewer bad moves are accepted by the algorithm. Finally at zero temperature, only good moves are accepted.

The VPR Router

The VPR router takes the output of the VPR placer as its input. The input describes a network of logic blocks whose physical locations are determined. The same architectural file that the placer uses also specifies the routing architecture for the router. Recall that the fundamental goal of the routing tool is to successfully connect all the nets through the routing network and to meet the timing constraints of the most timing-critical connections.

Since each physical routing resource can only be used by a single net at a time, the best connection choices for individual nets might conflict with each other. The VPR router uses the negotiation-based approach of the Pathfinder routing algorithm [Elbe95] to resolve these routing conflicts. It connects the logic blocks together through several routing iterations. During each iteration, the router completely routes the entire circuit; and except during the final iteration, each physical routing resource are allowed to be used by several nets at a time. The overuse is called *congestion* and the over-used routing resources are called *congested resources*.

During each iteration, the router connects one net at a time using the maze routing algorithm [Lee61]. For each net, the routing process is guided by a set of metrics that are based on the delay of the net and the congestion of the routing resources from all the previous routing iterations. These metrics are updated after each routing iteration to make already congested resources more costly to use as time progresses. The nets compete for congested resources based on these metrics. When a net is more timing-critical or has no other alternatives, it is given priority for the routing resources that it prefers. When a net is not timing-critical or has other equally good alternatives, it is forced to give up the congested resource that it occupies.

2.3 FPGA Architectures

This section provides a detailed description of two FPGA architectures proposed in previous FPGA studies including a conventional FPGA architecture described in [Betz99a] and a datapath-oriented FPGA architecture described in [Cher96]. These two architectures have been chosen because of their influence on the MB-FPGA architecture proposed in Chapter 3. Each FPGA is described in terms of its logic block architecture, its routing architecture, and its CAD flow. Enough details are given, and in some cases specific comments are made, to show

how specific architectural features of these FPGAs relate to the research described in this dissertation. Following these detailed architectural descriptions, several existing datapath-oriented architectures are briefly described. The section is concluded by a brief review of the various techniques used for modeling FPGA delay and area throughout this work.

2.3.1 A Conventional FPGA Architecture

The overall structure of the conventional FPGA architecture proposed in [Betz99a] is shown in Figure 2.2. It consists of a two-dimensional array of programmable logic blocks, called *logic clusters*, with horizontal routing channels between rows of logic blocks and vertical routing channels between columns of logic blocks. At the periphery of the architecture are the I/O blocks, which bring signals from outside into the architecture and send signals generated inside the architecture to the outside. At the intersection of a horizontal routing channel and a vertical routing channel is a switch block, which provides programmable connectivity between the horizontal and vertical channels. This architecture was developed by Betz et. al. as the base architecture for the development of the T-VPACK and the VPR tools described in Section 2.2. The architecture has been used in many architectural studies including [Betz97a] [Betz97b] [Betz98] [Betz99a] [Betz99b] [Betz00] [Chen03] [Cong03] [Harr02] [Li03] [Lin03] [Marq99] [Marq00a] [Marq00b] [Sank99] [Swar98] [Tess02] [Varg99]. In this thesis, the architecture is used as a comparison architecture for the experimental results presented Chapter 5. The structure of each architectural component is described in great detail in [Betz99a], so each of these components is described in turn.

2.3.1.1 Logic Clusters

The structure of a logic cluster, illustrated in Figure 2.3, consists of a set of cluster inputs, a set of cluster outputs, and several tightly connected Basic Logic Elements (BLEs). The out-

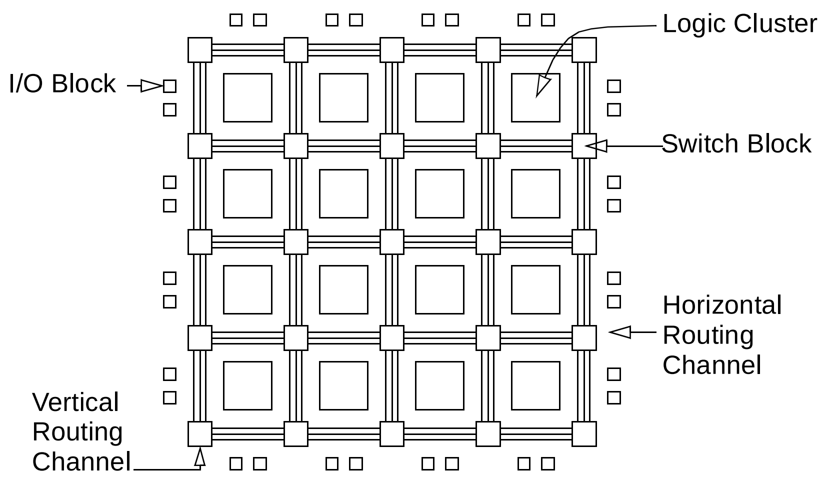


Figure 2.2: Overview of FPGA Architecture Described in [Betz99a]

puts of the logic cluster are directly connected to the outputs of the corresponding BLEs. The network that connects all the BLEs within a cluster is called the local routing network.

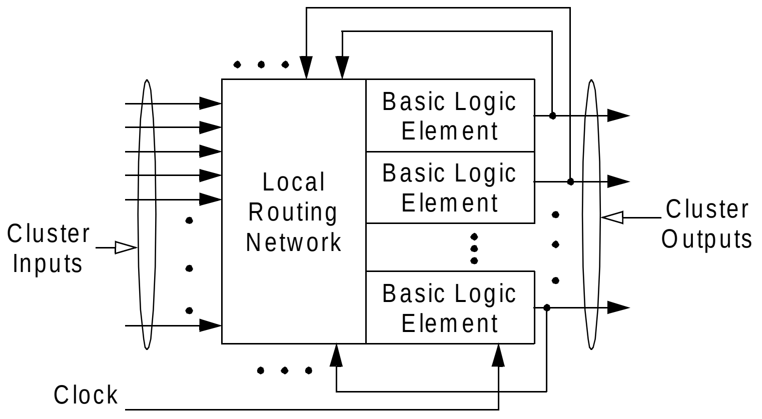


Figure 2.3: Logic Cluster

The detailed structure of a BLE is shown in Figure 2.4. It consists of a LUT, a D-type Flip-Flop (DFF), and a multiplexer. The LUT output is feed into the DFF input. The multiplexer is controlled by a Static Random Access Memory (SRAM) cell and is used to choose either the LUT output or the DFF output as the output of the BLE. The input of the BLE consists of inputs to the LUT and the clock input to the DFF.

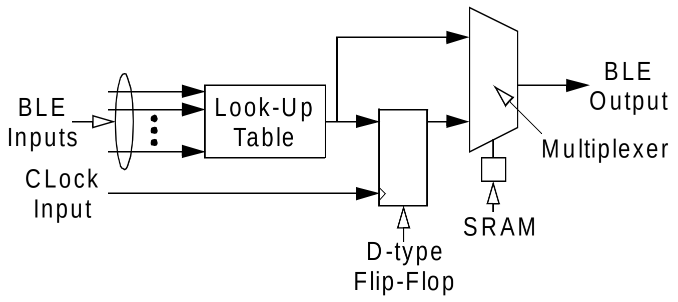


Figure 2.4: Basic Logic Element

The detailed structure of a LUT is shown in Figure 2.5. The LUT has *K* inputs and one output where *K* is specified as an architectural parameter of the architecture. It can be programmed to implement any *K*-input logic function. The LUT is implemented as a multiplexer whose select lines are the LUT inputs. These inputs select a signal from the outputs of 2^K SRAM cells to generate the LUT output.

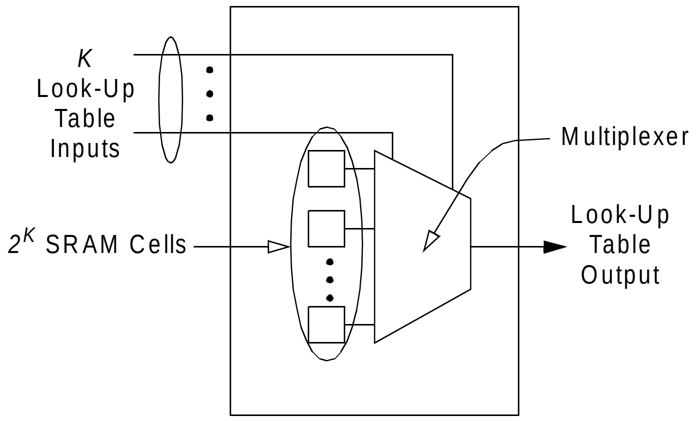


Figure 2.5: Look-Up Table

Local Routing Network

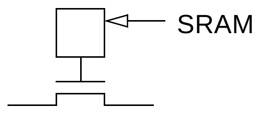
The inputs to a local routing network, as shown in Figure 2.3, consist of two types of signals. The first type is an input to the logic cluster. The second type is an output of a BLE in the cluster. Each cluster input or each BLE output connects to exactly one input of the local routing network. The outputs of the local routing network are connected to the BLE inputs; and there is exactly one network output for every BLE input.

The local routing network has a fully connected topology. Each output of the network can be connected to any input of the network. The topology is widely used in many subsequent FPGA studies including [Marq00b]. The topology also has the advantage of reducing the complexity of the packing tools [Betz99a] since any network input can be connected to any LUT input. Note that commercial devices including Virtex [Xili02], Stratix [Alte02], and Cyclone [Alte02], typically use a depopulated local routing network structure [Lemi01], which uses less area, but requires more complex packing tools.

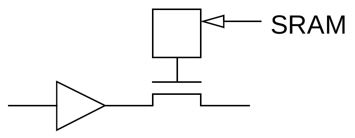
2.3.1.2 Routing Switches

Programmable switches, called routing switches, provide reconfigurable connectivity throughout the architecture. The architecture uses two types of routing switches, the pass transistor switch and the buffered switch. As illustrated in Figure 2.6a, a pass transistor switch consists of a single pass transistor controlled by an SRAM cell. The switch is bi-directional which allows electrical current to flow from either end of the switch to another.

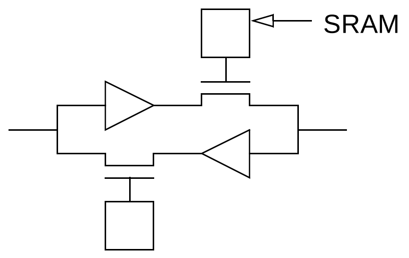
A buffered switch, shown in Figure 2.6b, consists of a buffer, a pass transistor, and an SRAM cell. Since buffers only allow electrical current to flow in one direction, buffered switches are uni-directional. A bi-directional switch can be built out of two buffered switches using the configuration shown in Figure 2.6c. Comparing the two types of switches, pass transistor switches are much smaller in size while buffered switches provide more driving strength and regenerate their input signals. Since pass transistor switches do not regenerate their input signals, the *RC* time constant of a signal grows quadratically as a function of the number of pass transistor switches that the signal passes and the total length of the wire [Betz99a]. As a



(a) Pass Transistor Switch



(b) Buffered Switch



(c) Bi-Directional Buffered Switch

Figure 2.6: Routing Switches

result, the pass transistor switches are much slower than the buffered switches for connecting long signal connections.

The FPGA architecture uses a technique called buffer sharing to save the implementation area of buffered switches. The technique shares a common buffer among several buffered switches that originate from a common source. An example is shown in Figure 2.7. In the figure, a source is connected to three sinks through three buffered switches. Without buffer sharing, three separate buffers are needed. With buffer sharing, only one buffer is used.

2.3.1.3 Routing Channels

As illustrated in Figure 2.8, each routing channel of the architecture consists of wire segments and routing switches. Note that for clarity only one horizontal routing channel is shown in the figure. The vertical channels that intersect the horizontal channel are not illustrated. The

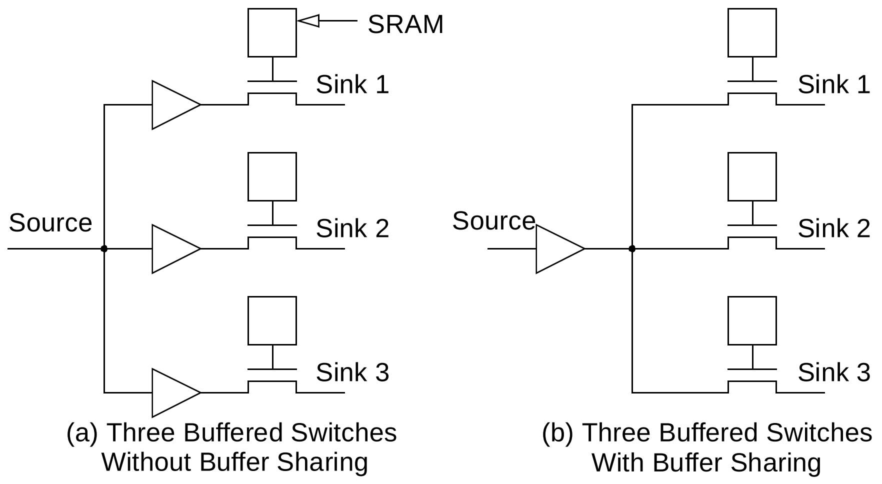

SRAM
Sink 1
Sink 2
Sink 3
Source
(a) Three Buffered Switches Without Buffer Sharing
Sink 1
Sink 2
Sink 3
Source
(b) Three Buffered Switches With Buffer Sharing

Figure 2.7: Buffer Sharing

number and types of wire segments and routing switches in each channel are specified as architectural parameters of the architecture. A wire segment starts at one switch block, spans several logic blocks, and ends at another switch block. The number of logic blocks that the segment spans is called the logical length of the segment. Routing switches are located in the switch blocks. They connect wire segments together to form one continuous track, called a routing track, that spans the entire length of the routing channel. In Figure 2.8, three routing tracks are illustrated. The top track contains wire segments of logical length one. The middle track contains wire segments of logical length two; and the bottom track contains wire segments of logical length four.

The choice of wire segment lengths is important to the overall performance of the architecture. Long wire segments are valuable for implementing signals that connect two far away logic blocks. By using long wire segments, a router can reduce the number of routing switches used to implement these long connections, and consequently reduce the delay of these connections. Appropriate combination of segment lengths also can be used to increase the logic density of the FPGA architecture. By evenly matched the segment lengths with net lengths, the

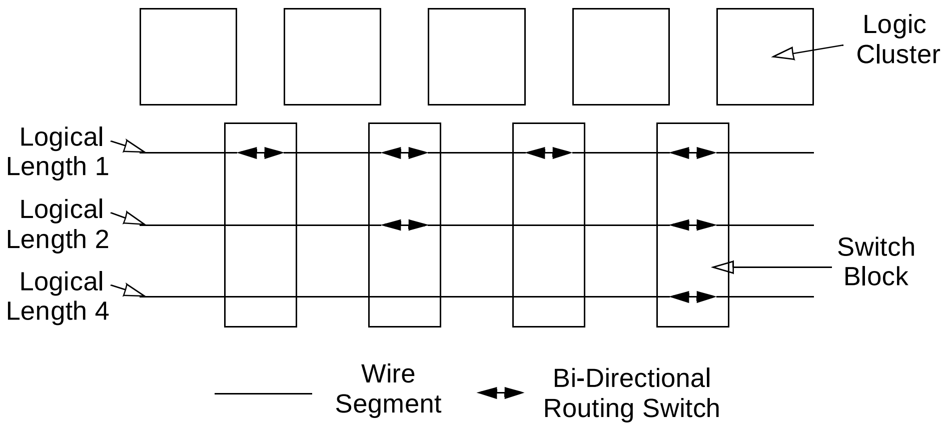


Figure 2.8: Routing Channel

total number of routing switches in any particular implementation of the architecture can be effectively reduced; and consequently the logic density of the architecture can be increased.

As shown in Figure 2.9, the starting positions of the wire segments with the same length are staggered in order to ease the physical layout of the architecture. With staggered starting positions, routing tracks in Figure 2.9 can be rearranged into the topology shown in Figure 2.10 to create identical tiles, each containing one logic block and its neighboring routing resources. With a tile based architecture, the physical layout of FPGAs can be greatly simplified. Instead of designing the layout of an entire FPGA chip, only the layout of one single tile has to be designed. The tile then can be duplicated along a two-dimensional array to create a complete FPGA layout. Note that for clarity only one horizontal routing channel is illustrated in Figure 2.9 and Figure 2.10; nevertheless, the same design principle applies for architectures with both horizontal and vertical routing channels.

2.3.1.4 Switch Blocks

A switch block consists of all the programmable switches located at the intersection of a horizontal routing channel and a vertical routing channel. The FPGA architecture described in [Betz99a] is designed with two types of switch blocks. One type is based on the disjoint topol-

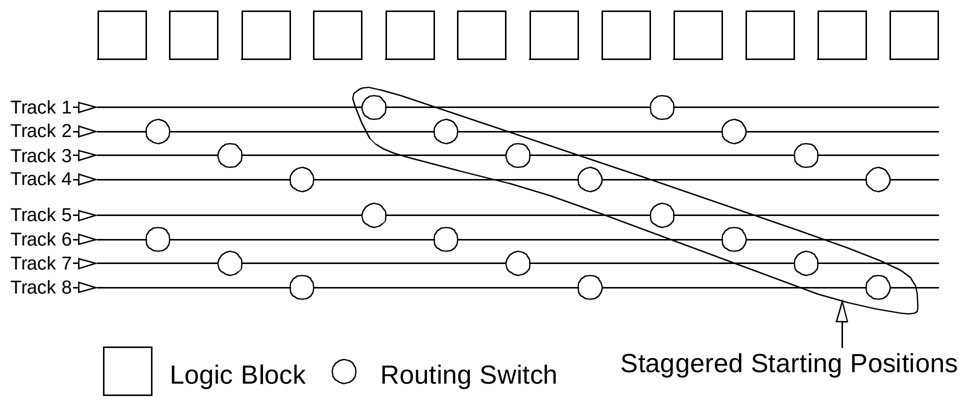


Figure 2.9: Staggered Wire Segments

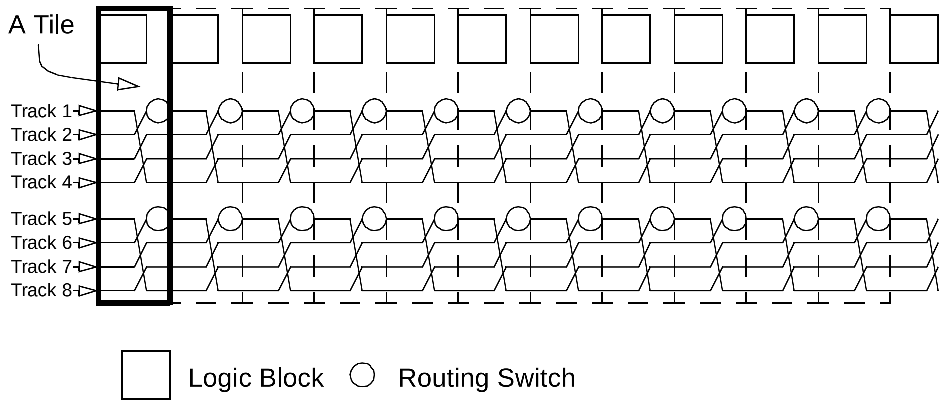


Figure 2.10: Tiles

ogy [Hsei90] and the other is based on the Wilton topology [Wilt97]. The disjoint topology is more popular and is described here. Note that other newer switch block topologies such as the Imran topology [Masu99] can also be used with the architecture; although they are not further discussed in this work.

The disjoint topology assumes that all routing channels contain the same number of routing tracks. For two intersecting channels, every track in the horizontal channel is connected to the same track number in the vertical channel by routing switches. The architecture described in [Betz99a] further assumes that the two connecting tracks must have the same segment length.

There are two configurations for two tracks that are connected at a switch block to intersect. In the first configuration shown in Figure 2.11a, both the horizontal segments and the vertical segments end at the switch block. This configuration uses six bi-directional switches to connect the segments together. Two of the six switches are part of the horizontal routing track or the vertical routing track. The remaining four switches are used to connect the horizontal track to the vertical track. In the second configuration, shown in Figure 2.11b, neither the horizontal segment nor the vertical segment ends at the switch block. For this configuration, only one bi-directional switch is needed to connect the horizontal and vertical tracks together. The wire segments are distributed in such a way so that the two connecting tracks can never intersect in the configuration shown in Figure 2.11c, where segments on one track end at the switch block, while the segment on the other track does not.

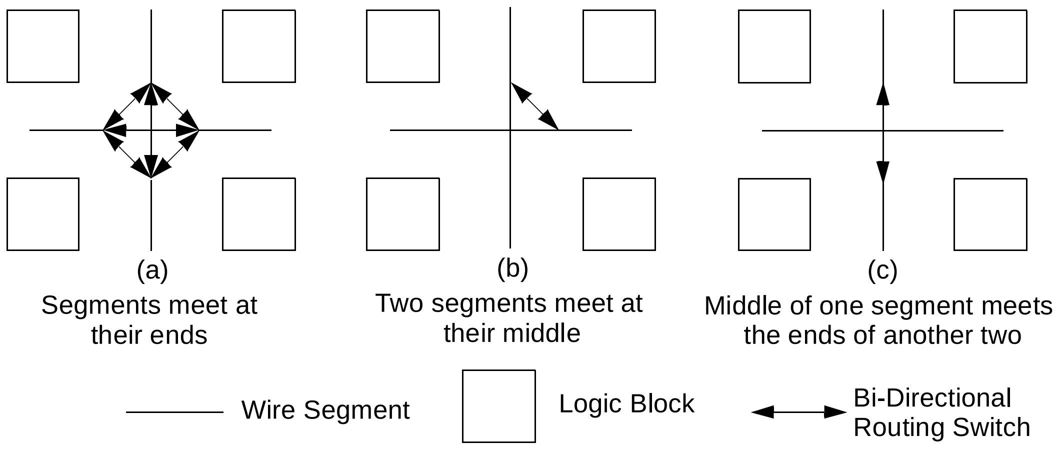

(a)
Segments meet at their ends
(b)
Two segments meet at their middle
(c)
Middle of one segment meets the ends of another two
Wire Segment
Logic Block
Bi-Directional Routing Switch

Figure 2.11: Different Topologies of A Horizontal Track Meeting A Vertical Track

2.3.1.5 Input and Output Connection Blocks

Logic clusters are connected to its neighboring routing channels through connection blocks. A collection of switches that connect all the inputs of a logic cluster to a routing channel is called an input connection block, while a collection of switches that connect all the outputs of a logic cluster to a routing channel is called an output connection block.

A portion of an input connection block that connect a logic cluster input to a neighboring routing channel is shown in Figure 2.12. It is implemented using a multiplexer. The output of the multiplexer is connected to the input of the logic cluster. The inputs of the multiplexer are connected to a set of routing tracks. Isolation buffers are used to electrically isolate the multiplexer inputs from the routing tracks [Betz99a], shielding the capacitance of the tracks from the input multiplexers.

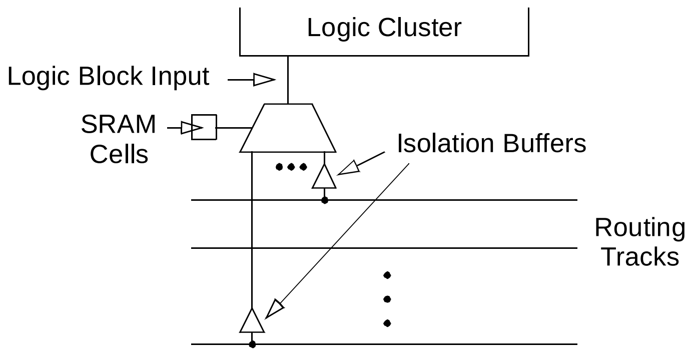


Figure 2.12: Input Connection Block

A portion of an output connection block that connects a logic cluster output to a neighboring routing channel is shown in Figure 2.13. It is implemented by connecting the logic cluster output through a shared driving buffer and dedicated pass transistors to a set of routing tracks. The configuration of both the input connection blocks and the output connection blocks are controlled by SRAM cells.

2.3.1.6 I/O Blocks

In the FPGA architecture described in [Betz99a], each I/O block contains an input pin for bringing signals into the FPGA and an output pin for sending internal signals to the outside of the FPGA. Both the input pin and the output pin are connected to the routing channels through

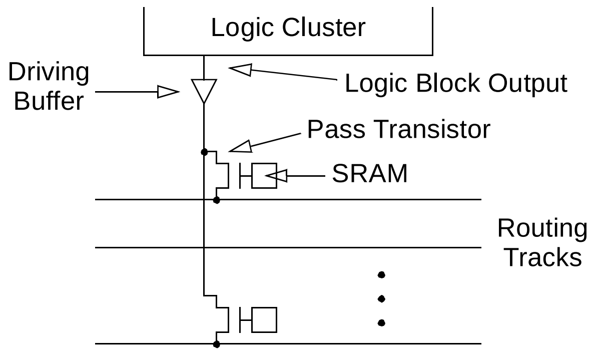


Figure 2.13: Output Connection Block

the same output connection block and the same input connection block that are discussed in Section 2.3.1.5, respectively.

2.3.2 DP-FPGA — A Datapath-Oriented FPGA Architecture

The work of this thesis is based on the DP-FPGA architecture described in [Cher94] [Cher96] [Cher97]. The overall structure of the DP-FPGA [Cher96] is shown in Figure 2.14. It consists of three high-level blocks including the memory block, the control block, and the datapath block. The memory block consists of banks of SRAM. It can be configured to implement memory systems of different width and depth. Of the two remaining blocks, the function of the control block is to implement non-datapath circuits while the datapath block is designed specifically for implementing datapath. The exact structure of the memory block and the control block was not specified in detail by the Cherepacha study. The study also did not specify how each block should be connected to the other two blocks. The focus of the Cherepacha study was on the architecture of the datapath block, which is described in detail below.

2.3.2.1 Overview of the Datapath Block

The datapath block of DP-FPGA is one of the first FPGA architectures that use the technique of configuration memory sharing (called *programming-bit sharing* in the DP-FPGA terminology [Cher96]) to create both the CMS routing resources and the CMS logic blocks. The

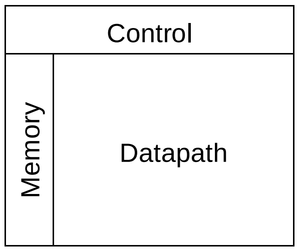


Figure 2.14: Overview of DP-FPGA Architecture

technique creates CMS resources by sharing a single set of configuration memory among several programmable resources. By sharing, the amount of configuration memory that is need to control the programmable resources is reduced and consequently, the implementation area of datapath applications, which contain a large amount of identical bit-slices, are minimized. The Cherepacha study demonstrates that there can be significant savings in logic block area when CMS logic blocks are used instead of conventional logic blocks for implementing datapath circuits. The effectiveness of the CMS routing resources, which account for the majority of FPGA area, however, was not investigated in detail by this work.

The Cherepacha study also only specified a subset of the datapath block architecture. These architectural features are described here. Because of the incompleteness in architectural specification, no CAD flow was ever designed for the DP-FPGA architecture. The overall structure of the datapath block is shown in Figure 2.15. Like the conventional FPGA architecture described in [Betz99a], it also consists of a two-dimensional array of logic blocks with horizontal routing channels between rows of logic blocks and vertical routing channels between columns. Each routing channel consists of two separate sub-channels. One, called data sub-channel, is designed to carry multiple-bit wide data. The other, called control sub-channel, is designed to carry one-bit wide data. The data sub-channels contain more routing tracks in the horizontal direction than in the vertical direction. The control sub-channels, on the other hand, contain more routing tracks in the vertical direction.

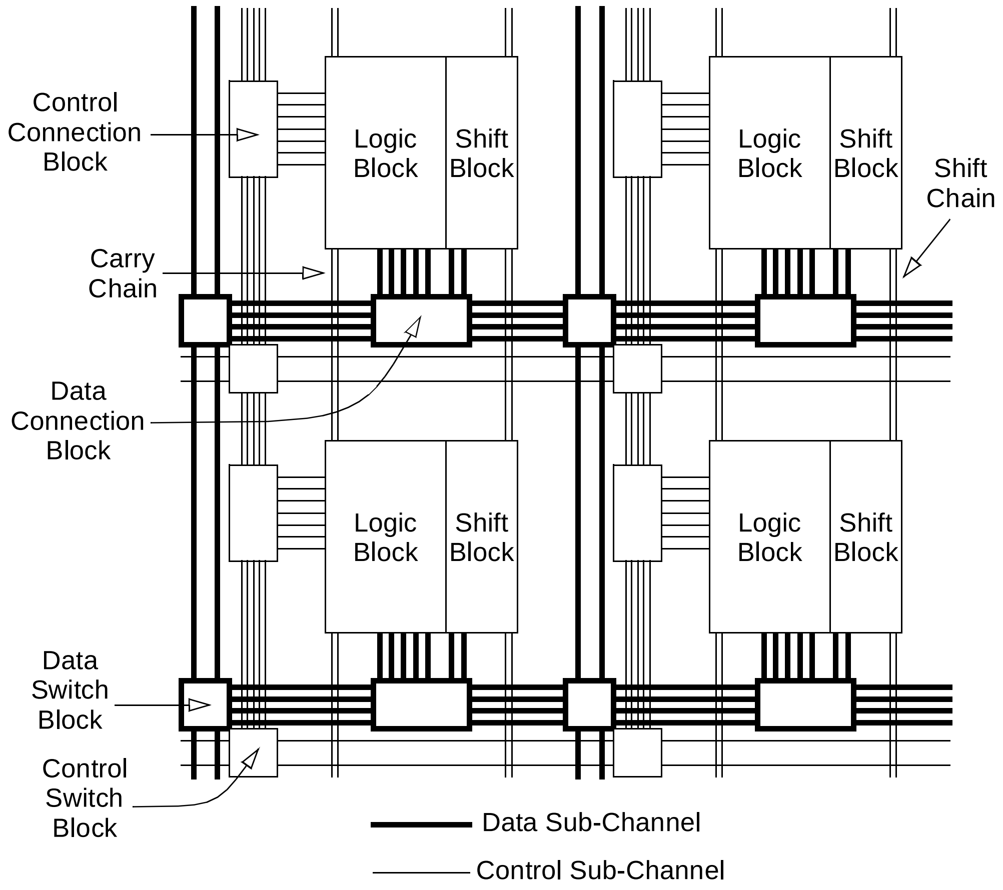


Figure 2.15: Overview of Datapath Block

The datapath block contains two types of switch blocks — the data switch blocks, which provide connectivity for the data sub-channels, and the control switch blocks, which provide connectivity for the control sub-channels. The Cherepacha study did not specify the exact topology of these switch blocks. Neither did it specify the exact topology of the control connection blocks, which connect the logic blocks to the control sub-channel. The Cherepacha study did specify in detail the structure of the logic blocks, shift blocks, and data connection blocks. It finds, with a few simplifying assumptions including the assumption that all transistors are minimum width, and without knowing the exact number of routing tracks per channel, that collectively these components can potentially double the area efficiency of the corresponding conventional components for datapath [Cher96]. Each of these architectural components is described in turn.

2.3.2.2 Arithmetic Look-Up Tables

The main building blocks of a DP-FPGA logic block are arithmetic LUTs. Structurally, these LUTs are more complex than the conventional LUTs used in the conventional architecture described in Section 2.3.1. An arithmetic LUT, shown in Figure 2.16, consists of *n* inputs, two outputs, two conventional LUTs each with *n*-1 inputs, two two-input multiplexers and an SRAM cell. The LUT has two modes of operations, the normal mode and the arithmetic mode, which are controlled by the SRAM cell. When the SRAM cell is set to be one, the arithmetic LUT is in the normal mode of operation. In this mode, it behaves as a conventional LUT with *n*-inputs and one output. The LUT output is presented on the output P shown in Figure 2.16 and the output G is ignored. When the SRAM cell is set to be zero, the LUT is in the arithmetic mode of operation. In this mode, one conventional LUT in the arithmetic LUT is used to generate the output at P, which is used as a propagate signal of a carry look ahead adder. The other conventional LUT in the arithmetic LUT is used to generate the output at G, which is used as a generate signal of a carry look ahead adder.

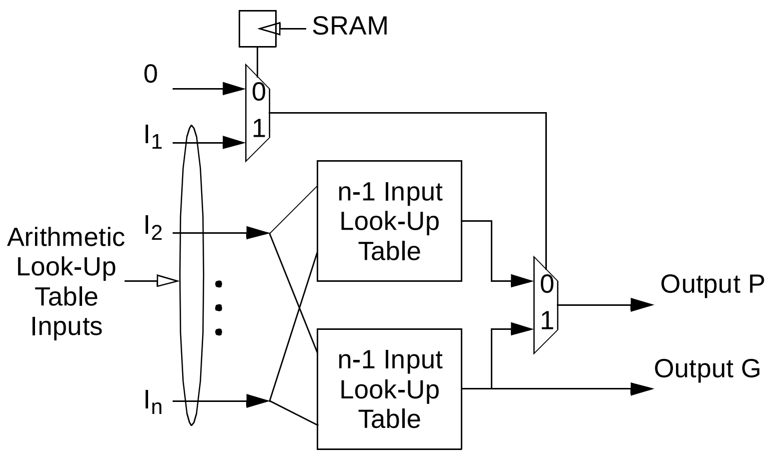


Figure 2.16: Arithmetic Look-Up Table

2.3.2.3 Logic Blocks

The connectivity of a DP-FPGA logic block is shown in Figure 2.17. The logic block has several input signals, called data inputs, and four output signals, called data outputs. The data inputs are connected to a neighboring data sub-channel through a data connection block, called the input data connection block. The data outputs are connected to the same sub-channel through another data connection block, called the output data connection block. The data outputs are further connected to a shift block whose outputs are connected to the data sub-channel. The logic block also has connections to the logic blocks above and below through carry signals.

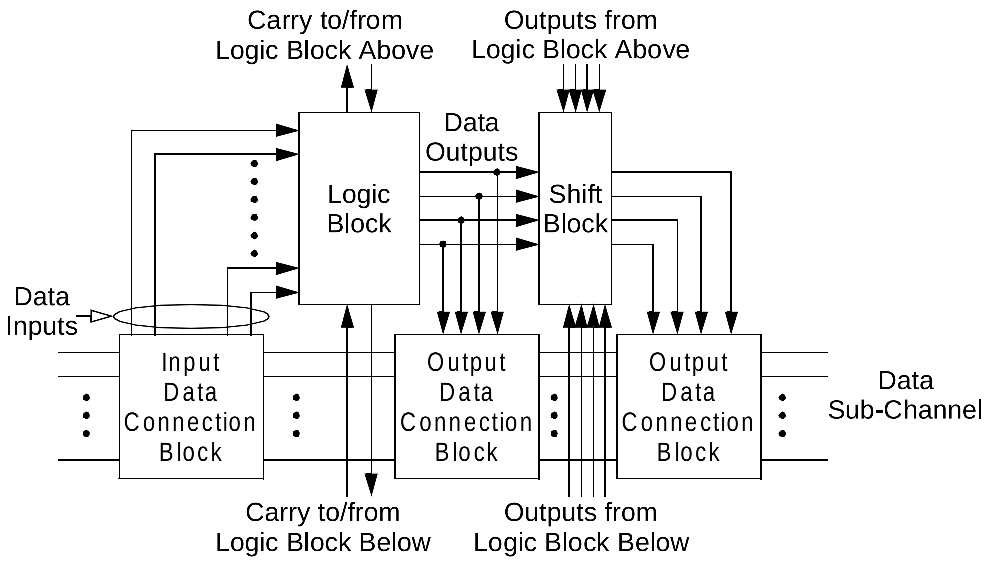

Carry to/from Logic Block Above
Outputs from Logic Block Above
Data Outputs
Logic Block
Shift Block
Data Inputs
Input Data Connection Block
Output Data Connection Block
Output Data Connection Block
Data Sub-Channel
Carry to/from Logic Block Below
Outputs from Logic Block Below

Figure 2.17: Logic Block Connectivity

The internal structure of the logic block is shown in Figure 2.18. It consists of four arithmetic LUTs, one carry block, four flip-flops, and four two-input multiplexers. All four arithmetic LUTs have the same number of inputs. Each LUT input is connected to a unique data input of the logic block. All four LUTs share a single set of configuration memory and are identically configured at all times. The SRAM cell that controls the operation mode of the arithmetic LUTs is also shared across all four LUTs.

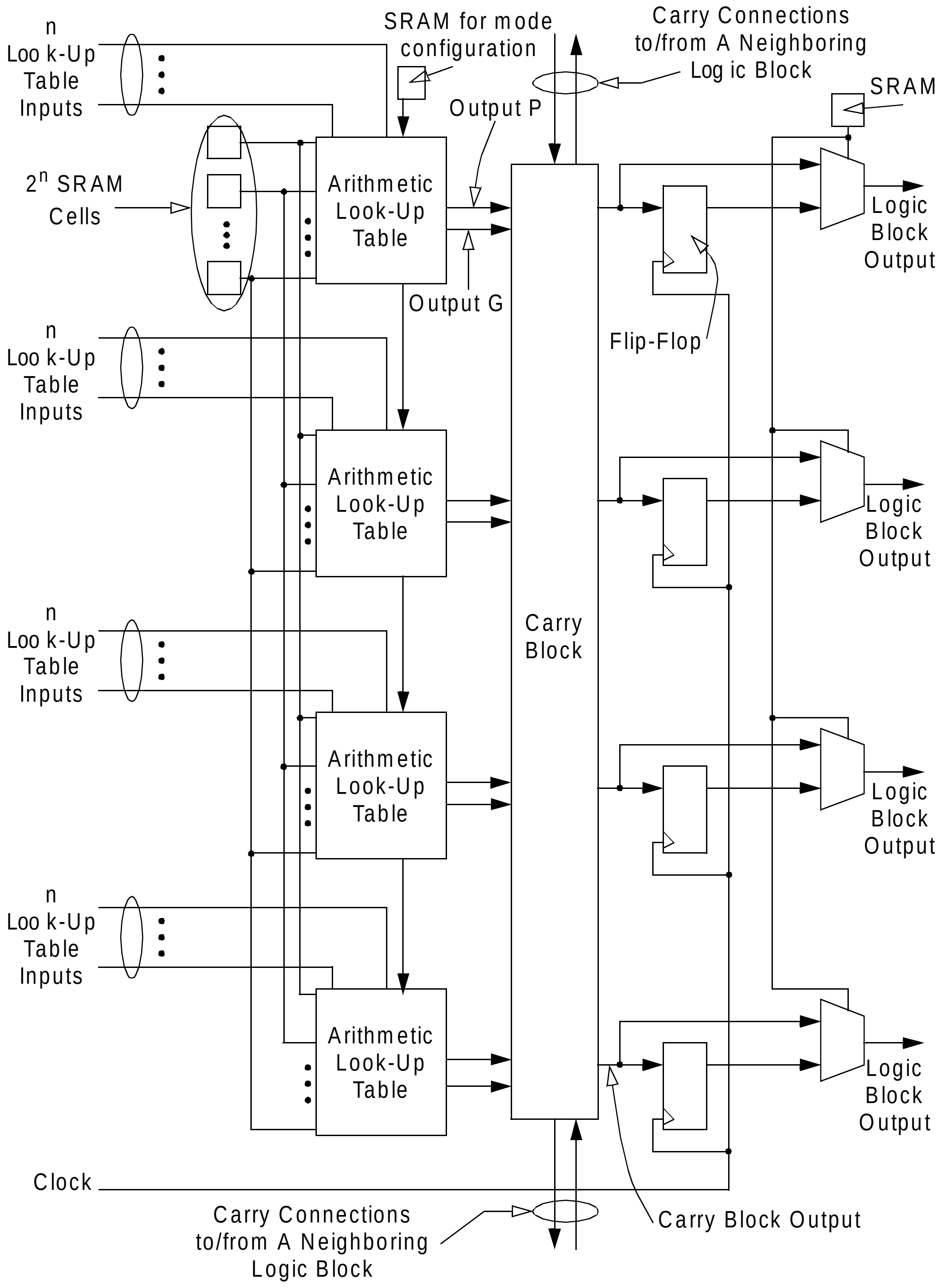


Figure 2.18: DP-FPGA Logic Block

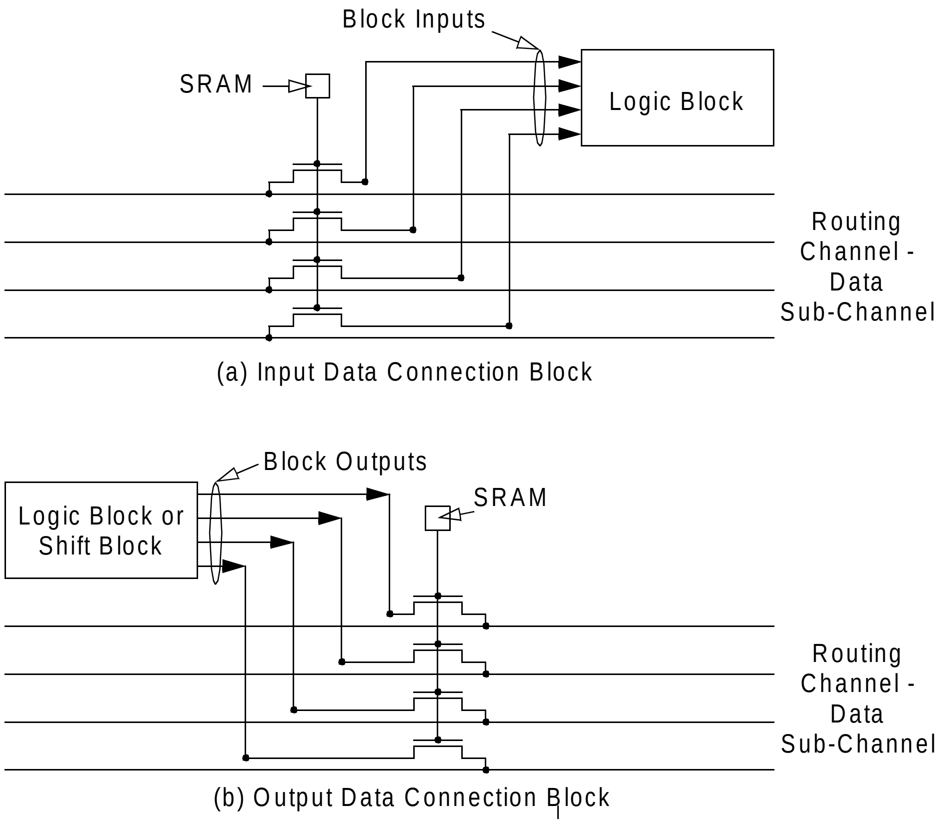
The outputs of the LUTs are fed to the carry block. When the LUTs are in the normal mode of operation, the P outputs are directly connected to the corresponding carry block outputs. When the LUTs are in the arithmetic mode of operation, each arithmetic LUT behaves as a bit-slice of a carry look ahead adder. The carry block generates carry signals based on the P outputs and the G outputs of the arithmetic LUTs. In the arithmetic mode, each output of the carry block represents a bit of the sum output of a carry look ahead adder.

Each carry block output is connected to a flip-flop input. Each two-input multiplexers is used to select either an carry block output or the corresponding flip-flop output to produce a logic block output. Note that all four two-input multiplexers also share a single SRAM bit that stores their programmable configuration.

Data Connection Blocks

There are two types of data connection blocks. One type, called the input data connection block, connects the logic block inputs to the data sub-channel. The other type, called the output data connection block, connects the logic block outputs or the shift block outputs to the data sub-channel. Both connection blocks are CMS routing resources, which uses configuration memory sharing to increase their logic density. A portion of the input data connection block is shown in Figure 2.19a. It connects four logic block inputs to four routing tracks in the data sub-channel through four pass transistors. The four pass transistors share a single configuration SRAM bit.

A portion of the output data connection block is shown in Figure 2.19b. It connects the four logic block outputs to four routing tracks in a data sub-channel through four pass transistors. Again these four pass transistors is controlled by a single bit of SRAM.



Block Inputs
SRAM
Logic Block
Routing Channel - Data Sub-Channel
(a) Input Data Connection Block
Block Outputs
Logic Block or Shift Block
SRAM
Routing Channel - Data Sub-Channel
(b) Output Data Connection Block

Figure 2.19: Data Connection Block

Shift Blocks

The shift block, shown in Figure 2.17, is a barrel shifter and is design to perform arithmetic and logical shift operations, which are commonly found in arithmetic applications, for multiple-bit wide data. The block is necessary for the DP-FPGA architecture due to the limited connectivity provided by the data connection blocks shown above. Without the shift blocks, the DP-FPGA architecture is only capable of performing coarse-grain shift operations in increments of four [Cher96]. Such limitation can greatly reduce the usefulness of the architecture.

To accommodate all possible shift operations, the shift block is elaborately designed. It can either left shift or right shift the output of the logic block and presents the shifted data at its output. The output is then connected to the data sub-channel through the output data connection block. For each shift operation, new data can be shifted in from several sources including the outputs of the logic blocks above and below, constant 0s, and constant 1s.

Several shift blocks can be used to shift multiple-bit wide data generated by multiple logic blocks when these logic blocks are physically placed adjacent to one another. The placement is necessary in order to allow data to be cascaded from one shift block to another through the *outputs from logic block above* and the *outputs from logic block below* connections shown in Figure 2.17.

2.3.4 Other Datapath-Oriented Field-Programmable Architectures

Having described in detail the FPGA architectures that form the basis of this work, this section provides a general survey on the field of datapath-oriented field-programmable architectures, which are typically designed for arithmetic-intensive applications. These datapath-oriented architectures can be classified into four classes, including the processor-based architectures, the static Arithmetic Logic Unit (ALU)-based architecture, the dynamic ALU-based architectures, and the LUT-based architectures. Each of these architectural classes is described in turn; and the section is concluded by a brief review of the various datapath-oriented features employed in the current state-of-the-art commercial FPGAs.

Note that the first three architectural classes are built around arrays of small processors or ALUs, which are considerably more complex than LUTs. As a result, these devices usually have quite different routing demands and contain substantially different routing resources than the conventional FPGAs. The fourth class of architectures are more FPGA-like — each contains an array of LUT-based logic blocks and segmented routing resources. As a result, this class of devices are more closely related to the current work and can benefit the most from its results.

2.3.4.1 Processor-Based Architectures

The architectures of PADDI-1 [Chen92], PADDI-2 [Yeun93], RAW machine [Wain97], and REMARC [Taka98] all consist of an array of processors. These architectures can be said to be reconfigurable on a cycle-by-cycle basis since, whenever a processor executes a new instruction, the behavior of the processor is changed. This cycle-by-cycle reconfigurability is quite different from the conventional FPGAs whose logic blocks are configured only once — at the beginning of a computation process.

In particular, PADDI-1, PADDI-2, and REMARC all use simple 16-bit wide processors that can perform addition, subtraction, and several logical operations in hardware. Each processor also contains a 16-bit wide register file (PADDI-1 and PADDI-2) or data memory (REMARC) for storing data. Each also has enough instruction memory to store a maximum of 8 (PADDI-1 and PADDI-2) or 32 (REMARC) instructions. Note that these instructions are stored in fully decoded forms so each instruction might take up to 32 to 53-bits of storage space. The instructions are executed in an order either as indicated by a global program counter (PADDI-1 and REMARC) or as specified in the next-program-counter field of each instruction (PADDI-2).

The RAW machine is composed of an array of full-scale 32-bit wide processors, each containing a large instruction memory and data memory, as well as a register file. The ALU inside each processor can perform a variety of arithmetic and logical operations including hard-wired multiplication and division. The processor also contains a substantial amount of reconfigurable logic in the form of LUTs and flip-flops. Note that many specifics of the RAW machine including the number of instruction and data memory entries, the amount of registers in each register file, the exact structure of the ALU, and the size of the reconfigurable logic are variable architectural parameters.

The processors communicate with each other through global connection networks, which vary widely from architecture to architecture. In particular, the PADDI-1 device employs a crossbar network that connects two rows of four processors together. The connections are made in chunks of 16-bit wide buses. The PADDI-2 architecture is built upon the PADDI-1 architecture. Here eight processors are connected into a cluster using the crossbar network of PADDI-1. Sixteen clusters are then grouped into two rows of eight clusters. Between these two rows are several horizontal routing buses that run the full length of the row. The clusters are connected to the buses through their input or output pins. For better performance, each routing bus is broken into segments using programmable switches at pre-determined intervals. The buses are time-shared resources. To communicate, a processor has to use a set of pre-defined communication protocols to claim a bus. Once a bus is claimed, the communication can be bi-directional — either to or from the initiating processor.

Each REMARC device contains 64 processors in an 8 by 8 array. There is only one 16-bit wide bus running horizontally or vertically in between every two rows or two columns of processors. The communication is again time-shared and uses a set of pre-defined protocols. Note that in this architecture, unlike the conventional FPGAs, a horizontal bus does not connect to a vertical bus.

Finally, the RAW machine has an elaborate routing architecture. Similar to REMARC, its processors are arranged in an array structure. There are a number of 32-bit wide buses running horizontally or vertically in between two rows or two columns of processors. Like conventional FPGAs, at the intersection of a horizontal and vertical routing channel, there is a switch block. Unlike conventional FPGAs, however, each switch block contains a set of instruction memory (controlled by the program counter of a nearby processor) whose content controls the cycle-by-cycle connectivity of the switch block. The switch block also can perform wormhole routing of packets generated by the processors using the addresses contained in the header of each packet.

2.3.4.2 Static ALU-Based Architectures

Unlike processor-based architectures, static ALU-based architectures, including Colt [Bitt96], DReAM [Also00], and PipeRench [Gold00], do not contain instruction memory, the program counter, and their associated control logic. Instead, the configuration of each ALU is directly controlled by the configuration memory. Nevertheless, ALU-based architectures still can be rapidly reconfigured since these architectures consume significantly less configuration memory than the traditional FPGAs.

A Colt [Bitt96] logic block is called an IFU, which contains a 16-bit wide ALU and several pipeline registers. The device consists of 16 IFUs placed in a 4 by 4 array. Each IFU is connected to its immediate neighbors through a set of nearest neighbor interconnects. Two 16-bit wide inputs of each IFU located at the top row of the array and one 16-bit wide output of each IFU located at the bottom row of the array are connected together by a full crossbar (called the smart crossbar) which also provides partial connectivity to chip-level I/Os.

A DReAM [Also00] logic block is called a RPU, which contains two 8-bit wide ALUs and two banks of 8-bit wide memory. Four RPUs are grouped into a cluster. Within the cluster, RPUs communicate with each other through a set of 16-bit wide cluster-level local interconnects. Nine clusters in a 3 by 3 array form a DReAM device. The array is interconnected by a global routing network, which is similar in topology to a conventional FPGA global routing network. As in conventional FPGAs, the horizontal and vertical routing channels are connected together by switch blocks at their intersections. Unlike conventional FPGAs, however, the routing tracks are grouped into 16-bit wide buses; and the RPUs communicate across these buses through a set of pre-defined communication protocols. Note that beside the RPUs, each

DReAM device also contains a global communication unit whose function and structure is beyond the scope of this work.

Each ALU-based logic block of PipeRench [Gold00] is called a stripe, which contains sixteen 8-bit wide ALUs and a set of registers. Stripes in a PipeRench device are vertically stacked; and the physical routing network of the device only provides connectivity between two adjacent stripes. Communication across distant stripes is achieved through rapid reconfiguration and by storing data in the internal registers of a stripe. The technique, called virtual global connection, is described in more detail in [Gold00]. Although essential to the structure of PipeRench, the technique cannot be readily applied to the traditional FPGAs and is not described in detail here.

2.3.4.3 Dynamic ALU-Based Architectures

Like the static ALU-based devices, the RaPiD [Ebel96], MATRIX [Mirs96], and Chess [Mars99] architectures contain only ALU-based logic blocks and no instruction memory. These ALUs, however, not only can be configured by configuration memory but also by data from the computation process itself. This extra level of flexibility increases the functionality of the architectures at the expense of increased architectural complexity and hardware cost.

In particular, each RaPiD device is composed of identical functional units, which consist of groups of 16-bit wide datapath components including ALUs, registers, RAM blocks, and integer multipliers. The functional units are linearly placed in a row and connected to a set of routing tracks through programmable switches. These tracks are grouped into 16-bit wide buses and run horizontally across the full length of the row. To increase speed, each track is broken into a series of wire segments, which are interconnected by programmable switches.

The MATRIX architecture consists of 8-bit wide ALUs and memory blocks placed in an FPGA-like two-dimensional array. The array is connected by FPGA-like routing resources including nearest neighbor connections, length four routing wires, and global routing wires. The architecture is also deeply pipelined to increase the clock frequency of its applications.

Each Chess device consists of a set of 4-bit wide ALUs placed in an array, which is interspersed by RAM blocks. The ALUs and RAM blocks are connected by an FPGA-like global routing network. The ALUs are also directly connected to their immediate neighbors by a set of nearest neighbor interconnects.

2.3.4.4 LUT-Based Architectures

Similar to DP-FPGA, Garp [Haus97] and the mixed-grain FPGA [Leij03] are LUT-based FPGAs that target datapath applications. A Garp device is designed as a reconfigurable fabric that serves as a co-processor to a MIPS core. The architecture consists of an array of 32 rows by 24 columns of logic blocks. Each logic block contains two LUTs that are controlled by a single set of configuration memory. Each LUT has four inputs and is connected by a set of fast carry connections to other LUTs that are on the same row. The global routing network of Garp is similar to the global routing network of a conventional FPGA in topology. However, unlike the conventional routing network where wire segments are connected together by routing switches, wire segments of Garp remain unconnected to each other. The exclusion of routing switches significantly reduces the configuration memory required to control a Garp device and can lead to faster reconfiguration (by allowing the parallel loading of locally stored configuration data). It also, however, severely limits the possible applications of the Garp architecture.

The mixed-grain architecture as proposed in [Leij03] contains a single 4-LUT in each of its logic blocks. The logic blocks can be configured into two modes including the random-logic mode and the arithmetic mode. In the random-logic mode, the logic block behaves as a single 4-LUT. In the arithmetic mode, the 4-LUT is decomposed into 4 2-LUTs to implement four distinct bit-slices. In this mode, each 2-LUT can be used as a part of a full adder, a part of a 2:1 multiplexer, or a two input Boolean equation. The global routing network of the mixed-grain architecture is similar to the conventional FPGA architecture in topology. The routing tracks, however, are grouped into 2-bit wide buses. In the random-logic mode, the bus is used to route a single bit of data; and in the arithmetic mode, the bus is used to route two bits of data at a time. The architecture also contains some special routing tracks, each independently controlled by a single set of configuration memory, dedicated for routing the control signals of the logic blocks.

Note that although the work in [Leij03] has defined the basics of the mix-grain FPGA routing architecture, the work did not measure its area efficiency by actually placing and routing a set of benchmark circuits on the architecture. Instead, the routing area required to implement several simple benchmarks are roughly estimated by counting the total number of logic block pins required to implement each circuit. This method is much less accurate than area measurements obtained by actually implementing a set of well-chosen benchmarks on a given architecture [Betz99a]. Furthermore, the feasibility of the routing architecture has not been exactly proven since no circuit has been actually implementing on the architecture.

2.3.4.5 Datapath-Oriented Features on Commercial FPGAs

Commercially available general purpose FPGAs have been incrementally adding datapath-oriented features throughout the years. The earliest adopted datapath-oriented features is carry chains; and in recent years, more complex datapath-oriented features like DSP blocks [Alte02] (Altera *Stratix*) and multipliers [Xili02] (Xilinx *Virtex II*) have been added to existing architectures. Unlike the research done in this work, however, these features are mainly aimed at improving the performance of specific arithmetic functions through heterogeneous architectures. The heterogeneity introduced by the DSP and multiplier blocks often increases the complexity of FPGA CAD flow, which potentially can introduce inefficiency in area utilization.

Finally, even though routing area accounts for a majority of the total FPGA area, none of the existing commercial FPGAs utilizes the CMS routing resources, which employs configuration memory sharing to increase the area efficiency of routing resources in datapath-oriented applications. Furthermore, little research has been done on designing automated CAD tools that can capture datapath regularity and efficiently utilize the regularity on CMS logic or routing resources.

2.3.5 Delay and Area Modeling

Given an FPGA architecture, three of the most important questions that can be asked about the architecture are:

1. What is the performance of the architecture?
2. How much area does the architecture consume? and
3. How much power do the applications implemented on the architecture demand?

To address the first two questions, one needs a method of measuring the time that it takes a signal to propagate from one point of the architecture to another and the implementation area of the FPGA. One obvious approach to addressing these two questions is to physically implement the architecture and measure the delay and area. However, this approach is too time consuming to be practical and can be influenced by the skills of the physical designer.

In this work, a simpler and more efficient approach is used. The approach is described in detail in [Betz99a]. In [Betz99a], each type of transistor in the FPGA architecture is sized using a set of guidelines to obtain the best transistor size for performance and area. Then the Elmore delay model [Elmo48] [Okam96] is used to calculate the delay between any two points on the architecture; and area (called the minimum-width transistor area) is calculated by totaling the area consumed by all transistors in the architecture, and dividing this total area into the area of a minimum width transistor. Commercially, this methodology was used in the design of the Stratix and Cyclone FPGAs [Leve03] and was shown to give accurate results. Note that the third question, power, is not investigated in this work.

2.4 Summary

This chapter has described a typical FPGA CAD flow, a conventional FPGA architecture, as well as the datapath-oriented DP-FPGA architecture that form the basis of this work. Several representative datapath-oriented reconfigurable/FPGA architectures have also been briefly described. Note that few prior work has invested effort into the detailed architecture of datapath-oriented FPGAs. Next chapter proposes the details of such an architecture.

3 A Datapath-Oriented FPGA Architecture

3.1 Introduction

This chapter presents a new FPGA architecture that contains multi-bit logic and CMS routing resources designed specifically for arithmetic-intensive applications. The architecture is unique in that its routing channels contain a mixture of conventional fine-grain routing resources and datapath-oriented CMS routing resources. The combination allows a homogenous architecture for the efficient implementation of large datapath circuits as well as small non-datapath circuits. This architecture is used as the base architecture for designing the CAD tools presented in Chapter 4, Chapter 5, and Chapter 6. Various parameters of this architecture are explored in Chapter 7 and Chapter 8 to determine the effect of CMS routing on the area efficiency of the architecture.

An overwhelming majority of circuits in an arithmetic-intensive application are datapath. These circuits have highly regular and bit-sliced structures. The remainder of the circuits in these applications are non-datapath, which are designed to control the operations of the datapath circuits. Each non-datapath circuit typically resembles a somewhat random-looking network of logic gates and contains little regular structure, which will be called *irregular logic*.

Since datapath circuits are highly regular, CMS resources created through configuration memory sharing can be effectively used to increase their area efficiency. Non-datapath circuits, on the other hand, cannot benefit from configuration memory sharing since they do not possess much regularity. In fact, non-datapath circuits implemented on CMS resources often consume significantly more area since they can only utilize a small fraction of these identically configured resources.

To efficiently implement both datapath and non-datapath circuits, the DP-FPGA architecture, discussed in Chapter 2, uses a heterogeneous structure that contains one dedicated region for implementing datapath circuits and another dedicated region for implementing non-datapath circuits. The use of dedicated regions for each type of circuit, however, has its own shortcomings including the need of more complex placement techniques, inherent issues of heterogeneity, inefficient utilization of programmable resources, and increased complexity in CAD tool design. The architecture described in this chapter, called the Multi-Bit FPGA (or MB-FPGA for short) architecture, eliminates these shortcomings through the use of a homogenous architecture. The architecture is carefully designed to take advantage of the commonalities between datapath and non-datapath circuits. It also leverages the assumption that an overwhelming majority of its target applications are datapath in structure. Comparing to the original DP-FPGA architecture, the MB-FPGA design is much simpler in structure and has no restrictions on placement — properties that can potentially increase the utilization of programmable resources. The homogenous structure of MB-FPGA also greatly simplifies the design of the CAD tools.

The remainder of this chapter describes the MB-FPGA architecture in detail. Section 3.2 motivates the development of the MB-FPGA architecture through a review of the DP-FPGA architecture discussed in Chapter 2 and a discussion on its shortcomings. Section 3.3 presents other motivations and lists the design goals of the MB-FPGA. Section 3.4 presents a structural model of the arithmetic-intensive applications. A description of the general approach for designing the MB-FPGA architecture follows in Section 3.5. Section 3.6 presents a detailed description of the MB-FPGA architecture; and finally Section 3.7 presents concluding remarks.

3.2 Motivation

The MB-FPGA architecture evolves from the DP-FPGA architecture. The primary goal in designing the MB-FPGA architecture is to improve the DP-FPGA architecture using recent advances in conventional FPGA research and to create a complete FPGA architectural description that can be supported by modern CAD tools. As a result, the first step in designing the MB-FPGA architecture was to identify the shortcomings of the original DP-FPGA architecture; and improving upon these shortcomings has became the main motivation of the MB-FPGA design process. This section presents each of these identified shortcomings in order of their importance.

3.2.1 Heterogeneous Architecture

The DP-FPGA architecture implements logic in an arithmetic-intensive application in two distinct high-level blocks. The datapath block implements the datapath circuits; and the control block implements the non-datapath circuits. Inside the datapath block there are two distinct routing channels. One type of routing channel, the control sub-channel, is used to bring single-bit signals generated by the control block into the datapath block. The other type of routing channel, the datapath sub-channel, is used to connect multi-bit signals inside the datapath block.

This heterogeneous architecture is inefficient for implementing large arithmetic-intensive applications with complex structures for two reasons: First, the architecture predetermines two separate placement regions, one for datapath logic and the other for non-datapath logic. This limits the placement options of logic blocks and can lead to placement that causes critical paths to be far longer than necessary because they are forced to crisscross between the two major placement regions. Second, the architecture pre-allocates a fixed amount of resources for each high-level block. Consequently, a circuit can fit in the architecture only if both of its datapath and non-datapath components fit in their respective high-level blocks. If any one type of logic does not fit in its high-level block, the entire circuit will not fit. As a result, for many applications, one of the high-level blocks is often severely underutilized; and the overall area efficiency suffers.

The heterogeneity of DP-FPGA further complicates CAD tool design. A CAD flow designed for the heterogeneous architecture has to assign logic to either the datapath block or the control block. This assignment usually takes place at a very early stage of the CAD flow, often right before packing, since each of the high-level blocks has a completely different logic block design. Normally datapath logic is assigned to the datapath block; and non-datapath logic is assigned to the control block. For good performance, however, it is sometimes beneficial to implement some non-datapath logic in the datapath block and vice-versa. Identifying these special cases needs the help of good timing and area information for each possible implementation, but the timing and area information at this early stage of the CAD flow often is highly inaccurate. This makes the design of good CAD algorithms difficult. A homogenous architecture, on the other hand, avoids the differentiation process entirely; therefore, it greatly simplifies the CAD design process.

3.2.2 Logic Block Efficiency

In addition to the difficulties due to heterogeneity, the logic block architecture of the DP-FPGA datapath block is inefficient for implementing large arithmetic-intensive applications. This inefficiency manifests in two ways. First, modern complex datapath circuits often contain bit-slices that resemble non-datapath circuits. These bit-slices are often large enough to require tens or hundreds of LUTs to implement. Recent research on conventional FPGA architectures [Betz97b] [Betz98] [Betz99a] suggests that, for implementing non-datapath circuits, a good conventional logic block architecture should contain four to ten tightly connected LUTs; and logic blocks containing only one LUT are extremely inefficient. Although a DP-FPGA logic block contains four LUTs, these LUTs only can be used to implement logic from four separate bit-slices. As a result, to each bit-slice, the DP-FPGA logic block resembles a conventional logic block containing only one LUT. This small logic block capacity is inefficient for implementing modern large bit-slices. To increase efficiency, each logic block of the MB-FPGA architecture contains several groups of tightly connected LUTs; and each group is used to implement logic from a single bit-slice.

Second, all LUTs in a DP-FPGA logic block are always identically configured through configuration memory sharing. The routing resources associated with each LUT are also identically configured at all times. In a homogenous architecture, this degree of configuration memory sharing can lead to poor utilization of both logic and routing resources since the logic blocks are not only used to implement highly regular datapath logic, but also irregular non-datapath logic. When a DP-FPGA logic block is used to implement non-datapath logic, only one of the four LUTs can actually be used. The other three LUTs, all sharing the same configuration memory, are wasted since they cannot be configured to implement any other logic functions. As a result, the logic capacity of the DP-FPGA logic block when used to implement non-datapath logic is only one fourth of the logic capacity of the same logic block when used to implement datapath logic. To overcome this inefficiency, the MB-FPGA uses a multi-bit logic block architecture called super-clusters, which is parameterized to have either no configuration memory sharing (which will be the focus of all the investigations conducted in this work), full configuration memory sharing, or lesser than the full degree configuration memory sharing.

3.2.3 Parameterization

The other major motivation of designing the MB-FPGA architecture is to create a highly parametrized platform for investigating the effect of various datapath-oriented CMS resources. The design should be highly parameterized. It should have the same degree of parameterization as the one used by the VPR tools discussed in Chapter 2 and should include new parameters to characterize the multi-bit logic and CMS routing resources.

3.3 Design Goals of MB-FPGA

Having discussed the motivations for creating the MB-FPGA architecture, the five design goals of the MB-FPGA architecture are summarized below:

1. The architecture should be a homogenous architecture with datapath-oriented multi-bit logic and CMS routing, supporting the area-efficient implementation of arithmetic-intensive applications.
2. The architecture should be equally capable of implementing datapath circuits as well as small non-datapath circuits. (Note that this design goal was not verified experimentally in this work.)
3. Each logic block should be able to implement several bit-slices at once; it should be large enough to efficiently implement logic from each distinct bit-slice.
4. The architecture should be supported by a modern CAD flow, which should contain new algorithms designed to support the new datapath-oriented architectural features.
5. The architecture can be used as a highly parameterized platform for architectural experimentation. Particularly, the datapath-oriented CMS resources should be designed to have a variable degree of configuration memory sharing.

3.4 A Model for Arithmetic-Intensive Applications

Since the primary purpose of the MB-FPGA architecture is to provide an area efficient platform for implementing arithmetic-intensive applications, an appropriate model must be defined for these applications. The model presented here is based on the widely used digital design principles described by several textbooks on digital design including [Hama02] [Katz94] [Kore02] [West92]. The model is further verified by analyzing several arithmetic-intensive applications including two CPU designs [Sun99] [Ye97] and a graphics accelerator design [Ye99a] [Ye99b].

The hardware implementation of an arithmetic-intensive application typically consists of two interconnected modules, as shown in Figure 3.1. One module is called the arithmetic module and the other is called the control module. The arithmetic module is designed to perform all arithmetic operations of the application. It consists of a collection of interconnected datapath circuits. It also has several multiple-bit wide data inputs and multiple-bit wide data outputs. The control module, on the other hand, is designed to control and sequence the operations performed by the arithmetic module. It consists of a collection of non-datapath circuits which typically are finite-state machines. Its inputs consist of external commands from outside of the application and the current state of the datapath generated by the arithmetic module. In terms of the amount of logic that one contains, the arithmetic module typically is much larger than the control module. This is especially true for applications that deal with complex arithmetic operations or have very wide data inputs and outputs.

Note that the arithmetic and the control modules are only conceptual entities designed to ease the design process of arithmetic-intensive applications. During the actual placement process, circuits in each module need not to be physically placed in their own distinct regions. Most of the time, it is actually beneficial in terms of maximizing performance to place the control logic close to the corresponding datapath that it controls instead of placing it close to other control circuits.

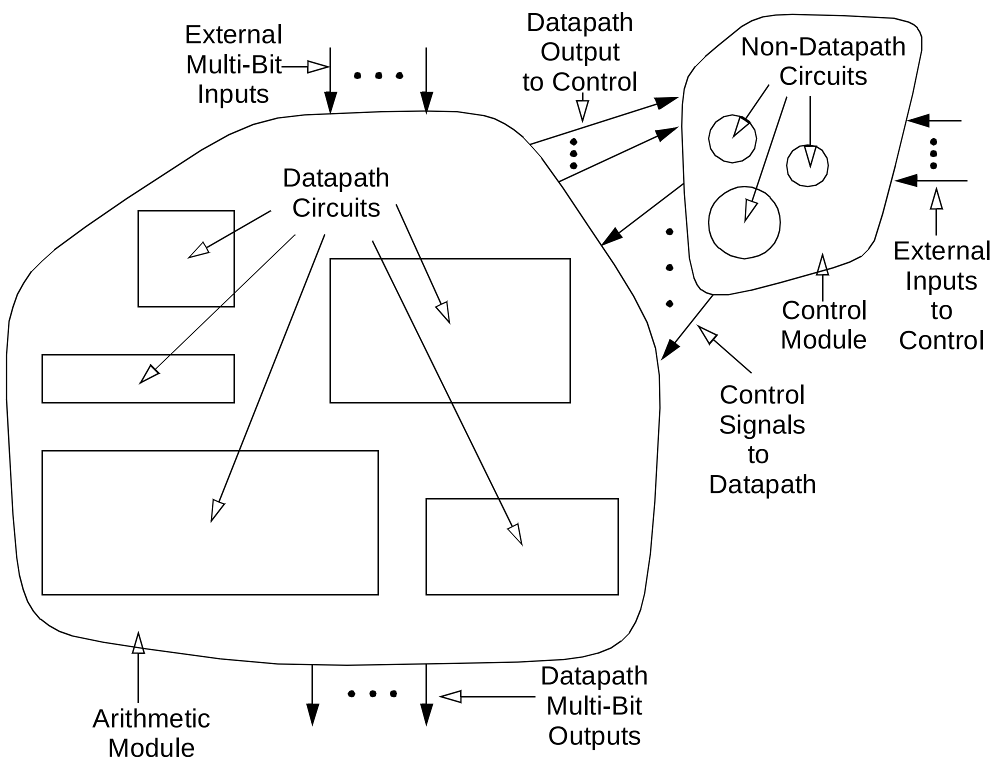

External Multi-Bit Inputs
Datapath Output to Control
Non-Datapath Circuits
Datapath Circuits
External Inputs to Control
Control Module
Control Signals to Datapath
Arithmetic Module
Datapath Multi-Bit Outputs

Figure 3.1: Arithmetic-Intensive Application

The structure of a datapath circuit is shown in Figure 3.2. It consists of a set of interconnected bit-slices, each of which processes a set of bits from the input data. As illustrated, the bit-slices are lined up from left to right according to the bit positions of the bits that they process. The bit-slice that processes the most significant bit is positioned at the extreme left of the figure.

The width of a datapath circuit is defined to be the number of bit-slices that it contains. This width usually is the same as the width of the input and output data of the datapath circuit. For example, a circuit that processes 32-bit wide data usually consists of 32 bit-slices. A datapath circuit can usually be expanded to process wider data by adding more bit-slices to its structure.

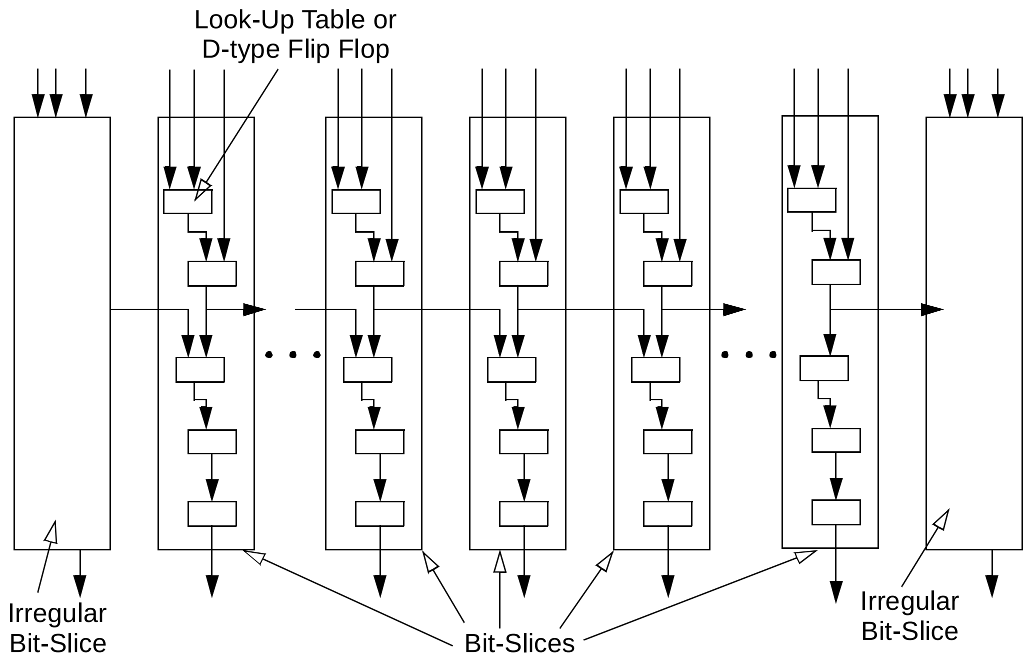


Figure 3.2: Datapath Structure

Within each bit-slice, there is a network of interconnected logic and storage, which is typically implemented by LUTs and DFFs, respectively. The network typically resembles the network found in a non-datapath circuit. With the exception of the bit-slices at both ends of a datapath circuit, most bit-slices in a datapath circuit usually contain identical networks — they have exactly the same number of DFFs, exactly the same number of LUTs for each available LUT configuration, and exactly the same connections that connect the LUTs and DFFs together. The bit-slices at the ends of a datapath often are structurally different from other bit-slices since they have to deal with the boundary conditions for processing the least and the most significant bits of inputs. Most often, however, the differences are small.

Bit-slices in a datapath circuit often have similar external connections. For example, assume that there are two datapath circuits, circuit A and circuit B. Also assume that the bit-slices in each circuit are assigned consecutive index numbers with the right most bit-slice assigned zero. If a bit-slice in circuit A with index number i is connected to a bit-slice with index number j in datapath circuit B, the bit-slice with index number i + p in circuit A usually has corresponding connections to the bit-slice with index number j + p in circuit B. If circuit B has less than j + p slices, bit-slice i + p in circuit A often is connected to the bit-slice in circuit B with index number (j + p) modulo n_B, where n_B is the total number of bit-slices in B. Note that, in both cases, A and B can be used to represent the same datapath circuit.

3.5 General Approach and Overall Architectural Description

Designing FPGA architectures is a complex problem; and there are no prescribed design methodologies. The general approach taken here, in designing the MB-FPGA architecture, is to first conceive an efficient architecture for implementing datapath circuits since these circuits account for the majority of logic in arithmetic-intensive applications. Then the design is modified to accommodate non-datapath circuits.

The major design effort of the MB-FPGA architecture involves in creating an efficient partitioning methodology that can effectively divide datapath circuits into logic blocks, which are suitable to be automatically placed by modern placement algorithms like the various versions of the simulated annealing algorithms described in [Betz99a] [Kirk83] [Marq00a] [Sech85] [Sech86] [Sech87] [Sun95] [Swar95]. This methodology significantly influences the design of the MB-FPGA architecture, and is described next along with a brief overview of MB-FPGA. Identifying the similarities between the datapath circuits and non-datapath circuits also significantly influences the design of MB-FPGA; and it is described in turn after the description on the partitioning methodology.

3.5.1 Partitioning Datapath Circuits into Super-Clusters

The sizes and the widths of datapath circuits vary greatly from application to application. Even within a single application, there can be some degree of discrepancy in the sizes and the widths of datapath circuits. The MB-FPGA architecture accommodates these varying sizes and widths by partitioning each datapath circuit into fixed sized chunks. Each chunk is suitable for implementation on a MB-FPGA logic block, which is called a super-cluster. The partitioning process attempts to capture the regularities of each datapath circuit and map the captured regularities onto the datapath specific features of the super-clusters.

The MB-FPGA architecture is designed with the assumption that the partitioning process starts from one end of a datapath circuit and progresses to the other end. In this process, every M neighboring bit-slices are grouped together into a group. For each group, N LUTs and their associated DFFs are selected from each bit-slice and are mapped onto a super-cluster. This process is repeated until all LUTs and DFFs are mapped onto super-clusters. Within each super-cluster, LUTs and DFFs from each bit-slice are kept together in a distinct sub-structure of the super-cluster, called a cluster. Note that if the copies of a particular LUT configuration exist across several bit-slices in a particular group, the partition process will keep these copies in a single super-cluster.

The partitioning process exposes intra-bit-slice connections and turns them into inter-super-cluster connections. For super-clusters that are used to implement groups of identical bit-slices, these inter-super-cluster connections form M-bit wide buses. Signals in each of these buses share the same source and sink super-clusters. Furthermore, comparing to all the other signals in the same bus, each of these signals has a unique source cluster and a unique sink cluster. When the super-clusters are connected together during the routing process, these buses can be routed through CMS routing resources that use configuration memory sharing to improve the area efficiency.

3.5.2 Implementing Non-Datapath Circuits on the MB-FPGA Architecture

The MB-FPGA architecture uses the similarities between the structure of a bit-slice and the structure of a non-datapath circuit to implement non-datapath circuits in the super-clusters. For implementing a non-datapath circuit, a super-cluster is broken down into its constituent clusters, each of which is used to implement a portion of the non-datapath circuit. Since bit-slices are structurally similar to non-datapath circuits, clusters, which are designed for the efficient implementation of bit-slices, can also be efficient for implementing many non-datapath circuits.

When clusters are used to implement a non-datapath circuit, CMS routing resources that are created through configuration memory sharing can become inefficient. To increase the area efficiency of implementing both datapath and non-datapath circuits, only a portion of the MB-FPGA routing resources are CMS resources. The remaining resources are conventional resources that do no share their configuration memory bits.

3.6 The MB-FPGA Architecture

The overall structure of the MB-FPGA architecture is shown in Figure 3.3. It consists of a two-dimensional array of super-clusters interconnected by horizontal and vertical routing channels. The connectivity between the super-clusters and the routing channels is provided by switch blocks and connection blocks. On the periphery of the architecture are the I/O blocks that connect the architecture to the outside world.

The MB-FPGA architecture has several unique datapath-oriented features including a hierarchical logic block architecture that consists of super-clusters and clusters. It also has routing channels that contain a mixture of configuration memory sharing routing tracks, called CMS routing tracks, and conventional routing tracks, called fine-grain routing tracks. Each of these features is described in turn.

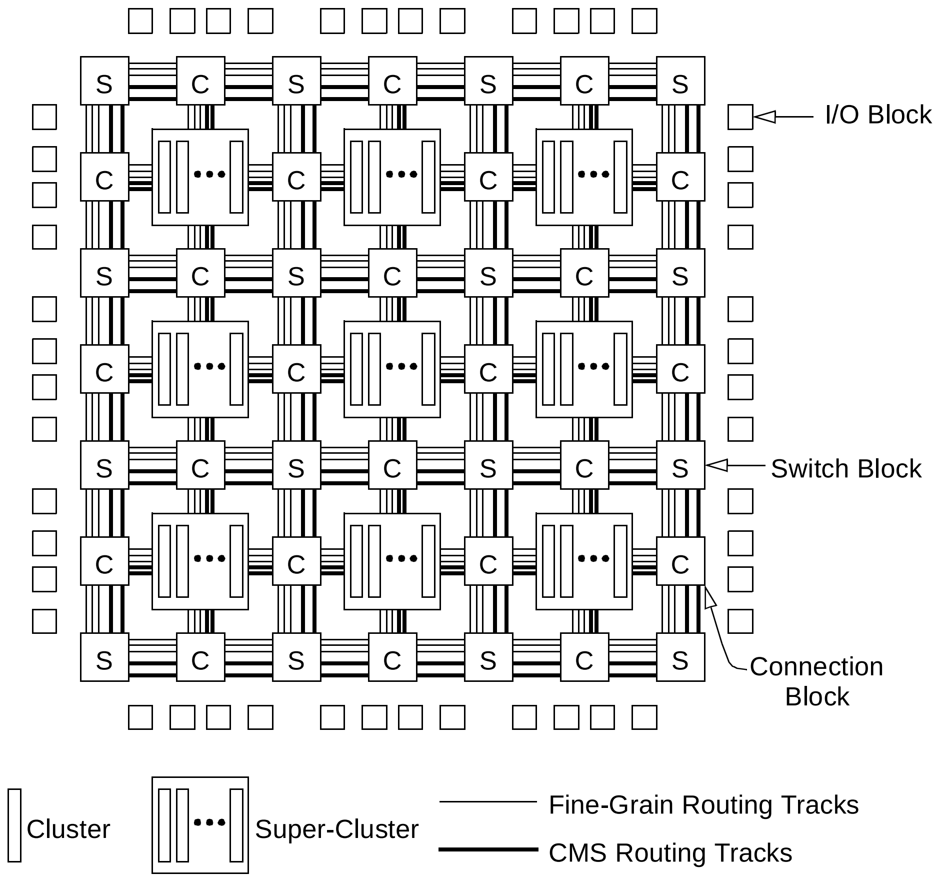


Figure 3.3: Overview of MB-FPGA Architecture

3.6.1 Super-Clusters

The structure of an MB-FPGA super-cluster, shown in Figure 3.4, consists of a set of super-cluster inputs, a set of super-cluster outputs, a set of carry inputs, a set of carry outputs, and several loosely connected clusters. The external interface of an MB-FPGA cluster is shown in more detail in Figure 3.5. It consists of a set of cluster inputs and a set of cluster outputs, a set of carry inputs, and a set of carry outputs. The number of carry inputs is equal to the number of carry outputs for each cluster. The number of carry inputs of a cluster is also equal to the number of carry inputs of a super-cluster and the number of carry outputs of a super-cluster. We believe the logic function of the MB-FPGA cluster can be based on any conventional FPGA logic block that is efficient at implementing non-datapath circuits.

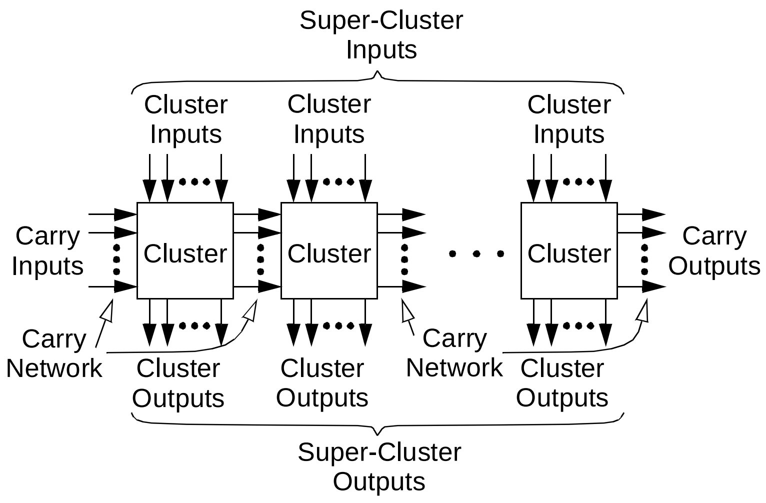


Figure 3.4: Super-Cluster with M Clusters

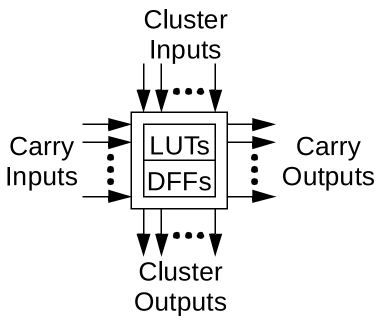


Figure 3.5: Cluster

The total number of super-cluster inputs is equal to the total number of cluster inputs of all the clusters in a super-cluster; and each super-cluster input is directly connected to a corresponding cluster input. Similarly, the total number of super-cluster outputs is equal to the total number of cluster outputs; and each super-cluster output is directly connected to a corresponding cluster output.

The network that connects all the clusters within a super-cluster together is called the carry network. It is created by connecting the carry outputs of each cluster to the carry inputs of one of its neighboring clusters to form a carry chain. The carry inputs of the super-cluster is connected to the carry inputs of the cluster at one end of the carry chain; and the carry outputs of the super-cluster is connected to the carry outputs of the cluster at the other end.

As it is apparent from the super-cluster structure, a super-cluster containing M clusters can be used to implement an M-bit wide datapath circuit whose bit-slices are no bigger than the logic capacity of a single cluster. Larger datapath circuits can be decomposed into smaller datapath circuits each of which can be implemented by a single super-cluster. When implementing non-datapath circuits, the behavior of the super-cluster is similar to the behavior of M conventional logic clusters.

3.6.1.1 Clusters

The design of the MB-FPGA cluster is based on the clusters used in [Betz99a], which are presented in Chapter 2. The overall structure of the cluster is shown in Figure 3.6. It consists of several tightly connected BLEs and a set of input and output signals as specified in Figure 3.5. The number of cluster outputs is equal to the number of BLEs in the cluster; and each cluster output is directly connected to a BLE output. Note that, for the ease of discussion, it is assumed that there are an even number of BLEs in a cluster; although, with some minor adjustments, the architecture described below applies equally well to any cluster containing an odd number of BLEs.

The number of carry outputs is equal to half of the number of BLEs in a cluster in order to match the nature of the arithmetic structures targeted by MB-FPGA. To connect carry outputs, BLEs are grouped into groups of two. One BLE in the group is called a carry BLE; and the other BLE is called a sum BLE. The output of a carry BLE is connected to a cluster carry output as well as a cluster output. The output of a sum BLE, on the other hand, is only connected to a cluster output. Both the cluster inputs and the cluster carry inputs are connected to the local routing network.

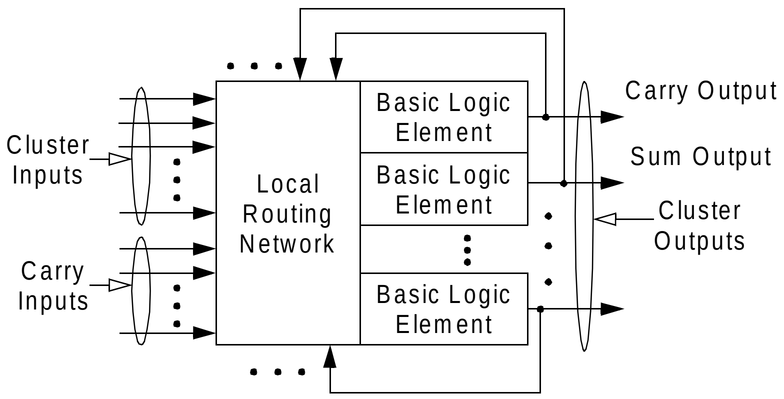


Figure 3.6: A Modified Cluster from [Betz99a]

Local Routing Network

The local routing network, as shown in Figure 3.7, consists of two separate networks. The first network, labelled local network 1, has a fully connected topology. The inputs to the network consist of two types of signals. The first type is a cluster input; and the second type is an output of a BLE in the same cluster. Each of these signals is connected to exactly one network input. The number of outputs of the network is equal to the total number of BLE inputs in the cluster.

The second network, labelled local network 2, connects the inputs of each BLE to the corresponding outputs of the first network. It also connects the cluster carry inputs to the BLEs. As shown in Figure 3.7, for every BLE, one BLE input is connected to both a carry input and an output of the first network through a two-input multiplexer controlled by an SRAM cell. All the other BLE inputs are directly connected to their corresponding outputs from the first network. Since each cluster contains half as many cluster carry inputs as the total number of BLEs, each carry input is shared by two BLEs — one BLE is a carry BLE and the other is a sum BLE.

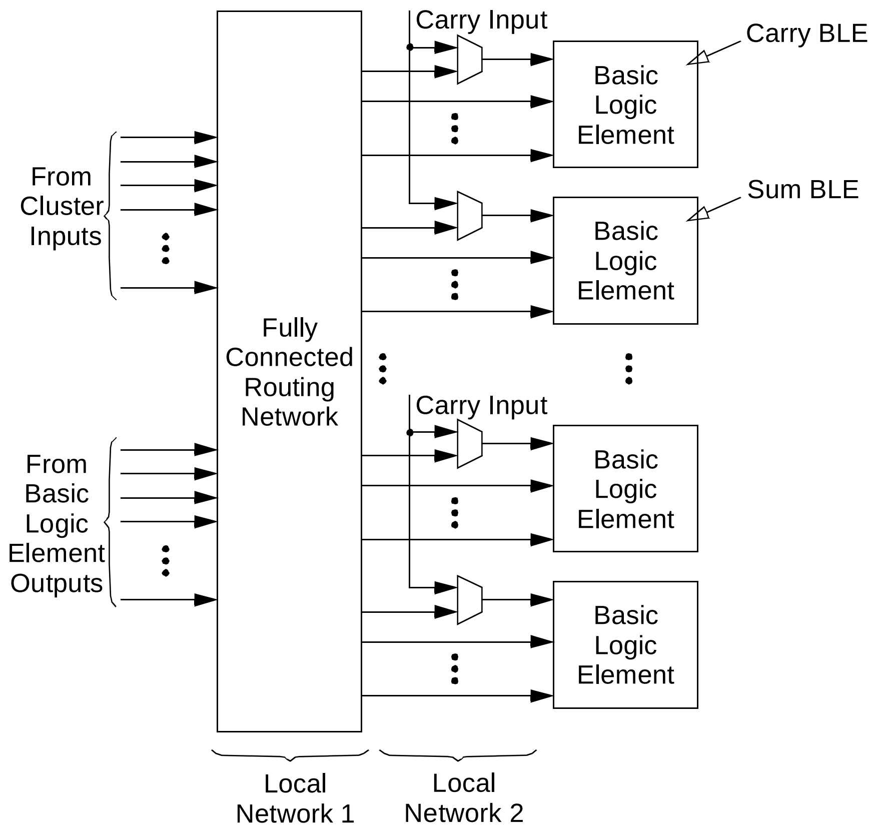


Figure 3.7: Local Routing Network

Carry Network in Detail

The carry network is designed to reduce the routing delay of long carry chains which often exist in datapath circuits. A detailed illustration of a carry network is shown in Figure 3.8. It consists of chains of BLEs connected across the cluster boundaries. In a cluster, the carry signals are generated by carry BLEs; and each of these carry signals is connected to one carry BLE and one sum BLE in a neighboring cluster. When implementing ripple carry adders, the carry BLEs and the sum BLEs are used to implement the carry generation logic and the sum generation logic respectively. Note that this structure is different from the carry chains in commercial architectures which propagate within individual clusters.

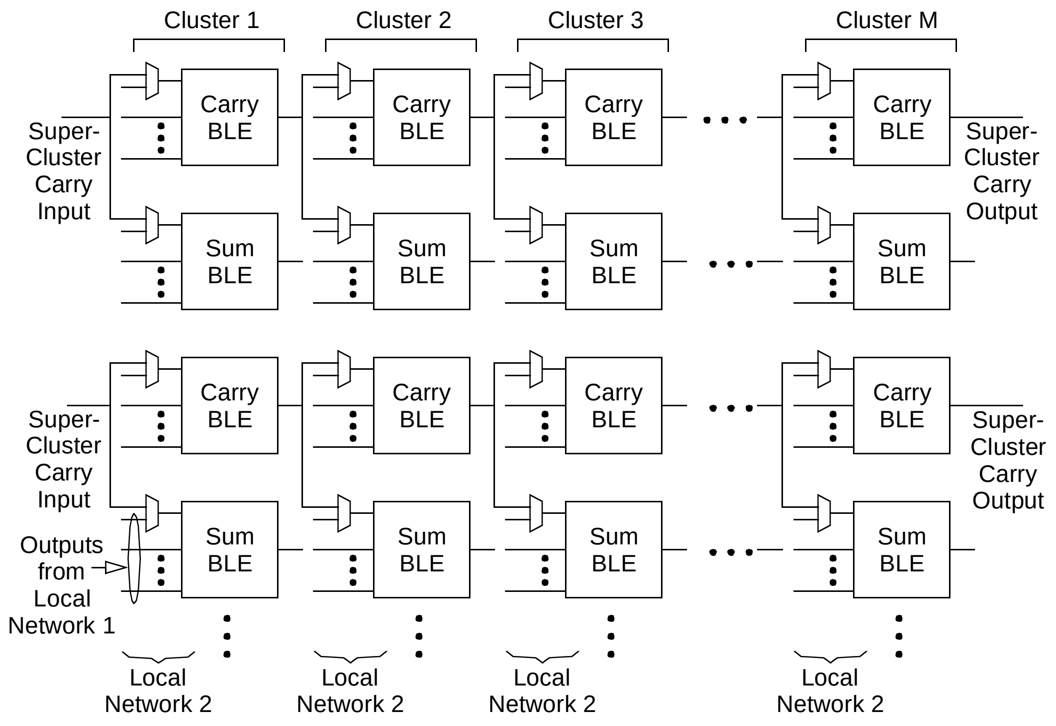

Cluster 1
Cluster 2
Cluster 3
Cluster M
Carry BLE
Sum BLE
Super-Cluster Carry Input
Super-Cluster Carry Output
Outputs from Local Network 1
Local Network 2

Figure 3.8: Carry Network

3.6.1.2 Configuration Memory Sharing

As illustrated in Figure 3.9, in an MB-FPGA cluster, several BLEs and their associated local routing resources can share configuration memory across cluster boundaries. The exact number of BLEs that share configuration memory is an architectural parameter of the MB-FPGA architecture and is selected according to the regularity of the target applications. Note that this architectural feature is not further explored in this thesis; and its characteristics will be fully studies in detail in future investigations. For the remainder of the thesis, it is assumed that each BLE is always assigned its own unique set of configuration memory.

3.6.2 Routing Switches

The MB-FPGA architecture can use three types of routing switches including the pass transistor switches and the bi-directional buffered switches used in [Betz99a], the uni-direc-

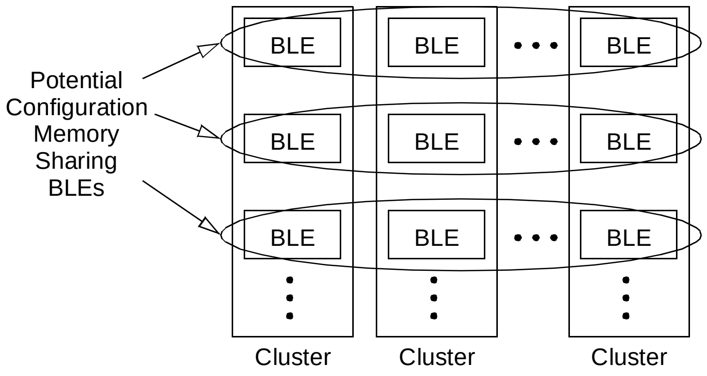


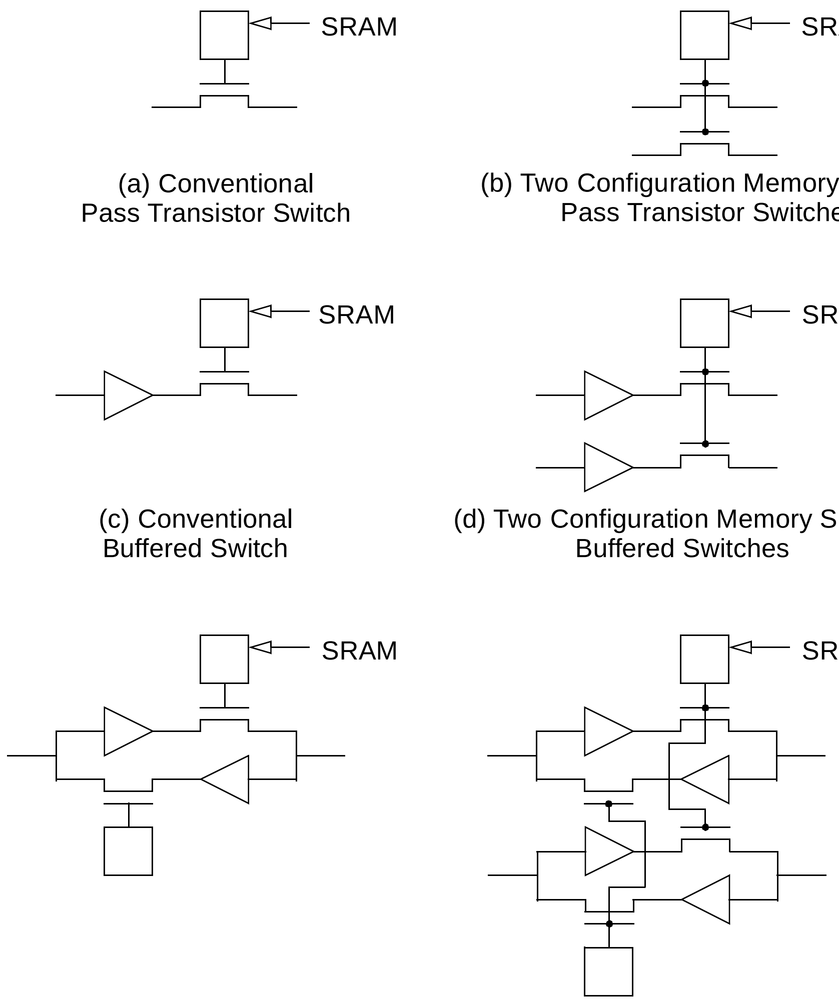
Figure 3.9: BLEs and Configuration Memory Sharing

tional buffered switches used in [Lemi02]. As described in Chapter 2, the buffered switches can use buffer sharing to reduce their implementation area. The MB-FPGA architecture also contains switches that share a single set of configuration memory. These switches are called CMS routing switches. Several examples of the CMS switches are shown in Figure 3.10 along with their conventional fine-grain counter parts.

3.6.3 Routing Channels

Each routing track in an MB-FPGA routing channel consists of wire segments connected by routing switches. These routing switches can either be conventional or configuration memory sharing. Each group of tracks that share a single set of CMS switches is called a *routing bus*. The number of tracks in a routing bus is called the granularity of the routing bus. As discussed in the general approach section, the task of transporting multi-bit wide buses from one location to another occurs frequently in datapath applications. By sharing configuration memory, CMS routing tracks can route bus signals in less area than the conventional routing tracks.

Three routing buses, each with a granularity value of two, are shown in Figure 3.11. These routing buses contain wire segments with logical length of one, two, and four respectively. Note that all routing tracks in a routing bus have the same starting position; and the starting position of the routing bus is defined to be the starting position of all the routing tracks



(a) Conventional Pass Transistor Switch

(b) Two Configuration Memory Sharing Pass Transistor Switches

(c) Conventional Buffered Switch

(d) Two Configuration Memory Sharing Buffered Switches

(e) Conventional Bi-Directional Buffered Switch

(f) Two Configuration Memory Sharing Buffered Switches

Figure 3.10: Routing Switches

in the routing bus. To conform to the single tile layout methodology outlined in Chapter 2, the starting positions of routing buses in the MB-FPGA architecture are staggered just like the starting positions of the routing tracks discussed in Chapter 2.

Not all signals in a datapath circuit can be grouped into buses. For example, control signals from control logic seldom can be grouped into buses. Since it is inefficient to use a wide routing bus to route one-bit wide signals, the MB-FPGA routing channels also contain conventional routing tracks, which are called fine-grain routing tracks. Note that, for the study done in

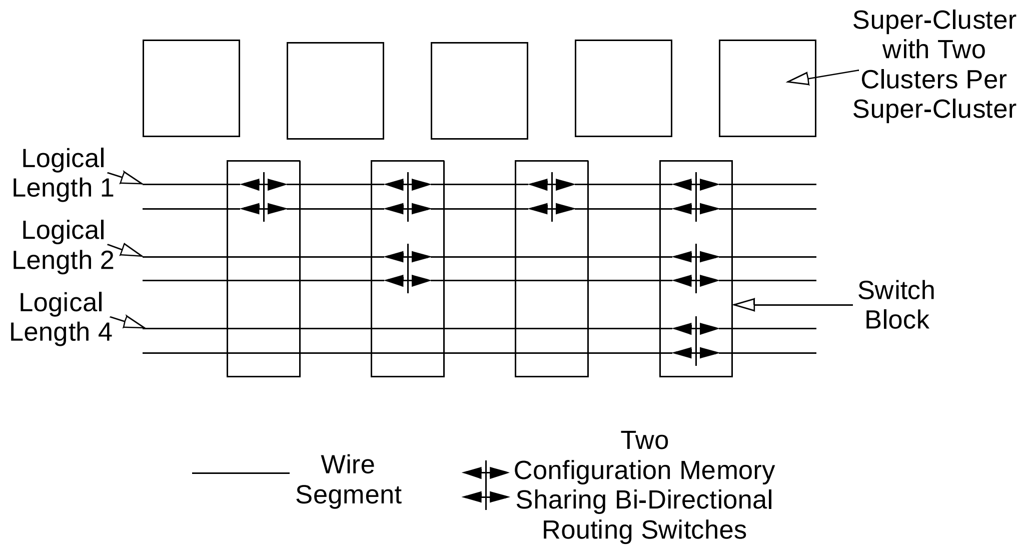

Super-Cluster with Two Clusters Per Super-Cluster
Logical Length 1
Logical Length 2
Logical Length 4
Switch Block
Wire Segment
Two Configuration Memory Sharing Bi-Directional Routing Switches

Figure 3.11: CMS Routing Tracks With A Granularity Value of Two

this work, each MB-FPGA routing channel is assumed to contain a mixture of fine-grain routing tracks and CMS routing tracks. The granularity value of the CMS tracks is set to be equal to the number of clusters in a super-cluster.

3.6.4 Switch Blocks

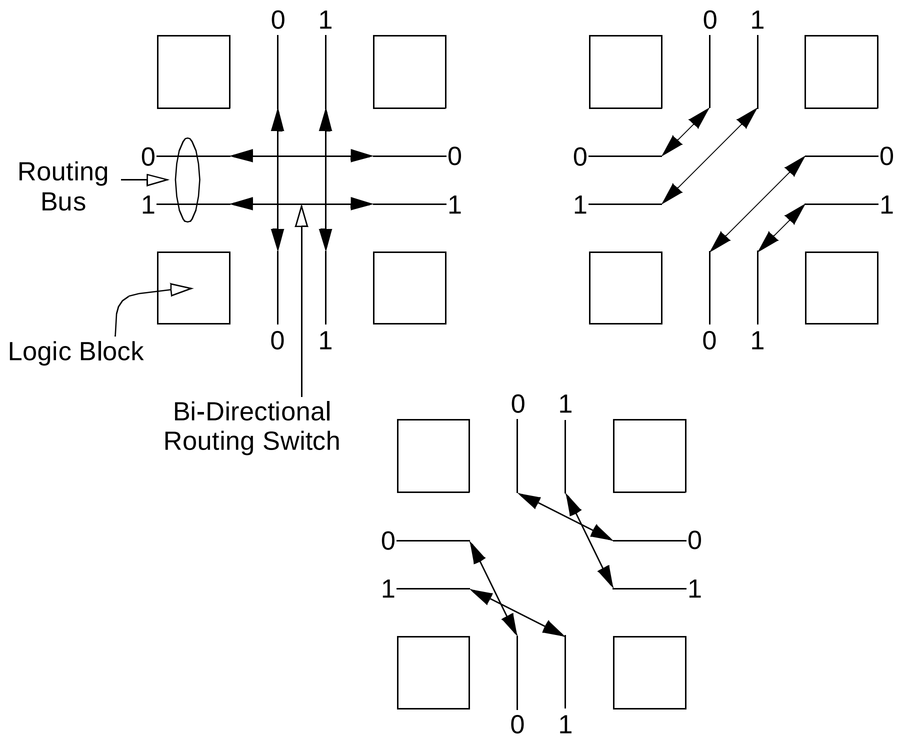
A variety of switch block topologies can be used to create the MB-FPGA switch blocks. Most of these topologies are derived from existing conventional switch block topologies like the disjoint topology [Hsei90] [Lemi97] and the Wilton topology [Wilt97]. This section describes the design decisions that were made for the MB-FPGA routing architecture and their justifications.

1. A conventional switch block topology defines the connectivity between routing tracks at the intersection of a horizontal and a vertical routing channel. When the same topology is applied to CMS routing tracks, the connectivity defined for any two routing tracks in the original topology should be used to define the connectivity for two corresponding routing buses in the MB-FPGA architecture. This is the essential notion of

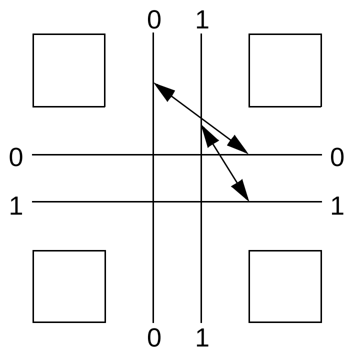
CMS routing. Each routing bus should have the same position and the same orientation as its corresponding routing track relative to the switch block.

2. When connecting two routing buses together, the corresponding bits in each bus should always be connected together. An example is shown in Figure 3.12. In the figure, each routing bus is two-bit wide; and each track in a routing bus is labelled with an index number of either 0 or 1. For each case, only tracks with the same index number are connected to each other in the switch block.
3. For routing channels that contain both CMS routing tracks and fine-grain routing tracks, only CMS tracks can be connected to CMS tracks; and only fine-grain tracks can be connected to fine-grain tracks. This significantly reduces the complexity of the switch blocks. However, connecting fine-grain tracks to CMS tracks and vice versa may be beneficial. It is left as an area for future work.
4. The switches that connect two routing buses together must share a single set of configuration memory bits; and the switches that connect two fine-grain routing tracks together must be conventional.
5. In each switch block, the switch block topology that connects CMS routing tracks together does not have to be the same as the switch block topology that connects the fine-grain routing tracks together.

Note that there are many other choices that could be made in this design. For example, like the DP-FPGA architecture, hardware shifters can be added to the switch blocks to support shifting operations. For the purpose of this thesis, however, the decisions listed above are always used unless it is stated otherwise.



(a) Six ways of connecting segments that meet at their Ends



(b) Connecting two segments that meet at their middle

Figure 3.12: Connecting Routing Buses

3.6.5 Input and Output Connection Blocks

A portion of an input connection block of an MB-FPGA super-cluster is shown in Figure 3.13. Note that, for the ease of illustration, the super-cluster is drawn with four clusters. The topology discussed here, however, applies equally well to super-clusters containing any number of clusters. As shown in the figure, each input pin of a super-cluster can be connected to a fixed number of fine-grain routing tracks. For each super-cluster with M clusters, M super-cluster inputs, each connected to an input of a unique cluster, are grouped together to form an M-bit wide bus, called an *input bus*. Each input bus is connected to a fixed number of routing buses; and there are as many input buses as the number of cluster inputs.

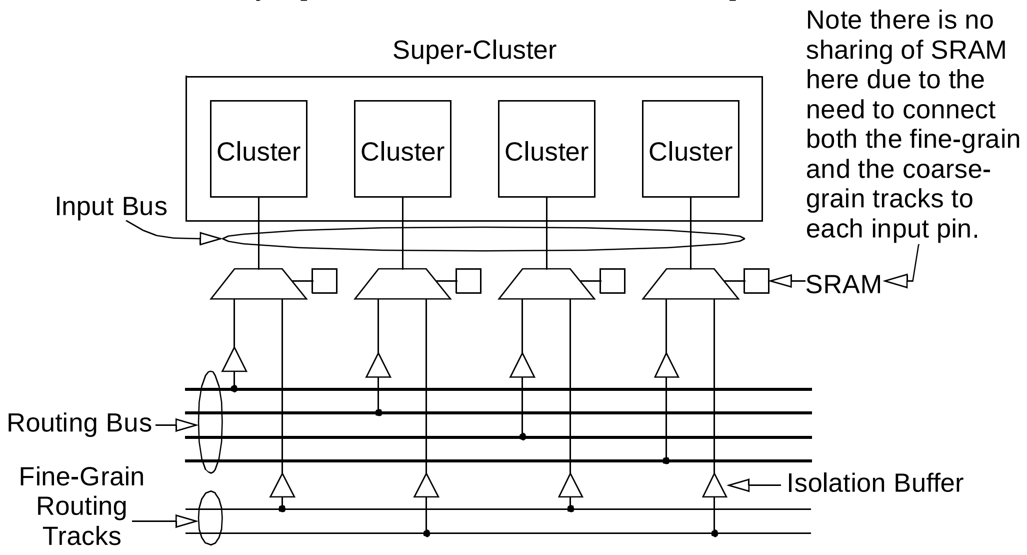

Super-Cluster
Cluster
Cluster
Cluster
Cluster
Note there is no sharing of SRAM here due to the need to connect both the fine-grain and the coarse-grain tracks to each input pin.
Input Bus
SRAM
Routing Bus
Fine-Grain Routing Tracks
Isolation Buffer

Figure 3.13: Input Connection Block (M=4)

A portion of an output connection block of a MB-FPGA super-cluster is shown in Figure 3.14. Again, for the ease of illustration, the super-cluster is drawn with four clusters. Individually, each output pin of the super-cluster can be connected to a fixed number of fine-grain routing tracks. As with super-cluster inputs, M super-cluster outputs, one from an output of a unique cluster, are grouped into an M-bit wide bus, called an *output bus*. Each output bus is connected to a fixed number of routing buses; and there are as many output buses as the number of cluster outputs.

As shown in Figure 3.13 and Figure 3.14, when connecting an input/output bus to a routing bus, the corresponding bits of each bus are connected together. The programmable switches in each bus-to-bus connection of the output connection blocks are able to share a single set of configuration memory.

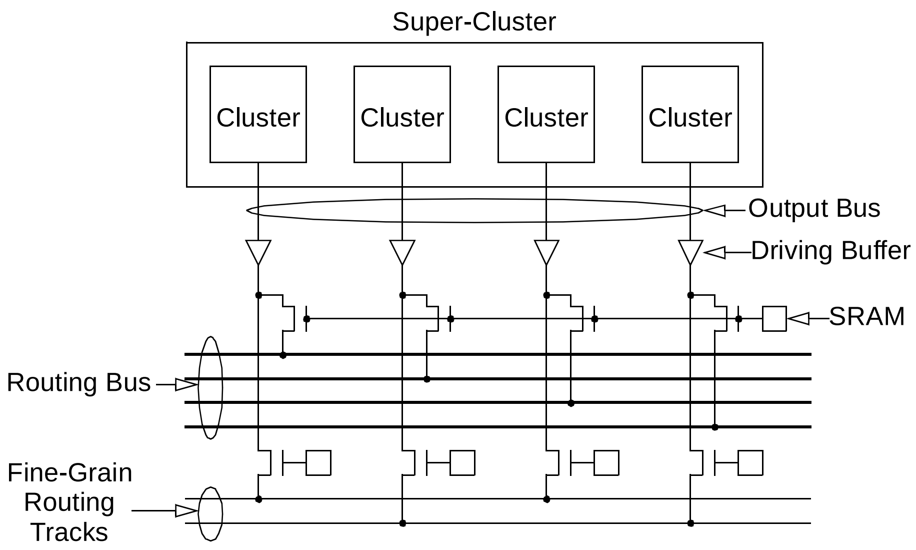


Figure 3.14: Output Connection Block (M=4)

3.6.6 I/O Blocks

As in most conventional FPGA architectures, all I/O blocks of MB-FPGA reside on the periphery of the architecture. Each I/O block is bi-directional — each I/O block contains one input pin and one output pin. Both the input pin and the output pin have the same connection patterns to the fine-grain and CMS routing tracks in their neighboring routing channel. Each I/O block input or output pin can be connected to a fixed number of fine-grain routing tracks.

M I/O block input pins or M I/O block output pins are grouped together to form M-bit wide buses, where M is equal to the number of clusters in a super-cluster. The buses formed by input pins are called *pad-input buses*; and the buses formed by the output pins are called *pad-output buses*. Each pad-input or pad-output bus is connected to a fixed number of CMS routing buses. Similar to the input bus to routing bus connections and output bus to routing bus connections, when connecting a pad-input or a pad-output bus to a routing bus, the corresponding bits of each bus are connected together. For pad-output buses, the programmable switches in each bus-to-bus connection share a single set of configuration memory.

3.7 Summary

This chapter has described a new datapath-oriented FPGA architecture, called MB-FPGA. The architecture is homogenous and contains a mixture of CMS and fine-grain resources designed for the area efficient implementation of arithmetic-applications. The architecture is also highly parameterized and can be used in architectural experiments to investigate the effect of CMS routing resources.

4 An Area Efficient Synthesis Algorithm for Datapath Circuits

4.1 Introduction

This chapter presents a new kind of synthesis algorithm that has been designed specifically to synthesize datapath circuits into datapath-oriented FPGAs, such as the one described in Chapter 3. The algorithm is unique in that it preserves the regularity of datapath circuits while achieving the area efficiency of more conventional synthesis algorithms that do not preserve datapath regularity. The algorithm is used as a part of a CAD flow in Chapter 7 and Chapter 8 to explore the architectural parameters and the area efficiency of the datapath architecture described in Chapter 3.

Synthesizing datapath circuits for datapath-oriented FPGAs can be more difficult than conventional synthesis because the algorithm must preserve the regularity of datapath circuits in order to utilize the special features of datapath-oriented FPGAs. The task of preserving regularity limits the optimization opportunities that conventional synthesis algorithms can exploit, which can severely limit the effectiveness of these algorithms. This is especially true for tools that focus on minimization of area. To preserve regularity, conventional algorithms must limit their area optimizations strictly within the boundaries of bit-slices. As a result, the optimization opportunities that exist across bit-slice boundaries cannot be effectively exploited by these algorithms. For FPGA architectures like the MB-FPGA architecture, which are designed to improve the area efficiency of FPGAs through the utilization of datapath regularity, improving the area efficiency of their synthesis algorithms is essential to achieve the overall effectiveness of the architecture. The algorithm described here, called the Enhanced Module Compaction (EMC) algorithm, addresses the issue of area efficiency by using several high level techniques that discover optimization opportunities across bit-slice boundaries and transform these bit-slices into more area-efficient implementations while still preserving their regularity.

EMC has been used to obtain excellent synthesis results for several datapath circuits from the Pico-Java processor [Sun99]. The results show that the algorithm is able to preserve datapath regularity while achieving the area efficiency of conventional synthesis algorithms that do not preserve datapath regularity.

This chapter is organized as follows: Section 4.2 motivates the development of an area efficient, datapath-oriented synthesis algorithm by reviewing several existing algorithms. Section 4.3 presents the datapath circuit representation used in the EMC algorithm. Section 4.4 describes the algorithm in detail. Section 4.5 presents the synthesis results of the algorithm; and Section 4.6 gives concluding remarks.

4.2 Motivation and Background

The most effective way of preserving datapath regularity is to preserve bit-slices. Preserving bit-slices, however, can often interfere with area optimizations. Consider Figure 4.1, which shows three interconnected bit-slices. All three bit-slices have identical internal structures consisting of three interconnected 3-input LUTs, A, B, and C. The external connection of these bit-slices, however, are slightly different. Bit-slice inputs a, b, and c are connected to a different set of external inputs for each bit-slice. Inputs d, e, and f, on the other hand, share a single set of external inputs across all three bit-slices. The remaining bit-slice input, g, is connected to a constant zero for bit-slice 1, to a constant one for bit-slice 2, and to a registered version of the output of LUT B for bit-slice 3.

The simplest way of preserving bit-slices is to perform area optimization strictly within bit-slice boundaries without considering the bit-slice I/O connections. Using this technique, each bit-slice in Figure 4.1 needs exactly three 3-input LUTs to implement. To be more area

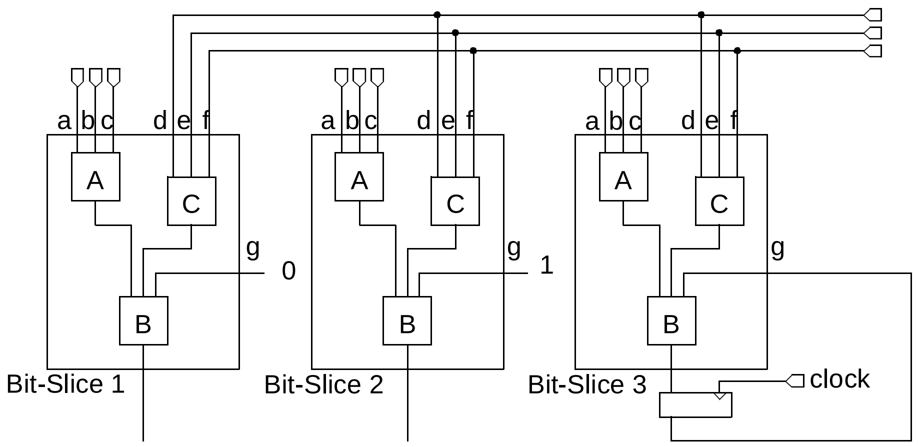


Figure 4.1: Regularity and Area Efficiency

efficient, area optimizations can peer across these boundaries without destroying them. In this case, LUT C can be extracted out of the three bit-slices and shared as shown in Figure 4.2. The total implementation area of the circuit is reduced, since each bit-slice only needs two 3-input LUTs to implement. However, since LUT C does not belong to any bit-slice, irregularity is introduced into the circuit. The implementation area can be further reduced through the introduction of even more irregularities. For example, LUT B in bit-slice 1 and bit-slices 2, can be implemented using a smaller 2-input LUT as shown in Figure 4.3. If the target FPGA allows more than one type of LUT, this implementation requires even less area to implement at the expense of increased irregularity. Although this is a simple example, it illustrates the essence of the problems that occur while preserving datapath regularity during synthesis.

As discussed in Chapter 2, existing synthesis algorithms that preserve datapath regularity can be classified into four categories: regularity preserving logic transformation-based synthesis [Kutz00a] [Kutz00b], hard-boundary hierarchical synthesis (synthesis that performs optimizations strictly within bit-slice boundaries), template mapping [Call98] [Cora96] [Nase94] [Nase98], and module compaction [Koch96a] [Koch96b]. Among these four algorithms, regularity preserving logic transformation relies on extracting regularity directly from datapath cir-

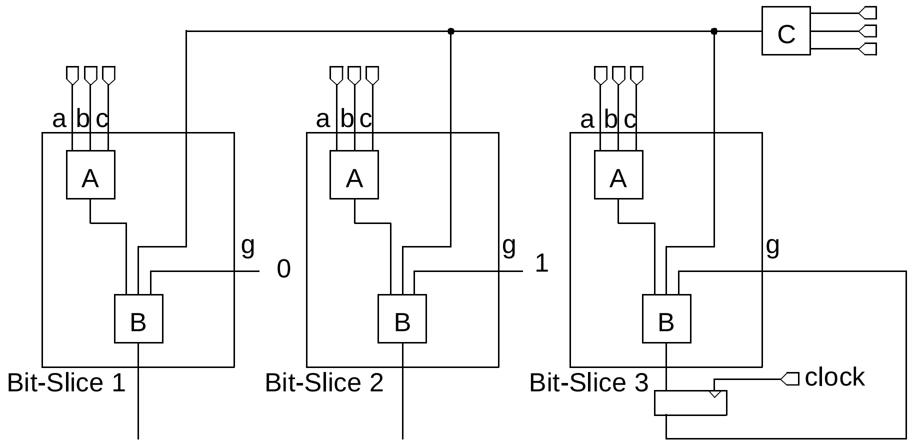


Figure 4.2: Share Look-Up Table C

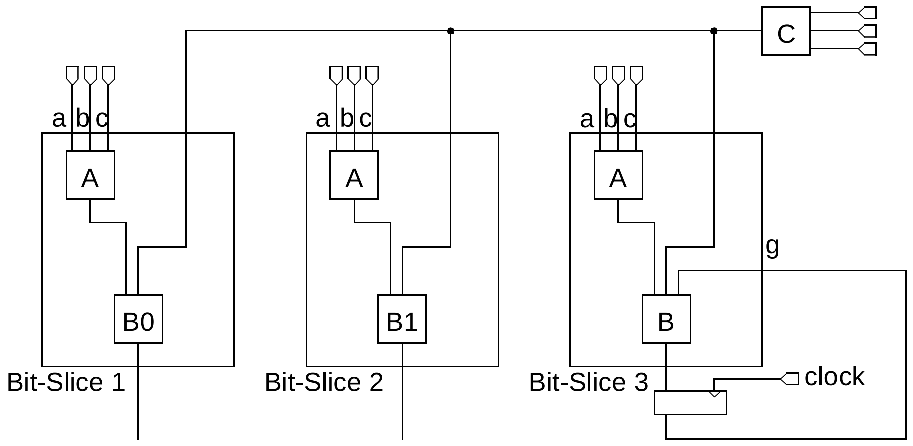


Figure 4.3: Simplify Look-Up Table B

cuits without the guidance of bit-slices. Its effectiveness in preserving datapath regularity is limited because this is a difficult problem that does not leverage information that exists in the user design.

Hard-boundary hierarchical, template mapping [Call98] [Cora96] [Nase94] [Nase98], and module compaction [Koch96a] [Koch96b] algorithms, all make use of user-defined regularity information in the form of bit-slices. All three, however, are not very area efficient. For example, when the Synopsys FPGA Compiler is used to synthesize a series of 15 datapath circuits from the Pico-Java processor [Sun99] using the hard-boundary hierarchical synthesis, there is an average area increase of 38% compared to synthesis that destroys datapath regularity (which will be called flat synthesis in the remainder of this thesis). The detailed results of this experiment after synthesis are summarized in Table 4.1, where column 1 lists the name of each circuit. Column 2 and 3 list LUT count for the flat synthesis and the hard-boundary hierarchical synthesis, respectively. Column 4 lists the percentage of LUT increase for hard-boundary hierarchical synthesis as compared to the flat synthesis. Column 5, 6, and 7 summarize the same information for the DFF count.

For the template mapping, using the technique discussed in Chapter 2, [Cora96] reports an area increase of over 48%. The module compaction algorithm, on the other hand, has the shortcoming of being designed specifically for the Xilinx XC4000 FPGA architecture. As a result, the algorithm uses dedicated hardware in XC4000 to implement some of its preserved regular structures. This artificially reduces the total number of LUTs needed to implement a circuit. Furthermore, there are limited experimental results on the algorithm's area efficiency — only two area results were ever published. Both are for relatively small circuits. Even with dedicated hardware support, the bigger circuit, containing only 712 4-LUT, still has a worst case area inflation of 17%. The much smaller circuit, containing only 112 4-LUT, was shown to have an area reduction of 16%.

4.3 Datapath Circuit Representation

The goal of this work is to develop an algorithm that preserves datapath regularity. To do this requires an appropriate format for specifying datapath regularity. The format used in this thesis consists of a netlist of datapath components, described in VHDL or Verilog, which is called the *top-level netlist*. All datapath components used in the netlist are instantiated from a predefined datapath component library. This library contains fundamental datapath building blocks such as multiplexers, adders/subtracters, shifters, comparators, and registers.

	#LUTs			#DFFs		
	Flat	HB	% inflat.	Flat	HB	% inflat.
dcu_dpath	960	1190	24%	288	288	0.0%
ex_dpath	2530	3517	39%	364	364	0.0%
icu_dpath	3120	4430	42%	355	356	0.28%
imdr_dpath	1182	1548	31%	170	170	0.0%
pipe_dpath	443	549	24%	218	220	0.92%
smu_dpath	490	568	16%	190	190	0.0%
ucode_dat	1243	1362	9.6%	224	224	0.0%
ucode_reg	78	172	121%	74	80	8.1%
code_seq_dp	218	366	68%	216	226	4.6%
exponent_dp	477	725	52%	64	64	0.0%
incmod	779	1207	55%	72	72	0.0%
mantissa_dp	846	1167	38%	192	192	0.0%
multmod_dp	1558	2275	46%	193	193	0.0%
prils_dp	377	675	79%	0	0	0.0%
rsadd_dp	346	526	52%	0	0	0.0%
Total	14647	20277	38%	2620	2639	0.73%

Table 4.1: Area Inflation for Hard-Boundary Hierarchical Synthesis

These datapath components are in turn composed of bit-level structures that are called *bit-slice netlists*. A bit-slice netlist is a netlist of logic gates, representing the function of a single bit-slice of a datapath. The bit-slice netlist is typically instantiated multiple times in a datapath component. At this level the netlist is called the *datapath component level netlist*.

The number of bit-slice netlist instantiations corresponds to the width of the datapath. All instantiations are assigned a unique *bit-slice number* from one to the width of the datapath with the least significant bit-slice labeled one.

An example of a 4-bit ripple carry adder datapath component is shown in Figure 4.4. The bit-slice netlist of this datapath component is a netlist of logic gates defining a full adder. This design is instantiated four times to form the 4-bit adder.

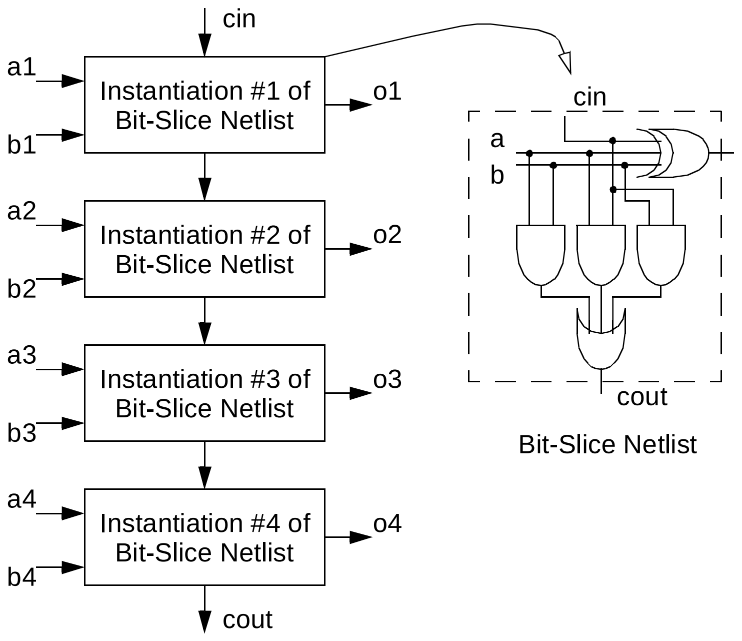

cin
a1
b1
Instantiation #1 of Bit-Slice Netlist
o1
a2
b2
Instantiation #2 of Bit-Slice Netlist
o2
a3
b3
Instantiation #3 of Bit-Slice Netlist
o3
a4
b4
Instantiation #4 of Bit-Slice Netlist
o4
cout
cin
a
b
cout
Bit-Slice Netlist

Figure 4.4: 4-bit Ripple Adder Datapath Component

4.4 The EMC Synthesis Algorithm

Figure 4.5 gives the overall flow of the new datapath-oriented EMC synthesis algorithm, which consists of four major steps: word-level optimization, module compaction, bit-slice I/O optimization, and within bit-slice boundary synthesis. In the first three steps, the top-level netlist is optimized for area through the transformation and merging of its datapath components into more area efficient implementations. During these steps, some logic might be created which does not belong to any specific bit-slices. For example, new logic might be created to generate signals that fan out to multiple bit-slices such as the example in Figure 4.2. This logic is called *irregular logic* (previously known as non-datapath, to distinguish it from logic that fits nicely into a datapath) and is represented directly as logic gates in the top-level netlist. Each one of these three steps is discussed in more detail in the subsections that follow.

In the final and fourth step, each bit-slice is synthesized and mapped into 4-input LUTs and DFFs through the use of a conventional synthesis algorithm. The irregular logic gates are

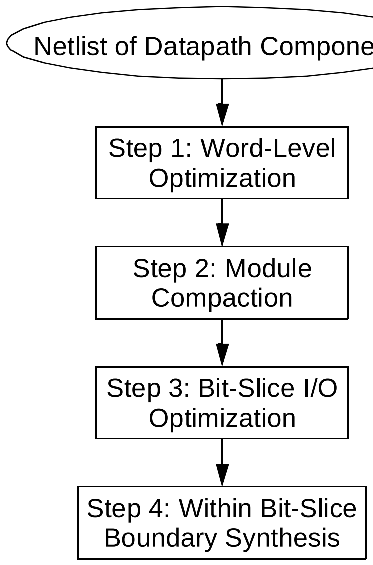

Netlist of Datapath Components
Step 1: Word-Level Optimization
Step 2: Module Compaction
Step 3: Bit-Slice I/O Optimization
Step 4: Within Bit-Slice Boundary Synthesis

Figure 4.5: Overall Synthesis Flow

also synthesized and mapped into LUTs and DFFs independently from the datapath components using the same conventional synthesis algorithm.

4.4.1 Word-Level Optimization

During the first step, two types of word-level optimizations are performed. One is used to extract common sub-expressions across bit-slice boundaries. The other uses operation reordering to reduce area. These two optimizations are performed manually. Their algorithms, which are suitable for automation, are presented here.

4.4.1.1 Common Sub-expression Extraction

Each datapath component represents a set of arithmetic operations. In the top-level netlist, datapath components are connected together to form complete mathematical functions. Each of these functions has multiple bit outputs, where the output bits can be individually described using logic expressions. Often, common sub-expressions exist across these logic expressions. More precisely, let both x ($[x_0 , x_1 , ... , x_n]$) and y ($[y_0 , y_1 , ... , y_n]$) be bit vectors of width n. Let $y = f(x)$ be a mathematical function of x. Each individual bit of y can be expressed in terms of bits of x as follows:

$y_0 = f_0 (x_0 , x_1 , ... , x_n)$

$y_1 = f_1 (x_0 , x_1 , ... , x_n)$

...

$y_n = f_n (x_0 , x_1 , ... , x_n)$

If there exist a function $g (x_0 , x_1 , ... , x_n)$, such that:

$y_0 = f'_0 (g (x_0 , x_1 , ... , x_n) , x_0 , x_1 , ... , x_n)$

$y_1 = f'_1 (g (x_0 , x_1 , ... , x_n) , x_0 , x_1 , ... , x_n)$

...

$y_n = f'_n (g (x_0 , x_1 , ... , x_n) , x_0 , x_1 , ... , x_n)$

then $g(x)$ is called a common sub-expression of $f_0(x), f_1(x), ... , f_n(x)$. The implementation area of mathematical functions can be reduced by discovering and extracting these common sub-expressions so that they are only implemented once.

In a conventional synthesis process, common sub-expressions are extracted through logic transformations. This extraction process usually destroys the regularity of datapath circuits, since conventional synthesis independently transforms logic expressions one bit at a time. We have found that many of these common sub-expressions can be discovered at the word-level where entire datapath components are examined based on their functionality. Most importantly, datapath regularity can be easily preserved by extracting these common sub-expressions at the word-level where datapath structures remain clearly identifiable.

For the benchmarks investigated in this work, the most effective word-level transformation that extracts common sub-expressions is *multiplexer tree collapsing*. In a multiplexer tree, the multiplexers, their data inputs, outputs, and the interconnection signals form a tree topology. Each node of the tree, which has multiple inputs and a single output, represents a multiplexer. Each input of a node corresponds to a multiplexer data input. The output of a node corresponds to a multiplexer output. An edge in the graph represents a net connecting a multiplexer output to a multiplexer data input, a primary input, or the primary output of the multiplexer tree.

A multiplexer tree sometimes can be substituted by a single multiplexer, which requires much less logic to implement, as illustrated in Figure 4.6. Here the multiplexer tree in the left circuit is substituted by a single multiplexer in the right circuit. To implement the two multiplexers and the **and** gate in the left circuit, two 4-input LUTs are needed for every bit-slice as indicated by the shaded regions in the figure. To implement the multiplexer and the **and** gate in the right circuit, only one 4-input LUT is needed for every bit-slice. The extra irregular logic in the right circuit is the common sub-expression extracted by the transformation. It usually is shared by several bit-slices, so its area cost is small in wide datapath circuits.

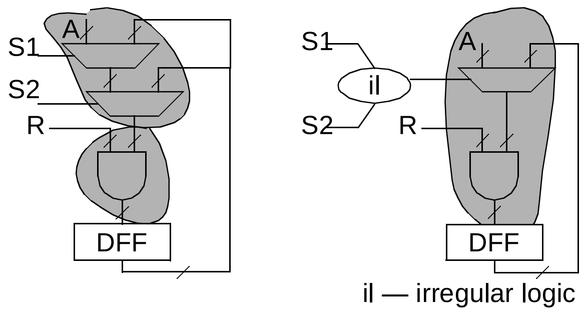


Figure 4.6: Mux Tree Collapsing Example

The algorithm used to collapse multiplexer trees is as follows: First, multiplexer trees are identified in the top-level netlist. This is easy to perform since the functionality of each datapath component is known. The total number of unique data inputs to each tree is then identified. Each tree is replaced by a single multiplexer whose width is equal to the number of unique data inputs of the tree. Each input of the new multiplexer is connected to a unique multiplexer tree primary data input. The output of the new multiplexer is connected to the primary output of the tree. Finally, the select signal of the new multiplexer is generated using the select signals of the original multiplexer tree. If the replacement reduces the area cost in terms of LUT and DFF count, it is retained. Otherwise, the replacement is rejected.

4.4.1.2 Operation Reordering

The second word-level transformation uses operation reordering to reduce area. In particular, the optimization reorders result selections into operand selections. Arithmetic operators such as multiplications are, in general, much more expensive than multiplexers. In the event that several identical operations are performed on independent data sets and only one result is used, it usually is much cheaper to preselect the input data than to perform all operations in parallel and select the final results.

An example is given in Figure 4.7. Here the result of two addition operations is selected by a 2:1 mux. The operation can be more efficiently performed by preselecting adder inputs and using a single adder instead of two. Before optimization, five 4-input LUTs are needed to implement the function. After optimization, only four 4-input LUTs are needed to implement the same function. This optimization is not obvious at the bit-slice level, since **cout0a** and **cout0b** appear to be two independent signals. However, when viewed from the top-level netlist, the optimization is clearly identifiable.

More generally, assume that there is a function $y = f(x)$ where x ($[x_0, x_1, \ldots, x_n]$) is an n-bit wide bit vector and y ($[y_0, y_1, \ldots, y_p]$) is a p-bit wide bit vector. The function:

```
if (s == 0) then
    z = f( u )
else
    z = f( v )
```

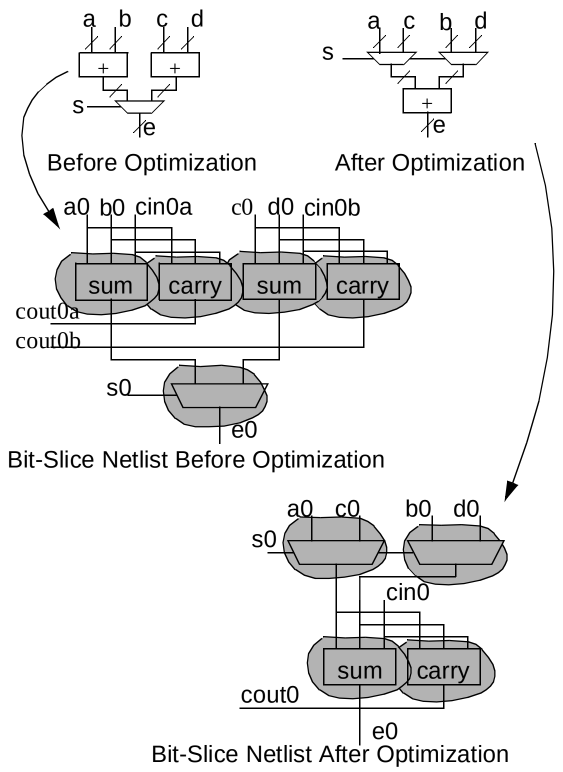


Figure 4.7: Result Selection to Operand Selection Transformation

can be sometimes more cheaply implemented as:

```
if (s == 0) then
    w = u
else
    w = v
z = f ( w )
```

if *f(x)* requires more area to implement than the extra multiplexers. The algorithm would search for multiplexers whose data inputs are from the outputs of identical functions where these outputs have no other fan-outs. It then compares the area implementation cost of *f(x)* with the area implementation cost of the multiplexers. If the area cost of *f(x)* is greater than the area cost of the additional multiplexers, the transformation is performed.

4.4.2 Module Compaction

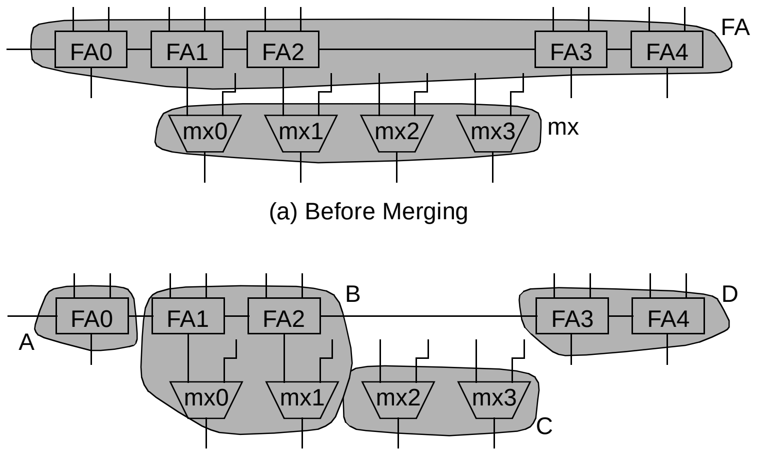
The goal of module compaction is to create larger bit-slices, which can be more efficiently synthesized by the conventional synthesis algorithms. This compaction process is performed as the second step of the optimization. Here two connected bit-slice netlists are iteratively merged together to form a larger bit-slice netlist. The algorithm repeatedly iterates through the input netlist until there are no eligible datapath components left to be merged. By creating larger bit-slice netlists, more optimization opportunities are created for the conventional synthesis stage shown in Figure 4.5, where the synthesis is restricted to within the boundaries of bit-slice netlists. This merging process is similar to the module compaction algorithm proposed by Koch in [Koch96a] as discussed in Chapter 2. It differs in its merging criteria; unlike Koch's algorithm, it does not depend on any placement information. As a result, the EMC algorithm can be more easily integrated into existing CAD flows.

The basic merging operation is a pattern identification process. Two groups of bit-slices from two datapath components are merged if the following conditions are met:

1. These two groups contain equal numbers of bit-slices.
2. All bit-slices in each group have consecutive bit-slice numbers as defined in Section 4.3.
3. All bit-slices in one group are identically connected to their corresponding bit-slices in the other group. Here two corresponding bit-slices are defined to be bit-slices from two distinct groups, each with the same offset from the lowest bit-slice number in its group.

Each merging operation creates a new datapath component. The bit-slice netlist of the new component combines the two original bit-slice netlists. If a merging group does not include all the bit-slices of its datapath component, the remaining slices in the component are split into two datapath components — one with all the bit-slices whose bit-slice numbers are smaller than the bit-slice numbers of the merging group, and the other with all the bit-slices whose bit-slice numbers are larger than the bit-slice numbers of the merging group.

An example of module compaction is shown in Figure 4.8. Here, before merging, as shown in Figure 4.8a, there are two datapath components with labels **FA** and **mx**, respectively. One component contains five slices of full adders labeled **FA0** to **FA4**. The other component contains four slices of single-bit 2:1 multiplexers labeled **mx0** to **mx3**. Based on the merging rules stated above, two full adders, **FA1** and **FA2** can be merged with two single-bit 2:1 multiplexers, **mx0** and **mx1**, to form a new datapath component. The remaining three full adders are broken into two new components after merging. The four new datapath components created by the merging process is shown in Figure 4.8b. They are labeled **A**, **B**, **C**, and **D**.



(b) After Merging

Figure 4.8: A Bit-Slice Netlist Merging Example

Two extra conditions are imposed to prevent a carry type signal from causing all bit-slices connected to it to be merged into a single component. For example, consider a second merging iteration on the circuit of Figure 4.8b after the initial merging described above. Datapath component **A** will be qualified to be merged with the first slice of datapath component **B** since they are connected by the carry signal. Then, in the third merging iteration, component **A** and **B** will be completely merged into a single bit-slice. After two more iterations, the carry chain will cause **A**, **B**, and **D** in the figure to be merged into a single bit-slice, which completely destroys the regularity of the datapath.

To prevent this, first, merging operations are ordered so that operations that will create the widest datapath components are performed first. Second, for every bit-slice netlist an *ancestors* field is defined, which is a set of bit-slice netlists. Initially each bit-slice netlist has only itself in its *ancestors* set. When two bit-slice netlists are merged, the *ancestors* set of the new bit-slice netlist is the union of the *ancestors* sets of the two merging bit-slice netlists. If the intersection of the *ancestors* of two bit-slice netlists is not empty, these two bit-slice netlists cannot be merged together.

With the *ancestors* field, nothing can be merged during the second merging iteration in Figure 4.8b, since all mergable component pairs, (**A**, **B**) and (**B**, **D**) share one common ancestor — **FA**.

4.4.3 Bit-Slice Netlist I/O Optimization

Each bit-slice netlist has a set of predefined I/O signals that enter and exit the netlist. Depending on the usage of these signals, some of them can be eliminated and converted into internal signals of the netlist. Since each bit-slice netlist is synthesized using a conventional synthesis algorithm in the final step of the synthesis flow, converting I/O signals into internal signals can reduce the implementation area of bit-slices by providing extra information to the conventional synthesizer. In this optimization process, four types of bit-slice I/O signals are converted into internal signals of bit-slices. Each is discussed below.

Before any I/O optimization is performed, each datapath component in the top-level netlist is first divided into **m**-bit wide subcomponents, where **m** is specified by the user. Each subcomponent is a self-contained datapath component with its own bit-slice netlist definition and a netlist of **m** instantiations of the bit-slice netlist. The division starts from the least significant bit of each datapath component and groups adjacent **m** bit-slices into a subcomponent. If the width of the datapath component is not an integer multiple of **m**, the subcomponent containing the most significant bits will be less than **m**-bit wide. The variable **m** is called the *granularity* of the synthesis flow. A larger **m** preserves more datapath regularity typically at the expense of increased area, while a smaller **m** decreases area at the expense of preserving less datapath regularity. After division, each original datapath component in the top-level netlist is substituted by its corresponding subcomponents.

The first type of I/O optimization is constant absorption. When an input of a bit-slice netlist is always connected to the same constant value (either zero or one) for all instantiations of the netlist in a datapath component, this input signal is converted into a constant internal signal of the netlist.

The second type of I/O optimization is feedback absorption. When a connection exists between a bit-slice netlist input and a bit-slice netlist output for all instantiations of the netlist, this input signal is converted into an internal signal and reconnected to the corresponding output inside the netlist.

An example of feedback absorption is shown in Figure 4.9. Here **Datapath Component A** consists of four bit-slices, which are all instances of the same bit-slice netlist. Since each of the slice inputs labeled **Ai1** is connected to a corresponding slice output labeled **Ao** from the same slice, **Ai1** is eliminated as an input of the bit-slice netlist and is converted to an internal signal. **Ai1** is reconnected to **Ao** inside the netlist.

The third type of I/O optimization is duplicated input absorption. When two bit-slice netlist inputs are connected together for all instantiations of the design, one of the input signals is converted into an internal signal and is reconnected to the other input signal inside the netlist.

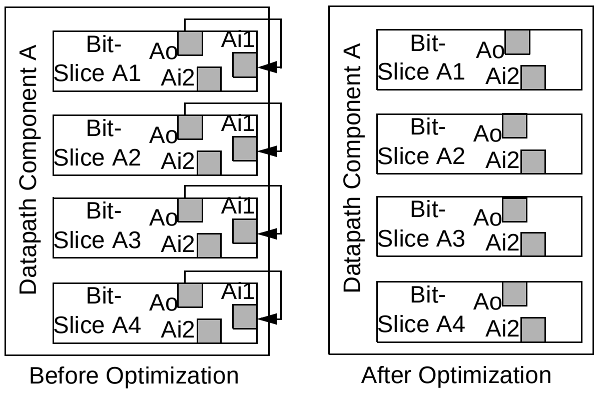


Figure 4.9: Feedback Absorption Example

An example of duplicated input absorption is shown in Figure 4.10. As before, **Datapath Component A** consists of four bit-slices, which are all instances of the same bit-slice netlist. Since each of the slice inputs labeled **Ai1** is always connected to a corresponding slice input labeled **Ai2** in the same slice, **Ai2** is eliminated as an input of the bit-slice netlist and is converted to an internal signal. **Ai2** is reconnected to **Ai1** inside the netlist.

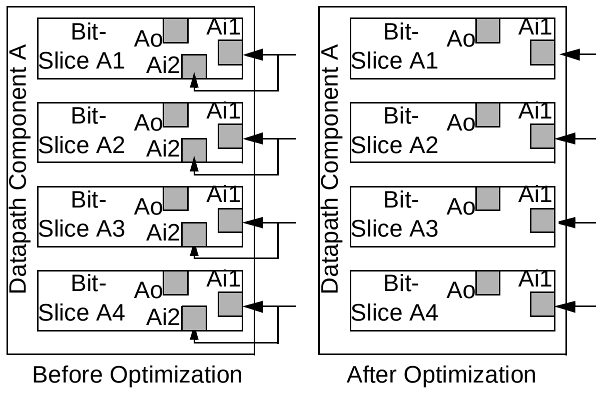


Figure 4.10: Duplicated Input Absorption

The last type of I/O optimization is unused output elimination. When a bit-slice netlist output does not connect to any other signals in all instantiations of the bit-slice netlist, this output signal is converted into an internal signal of the bit-slice netlist, which will permit the downstream synthesis to optimize it away.

4.5 Experimental Results

The experimental results from applying the EMC synthesis on fifteen datapath benchmarks are presented in this section. These fifteen circuits are from the Pico-Java processor [Sun99], which is a 32-bit processor; and the benchmark set covers all major datapath components of the processor. Note again that the word-level optimizations, described in Section 4.4.1, were performed manually. The other two optimization steps were done by automated algorithms implemented in the C-language. The Synopsys FPGA Compiler [Syno99] is used to perform the within bit-slice boundary synthesis. Unless specified otherwise, all the data presented here are synthesized using a granularity value (**m**), as defined in Section 4.4.3, of four. In the remainder of this section, the area inflation of EMC is first discussed; and then the regularity results are presented in turn.

4.5.1 Area Inflation

For every benchmark circuit, the final LUT and DFF count of the EMC synthesis is compared with the counts achieved by a fully conventional synthesis flow that flattens the datapath hierarchy. The conventional flow is also performed by the Synopsys FPGA Compiler. In order to assure that the best achievable conventional synthesis results are used to be compared with the EMC synthesis, two conventional synthesis flows are used. One flow directly synthesizes the input netlist of the EMC algorithm. The other flow resynthesizes the output netlist of the EMC algorithm. In some cases, one flow offers slightly better results than the other; and the best result is always used as the flat synthesis result.

Table 4.2 summarizes the LUT and DFF inflation of each benchmark circuit for the hard-boundary hierarchical synthesis, and the new EMC synthesis. Each inflation figure is calculated by comparing the regularity preserving synthesis with the best conventional synthesis results. The formula, $inflation = \frac{DA}{FA} - 1$, is used to calculate the inflation for both LUTs and DFFs. In the formula, *DA* represents the synthesis area of the regularity preserving synthesis; *FA* represents the synthesis area of the flat conventional synthesis.

	Best Flat Synthesis Area		Area Inflation for Regularity Preserving Synthesis							
			Hard-Boundary Hierarchical		EMC — Word-Level Optimization Only		EMC — Word-Level Optimization and Module Compaction		Full EMC	
	#LUT	#DFF	LUT	DFF	LUT	DFF	LUT	DFF	LUT	DFF
dcu_dpath	960	288	24%	0.0%	22%	0.0%	8.1%	0.0%	0.63%	0.0%
ex_dpath	2530	364	39%	0.0%	38%	0.0%	26%	0.0%	0.91%	0.0%
icu_dpath	3120	355	42%	0.28%	24%	0.28%	23%	0.28%	3.7%	0.0%
imdr_dpath	1182	170	31%	0.0%	31%	0.0%	29%	0.0%	3.1%	0.0%
pipe_dpath	443	218	24%	0.92%	19%	0.92%	19%	0.92%	6.3%	0.0%
smu_dpath	490	190	16%	0.0%	16%	0.0%	9.4%	0.0%	0.61%	0.0%
ucode_dat	1243	224	9.6%	0.0%	9.6%	0.0%	9.6%	0.0%	4.9%	0.0%
ucode_reg	78	74	121%	8.1%	113%	8.1%	113%	8.1%	5.1%	0.0%
code_seq_dp	218	216	68%	4.6%	53%	4.6%	53%	4.6%	2.3%	0.0%
exponent_dp	477	64	52%	0.0%	31%	0.0%	26%	0.0%	5.0%	0.0%
incmod	779	72	55%	0.0%	50%	0.0%	36%	0.0%	11%	0.0%
mantissa_dp	846	192	38%	0.0%	29%	0.0%	28%	0.0%	3.8%	0.0%
multmod_dp	1558	193	46%	0.0%	42%	0.0%	15%	0.0%	4.9%	0.0%
prils_dp	377	0	79%	0.0%	77%	0.0%	36%	0.0%	2.9%	0.0%
rsadd_dp	346	0	51%	0.0%	51%	0.0%	-2.6%	0.0%	-12%	0.0%
Total	14647	2620	38%	0.73%	32%	0.73%	22%	0.73%	3.2%	0.0%

Table 4.2: LUT & DFF Inflation for Regularity Preserving Synthesis

Column one of the table lists the name of each benchmark circuit. Columns two and three give the LUT and DFF count of each circuit from the best conventional synthesis flow. Columns four and five restate the inflation figures of the hard-boundary hierarchical synthesis, which were shown in Table 4.1. Here, synthesis is performed without the first three optimization steps of the EMC algorithm. Columns six to eleven summarize the inflation figures for various configurations of EMC to show the contributions of each individual optimization step. In particular, columns six and seven show the inflation figures when only step 1 (word-level optimization) and step 4 (within bit-slice boundary synthesis) are performed. Columns eight and nine list the inflation figures when step 1, step 2 (module compaction), and step 4 are performed. Finally columns ten and eleven list the inflation figures for the full EMC synthesis.

The average LUT inflation of the hard-boundary hierarchical synthesis is 38% and the average DFF inflation is 0.73%. The word-level optimization reduces the average LUT inflation to 32%; and the combined word-level optimization and module compaction reduce the average LUT inflation further down to 22%. Using the full EMC algorithm, the average LUT inflation is finally reduced to 3.2% and the DFF inflation is reduced to zero. Note that the bit-slice netlist I/O optimization contained in the full EMC algorithm utilizes many optimization opportunities created by the previous module compaction step, which creates bit-slices containing many common I/O signals through the merging of smaller bit-slices.

The benchmarks exponent_dp, icu_dpath, and code_seq_dp benefited the most from word-level optimization. Their LUT inflation figures were reduced by 21, 18, and 15 percentage points, respectively. Benchmarks rsadd_dp, prils_dp, and multmod_dp, on the other hand, benefited the most from module compaction. Their LUT inflation was reduced by 54, 41, and 27 percentage points, respectively, when module compaction is performed on top of word-level optimization. Finally, benchmarks ucode_reg, code_seq_dp, and prils_dp benefited the most from bit-slice I/O optimization. The LUT inflation figures were reduced by 108, 51, and 33 percentage points, respectively, when the optimization is performed on top of word-level optimization and module compaction.

The numbers from Table 4.2 show that the EMC algorithm does not significantly increase the LUT and DFF count for the benchmarks as compared with flat synthesis and is much more area efficient than the hard-boundary hierarchical synthesis. For the circuit, rsadd_dp, the EMC algorithm even discovered more optimizations than the conventional synthesis, resulting in much smaller area.

Finally, Table 4.3 presents LUT count inflation as a function of **m,** the granularity of synthesis. DFF count inflation remained at zero with increasing **m**. The table shows that the LUT inflation increases from 3.5% to 7.4% as **m** is increased from 4 to 32. The cause of this increase is the less efficient I/O optimization as described in Section 4.4.3 as a result of the increased datapath component width.

m	1	4	8	12	16	20	24	28	32
Avg. LUT Inflation (%)	0.0	3.5	4.6	6.3	6.7	6.5	6.7	6.8	7.4

Table 4.3: LUT Count Inflation as a Function of Granularity

4.5.2 Regularity

Various aspects of the datapath regularity were measured for the circuits after they are synthesized by the EMC algorithm in order to illustrate that the regularity was preserved. The granularity of the synthesis, **m**, is again set at four. Higher granularities typically result in higher regularity.

4.5.2.1 Logic Regularity

To measure the logic regularity preserved by the synthesis process. The benchmark circuits are first synthesized by the full EMC algorithm. LUTs and DFFs in the synthesized circuit are divided into two groups. The first group is called regular logic, which, as defined in Section 4.3, contains LUTs and DFFs that belong to datapath components. The second group is called irregular logic, which contain LUTs and DFFs that do not belong to any datapath component. Note that several optimizations described in Section 4.4 create irregular logic during the optimization process, so the proportion of regular logic as compared to irregular logic in a netlist is changed by the synthesis process.

Before synthesis, nearly all logic in the benchmarks is regular. After synthesis, the detailed regularity results for each circuit is shown in Table 4.4. In the table column 1 lists the name of each circuit. Column 2, 3, and 4 list the number of LUTs, the number of DFFs, and the total number of LUTs and DFFs for each circuit after EMC synthesis. Column 5, 6, and 7 list the number of LUTs, the number of DFFs, and the total number of LUTs and DFFs that are preserved in datapath for each circuit after synthesis. Finally column 8 lists the total percentage of LUTs and DFFs that are in datapath. Overall, the EMC algorithm preserves a significantly amount of logic regularity. Here, 90% of the total number of LUTs remain in datapath components after synthesis, while only 10% of the logic resides in irregular logic.

4.5.2.2 Net Regularity

The regularity of nets after synthesis was also measured. Table 4.5 shows two major types of *two-terminal connections* (defined as a logical connection containing one LUT/DFF output pin and one LUT/DFF input pin) exist in datapath benchmarks after synthesis — bus and control signals. The first column of Table 4.5 lists the name of each benchmark circuit. The second column lists the total number of two-terminal connections in each circuit.

A two-terminal bus is defined as an **m**-bit wide bus (four in this table) that connects one datapath component to another and obeys the following two conditions: First, each bit of the bus must be generated by a distinct bit-slice in the source datapath component and absorbed by a distinct bit-slice in the sink datapath component. Second, the source bit-slice and the sink bit-slice must have the same bit-slice number. The topology of a 4-bit wide bus is shown in Figure 4.11. On average 48% of two-terminal connections in these benchmarks can be

	#LUT	#DFF	#LUT + #DFF	#LUT in Datapath	#DFF in Datapath	#LUT + #DFF in Datapath	%LUT & DFF in Datapath
dcu_dpath	966	288	1254	900	288	1188	95%
ex_dpath	2553	364	2917	2390	350	2740	94%
icu_dpath	3235	355	3590	3108	352	3460	96%
imdr_dpath	1218	170	1388	1132	160	1292	93%
pipe_dpath	471	218	689	387	188	575	83%
smu_dpath	493	190	683	428	190	618	90%
ucode_dat	1304	224	1528	1224	224	1448	95%
ucode_reg	82	74	156	68	64	132	85%
code_seq_dp	223	216	439	52	152	204	46%
exponent_dp	501	64	565	320	64	384	68%
incmod	867	72	939	772	64	836	89%
mantissa_dp	878	192	1070	772	192	964	90%
multmod_dp	1634	193	1827	1388	152	1540	84%
prils_dp	388	0	388	324	0	324	84%
rsadd_dp	305	0	305	281	0	281	92%
Total	15118	2620	17738	13546	2440	15986	90%

Table 4.4: Logic Regularity

grouped into 4-bit wide busses. The percentage number for each benchmark is summarized in column three of Table 4.5.

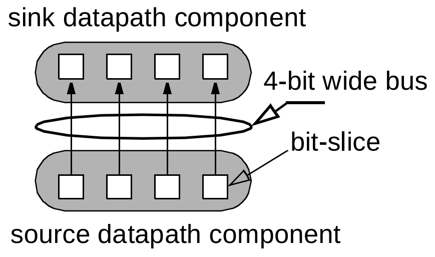


Figure 4.11: 4-bit Wide Bus Topology

A control net is a single net that enters a datapath component and fans out to all **m** bit-slices (4 in this table). The topology of a 4-bit control net is shown in Figure 4.12. The control

	Total Two-Terminal Conn.	Percentage of Two-Terminal Conn. that are 4-Bit Wide Busses	Percentage of Two-Terminal Conn. that are Fan-Out Four Control Signals
dcu_dpath	2232	49%	43%
ex_dpath	6547	52%	39%
icu_dpath	8047	47%	36%
imdr_dpath	3100	50%	36%
pipe_dpath	1049	48%	42%
smu_dpath	1167	48%	25%
ucode_dat	3143	52%	41%
ucode_reg	194	72%	21%
code_seq_dp	799	58%	18%
exponent_dp	1362	32%	23%
incmod	2013	42%	33%
mantissa_dp	2533	47%	36%
multmod_dp	3380	39%	25%
prils_dp	864	41%	32%
rsadd_dp	722	52%	27%
Total	37152	48%	35%

Table 4.5: Net Regularity

nets on average consist of 35% of the total two-terminal connections in these benchmarks. The detailed percentage number for each benchmark is shown in column four of Table 4.5.

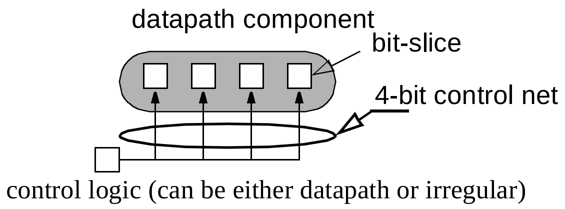


Figure 4.12: 4-bit Control Net Topology

Overall, there are 83% of two-terminal connections that belong to either a bus or a control net. There are few two-terminal connections that belong to both a bus and a control net at the same time. Note that in order to standardize the amount of logic contained in a datapath component in the above definition of bus and control signals, a datapath component is defined to be a group of LUTs and DFFs that can be fitted into an MB-FPGA super-cluster. The super-cluster contains four clusters; and each cluster contains ten inputs and four BLEs. The packing algorithm that will be described in Chapter 5, is used to create these fixed sized datapath components before the regularity measurement.

4.6 Conclusion

This chapter presented the EMC synthesis algorithm targeting datapath-oriented FPGAs. It empirically demonstrated that the algorithm is nearly as efficient in terms of LUT/DFF usage as the conventional flat synthesis algorithms. In terms of LUT and DFF count, the algorithm produces circuits on average with only 3%–8% LUT inflation and no increase in register count. The regularity of the fifteen benchmark circuits was also measured. The results show that there is a high degree of regularity in these synthesized benchmarks, with 48% of two-terminal connections that can be grouped into 4-bit wide busses and 35% of two-terminal connections from highly regular control signals with at least 4-bit fan-out.

5 A Datapath-Oriented Packing Algorithm

5.1 Introduction

This chapter presents a new kind of packing algorithm that has been designed specifically for datapath circuits. The algorithm is unique in that it preserves the regularity of datapath circuits and maps the preserved regularity onto the super-clusters of the MB-FPGA architecture described in Chapter 3. This algorithm and software is employed in Chapter 7 and Chapter 8 to investigate the effect of CMS routing on the area efficiency of FPGAs.

Packing that preserves datapath regularity can be more difficult than classical packing as described in [Betz97a] [Marq99] [Bozo01] because the task of preserving datapath regularity limits the flexibility of the packing tools. Recall that the MB-FPGA architecture captures datapath regularity by implementing identical portions of neighboring bit-slices in a single super-cluster. Within the super-cluster, logic from each bit-slice is implemented in individual clusters. Under this architecture, when a LUT or a DFF from a datapath is packed into a cluster, other LUTs and DFFs from the same bit-slice should be given priority for implementation in the same cluster. Similarly, LUTs and DFFs from neighboring bit-slices should be given priority for implementation in the super-cluster that the cluster resides in. The algorithm described here, called the coarse-grain node graph (CNG) packing algorithm, addresses this issue of preserving datapath regularity by simultaneously packing several identical LUTs or DFFs from neighboring bit-slices. It also uses specially designed metrics to optimize the implementation area of datapath circuits and the delay of timing-critical nets.

CNG has been used to obtain excellent packing results for the datapath circuits from the Pico-Java processor [Sun99]. It is able to pack for a wide range of super-cluster configurations while preserving a high degree of datapath regularity; and its area efficiency and performance approach those of the traditional packing algorithms, which do not preserve datapath regularity.

This chapter is organized as follows: Section 5.2 motivates the development of the datapath-oriented packer; Section 5.3 defines the datapath-oriented packing problem; Section 5.4 presents the model that is used to represent datapath circuits for the CNG packing algorithm; Section 5.5 describes the algorithm in detail; Section 5.6 presents the results from the tests of the packer; and Section 5.7 gives concluding remarks.

5.2 Motivation

A key problem in packing for the MB-FPGA architecture is that packing choices made for one LUT or one DFF from a bit-slice may limit the packing choices of another LUT or DFF. Consider Figure 5.1, which shows two groups of identical bit-slices. The first group consists of bit-slice 1, 2 and 3; and the second group consists of bit-slice 4, 5, 6, 7, and 8. For the purpose of timing analysis, each LUT is assumed to have a logic delay of 1 time unit. Each connection between any two LUTs is also assumed to have a propagation delay of 1 time unit if it is implemented inside a super-cluster and 10 time units if it is implemented outside a super-cluster. In the figure, assuming that all connections are implemented outside super-clusters, the most timing-critical path is the path that connects bit-slice 2, 5, 6, 7, and 8 together through LUT A and LUT D in bit-slice 2, LUTs labeled E in bit-slice 5, 6, and 7, and LUT E and LUT F in bit-slice 8. The total delay of this critical path is 60 time units.

Now, assume that each target super-cluster contains three clusters and each cluster contains two BLEs. Naively, a timing-driven packer, which also attempts to preserve datapath regularity, might pack bit-slice 1 first. Considering bit-slice 1 in isolation, the packer will try to minimize the local critical path that connects LUT A, B, and C together. As a result, LUT A and B will be packed into a single cluster and LUT C and D will be packed into another clus-

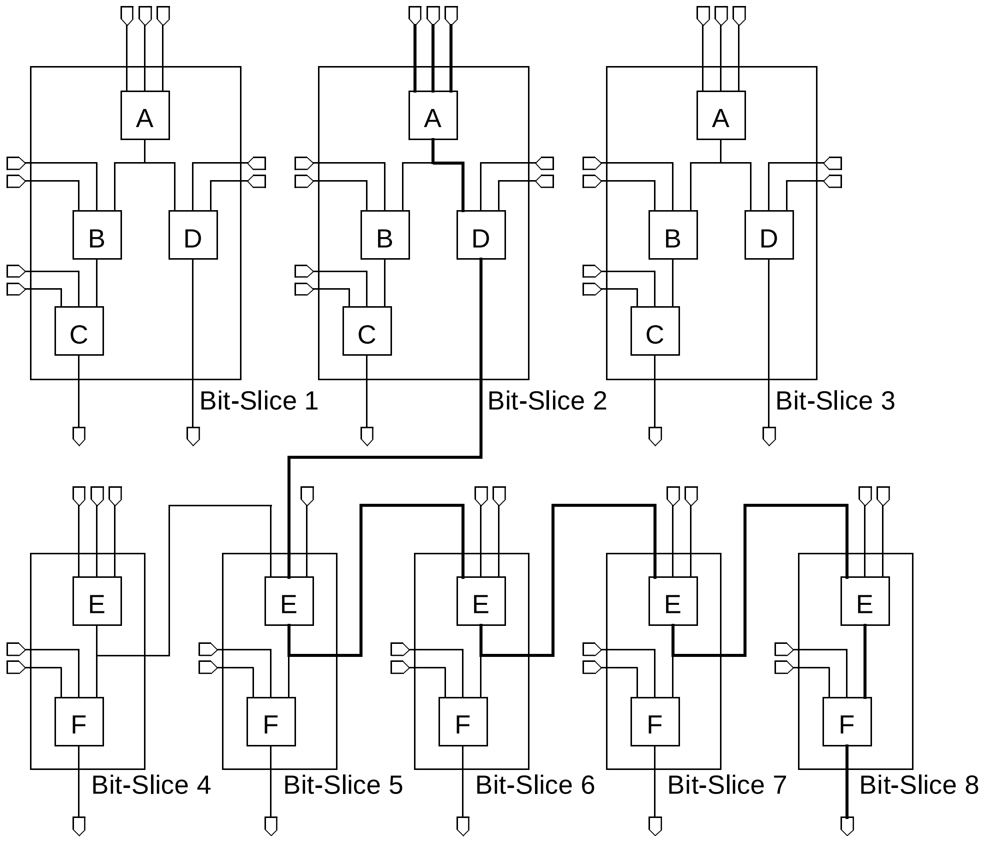


Figure 5.1: Regularity and Performance

ter. Once bit-slice 1 is packed, in order to preserve datapath regularity, bit-slice 2 and 3 will have to be packed using exactly the same configuration as bit-slice 1. After packing these three bit-slices into clusters, the three clusters containing LUT A and B can be grouped into a super-cluster; and the clusters containing LUT C and D can also be grouped into a super-cluster. Similarly, bit-slice 4, 5, 6, 7, and 8 can be packed into two super-clusters; and the entire packing solution is shown in Figure 5.2. This naive packing solution, however, is sub-optimal since LUTs on the critical path are separated into six clusters across four super-clusters, resulting in a critical path delay of 33 time units.

A better solution is for the packer to pack bit-slice 2 first. In this case, LUT A and D are packed into a cluster and LUT B and C are packed into another cluster based on the real criti-

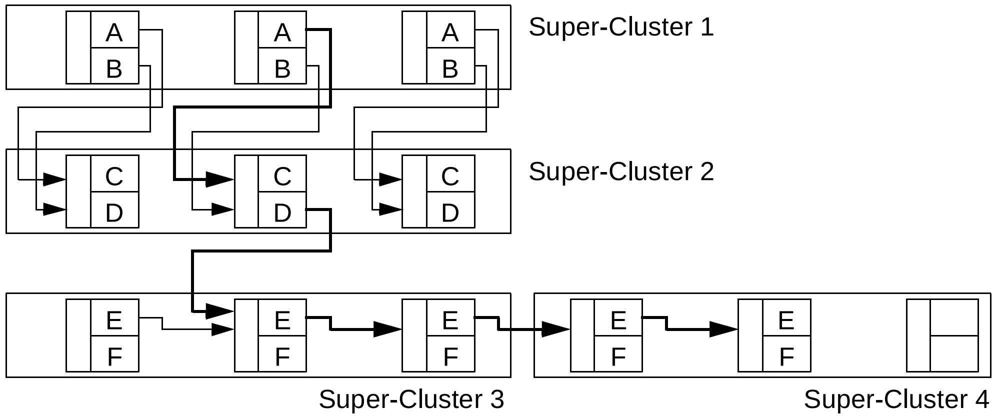


Figure 5.2: A Naive Packing Solution

cal path. As shown in Figure 5.3, in this solution, LUTs on the critical path are contained in only five clusters across three super-clusters, consisting of super-cluster 1, 3, and 4. Consequently, this packing solution is much faster that the previous one and has a critical path delay of 24 time units.

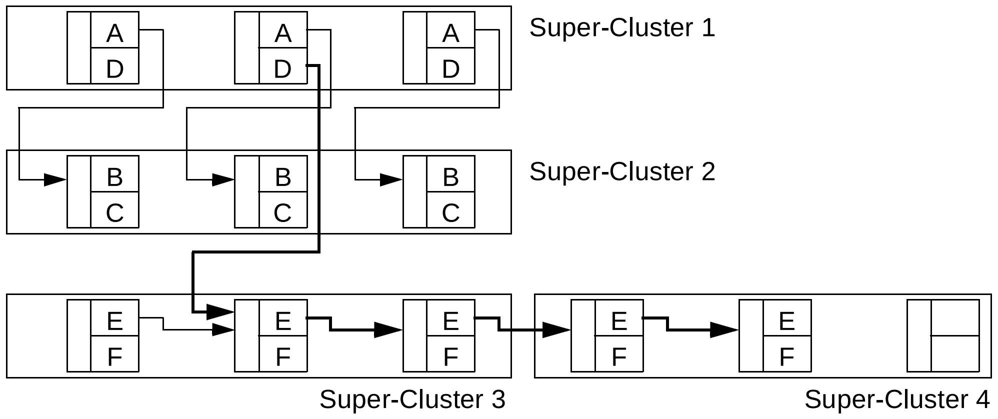


Figure 5.3: A Better Packing Solution

Note that LUTs on the part of the critical path that connects bit-slice 5, 6, 7, and 8 together are packed into separate clusters in order to preserve datapath regularity. Also, the carry chains in the super-clusters are used to improve the performance of this section of the critical path. Although this is a simple example, it illustrates the essence of the problem that arises from the need of preserving datapath regularity.

Common approaches used for packing of other FPGA architectures are not suitable for the super-clusters of the MB-FPGA architecture. VPACK [Betz97a], T-VPACK [Marq99], RPACK [Bozo01], and T-RPACK [Bozo01] are all ineffective because, as discussed in Chapter 2, they are all designed for purely cluster based FPGA architectures. Each algorithm can only pack one LUT and one DFF at a time while completely ignoring the regularity of the datapath circuits.

5.3 General Approach and Problem Definition

The input to the CNG packing algorithm is derived from the output of the EMC algorithm as described in Chapter 4, which consists of a netlist of LUTs and DFFs. The netlist also contains a description of the boundaries of each bit-slice in the input circuit. This netlist is first processed to create BLEs by iterating through each of its bit-slices. Within a bit-slice, a LUT is grouped with a DFF to form a BLE if the connection between the LUT and the DFF has the following three properties:

1. The input of the DFF is directly connected to the output of the LUT.
2. The DFF is the only sink of the LUT output.
3. The DFF is in the same bit-slice as the LUT.

Otherwise, the LUT is assigned to a BLE by itself. Each DFF that cannot be grouped with any LUTs based on the three properties above is also assigned to a BLE by itself. Note that property 1 and 2 are the criteria that the more traditional packing algorithms, including VPACK, T-VPACK, RPACK, and T-RPACK, use to group LUTs and DFFs into BLEs. Property 3 is added to ensure the preservation of datapath regularity during the grouping process.

After the creation of BLEs, the task of the CNG packing algorithm is to assign each BLE first to a super-cluster of the MB-FPGA and then to a cluster in the super-cluster. Each super-cluster is assumed to have a fixed number, M, of clusters. Each cluster is assumed to contain a fixed number, I, of cluster inputs and a fixed number, N, of BLEs. The top priority of the assignment is to preserve datapath regularity of the input circuit. When possible, the assignment should also minimize the total number of super-clusters used (to minimize the implementation area) and the critical path delay of the input circuit.

5.4 Datapath Circuit Representation

Since the primary purpose of the CNG packing algorithm is to preserve datapath regularity, an appropriate format for specifying datapath regularity must be defined for the packer. The format used in this chapter consists of a graph, G(V,A), which is called the *coarse-grain node graph.* The nodes, V, of G(V,A), represent the BLEs; and the edges, A, of G(V,A) represent the two-terminal connections that connect the BLEs together. Each node of G(V,A) can contain either one or several identical BLEs; and the number of BLEs contained in the node is called the *granularity* of the node. A node containing one BLE is called a *fine-grain node;* and it represents a BLE that does not belong to any datapath component. A node containing more than one BLE, on the other hand, is called a *coarse-grain node;* and each BLE in the node is from a unique bit-slice of a datapath component.

An example of the coarse-grain node graph is shown in Figure 5.4, which represents the datapath circuit shown in Figure 5.5. The graph consists of 11 interconnected nodes representing the 25 BLEs in the circuit. Nodes A through F are 3-bit wide coarse-grain nodes. A, B, C, and D represent the corresponding BLEs in bit-slices 1, 2 and 3, while nodes E and F represent BLEs labeled E and F in bit-slices 4 and 5, respectively. E’ and F’ are 2-bit wide coarse-grain nodes. E’ represents the two BLEs labeled E in bit-slices 7 and 8, while F’ represents the BLEs labeled F in the same two bit-slices. Finally nodes G, H, and I are fine-grain nodes, which represent BLEs with the corresponding labels in the irregular logic part of the circuit.

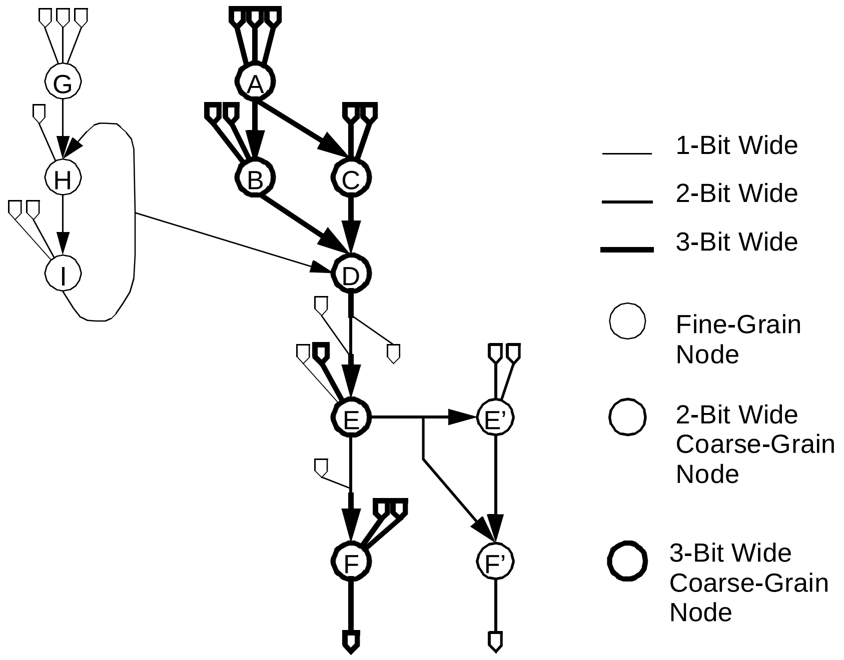


Figure 5.4: Coarse-Grain Node Graph

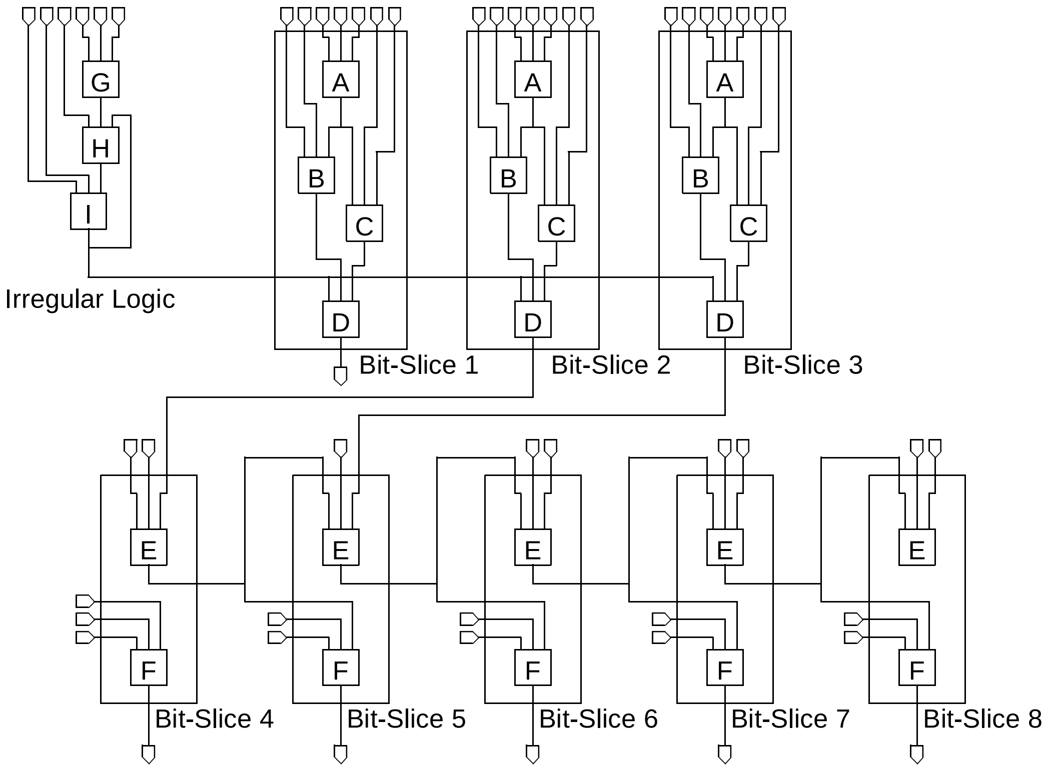


Figure 5.5: Datapath Circuit Represented by the Coarse-Grain Node Graph

5.5 The CNG Packing Algorithm

The overall flow of the packing algorithm is shown in Figure 5.6. It consists of two major steps. In the first step, initialization, the algorithm adjusts the granularity of the coarse-grain node graph and performs timing analysis on the input circuit. In the second step, packing, the algorithm groups nodes into super-clusters. Note that this algorithm is derived from the T-VPACK algorithm [Marq99], which is modified to accommodate the unique features of the MB-FPGA super-clusters. The basic principles described here, however, can also be employed to transform other packing algorithms, including VPACK [Betz97a], RPACK [Bozo01], and T-RPACK [Bozo01], into datapath-oriented algorithms.

5.5.1 Step 1: Initialization

Step 1, initialization, consists of two sub-steps. First, each coarse-grain node whose granularity value is greater than the granularity value of the target architecture (such as the M value for the MB-FPGA presented in Chapter 3) is transformed into a set of nodes. Each node in the set has a granularity value that is smaller than or equal to the granularity of the super-clusters. Timing analysis is then performed on the input circuit. Each of these sub-steps is described in turn.

5.5.1.1 Breaking Nodes

Given a coarse-grain node that is more than M bits wide, the breaking nodes function starts at the most significant bit of the node and continuously groups M neighboring BLEs into new coarse-grain nodes. If there are less than M BLEs remaining at the least significant end, these remaining BLEs are grouped by themselves into a coarse-grain node that is less than M-bit wide. These newly formed nodes are then used to substitute the original node in the coarse-grain node graph.

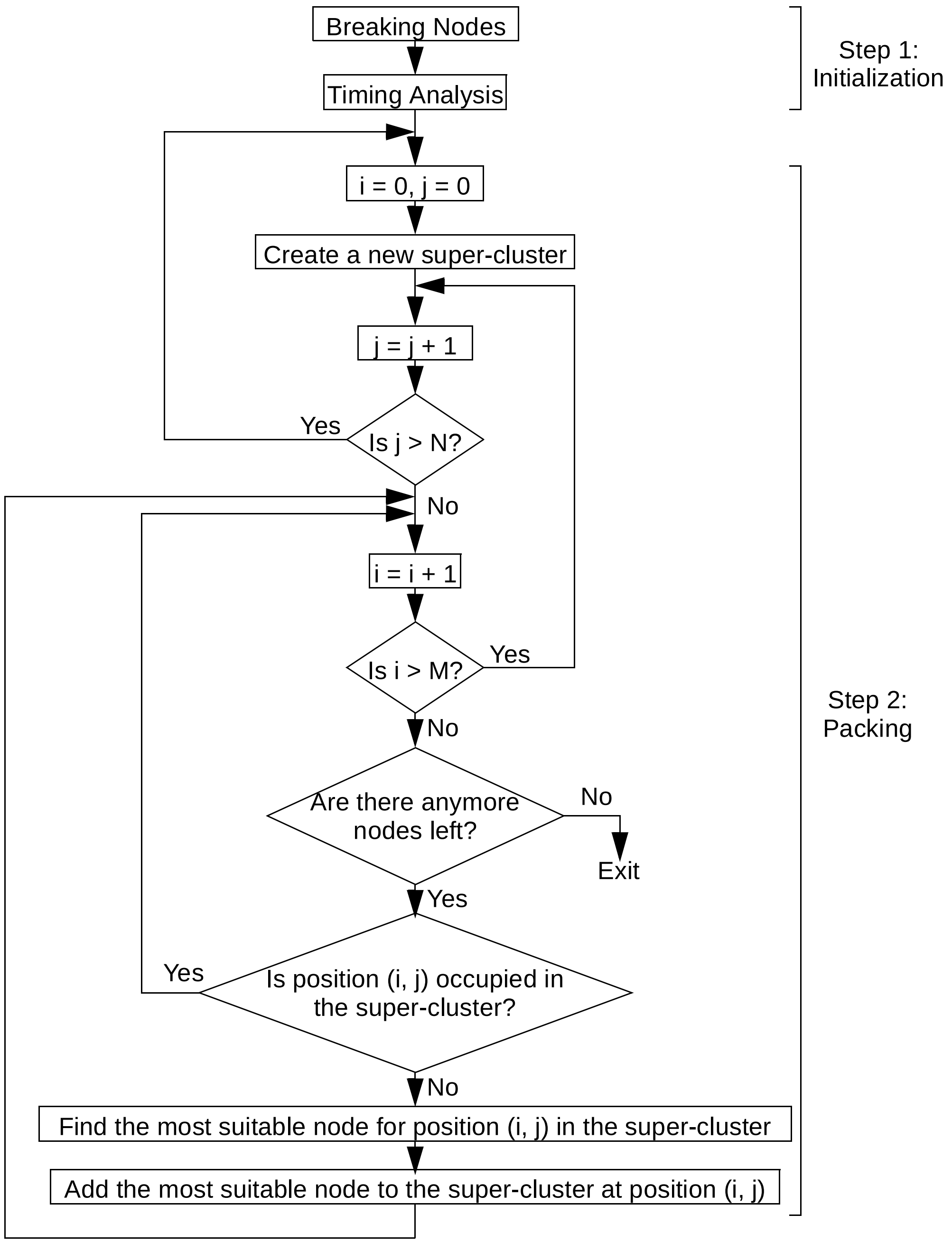

Breaking Nodes
Timing Analysis
Step 1: Initialization
i = 0, j = 0
Create a new super-cluster
j = j + 1
Is j > N?
Yes
No
i = i + 1
Is i > M?
Yes
No
Are there anymore nodes left?
No
Exit
Yes
Is position (i, j) occupied in the super-cluster?
Yes
No
Find the most suitable node for position (i, j) in the super-cluster
Add the most suitable node to the super-cluster at position (i, j)
Step 2: Packing

Figure 5.6: Overview of the CNG Packing Algorithm

5.5.1.2 Timing Analysis and Criticality Calculation

During timing analysis, the propagation delay and the expected arrival time of each BLE input or output pin is first calculated for the input circuit. The slack of each net is then derived from the delay and the expected arrival time metrics. For all of the above timing calculations, the logic delay of each BLE is set to be 1 time unit and the propagation delay of each two-terminal connection that connects two BLEs together is set to be 10 time units. Finally, the clock cycle time of the input circuit is set to be the delay of the most critical path of the circuit. These values were shown to generate good packing results for the T-VPACK algorithm in [Marq99] from which the current algorithm is derived.

The final step of the timing analysis calculates the criticality value for each net in the input circuit as defined in [Marq99]. For completeness, the criticality definition is briefly summarized here. The criticality is used to represent the slack information in a normalized form. The formula for calculating a criticality value from a corresponding slack value is shown in Equation 5.1. Here, the slack value is normalized by dividing into the maximum slack of the circuit, max_slack. The normalized value is then subtracted from one to derive the corresponding criticality value. For nets that are on the critical path of a circuit, whose slack value is zero, the corresponding criticality value is one. Less critical nets have smaller criticality values; and the least critical nets, whose slack values are equal to max_slack, have a corresponding criticality value of zero.

$$\text{criticality} = 1 - \frac{\text{slack}}{\text{max\_slack}} \qquad \text{Equation 5.1}$$

5.5.2 Step 2: Packing

During step 2, new super-clusters are created one at a time and each super-cluster is filled with nodes from the coarse-grain node graph. Nodes are added to a super-cluster in a predetermined order. Assuming that the *i*th BLE in the *j*th cluster is denoted by the pair of integers (i, j), a node is added to position (1, 1) first. Then, as shown by Figure 5.7, nodes are sequentially added to positions (2, 1), (3, 1), ..., (N, 1), (1, 2), (2, 2), ... (N, 2), ..., (1, M), (2, M), ... (N, M), if these positions are not already occupied by BLEs.

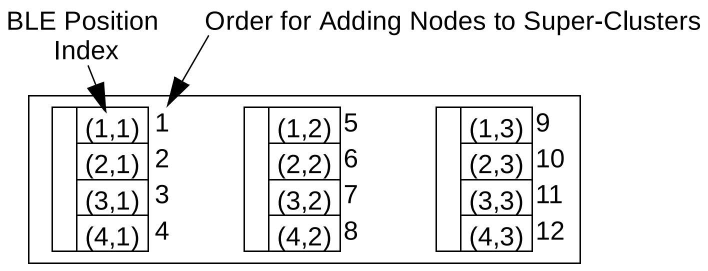


Figure 5.7: Order for Filling Super-Cluster with N = 4, M = 3

This order of adding nodes to super-clusters guarantees that if no BLEs have been added to position (i, j), BLE positions (i, j + 1), (i, j + 2), ..., (i, M) will also be unoccupied. To find the most suitable node for BLE position (i, j), CNG first finds all nodes whose granularity is less than M - j + 1. If (i, j) is equal to (1, 1), then the seed criticality function is used to select the most suitable node from this group of nodes. Otherwise, the attraction criticality function is used instead. Once the most suitable node with a granularity value of m is determined, the BLEs in this node are added to consecutive BLE positions (i, j), (i, j + 1), ..., (i, j + m - 1) with the least significant BLE added to position (i, j) and the most significant BLE added to position (i, j + m - 1).

Note that because of the carry network, not all BLE positions in an MB-FPGA cluster are equivalent. This lack of equivalency is the reason why CNG must select nodes for each specific positions in a super-cluster. An example is shown in Figure 5.8. Here there are three BLEs, A, B, and C, in a super-cluster. These BLEs are connected by a carry chain through the carry network. In the figure, the BLE position (1, 1) is equivalent to the BLE position (3, 1); therefore, BLE A can be moved to position (3, 1) provided that BLEs B and C are also moved to position (3, 2) and (3, 3) respectively. However, BLE A cannot be moved to position (2, 1) or (4, 1) since these two positions are not equivalent to BLE position (1, 1) due to the difference in their carry connections.

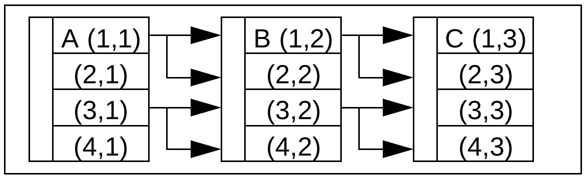


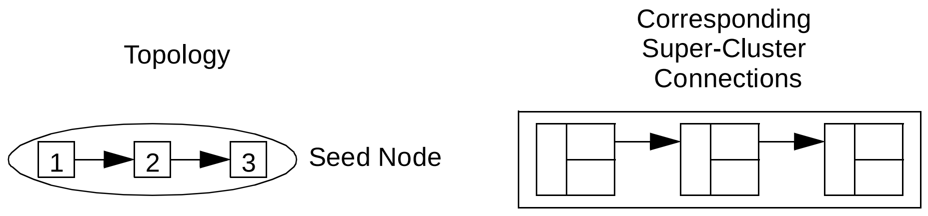
Figure 5.8: Equivalence of BLEs in Clusters

The remainder of this section describes the two criticality functions, including the seed criticality function and the attraction criticality function, which are used in the packing process. Each of these functions is described in detail in turn.

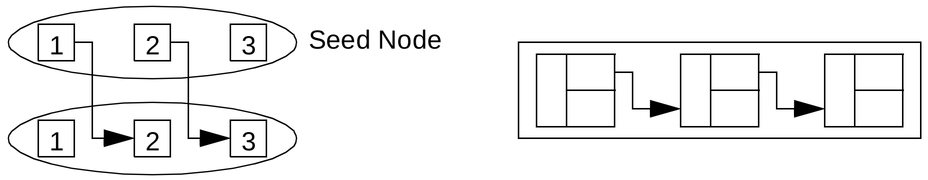
5.5.2.1 Calculating Seed Criticality

The first node added to a super-cluster is called a seed. This seed node is selected using a metric called the seed criticality metric, which measures the maximum possible speed improvement of building super-clusters based on one particular node as the seed node. Implementing a seed node in a super-cluster by itself does not necessarily improve the performance of a circuit; however, when a subsequent node, A, is added to the same super-cluster, many two-terminal connections that connect the seed node and node A together can then be implemented in the local routing networks or the carry network of the super-cluster. Because these networks are inside the super-clusters, they are inherently much faster than the global routing network. Consequently, the performance of the circuit is improved. Note that each of the two-terminal connections that can be implemented inside the super-cluster is called a *potential local connections* of the seed node.

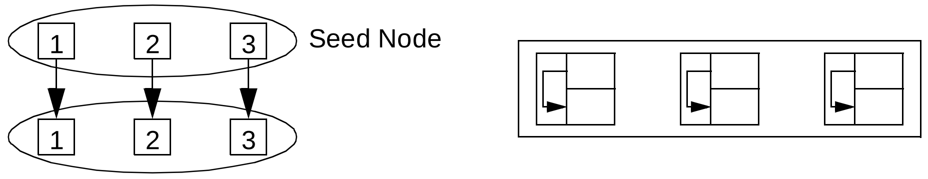
Potential local connections can be identified using a pattern matching process. This process first labels all the BLEs in each node consecutively from 1 to m, where m is the granularity of the node. The BLE at the least significant position is labeled 1; and the BLE at the most significant position is labeled m. Potential local connections are then identified by matching each two-terminal connection of the seed node against the four topologies shown in Figure 5.9. If the connection matches one of the topology, then it is a potential local connection. Otherwise it is not.



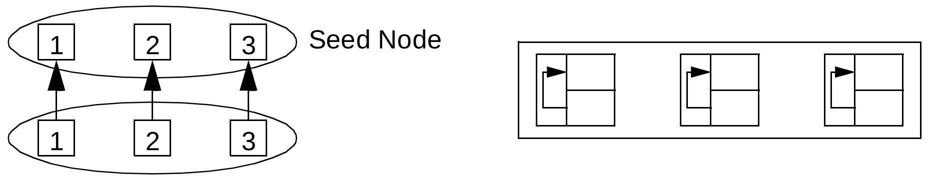
(a) Two Two-Terminal Connections in Topology 1 Configuration



(b) Two Two-Terminal Connections in Topology 2 Configuration



(c) Three Two-Terminal Connections in Topology 3 Configuration



(d) Three Two-Terminal Connections in Topology 4 Configuration

Figure 5.9: Topology for Identifying Potential Local Connection

Figure 5.9a shows two two-terminal connections in the configuration of topology 1. One connection connects BLE 1 to BLE 2, while the other connection connects BLE 2 and BLE 3 together. In general, in topology 1, both the source BLE and the sink BLE of the two-terminal connection is in the seed node. The index of the source BLE is one less than the index of the sink BLE. As shown by Figure 5.9a, topology 1 connections can be implemented in the carry networks of the super-clusters.

Figure 5.9b and Figure 5.9c shows connections in topology 2 and topology 3 configurations, respectively. As shown, in topologies 2 and 3, the source and the sink BLEs exist in two distinct nodes; and for either topology 2 or 3 the source BLE is always in the seed node. Furthermore, for topology 2, the index of the source BLE is one less than the index of the sink BLE. For topology 3, on the other hand, the source BLE and the seed BLE have the same index. Topology 2 can be implemented in the carry networks; and topology 3 is suitable for implementation in the local routing networks of the super-clusters. Finally, topology 4 is almost identical to topology 3 with the exception that the sink BLE is in the seed node.

$$\text{seed\_criticality}(n) = \max(S(n)) + \varepsilon \times \text{cnt}(S(n)) + \varepsilon^2 \times d(n) \quad (\varepsilon \ll 1) \qquad \text{Equation 5.2}$$

The formula for calculating seed criticality is shown in Equation 5.2. In the equation, the function max(X) returns the maximum value in a set, X, of real numbers. Function cnt(X), returns the number of elements that are equal to max(X) in the set X. S(n) is the complete collection of all the net criticality values from all the potential local connections of node n.

The function, max(S(n)), corresponds to the maximum speed improvement achievable by implementing n as a seed node. cnt(S(n)) is a tie breaker; and it counts the number of potential local connections that can achieve the maximum speed improvement. Note that max(S(n)) and cnt(S(n)) are analogous to the base seed criticality and the number of path affected metrics used in [Marq99], respectively. These functions, however, are more general in nature and are applicable to a wider range of FPGA clustering architectures than the fully connected topology assumed by [Marq99].

The metric distance to source, d(n), on the other hand, is an unmodified version of the same metric defined in [Marq99], which is described in detail in [Marq99]. Nodes with the same max(S(n)) values usually are connected together by a single critical path. d(n) measures the order of these nodes along the critical path. Everything else being equal, the node that is the furthest from the source of the critical path is given the highest priority for implementation as a seed node. This allows the packing process to start packing from one end of the critical path instead of from the middle. As suggest by the study done in [Marq99], this packing order minimizes the overall critical path delay of a circuit.

5.5.2.2 Calculating Attraction Criticality

Once a seed is added to a super-cluster, CNG fills the super-cluster based on the attraction criticality metric. Each node in the coarse-grain node graph has an attraction criticality metric as shown in Equation 5.3. The metric consists of four parts: the base seed criticality, B(n), accounts for the performance improvement of implementing a node in a super-cluster; shared I/O count, C(n), accounts for the number of additional cluster I/Os that is needed to implement a node in a super-cluster; and finally secondary attraction criticality, B_p(n), and common I/O count, C_p(n), account for the closeness of the placement resulting from adding a node to a super-cluster. These four parts are weighted and summed into the attraction criticality metric. Each of these parts are described in turn.

$$\begin{aligned}&\text{attraction\_criticality}(n) = \\ &\alpha \times \left(\beta \times B(n) + (1-\beta) \times \frac{C(n)}{P_{max}}\right) + \\ &(1-\alpha) \times \left(\tau \times B_p(n) + (1-\tau) \times \frac{C_p(n)}{M \times P_{max}}\right)\end{aligned}$$

Equation 5.3

Base Seed Criticality

As shown in Figure 5.10, after a node is added to a super-cluster, the connections between the node and the super-cluster can be classified into two types. The first type consists of connections that can be implemented in the local routing networks of the clusters or the carry network that connects the clusters together. The second type consists of connections that have to be routed through the global routing network. The implementation of the first type of connections often results in increased performance; and this increase in performance is measured by the base attraction criticality in Equation 5.3. It is equal to the maximum criticality among all type one connections in addition to all internal connections of the node that can be implemented in the carry network.

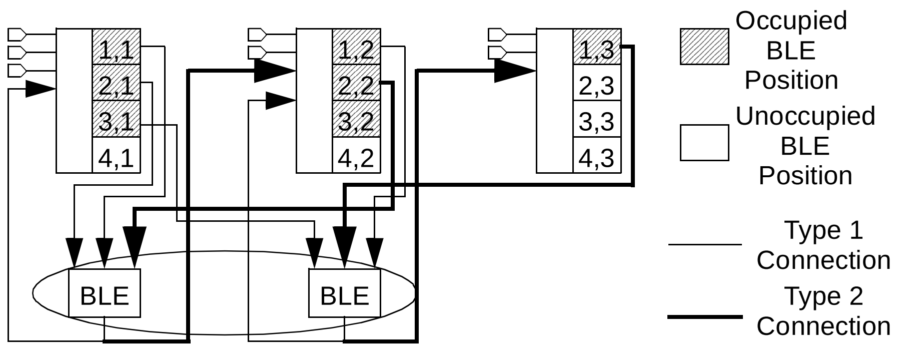


Figure 5.10: Adding a Node to a Super-Cluster at Position (4,1)

Secondary Attraction Criticality

Adding a node to a super-cluster also makes all BLEs in the node physically closer to the BLEs in the super-cluster. This physical closeness makes type two connections potentially much faster than any connection that connects two separate super-clusters. The secondary attraction criticality is used to measure this speed up. It is equal to the maximum criticality among all type two connections in addition to all internal connections of the node that must be routed through the global routing network.

Shared I/O Count

Since cluster inputs are limited routing resources, it is important to minimize the usage of cluster inputs when adding nodes to a super-cluster. As described in [Marq99], it is preferable to choose BLEs with the following three types of I/Os for a cluster:

1. a BLE input that is connected to the same net as one of the cluster inputs
2. a BLE input that is connected to one of the cluster outputs
3. a BLE output that is connected to a cluster input

When adding a BLE with type 1 or 2 inputs to a cluster, the duplicated BLE inputs do not require additional cluster inputs. Adding a BLE with a type 3 output to a cluster eliminates the cluster input that is connected to the output of the BLE. The shared I/O count metric measures the I/O commonalities between a node and a super-cluster. It is equal to the total number of the three types of BLE I/Os in a node when each BLE is matched with its corresponding cluster. Note that, in Equation 5.3, P_{max} is defined to be the maximum possible value of the shared I/O count metric. It is used in the equation to normalize the shared I/O count to a value that is between 0 and 1.

Common I/O Count

Adding a node to a super-cluster might increase the number of common I/O signals shared by various clusters of a super-cluster. As shown in Figure 5.11, for two clusters from the same super-cluster, routing an input signal that is shared by the two clusters usually requires less resources than routing two distinct inputs. Similarly, routing the output of one cluster to another requires less routing resources if both clusters are in the same super-cluster. The common I/O count is used to account for this increase in routing efficiency. It is analogous to the shared I/O count metric. However, instead of measuring the number of I/O signals that are in common between each BLE and its corresponding cluster, the common I/O count is equal to the total number of BLE I/Os in a node that is in common with all the I/Os of a super-cluster excluding the signals that have already been counted by the shared I/O count metric.

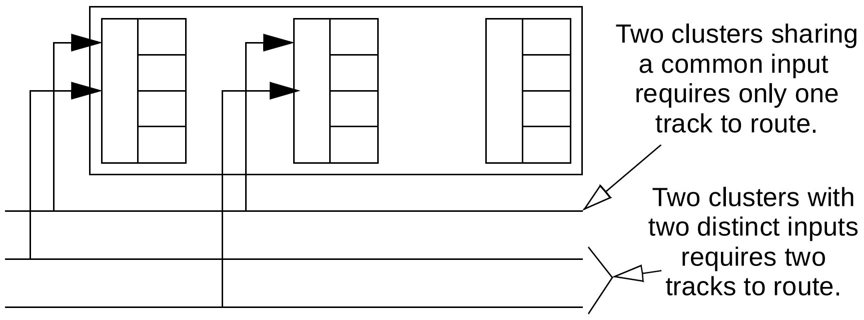

Two clusters sharing a common input requires only one track to route.
Two clusters with two distinct inputs requires two tracks to route.

Figure 5.11: Common Inputs Between Clusters in a Super-Cluster

Note that for all experiments performed in this thesis, α, β, and τ are set to be 0.85, 0.75, and 0.75 respectively. These values are experimentally shown to generate good packing results when the CNG algorithm is used to pack the 15 benchmark circuits from the Pico-Java processor [Sun99].

5.6 Results

CNG has been used to pack several benchmark circuits into various super-cluster architectures of MB-FPGA. The packing results shown in this section are based on the fifteen datapath circuits from the Pico-Java Processor from Sun Microsystems [Sun99]. Each circuit is first synthesized into several granularity values using the EMC algorithm as described in Chapter 4. Table 1 gives the name, size (number of BLEs) of each circuit for a given granularity value. For each synthesis granularity, the synthesized circuits are packed into a corresponding super-cluster architecture with the same granularity value. The detailed structure of these super-cluster architectures is described in detail in the next sub-section. The packing results for regularity, area, and performance are then described in turn.

Circuit Name	Number of BLEs Obtained by Each Synthesis Granularity									
	1	2	4	8	12	16	20	24	28	32
code_seq_dp	362	364	364	364	364	364	364	364	364	364
dcu_dpath	958	962	966	974	982	974	978	982	986	990
ex_dpath	2823	2747	2649	2719	2947	2955	2942	2938	2918	2938
exponent	467	517	517	539	567	565	565	565	565	565
icu_dpath	3254	3237	3245	3245	3273	3277	3281	3285	3289	3294
imdr_dpath	1286	1268	1255	1286	1288	1283	1291	1296	1294	1297
incmod	870	862	867	940	948	1005	1000	995	993	1021
mantissa_dp	912	919	942	966	971	982	983	983	977	983
multmod_dp	1602	1636	1634	1636	1636	1636	1653	1635	1637	1637
pipe_dpath	452	499	452	503	503	501	483	503	503	501
prils_dp	363	396	393	385	385	393	389	385	385	409
rsadd_dp	350	314	313	305	305	305	305	305	305	305
smu_dpath	561	557	557	560	563	561	543	563	563	561
ucode_dat	1264	1273	1304	1278	1282	1286	1290	1294	1298	1301
ucode_reg	78	80	82	86	86	94	90	86	106	110

Table 1: Experimental Circuits

5.6.1 Super-Cluster Architectures

The overall structure of a super-cluster used in the tests of the CNG algorithm contains a variable number, M, of clusters, where M is the granularity of the MB-FPGA architecture. Each cluster of the super-cluster contains 10 (I) input pins, a fully connected local routing network, and 4 (N) BLEs. An I value of 10 and an N value of 4 were chosen because they were shown to be the most efficient for the clusters of conventional FPGA architectures in [Betz97b] [Betz98] [Betz99a]. Several values of M were investigated. These values are the same as the ones shown in Table 1, namely 1, 2, 4, 8, 12, 16, 20, 24, 28, and 32.

5.6.2 Regularity Results

Two yardsticks are used to measure the amount of regularity preserved by the CNG packer. The first yardstick measures the percentage of BLEs in all datapath components of width M, where M is the number of clusters in a super-cluster. The second yardstick measures the percentage of BLEs in all datapath components which are at least 2-bit wide. Figure 5.12 plots these two metrics against the granularity of the target super-clusters over all benchmark circuits. As shown, over 85% of BLEs are in at least 2-bit wide datapath components regardless of the granularity of the target super-clusters. The percentage of BLEs in M-bit wide datapath components drops from over 90% when M is equal to 2 to slightly over 40% when M is equal to 20. The metric then increases again to slightly less than 55% when M is further increased to 32. (Note that the underlying cause of the variation of this metric is discussed in detail in Chapter 7.) These are excellent results especially considering that the conventional packers preserve little regularity. Note that since the packer does not break up coarse-grain nodes that represent datapath components when the synthesis granularity and the packing granularity are equal, any logic regularity from the synthesis process is completely preserved by the packer in these experiments.

5.6.3 Area Results

One penalty that the CNG algorithm pays in preserving datapath regularity is in area inflation. The average area consumed by super-clusters over all the benchmark circuits is shown in Figure 5.13, which plots the arithmetic average across the fifteen benchmark circuits against the granularity of the target super-clusters. Note that the area of each super-cluster is measured using the minimum-width transistor area as defined in [Betz99a], which was discussed in detail in Chapter 2. As shown, when M is equal to 1, the packer does not preserve any datapath regularity and is able to achieve the lowest area. The area slowly increases when

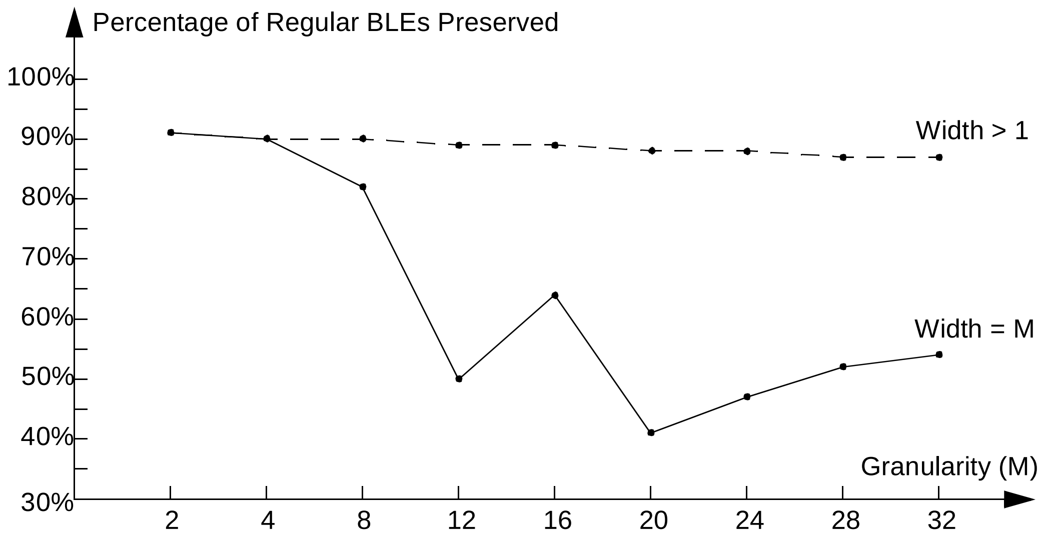


Figure 5.12: Regularity vs. Granularity

M is increased from 1 to 4. Then it rises quickly when M is increased from 4 to 12. The rate of increase then slows when M is further increased to 32. Overall the maximum area inflation occurs when M is equal to 32. Here the area is 18% larger than the area obtained when no datapath regularity was preserved (when M is equal to 1).

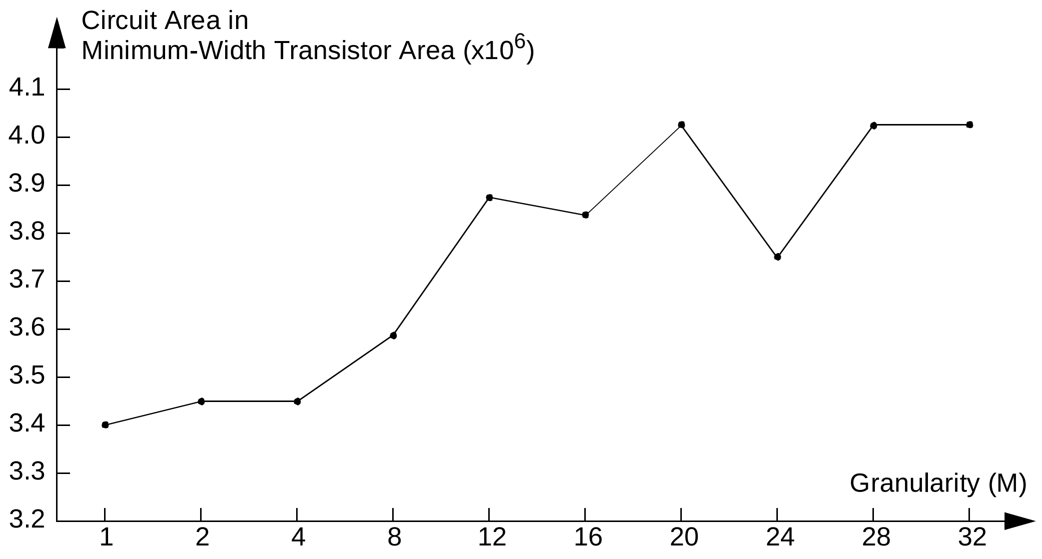


Figure 5.13: Area vs. Granularity

The reason for this increase in area is that as the granularity of packing increases, the packer has to deal with an increasing variety of datapath width. Geometrically, it is much harder to fit datapath components with a large variety of width into the structure of the super-clusters. Furthermore, at high granularity values, one super-cluster contains many more BLEs than super-clusters with lower granularity values. This reduces the number of super-clusters needed to pack a circuit; however, it also increases the number of wasted BLEs if a super-cluster is only partially filled.

Overall, the data in Figure 5.13 shows that the granularity values of 2, 4, and 8 represent a reasonable trade-off between regularity and area inflation. At these values, the packer preserves a large amount of regularity while only giving up between 1% to 6% of area.

5.6.4 Performance Results

The CNG packing results show that the preservation of regularity benefits the overall logic performance of datapath circuits when the carry networks is significantly faster than the local routing networks. This is true for the MB-FPGA architecture, since carry networks are more directly connected to the BLE inputs as compared to the local routing networks. The same is true for many commercial FPGAs where the speed of the carry networks is much faster than local routing [Alte02] [Xili02]. If the speed of the carry networks is similar to the speed of the local routing networks, on the other hand, the logic performance of the circuits degrades when regularity is preserved.

The geometric average of logic delay over the fifteen benchmark circuits are shown in Figure 5.14. Note that to calculate logic delay, the delay of the global routing network is set to be zero. There are two curves in the figure. The first curve labeled slow carry assumes that each two-terminal connection in both the local routing networks and the carry networks have 1 time unit of delay. The logic delay through a BLE is also assumed to be 1 time unit. The second curve labeled fast carry, on the other hand, assumes that the carry networks are much faster than the local routing networks. A connection in the carry networks is assumed to have

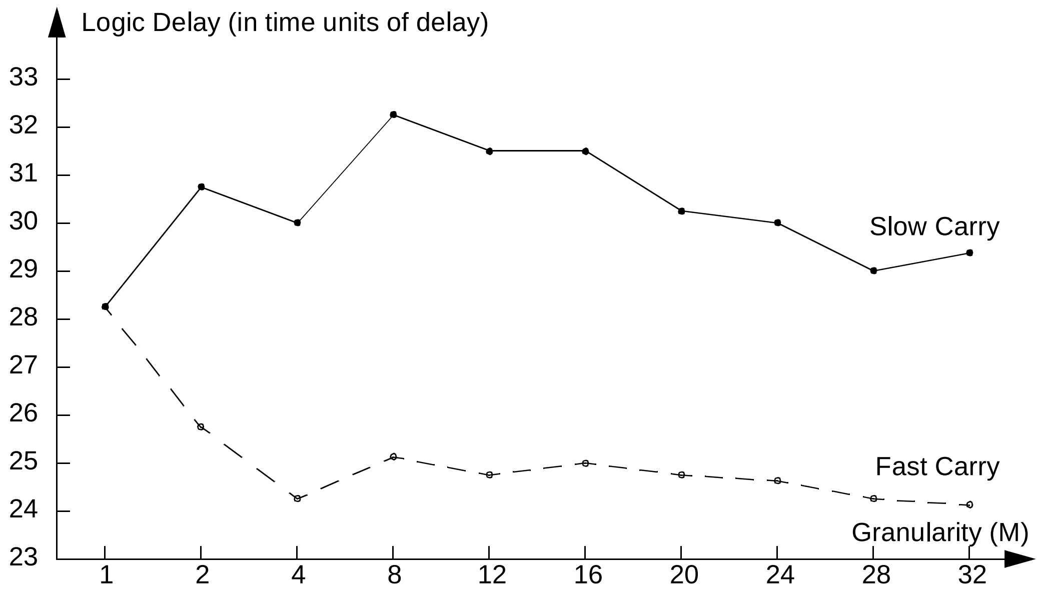


Figure 5.14: Delay vs. Granularity

0.01 time units of delay; and a connection in the local routing networks is assumed to have 1 time unit of delay. The logic delay through a BLE is still assumed to be 1 time unit.

The slow carry curve increases from 28.2 time units to slightly over 32 time units as M is increased from 1 to 8. The curve then decreases to around 29 time units as M is further increased to 32. The fast carry curve, on the other hand, decreases significantly from 28.2 to 24.3 when M is increased from 1 to 4. As M is further increased to 32, the curve stays within the range of 24 to 25 time units. These data demonstrate the benefit of identifying regularity and the carry chains associated with it for architectures with fast carry connections. For the benchmark circuits, a speed improvement around 14% is achieved for fast carry architectures. If the carry connections are not significantly faster than the connections in the local routing networks, however, there is a speed penalty of around 15% for the benchmark circuits.

5.7 Conclusions and Future Work

This chapter has described a new kind of packing algorithm that is designed specifically for preserving datapath regularity. The algorithm treats several BLEs, each from a unique bit-slice of a datapath, as a single group, and considers the packing options of the entire group at a time. The algorithm also uses various techniques to accommodate the special architectural features of the MB-FPGA architecture. The algorithm is able to preserve a large amount of datapath regularity and achieves good performance results while incurring a small amount of area inflation in logic area.

Future research should try to further reduce the logic area inflation of the algorithm. It also should find ways of further improving the performance of the packed circuits.

6 A Datapath-Oriented Routing Algorithm

6.1 Introduction

This chapter presents a new datapath-oriented routing algorithm that has been designed specifically for the MB-FPGA routing architecture described in Chapter 3. The algorithm is unique in that it balances the usage of conventional fine-grain routing tracks with datapath-oriented CMS routing tracks, allowing the MB-FPGA architecture to achieve maximum area savings over a wide range of datapath circuits. This algorithm and software is used in Chapter 7 and Chapter 8 to investigate the effect of CMS routing on the area-efficiency of FPGAs.

The routing problem for MB-FPGA is more difficult than classic routing because there are two distinct types of routing tracks and each type is designed for a specific purpose. The CMS routing tracks are primarily designed for routing a group of regularly structured connections from a common source to a common sink; the fine-grain routing tracks are designed to route irregular connections that cannot be grouped into any regular groups. It is usually beneficial to use each type of routing track for its intended purpose; however, when one type of routing track is over-subscribed, it might be beneficial to route the intended connections through the other type of routing track. As a result, groups of regular connections and individual irregular connections might compete for the same type of routing track. Resolving such competition is essential to achieving 100 percent routing completion. The algorithm described here, called the Coarse-Grain Resource (CGR) routing algorithm for MB-FPGA, addresses the issue of the balanced use of routing tracks by carefully considering the usage of each type of track and compensating the overused type with the under-used type. The algorithm also has the ability to optimize the routing delays of time-critical connections, by assigning the fastest paths to the connections that need them the most.

CGR has been used to obtain excellent routing results for the datapath circuits from the Pico-Java processor. The results show that CGR is able to route relatively large MB-FPGA architectures containing various proportions of CMS routing tracks; and it is able to effectively translate the regularity in inter-super-cluster connectivity into area savings.

This chapter is organized as follows: Section 6.2 motivates the development of the datapath-oriented router; Section 6.3 describes the placement algorithm that is used in conjunction with the CGR router; Section 6.4 defines the routing problem; Section 6.5 presents the model used to represent MB-FPGA architectures containing CMS routing tracks; Section 6.6 describes the CGR routing algorithm in detail; Section 6.7 presents the results from the tests of the router; and Section 6.8 gives concluding remarks.

6.2 Motivation

A key problem in datapath-oriented routing of MB-FPGA is to decide if a group of regular connections should be routed through the CMS routing tracks or the fine-grain routing tracks. This will be illustrated with a contrived example shown in Figure 6.1. It shows three views of the same section of an FPGA. The first view gives the routing options for a group of four regular connections that connect the super-cluster on the top-left corner to the super-cluster on the bottom-right corner of the figure. These four connections are collectively labelled as connection bus A. The second and the third view in the figure give the routing options for two individual connections labelled B and C respectively. In the figure, a routing switch is shown as an X, a wiring segment as a dotted line, and a possible route as a solid line. There are nine wire segments in the routing channel. Wire segments 1 through 4 are from a group of four CMS routing tracks. These four tracks form a single CMS routing bus by sharing a single set of configuration memory. Wire segments 5 through 9 are from five independent fine-grain routing tracks.

Now, assume a router that always routes groups of regular connections through the CMS routing tracks. Since wire segments 1, 2, 3, and 4 are the only CMS tracks, they must be selected for connection bus A. Then one of the connections B and C cannot be routed because they both rely on the same single remaining option, namely the wire segment numbered 5. Under these circumstances, a more flexible router would choose the fine-grain wire segment 6, 7, 8, and 9 for connection bus A. Connection B then can be routed through wire segment 1 of the CMS routing tracks. Since wire segment 2, 3, and 4 share the same set of configuration memory as wire segment 1, they also become unavailable to route the remaining connection. Finally connection C can be routed through wire segment 5. Although this is a simple example, it illustrates the essence of the problems that occur because of the differentiation of routing tracks into fine-grain tracks and CMS tracks.

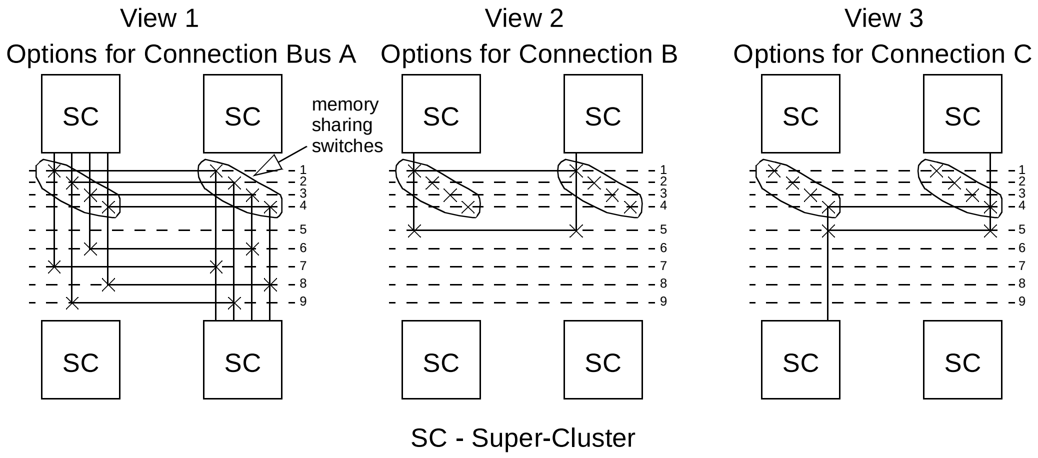


Figure 6.1: Example of Contention Between CMS and Fine-Grain Nets

Common approaches used for routing in other FPGA architectures are not suitable for MB-FPGA. Maze Router [Lee61], CGE [Brow92a] [Brow92b], Pathfinder [Ebel95], VPR [Betz99a] [Swar98], and NC [Chan00] are ineffective because, as shown in Chapter 2, none of them is equipped to deal with the CMS routing tracks. Each algorithm assumes the routing channels contain only fine-grain routing tracks, which do not share any configuration memory; and each algorithm completely ignores datapath regularity.

6.3 The MB-FPGA Placer

The input to the CGR routing algorithm is derived from the output of the CNG packing algorithm as described in Chapter 5. It consists of a netlist of super-clusters. These super-clusters are first placed by a datapath-oriented placer that employs the simulated annealing algorithm. The placer, called the MB-FPGA placer, is a modified version of the VPR placer [Betz99a] [Marq00a] as described in Chapter 2. It uses exactly the same annealing schedule and the same cost functions as the ones used by the VPR placer. It differs from the VPR placer in how clusters are moved during the annealing process. This difference arises from the need of preserving datapath regularity.

Like the VPR placer, the MB-FPGA placer first assigns each super-cluster to a random location. Then it iteratively improves the initial placement by moving either clusters or super-clusters. This ability of the MB-FPGA placer to move logic blocks from two different hierarchies — the cluster level and the super-cluster level — constitutes the major difference between it and the VPR placer, which only can move a single hierarchical level of logic blocks, namely the clusters. Moving two levels of logic blocks gives the MB-FPGA placer the unique ability of concurrently preserving datapath regularity, by moving individual super-clusters that contain datapath components, as well as maximizing placement efficiency when possible, by moving individual clusters residing in super-clusters that contain only irregular logic.

To create a move, an initial cluster, is first randomly selected. If the initial cluster is a part of a datapath, the entire super-cluster containing the cluster is moved to a new, randomly selected, location. If this location is already occupied by another super-cluster, this super-cluster is then moved to the original location containing the initial cluster. If the initial cluster is not a part of a datapath, another cluster position is randomly selected. If this position is unoccupied, the initial cluster is moved to the position. Otherwise, the cluster occupying the position (called the target cluster) is swapped with the initial cluster if the target cluster is not a part of any datapath. If the target cluster is a part of a datapath, the super-cluster containing the initial cluster is swapped with the super-cluster containing the target cluster. Finally, the move is evaluated using the cost functions, and is either accepted or rejected based on the evaluation result and the annealing schedule.

6.4 General Approach and Problem Definition

After the placement, the routing problem is transformed into the following: for each two point connection, the router must first identify if the connection belongs to any group of regular connections. It then should choose specific routing resources to implement the two point connection. The algorithm should prefer CMS routing tracks if the connection is a part of a group of regular connections. Otherwise, fine-grain routing tracks should be used when available.

The CGR router takes a routing resource graph and a netlist as its input. The routing resource graph represents a target MB-FPGA architecture; and the netlist represents a circuit of interconnected super-clusters in the target architecture. The router then finds a feasible (defined below) routing solution in the routing resource graph for each net in the netlist. The solution should minimize the delay of input circuit, in terms of minimizing the maximum propagation time of all signals in the placed and routed input netlist, as well as the amount of resources consumed by the input netlist.

A routing solution is defined to be a collection of interconnected nodes and edges that contains the source node and all the sink nodes of a net. To be feasible, the occupancy value — the total number of times that a node appears in all routing solutions — of each node in the routing solution must be less than or equal to its capacity value. Otherwise, the routing solution is said to be infeasible. A net is said to be routed, once a routing solution (either feasible or infeasible) is found.

6.5 MB-FPGA Architectural Representation

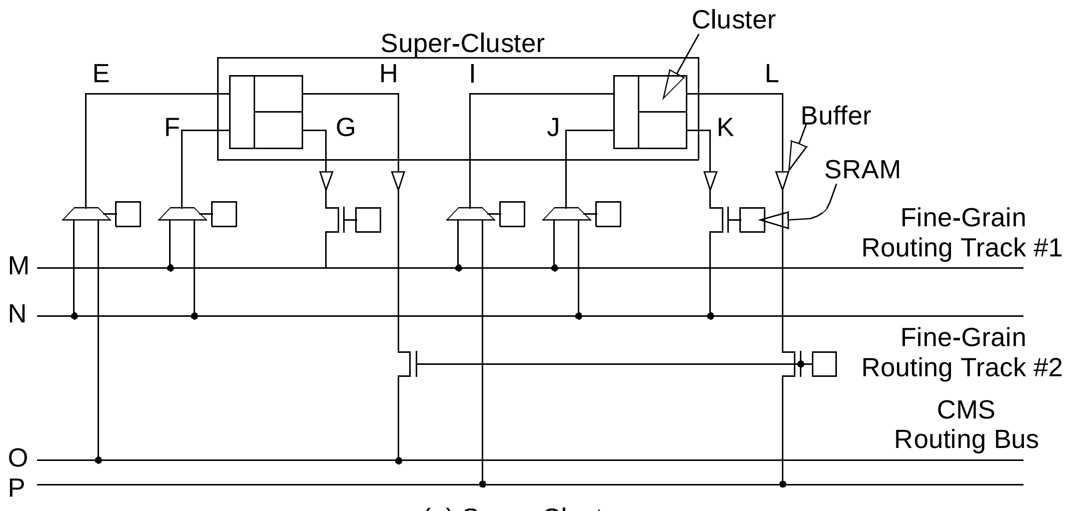
Since the primary purpose of the CGR routing algorithm is to provide a means of investigating the datapath-oriented routing architecture of MB-FPGA, an appropriate model must be defined to capture the unique features of MB-FPGA. The model that has been chosen is a *routing resource graph* G(V,A) consisting of a set of *nodes*, V, connected by *edges* A. Note that this model is similar to the routing resource graph of [Betz99a]; however, it is considerably more complex because it models the multi-bit logic, CMS routing tracks, and CMS switches, as well as the fine-grain resources. In the CGR routing resource graph, each node, V, represents a piece of routing resource. There are seven types of nodes, representing a signal source (called a *source node*), a signal sink (called a *sink node*), a super-cluster input pin, a super-cluster output pin, an input pin of an I/O block, an output pin of an I/O block, or a routing track (either a CMS track or a fine-grain track). Each edge in the routing resource graph represents a programmable routing switch.

Each node of the routing resource graph is associated with a *capacity value*. This value represents the maximum number of times that the routing resource represented by the node can be used in the final routing solution. For each pin or each routing track, this capacity value is one since each of these routing resources can only be legally used once. For each source or sink node, this capacity value is a positive number whose value depends on the equivalency of super-cluster output or input pins [Betz99a]. In general, if a source (sink) is connected to X super-cluster output (input) pins, these pins must be logically equivalent; and the capacity of the source (sink) is equal to X.

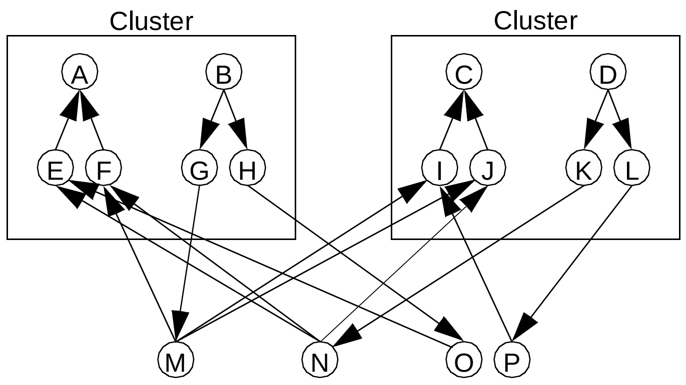
A super-cluster containing M clusters is modeled using M source nodes and M sink nodes. Each cluster is assigned one source node and one sink node. A source node is connected to all the output pins of its cluster; and its capacity is equal to the total number of BLEs (or output pins) in the cluster. A sink node is connected to all the input pins of its cluster; and its capacity is equal to the total number of cluster input pins.

Some of the nodes in a routing resource graph are grouped into *node-buses*. A node-bus can be M source nodes or M sink nodes of a super-cluster. It also can be M cluster input or output pins in an input bus or output bus, M routing tracks in a routing bus, M pad input pins or pad output pins in a pad-input bus or pad-output bus. (Recall that all the above bus types are defined in Chapter 3.) Edges connected to a node-bus are similarly grouped into *edge-buses* with each bus containing M edges. Each edge-bus is associated with a flag indicating if the switches in the bus share a single set of configuration memory. When the configuration memory is shared, all M switches in the bus must be turned on and off at the same time. When the state of these memory sharing switches are changed, the occupancy values of all the nodes connected immediately downstream to these switches must be increased and decreased simultaneously.

An example of the routing resource graph and its corresponding routing resources is shown in Figure 6.2. The super-cluster shown in the figure contains two clusters, meaning M = 2. Each cluster contains two input pins, two output pins, and is connected to a neighboring routing channel containing two fine-grain routing track and a two-bit wide routing bus. Also noted in the figure are the logic capacity of each node, the grouping of the node-buses, and the grouping of the edge-buses.



(a) Super-Cluster



Source Nodes (Capacity = 2 / Node): B, D
Sink Nodes (Capacity = 2 / Node): A, C
Input Pins (Capacity = 1 / Node): E, F, I, J
Output Pins (Capacity = 1 / Node): G, H, K, L
Fine-Grain Routing Tracks (Capacity = 1 / Node): M, N
CMS Routing Tracks (Capacity = 1 / Node): {O, P}
Node-Buses: {A, C}; {B, D}; {E, I}, {F,J}; {G, K}; {H, L}; {O, P}
Edge-Buses without Memory Sharing: {E->A, I->C}; {F->A, J->C}; {B->G, D->K};
{B->H, D->L}; {O->E, P->I}
Edge-Buses with Memory Sharing: {H->O, L->P}

(b) Routing Resource Graph

Figure 6.2: An Example Routing Resource Graph

6.6 The CGR Routing Algorithm

The overall flow of the CGR routing algorithm is shown in Figure 6.3. It consists of three major steps. In step 1, initialization, the algorithm identifies two types of bus structures — the *pin-buses* and the *net-buses* — in the input netlist. (These buses are defined in detail in Section

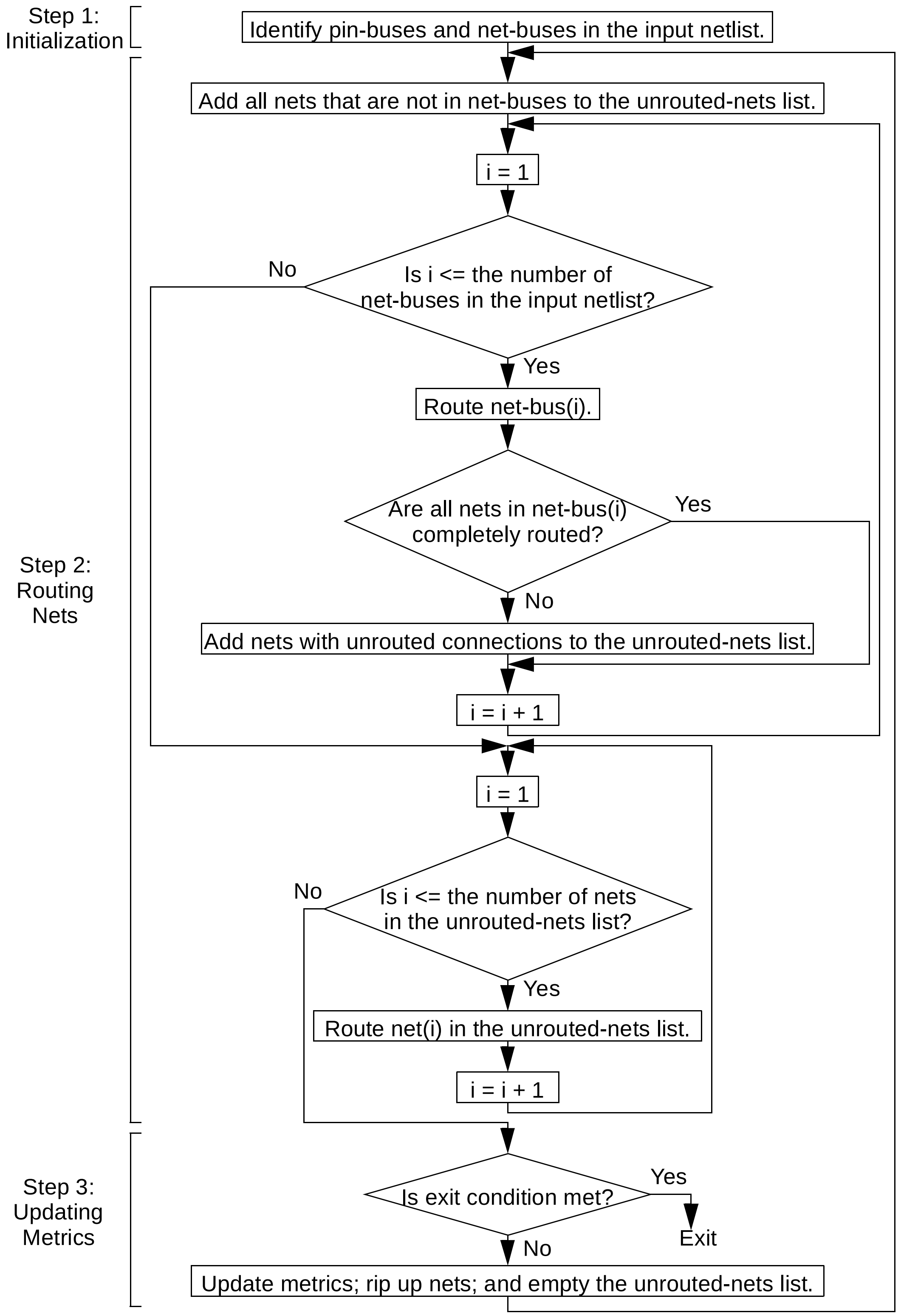
6.6.1.) Routing is then performed in step 2 where buses identified in step 1 are given priority for routing through the CMS routing tracks. In step 3, various cost metrics are updated according to the routing results from step 2. Once step 3 is completed, each net is reset to its unrouted state; and step 2 and step 3 are repeated in a new routing iteration. The repetition continuous until the exit condition is met. Note that the CGR algorithm described here is built upon the VPR routing algorithm as described in [Betz99a]; nevertheless, the basic principles presented here can also be used to transform other conventional routing algorithms into datapath-oriented routers. (VPR [Betz99a] is selected as the basis of this work over the original Pathfinder [Elbe95] router mainly due to its extensive support for FPGA architectural evaluation and modeling, which the Pathfinder router lacks.)

6.6.1 Step 1: Initialization

During step 1, CGR identifies inter-super-cluster connections that can be efficiently routed through the CMS routing tracks. For each input netlist, the initialization step first classifies all inter-super-cluster connections into two types of *two-terminal connections* (defined as a logical connection containing one super-cluster output pin and one super-cluster input pin) — groups of coarse-grain two-terminal connections called the *pin-buses* and individual fine-grain two-terminal connections. Then inter-super-cluster nets are similarly classified into groups of coarse-grain nets called net-buses and individual fine-grain nets. The amount of two-terminal connections or nets captured by these buses is a function of the amount of datapath regularity presented in the input netlist.

A *pin-bus* is a group of M two-terminal connections with the following four properties:

1. M is equal to the granularity of the MB-FPGA.
2. All connections must originate from the same source super-cluster and terminate at the same sink super-cluster.



Step 1: Initialization
Identify pin-buses and net-buses in the input netlist.
Add all nets that are not in net-buses to the unrouted-nets list.
i = 1
Is i <= the number of net-buses in the input netlist?
No
Yes
Route net-bus(i).
Are all nets in net-bus(i) completely routed?
Yes
No
Step 2: Routing Nets
Add nets with unrouted connections to the unrouted-nets list.
i = i + 1
i = 1
Is i <= the number of nets in the unrouted-nets list?
No
Yes
Route net(i) in the unrouted-nets list.
i = i + 1
Step 3: Updating Metrics
Is exit condition met?
Yes
Exit
No
Update metrics; rip up nets; and empty the unrouted-nets list.

Figure 6.3: Overview of the CGR Routing Algorithm

3. Each two-terminal connection must have a unique source cluster and a unique sink cluster.

4. For each two-terminal connection, its source cluster must have the same index as its sink cluster.

An example of a pin-bus is illustrated in Figure 6.4.

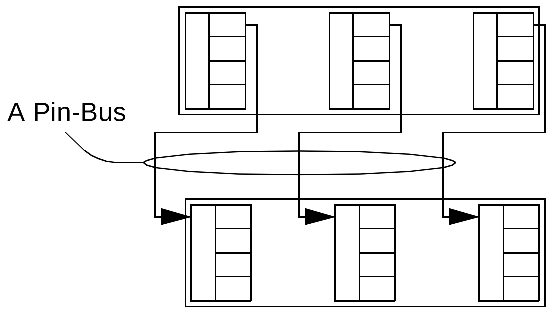


Figure 6.4: A Pin-Bus

Routing algorithms are most efficient when they are used to route a net, which contains a source and all the sinks of the source, at a time instead of individual two-terminal connections. To increase routing efficiency, pin-buses are grouped into net-buses. A *net-bus* is defined to be a group of M nets that contains at least one pin-bus, where M is equal to the granularity of the MB-FPGA. Note that a net-bus may also contain fine-grain two-terminal connections in addition to pin-buses. An example of net-buses is shown in Figure 6.5.

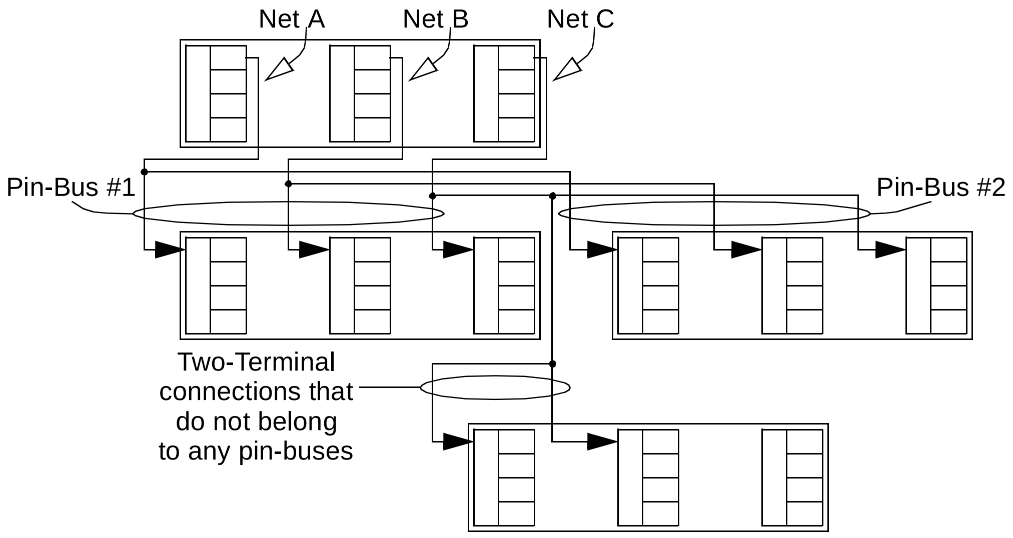

Net A
Net B
Net C
Pin-Bus #1
Pin-Bus #2
Two-Terminal connections that do not belong to any pin-buses

Figure 6.5: A Net-Bus Containing Net A, B, and C

6.6.2 Step 2: Routing Nets

After initialization, CGR first iterates through all the net-buses in the input netlist and routes the pin-buses through the CMS routing tracks. For each pin-bus, the algorithm also routes the first bit of the bus through the fine-grain routing tracks. The cost of routing the first bit is then compared with the cost of routing the entire pin-bus. When the CMS tracks are much more congested than the fine-grain tracks, the cost of routing the pin-bus through the CMS tracks will be higher than the cost of routing the first bit through the fine-grain tracks. In this case, the solution of routing the pin-bus through the CMS tracks is rejected. Instead each connection in the pin-bus is routed individually as described in the next two paragraphs. On the other hand, if using the CMS routing tracks incurs lower cost, the solution of routing the pin-bus through the CMS tracks is accepted.

During the routing of the net-buses, an unrouted-nets list is constructed. It contains three types of nets including all the fine-grain nets in the input netlist, nets in net-buses that contain fine-grain two-terminal connections, and net-buses containing pin-buses that are too expensive to be routed through the CMS routing tracks.

After iterating through the net-buses, CGR goes through the unrouted-nets list and routes each individual net in the list. Here only the unrouted connections in a net are routed. Again, a net is routed through both the fine-grain and the CMS routing tracks. The cost of using the fine-grain tracks is compared against the cost of using the CMS tracks. The lower cost option is always chosen as the final routing solution.

Note that as in other congestion-negotiation routing algorithms, each net-bus and each individual net is routed using the maze expansion algorithm, and the routing solutions are guided by the expansion cost. This cost is a combination of the congestion cost and the delay cost. It is also a function of the topology of the expansion. The congestion cost, delay cost, and the expansion cost are described below in more detail.

6.6.2.1 Congestion Cost

The CGR algorithm uses the same congestion cost function as the VPR router [Betz99a]. The VPR congestion cost function is briefly summarized here for completeness. A congestion cost is assigned to each node, n, in the routing resource graph. As shown in Equation 6.1, this congestion cost is a product of the base cost, b(n), the current congestion cost, p(n), and the historic congestion cost, h(n). The base cost, b(n), is a function of the routing resource type. Different routing resources are assigned different base cost values as shown in Table 6.1.

$$\text{congestion\_cost}(n) = b(n) \times p(n) \times h(n) \qquad \text{Equation 6.1}$$

Routing Resource	b(n)
Routing track	1
Super-Cluster Output pin	1
Super-Cluster input pin	0.95
Source	1
Sink	0

Table 6.1: b(n) Values for Each Type of Routing Resource

The current congestion cost, p(n), is defined to be a function of the difference between the current occupancy of the node and the capacity of the node as shown in Equation 6.2. The scaling factor, pfac, in Equation 6.2 is called the routing schedule of the router. The initial value of pfac is a small (< 0.5), so during early iterations, the current congestion of the node is a small part of the total cost of the node. This allows each node to be used more than its capacity allows. The value of pfac is increased by a factor of 1.5 to 2 during each routing iteration. So during the latter iterations, the current congestion cost becomes a significant factor in determining the total cost of a node. Consequently, at latter routing iterations, once a node reaches its full capacity, it no longer can be used in the routing solutions of other nets.

$$p(n) = 1 + \max(0, [\text{occupancy}(n) + 1 - \text{capacity}(n)] \times \text{pfac}) \quad \text{Equation 6.2}$$

The historic congestion cost, h(n), is an accumulation of the past congestion values; and it is defined by Equation 6.3. For the first iteration, I = 1, the historic congestion, $h(n)^1$ is set to be one. For each subsequent iteration, the difference between occupancy and capacity is scaled by a constant value, hfac, and is added to the historic congestion cost, $h(n)^{I-1}$, from the previous iteration to derive the current historic congestion cost, $h(n)^I$. The usual value of hfac is between 0.2 and 1.

$$h(n)^I = \begin{cases} 1 & I = 1 \\ h(n)^{I-1} + \max(0, \\ [\text{occupancy}(n) - \text{capacity}(n)] \times \text{hfac}) & I > 1 \end{cases} \quad \text{Equation 6.3}$$

6.6.2.2 Optimizing Circuit Delay

As with the VPR routing algorithm, CGR calculates the delay of each net using the Elmore delay model. For calculating the Elmore delay, the capacitance and the resistance of each routing resource are obtained through the SPICE simulations as described in [Betz99a]. The delay values are then used to determine the criticality of each two-terminal connections. Note that for all the experiments performed in this thesis, the device characteristics of the CMC 0.18 μm process is used in the SPICE simulations.

For each two-terminal connection, the criticality of the connection is calculated using Equation 6.4. Here *i* represents the source of a two-terminal connection and *j* represents the sink of the two-terminal connection. As shown by Equation 6.4, the criticality is a value between zero and one inclusively. It is a function of the slack of the connection and Dmax, the critical path delay of the circuit. When the total delay of a connection is close to the maximum delay of the circuit, the net has a very small slack. Consequently, its criticality is high. When the total delay of a connection is much smaller than the maximum delay of the circuit, the net has a very large slack. Consequently its criticality is low. Both the slack and Dmax values are re-calculated in each routing iteration based on the connectivity data obtained from the previous routing iteration.

$$\text{criticality}(i, j) = \max\left(\left[1 - \frac{\text{slack}(i, j)}{\text{Dmax}}\right], 0\right) \qquad \text{Equation 6.4}$$

6.6.2.3 Expansion Cost

When routing a net-bus, CGR expands one node-bus at a time as it is typically done in many other routing algorithms [Lee61] [Brow92a] [Ebel95] [Betz99a] [Chan00]. The expansion starts at the node-bus where the sources of the net-bus reside. It first expands into all the immediate neighboring node-buses. It then finds the node-bus with the least expansion cost. This node-bus is then expanded in turn. The expansion continues until the node-bus containing the targeted sinks is reached. When routing an individual net, the same algorithm is applied to individual nodes instead of node-buses.

Expansion Topologies

The expansion cost of CGR is not only designed to balance delay with congestion, but also to balance the use of CMS and fine-grain routing tracks based on the following two factors:

1. The number of net-buses and fine-grain nets in the input netlist.
2. The available number of CMS and fine-grain routing tracks in the MB-FPGA architecture.

The expansion cost also addresses the complication resulting from the “dual personality” of several types of routing resources — these resources can be considered either as an extension of fine-grain tracks or CMS tracks. For example, a super-cluster input pin can be considered either as an individual input pin or as a part of a super-cluster input bus. When connected to a signal from a fine-grain routing track, the pin behaves like the extension of the fine-grain routing tracks. When connected to a signal from a group of CMS routing tracks carrying a net-bus, on the other hand, the pin behaves like a part of the extension of the CMS routing tracks. Routing resources that behave similarly include super-cluster output pins, I/O block input and output pins, sources, and sinks.

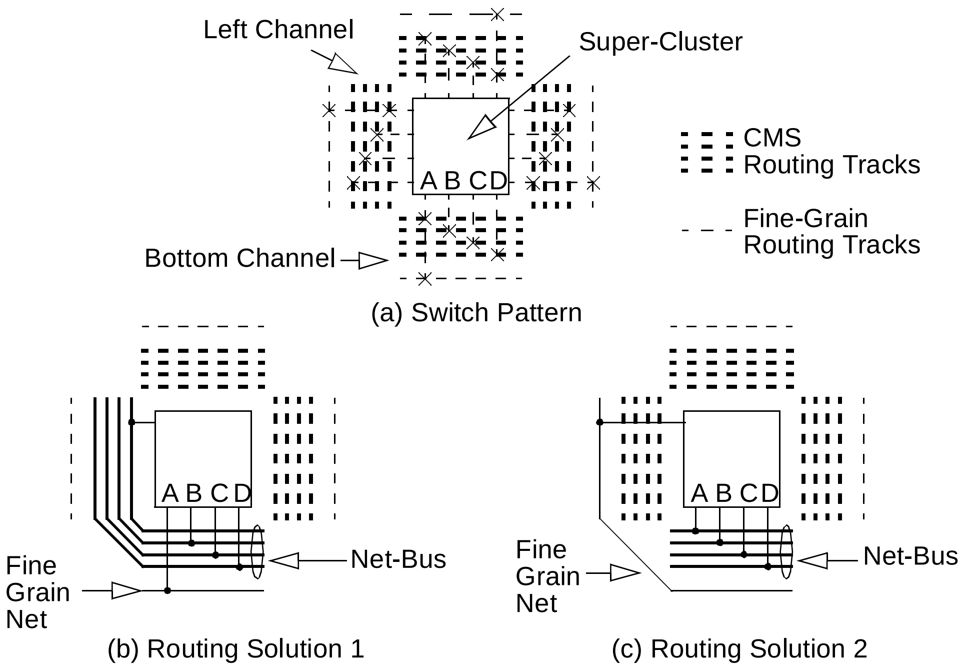


Figure 6.6: Competition for Resources

Because of this dual personality, fine-grain nets often compete with coarse-grain net-buses for resources. An example is shown in Figure 6.6, which shows three different views of the same MB-FPGA tile. The tile contains a super-cluster and four routing channels, which are connected by disjoint switch blocks [Hsei90]. In the figure, the unoccupied routing tracks are shown in dashed lines; and the occupied routing tracks are shown in solid lines. Finally, the switch pattern for the input connection blocks is shown in Figure 6.6a, where an X marks the position of a routing switch.

Figure 6.6b and Figure 6.6c shows two different routing solutions for routing a fine-grain net and a 4-bit wide net-bus. As shown, both the fine-grain net and a net in the net-bus can use pin A to get into the super-cluster. These two nets are said to be in competition for the same routing resource. In Figure 6.6b, the router first routes the fine-grain net through pin A. To avoid congestion, only three nets in the net-bus can get into the super-cluster from the bottom channel through pins B, C, and D. To complete the routing process, the net-bus needs to be routed from the bottom channel into the left channel and then into the super-cluster. In Figure 6.6c, pin A is used by the net-bus instead. All four bits of the net-bus can now be routed into the super-cluster through the bottom channel. However, one extra fine-grain routing segment has to be used in the left channel to connect the fine-grain net to the super-cluster.

This example shows the need of two separate expansion cost functions for dual-personality routing resources — one for the fine-grain expansions and the other for the coarse-grain expansions. Furthermore, these two cost functions should be consistently defined so that they can be fairly compared with each other. The CGR router accommodates the dual-personality routing resources by classifying all possible expansions into five expansion topologies. Each topology is completely defined by three items — the source node(s) of the expansion, the sink node(s) of the expansion, and the routing switch(es) that connect the source node(s) to the sink node(s). Each of these expansion topologies is then assigned a maximum of two unique cost functions, one for routing fine-grain nets and the other for routing coarse-grain net-buses. Furthermore, these functions are designed to share several key ingredients to ensure fair comparison. Each of these functions and its corresponding expansion topology are described in turn in more detail.

Expansion Cost Functions

As shown in Table 6.2, the expansion cost is calculated based on the expansion topologies. The table is divided into three major columns including the expansion topology column, the net type column, and the expansion cost column. The expansion topology column is then subdivided into five sub-columns. Each topology is labelled by a unique number in sub-column 1. Sub-column 2 and 3 list the granularity of the expansion source and the expansion sink respectively. Sub-column 4 states whether the edges connecting the expansion source and the expansion sink form a memory sharing edge-bus. Sub-column 5 lists several examples for each expansion topology. The net type column lists the two routing scenarios for each expansion — a single net or a net-bus. Finally, the expansion cost column lists the formulas for calculating the expansion cost under each scenario.

Note that, in these formulas, n_{to} is used to denote the expansion sink if it is a fine-grain node. On the other hand, if the expansion sink is a node-bus, N_{to} is used instead; and $N_{to}(l)$ is used to denote the lth bit in the node-bus, N_{to}. Finally, for the topologies where the expansion sink is a node-bus and a single net is being routed, $N_{to}(k)$ denotes the node, in the expansion sink, through which the single net will be actually routed. Similar convention applies to the expansion source, which is denoted by n_{from}, N_{from}, $N_{from}(l)$, or $N_{from}(k)$ for a given circumstance.

All formulas, shown in Table 6.2, are variations of Equation 6.5. The equation has three terms. The first term, C(n), represents the total congestion cost of all nodes in an expansion path that connects the source of a net to node n. The second term, D(n), represents the delay of the expansion path from the source to node n. These two terms are scaled by the criticality of the target sink. If the two-terminal connection connecting the source to the sink is very critical then the delay cost dominates. If the net is not critical then the congestion cost dominates. The

Expansion Topology					Net Type	Expansion Cost
#	From Node(s)	To Node(s)	Mem Shar	Examples		
1	fine-grain node	fine-grain node	N/A	fine-grain track -> fine-grain track	net	$[1 - \text{criticality}(i,j)] \times C(n_{\text{to}}) +$ $\text{criticality}(i,j) \times D(n_{\text{to}}) +$ $\text{future\_expansion\_cost}(n_{\text{to}})$ $C(n_{\text{to}}) = \text{congestion\_cost}(n_{\text{to}}) +$ $C(n_{\text{from}})$
					net-bus	N/A
2	fine-grain node	node-bus	N/A	fine-grain track -> super-cluster input pin	net	$[1 - \text{criticality}(i,j)] \times C(N_{\text{to}}(k)) +$ $\text{criticality}(i,j) \times D(N_{\text{to}}(k)) +$ $\text{future\_expansion\_cost}(N_{\text{to}}(k))$ $C(N_{\text{to}}(k)) = \text{congestion\_cost}(N_{\text{to}}(k)) +$ $C(n_{\text{from}})$
					net-bus	N/A
3	node-bus	fine-grain node	N/A	super-cluster output pin -> fine-grain track	net	$[1 - \text{criticality}(i,j)] \times C(n_{\text{to}}) +$ $\text{criticality}(i,j) \times D(n_{\text{to}}) +$ $\text{future\_expansion\_cost}(n_{\text{to}})$ $C(n_{\text{to}}) = \text{congestion\_cost}(n_{\text{to}}) +$ $C(N_{\text{from}}(k))$
					net-bus	N/A
4	node-bus	node-bus	no	CMS track -> super-cluster input pins; sources-> super-cluster input pins; super-cluster output pins -> sinks	net	$[1 - \text{criticality}(i,j)] \times C(N_{\text{to}}(k)) +$ $\text{criticality}(i,j) \times D(N_{\text{to}}(k)) +$ $\text{future\_expansion\_cost}(N_{\text{to}}(k))$ $C(N_{\text{to}}(k)) = \text{congestion\_cost}(N_{\text{to}}(k)) +$ $C(N_{\text{from}}(k))$
					net-bus	$\max([1 - \text{criticality}(i,j)] \times C(N_{\text{to}}(l)) +$ $\text{criticality}(i,j) \times D(N_{\text{to}}(l)) +$ $\text{future\_expansion\_cost}(N_{\text{to}}(l))\Big\|_{l=M}^{l=1})$ $C(N_{\text{to}}(l)) = \text{congestion\_cost}(N_{\text{to}}(l)) +$ $C(N_{\text{from}}(l))$

Table 6.2: Expansion Cost

Expansion Topology					Net Type	Expansion Cost
#	From Node(s)	To Node(s)	Mem Shar	Examples		
5	node-bus	node-bus	yes	super-cluster output pins -> CMS tracks; CMS tracks -> CMS tracks	net	$[1 - \text{criticality}(i, j)] \times C(N_{\text{to}}(k)) + \text{criticality}(i, j) \times D(N_{\text{to}}(k)) + \text{future\_expansion\_cost}(N_{\text{to}}(k))$ $C(N_{\text{to}}(k)) = \max\left(\text{congestion\_cost}(N_{\text{to}}(l))\Big\|_{l=M}^{l=1}\right) + C(N_{\text{from}}(k))$
					net-bus	$\max([1 - \text{criticality}(i, j)] \times C(N_{\text{to}}(l)) + \text{criticality}(i, j) \times D(N_{\text{to}}(l)) + \text{future\_expansion\_cost}(N_{\text{to}}(l))\Big\|_{l=M}^{l=1})$ $C(N_{\text{to}}(l)) = \text{congestion\_cost}(N_{\text{to}}(l)) + C(N_{\text{from}}(l))$

Table 6.2: Expansion Cost

third term represents the estimated future expansion cost, which is described in detail in [Betz99a]. Briefly, the future expansion cost is calculated by estimating the remaining expansion path that connects node n to the target sink. The total cost of this estimated path is then calculated and used as the future expansion cost.

$$\begin{aligned}\text{expansion\_cost}(n) = {} & [1 - \text{criticality}(i, j)] \times C(n) + \\ & \text{criticality}(i, j) \times D(n) + \\ & \text{future\_expasion\_cost}(n)\end{aligned} \qquad \text{Equation 6.5}$$

Expansion topology #1 is illustrated in Figure 6.7(a). In this topology, both the expansion source and the expansion sink are fine-grain nodes. The edge that connects these two nodes together does not belong to any edge-bus. For this topology, Equation 6.5 is used to calculate the expansion cost with n set to be n_{to}. The expansion cost for topology #2 and #3 are similarly defined.

Expansion topology #4 is illustrated in Figure 6.7(b). Here both the expansion source and the expansion sink are node-buses, and are connected by an edge-bus that does not share con-

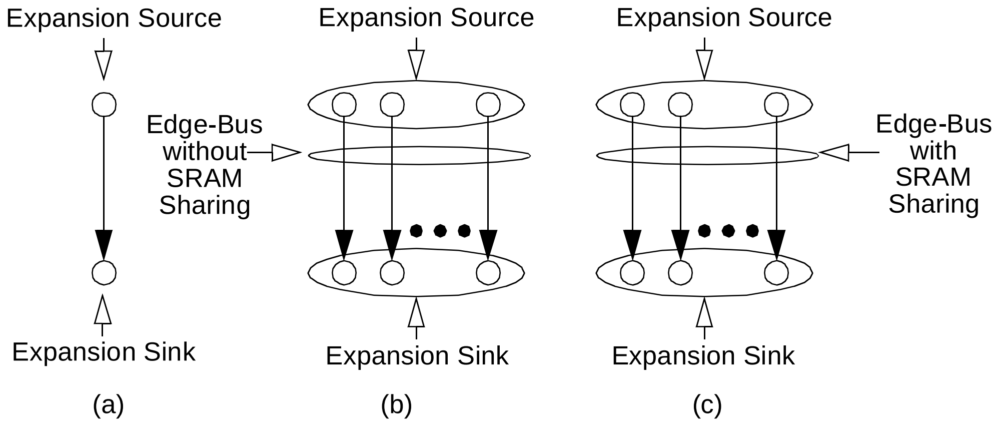


Figure 6.7: Expansion Topology

figuration memory. When only routing a fine-grain net through this expansion topology, Equation 6.5 is used to calculate the expansion cost with n set to be $N_{to}(k)$.

It is important to note that, due to the dual personality of some routing resources, nodes in a node-bus do not necessarily have the same congestion cost. An example is shown in Figure 6.8. Here the nodes in the node-bus represent four input pins. All four pins are connected to a net-bus through a CMS routing bus. Pin 0, however, has one more connection — it is also connected to a fine-grain net through a fine-grain routing track. The occupancy value of pin 0 is 2 while the occupancy values of pin 1, 2, and 3 are 1. Since the occupancy values are different, the expansion costs are also different. So when routing a net-bus through expansion topology #4, Equation 6.5 is used to calculate the expansion cost for each node in the expansion sink. The maximum cost is then used as the final expansion cost to account for any differences in either delay or congestion.

Expansion topology #5 is illustrated in Figure 6.7(c). Here both the expansion source and the expansion sink are node-buses. These two node-buses are connected by an edge-bus that shares a single set of configuration memory. When routing a fine-grain net through this expansion topology, the maximum congestion cost over all nodes in the expansion sink is first calcu-

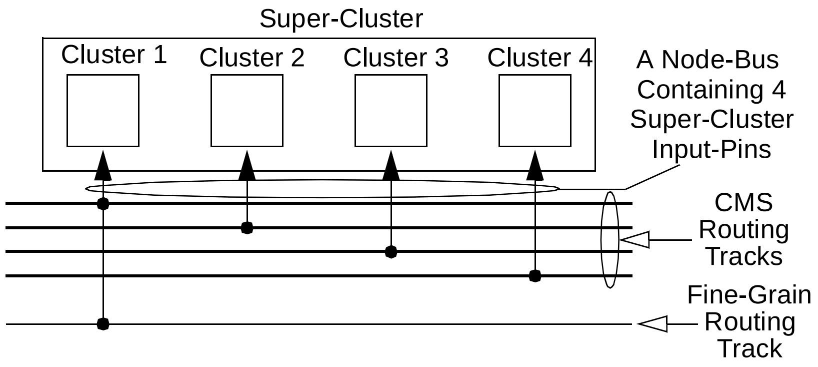


Figure 6.8: Double Connection in One Bit of A Node-Bus

lated. To account for the sharing of the configuration memory, the maximum congestion cost is added to the accumulated congestion cost of the expansion source to derive the accumulated congestion cost for node $N_{to}(k)$. Equation 6.5 is then used to calculate the expansion cost where n is substituted by $N_{to}(k)$. If a net-bus is routed through this expansion topology, on the other hand, Equation 6.5 is used to calculate the expansion cost for each node in the expansion sink. The maximum cost is then used as the expansion cost.

Note that to encourage net-buses to use the CMS routing tracks, the expansion cost of routing the first bit of each net-bus through the fine-grain routing tracks is penalized by a constant multiplication factor. In this thesis a multiplication factor of 20 is found to work well for the benchmarks.

6.6.3 Step 3: Updating Metrics

The tasks performed in step 3 include updating congestion cost for each node and calculating criticality values for each net. The exit condition is also checked at this step to see if the maximum number of routing iterations has been reached. Finally, CGR rips up all previously routed nets by setting the occupancy values of all nodes in the routing resource graph to zero. It then empties the unrouted-nets list in preparation for the next routing iteration.

6.7 Results

CGR has been used to route several industrial circuits implemented on the MB-FPGA architecture. The routing results shown in this section are based on the fifteen datapath circuits of the Pico-Java Processor [Sun99] described in Chapter 4 and 5. Each circuit is first synthesized using the EMC synthesis algorithm as described in Chapter 4. Then it is packed into super-clusters using the CNG packing algorithm described in Chapter 5. The placement algorithm, described in Section 6.3, is then used to place each circuit onto a square MB-FPGA that contains just enough super-clusters to accommodate the circuit.

Table 6.3 gives the name, size (the number of super-clusters, two-terminal connections, and pin-buses) of each benchmark circuit. The super-cluster architectures used to obtain these results will be presented in detail in the next sub-section along with the detailed routing architecture used in the investigation. After the architectural description, the routing results in terms of track count, routing area and routing performance are described in turn.

6.7.1 MB-FPGA Architecture

The detailed MB-FPGA architecture used in the tests of the CGR algorithm is presented here. This architecture is constructed based on the results of several previous studies on conventional cluster-based FPGA architectures described in [Betz97b] [Betz98] [Betz99a] [Betz99b] [Marq00b]. These architectural choices are described and justified in much more detail in Chapter 8. A brief summary of these choices is presented here.

Each super-cluster used in this investigation consists of 4 clusters. Each cluster contains 4 BLEs (with one 4-LUT in each BLE) and 10 inputs. At the super-cluster level, this translates to 40 super-cluster inputs grouped into 10 4-bit wide input buses and 16 super-cluster outputs grouped into 4 4-bit wide output buses. The input buses are distributed around the periphery of each super-cluster with 2 buses at the top, 2 buses on the right side, 3 buses at the bottom and

Circuit	#Super-Clusters	#Two-Term. Conn.	#Pin-Buses
code_seq_dp	23	799	116
dcu_dpath	63	2232	273
ex_dpath	176	6547	851
exponent_dp	32	1362	109
icu_dpath	217	8047	945
imdr_dpath	81	3100	387
incmod	57	2013	211
mantissa_dp	64	2533	298
multmod_dp	105	3380	330
pipe_dpath	29	1049	126
prils_dp	26	864	89
rsadd_dp	21	722	94
smu_dpath	36	1167	140
ucode_dat	83	3143	409
ucode_reg	6	194	35

Table 6.3: Experimental Circuits

3 buses on the left side of each super-cluster. The output buses are uniformly distributed around each super-cluster.

The switch block topology used in this study is the disjoint topology [Hsei90] for both the fine-grain tracks and the CMS tracks. For input/output connection blocks, each super-cluster input/output pin is connected to 40%/25% of the fine-grain tracks in each of its neighboring routing channels. Similarly each input bus/output bus is connected to 40%/25% of the 4-bit wide CMS routing buses in each neighboring channel. It is assumed that, for each circuit, the physical dimension of each I/O block is small enough so that the circuit will always be core-limited after placement. Each I/O block input or output pin is assumed to connect to 20% of the fine-grain routing tracks in its neighboring routing channel. Similarly, each pad-input or pad-output bus is assumed to connect to 20% of the CMS routing buses. Finally, each routing segments (either fine-grain or CMS) is assumed to expand two super-clusters. Note that the architectural generation methodology described in [Betz00] [Betz99a] is used to generate the actual topology of the input/output connection blocks based on the percentage values presented above.

Two architectural variables are used in the tests of the CGR routing algorithm. These variables are the number of CMS routing tracks and the number of fine-grain routing tracks in each routing channel. The testing process consists of first specifying a fixed number of CMS routing tracks per channel for routing a given circuit. The CGR router is then used to find the minimum number of fine-grain routing tracks required in each channel (in addition to the fixed number of CMS routing tracks) in order to successfully route the circuit. The methodology used in searching for the minimum number of fine-grain routing tracks is similar to the ones described in [Betz99a] [Betz99b]. The data collected from these tests are used to demonstrate the ability of the CGR algorithm to utilize the CMS routing tracks in place of the fine-grain routing tracks.

6.7.2 Track Count

The arithmetic average of the additional number of fine-grain routing tracks required to route a circuit for a specified number of CMS routing tracks is shown in Figure 6.9, where the average number of fine-grain tracks is plotted against the specified number of CMS tracks. The X-axis is track count in terms of the number of CMS routing tracks specified in the investigation. The Y-axis is also track count, but it is in terms of the number of tracks needed in order to successfully route a circuit.

Also plotted in the figure is the arithmetic average of the total number of tracks (fine-grain tracks plus CMS tracks) required per channel for routing a circuit against the specified number of CMS tracks. Finally, for the ease of reference, the number of CMS routing tracks specified in each investigation is transferred from the X-axis into the dashed line in the figure.

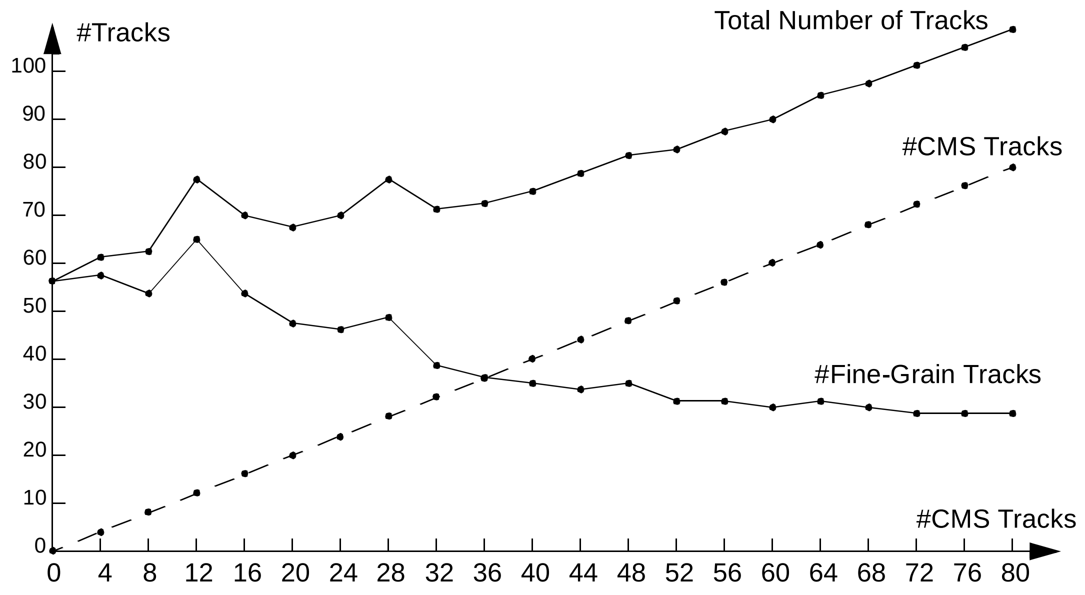


Figure 6.9: Track Count vs. #CMS Tracks per Channel

As shown, the average number of fine-grain tracks decreases from 56 tracks down to less than 30 tracks per channel as the average number of CMS tracks is increased from 0 to 80 tracks per channel. This decrease is due to the use of the CMS tracks in place of the fine-grain tracks by CGR. Note that the fine-grain tracks decreases significantly when the number of CMS tracks is increased from 0 to 40 tracks per channel. By adding these 40 CMS tracks, a maximum of 21 fine-grain tracks are eliminated. The rate of decrease slows significantly when the number of CMS tracks is further increased from 40 to 80 tracks per channel. By adding these 40 CMS tracks, only 5 fine-grain tracks are saved. This slow down is due to the fact that, in many circuits, the number of CMS tracks has reached saturation. In saturation, further increases in the number of CMS tracks will not reduce the number of fine-grain tracks since there are always nets that can never been routed through the CMS tracks (due to the limited connectivity of the CMS routing resources as described in Chapter 3).

6.7.3 Routing Area Results

The arithmetic average of the total area consumed by routing resources for each circuit is shown in Figure 6.10. The area is measured in terms of the minimum-width transistor area [Betz99a], which is described in detail in Chapter 2. In the figure, the X-axis is the number of CMS routing tracks specified by the investigation. The Y-axis is the area measured in minimum-width transistor area.

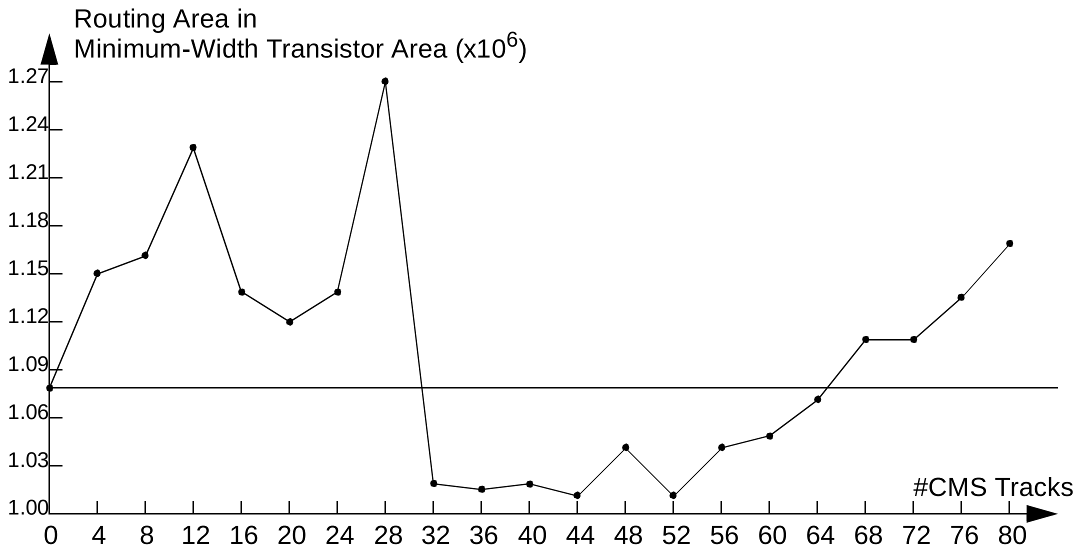


Figure 6.10: Area vs. #CMS Tracks

As shown, the graph can be divided into three regions. In the first region, where the architectures contain between 0 to 28 CMS tracks per channel, the routing area actually gets worse with the increased number of CMS tracks. This increase in area is due to the fact that the area efficiency of the CMS tracks is outweighed by the extremely small number of routing buses and the increased diversity in routing resources as explained in more detail in Chapter 8.

In the second region, as the number of CMS tracks is further increased, the routing area starts to decrease. At between 32 to 64 CMS tracks per channel, the routing area becomes smaller than the routing area of the architecture that contains no CMS tracks. Finally, in region three, as the number of CMS tracks reaches saturation, the routing area again starts to increase and becomes larger than the routing area of the architecture that contains no CMS routing tracks.

6.7.4 Routing Performance Results

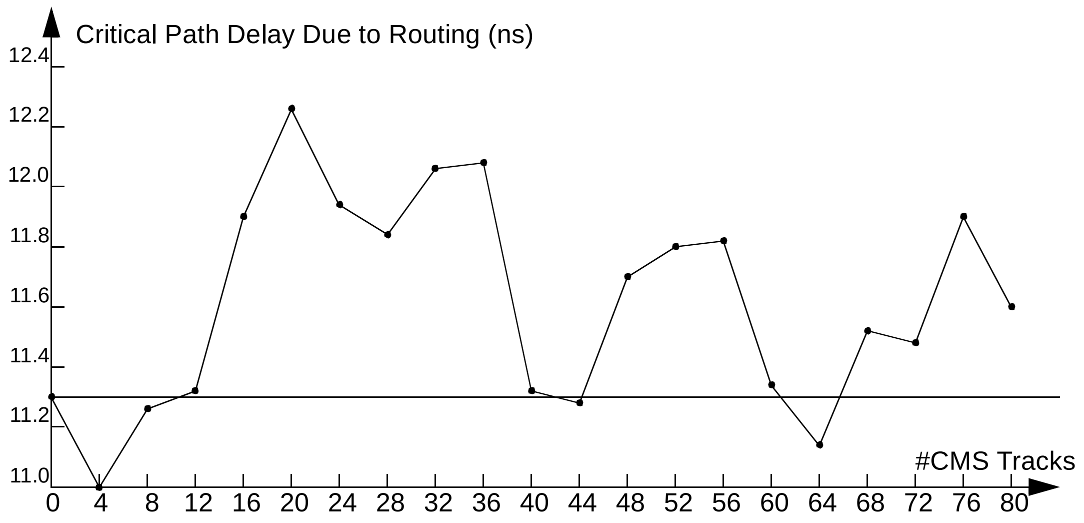


Figure 6.11: Delay vs. #CMS Tracks

The geometric average of the portion of critical path delay that is consumed by the routing resources for each circuit is shown in Figure 6.11. In the figure, the X-axis represents the number of CMS routing tracks specified in the investigation. The Y-axis represents the delay measured in nanoseconds. As shown, there are some speed penalties associated with using the CMS tracks for a majority of architectures. For 4, 8, 12, 40, 44, 60, and 64 CMS routing tracks per channel, however, the performance of these architectures are comparable to the architecture containing no CMS tracks.

6.8 Conclusions and Future Work

This chapter has described a new kind of routing algorithm that is designed specifically for the MB-FPGA architecture. The algorithm is able to balance the use of CMS and fine-grain routing resources to achieve maximum area savings over a wide range of datapath circuits.

The algorithm also is capable of optimizing the routing delays of time-critical connections and achieves good performance results.

Future research should improve upon the timing performance of the router. The goal should be to achieve consistently optimal routing delays regardless of the number of CMS routing tracks in each routing channel.

7 The Regularity of Datapath Circuits

7.1 Introduction

This chapter uses an experimental approach to investigate the effect of the granularity of the MB-FPGA architecture on the regularity of its circuit implementations. *Granularity,* as defined in Chapter 3, is equal to the number of clusters in a super-cluster or the number of CMS routing tracks in a routing bus. *Regularity* is a function of the number of identical bit-slices and buses in a circuit. The experiment consists of synthesizing and packing a set of data-path circuits onto several variants of the MB-FPGA architecture. Regularity is then measured over a range of granularity values, using the synthesis and packing algorithms described in Chapter 4 and 5 respectively.

Recall that the MB-FPGA architecture uses groups of CMS routing tracks, called routing buses, to route groups of signals, called buses, from their common source to their common sink super-clusters. When there is enough regularity, it is often preferable to have high granularity — wide routing buses — since in wide routing buses the configuration memory area can be amortized over a greater number of tracks, resulting in higher area efficiency. Too high a granularity value, however, can cause an over-supply of routing tracks in each routing bus, resulting in unused tracks and lower area efficiency. As a result, knowing the detailed relationship between the granularity and the regularity is essential in designing an area-efficient MB-FPGA architecture.

The MB-FPGA super-cluster introduced in Chapter 3 is used in the packing experiments conducted in this chapter. Figure 7.1 reproduces an overview of the super-cluster architecture for ease of reference. The super-cluster shown in the figure has a granularity value of M since it contains M clusters; and each cluster contains several BLEs, cluster inputs, and cluster outputs. The number of cluster outputs is always equal to the number of BLEs that it contains. The granularity of the super-cluster can be altered by changing the number of clusters; and individual cluster capacity can also be changed by varying the number of BLEs and the number of inputs per cluster.

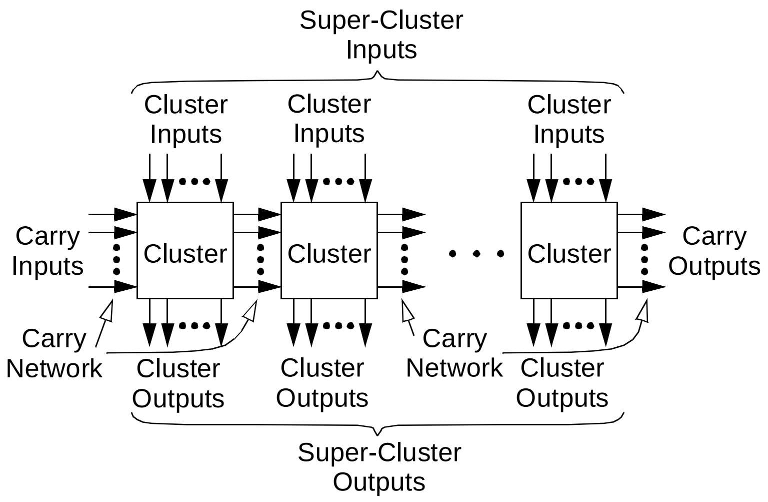


Figure 7.1: Super-Cluster with M Clusters

By definition, regularity can be measured using two distinct metrics, including the *logic regularity*, which measures the number of identical bit-slices in a datapath circuit, and the *net regularity*, which measures the number of buses in the same circuit. The specific questions concerning datapath regularity that are answered in this chapter include: what is the effect of the MB-FPGA granularity on the amount of logic regularity that can be captured by the synthesis and packing tools? and what is the effect of the MB-FPGA granularity on the amount of net regularity that can be captured? Section 7.4.1 and Section 7.4.2 define logic and net regularity in more detail and propose ways of measuring these regularity values. They also show that small granularity values of 2 and 4 are the best for capturing both types of regularity information.

This chapter is organized as follows: Section 7.2 presents assumptions that are made about the architecture of MB-FPGA; Section 7.3 describes the experimental procedure; the experimental results are presented in Section 7.4; and concluding remarks appear in Section 7.5.

7.2 MB-FPGA Architectural Assumptions

The MB-FPGA super-cluster architecture, used by the CNG packing algorithm, can be completely specified by four architectural parameters, including M, the granularity of the architecture, K, the number of inputs that each LUT contains, N, the number of BLEs in each cluster, and I, the number of cluster inputs. In this work, these values are selected based on the results of several previous studies on conventional FPGAs; and the same values are also used in the architectural study presented later in Chapter 8.

Throughout the experiments, K is set to be 4 since it has been shown that 4-LUTs are one of the most efficient LUT sizes [Rose90] [Ahme00] and this LUT size is also used in many commercial FPGAs [Alte02] [Xili02]. N and I are set to be 4 and 10 respectively since this combination was shown to be one of the most efficient by [Betz97b] [Betz98] [Betz99a] and is also used by many previous FPGA studies [Ahme00] [Betz99a] [Betz99b] [Lemi01] [Marq99] [Marq00a] [Marq00b] [Sank99]. Finally, M is the variable of the investigation and is varied from 2 to 32 for each experiment.

7.3 Experimental Procedure

The CAD flow used in the experiments is shown in Figure 7.2. As shown, the fifteen benchmark circuits from the Pico-Java processor [Sun99] are first synthesized by the EMC datapath-oriented synthesis algorithm into LUTs and DFFs. The synthesis process preserves the regularity of datapath circuits according to a given value of M. Using the preserved regularity, the CNG packing algorithm packs the synthesized circuits into a set of MB-FPGA super-clusters, again with respect to M. After packing, the logic regularity and the net regular-

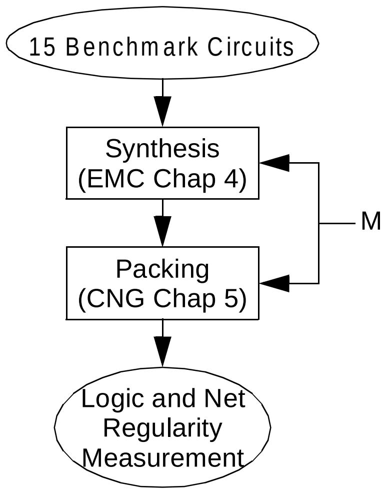


Figure 7.2: CAD Flow

ity of each circuit are measured. Finally, the CAD flow is invoked multiple times until all concerned values of M are investigated.

7.4 Experimental Results

This section first investigates the effect of granularity on logic regularity. Then the effect of granularity on net regularity is examined. Note that the various aspects of the benchmark circuits used in these experiments are summarized in previous chapters in Table 4.1, Table 5.1, and Table 6.1.

7.4.1 Effect of Granularity on Logic Regularity

After packing, datapath circuits are transformed into netlists of interconnected super-clusters. The logic regularity of these netlists, which is defined as the amount of logic that exists as datapath, can be measured by counting the number of identical BLEs in each super-cluster. More precisely, each super-cluster after packing can be divided into disjoint sets of BLEs. Each set, called a datapath component, has the following three properties:

1. Each BLE in the set is from an unique cluster that is different from the clusters of all the other BLEs in the set.
2. All BLEs in the set are configured to perform the same logic function.
3. The set contains as much BLEs as property 1 and 2 allows.

An example of datapath components is shown in Figure 7.3. In the figure, there is a super-cluster that contains four clusters; and each cluster contains four BLEs. The BLEs are configured to perform one of the six possible logic functions labeled as A, B, C, D, E, and F. As shown, the super-cluster can be divided into eight distinct datapath components based on the properties above. Four of the datapath components contain more than one BLE each and is indicated by unique shades in the figure. The remaining four datapath components contain one BLE each and are configured as function C, B, E, and F, respectively.

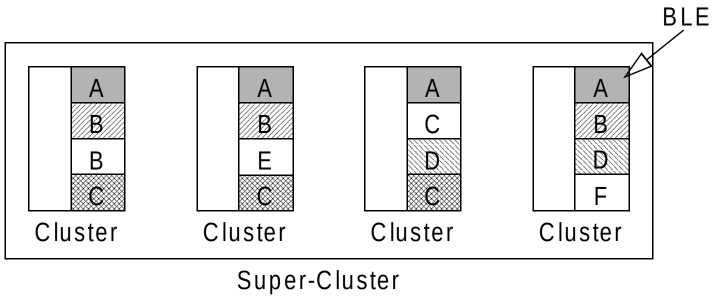


Figure 7.3: Dividing A Super-Cluster into Datapath Components

The datapath components can be classified according to their width, which is equal to the number of BLEs in each component. By definition, for a given MB-FPGA architecture, the maximum possible width of a datapath component is equal to M — the granularity of the architecture; and one-bit wide datapath components represent irregular logic.

Table 7.1 lists the percentage of BLEs that are in each width of datapath components as a function of the MB-FPGA granularity that was imposed during synthesis. Each percentage value is calculated by first summing the total number of BLEs in all datapath components of a given width over all the 15 benchmark circuits. Then the sum is divided by the total number of

BLEs in the benchmarks. Each column of the table corresponds to a different value of M. Each row corresponds to a different width of datapath components. Note that for each value of M represented in a given column, there are only M different widths of datapath components. The remaining rows in the column are labeled as *n.a.* and shaded in dark gray. Finally, cells that contain values that are less than one percent are shaded in light gray in the table.

DP Width	M — MB-FPGA Granularity Imposed in Synthesis								
	2	4	8	12	16	20	24	28	32
1	9.4%	9.9%	10%	11%	11%	12%	12%	13%	13%
2	**91%**	0.31%	0.051%	0.0%	0.049%	0.0%	0.0%	0.12%	0.049%
3	n.a.	0.23%	0.076%	0.075%	0.037%	0.074%	0.037%	0.0%	0.037%
4	n.a.	**90%**	7.0%	7.4%	6.1%	6.1%	6.6%	12%	5.9%
5	n.a.	n.a.	0.13%	0.12%	0.0%	0.0%	0.12%	0.64%	0.0%
6	n.a.	n.a.	0.49%	0.48%	0.15%	0.15%	0.48%	0.32%	0.15%
7	n.a.	n.a.	0.35%	0.0%	0.26%	0.17%	0.0%	0.094%	0.086%
8	n.a.	n.a.	**82%**	27%	14%	16%	27%	15%	14%
9	n.a.	n.a.	n.a.	1.0%	0.0%	0.22%	1.0%	0.0%	0.0%
10	n.a.	n.a.	n.a.	0.25%	0.0%	0.56%	0.25%	0.0%	0.0%
11	n.a.	n.a.	n.a.	0.55%	0.14%	0.0%	0.14%	0.15%	0.0%
12	n.a.	n.a.	n.a.	**52%**	2.0%	19%	0.074%	0.0%	0.074%
13	n.a.	n.a.	n.a.	n.a.	0.32%	1.5%	0.0%	0.0%	0.0%
14	n.a.	n.a.	n.a.	n.a.	0.95%	0.35%	0.0%	0.0%	0.0%
15	n.a.	n.a.	n.a.	n.a.	0.19%	0.19%	0.19%	0.0%	0.18%
16	n.a.	n.a.	n.a.	n.a.	**64%**	2.7%	2.9%	2.6%	2.7%
17	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%	0.0%	0.0%	0.0%
18	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%	0.0%	0.0%	0.0%
19	n.a.	n.a.	n.a.	n.a.	n.a.	0.24%	0.0%	0.26%	0.0%
20	n.a.	n.a.	n.a.	n.a.	n.a.	**41%**	1.1%	1.2%	1.1%
21	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%	0.0%	0.0%
22	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%	0.0%	0.0%
23	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.85%	0.62%	0.57%
24	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	**47%**	0.48%	0.44%
25	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%	0.0%
26	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%	0.0%
27	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.36%	0.33%
28	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	**52%**	4.1%
29	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.71%
30	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	2.0%
31	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.0%
32	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	**54%**

Table 7.1: % of BLEs Contained in Each Width of Datapath Components

The key conclusion to be drawn from Table 7.1 is that as indicate by the cells shaded in light gray, many entries in the table contain values that are less than one percent. As a result, for each M, a majority of BLEs is concentrated in only a small number of datapath widths. To illustrate this point further, the diversity of datapath component widths is analyzed in more detail next; the analysis is followed by an examination of the logic contained in the maximum width datapath components and in irregular logic; and finally the inherent regularity distribution of the benchmarks is analyzed; and architectural conclusions are drawn based on the results of logic regularity.

7.4.1.1 Diversity of Datapath Widths

Figure 7.4 is a plot of the diversity of datapath component widths versus granularity. The X-axis of the figure represents the granularity of the architecture, M. The Y-axis represents the number of distinct widths of datapath components. The figure contains four curves. The top curve represents the maximum possible number of datapath component widths for each granularity value. The remaining three curves represent the number of distinct widths of datapath components that contain more than 1%, 10%, and 40% of the total number of BLEs, respectively. As illustrated, as the minimum percentage value is increased from 1% to 40%, the diversity of the datapath components is reduced dramatically. For M = 32, for example, there are eight distinct widths of datapath components that contain more than 1% of the total number of BLEs, while only three distinct widths of datapath components contain more than 10% of the total number of BLEs. Finally, for each granularity, only one distinct width of datapath component contains more than 40% of the total number of BLEs. Note that the datapath component widths that contain more than 40% of the total number of BLEs are highlighted in bold prints in Table 7.1. The table clearly shows that, for a given value of M, the datapath component width that contains more than 40% of the total number of BLEs is always equal to the maximum possible datapath component width of the given granularity.

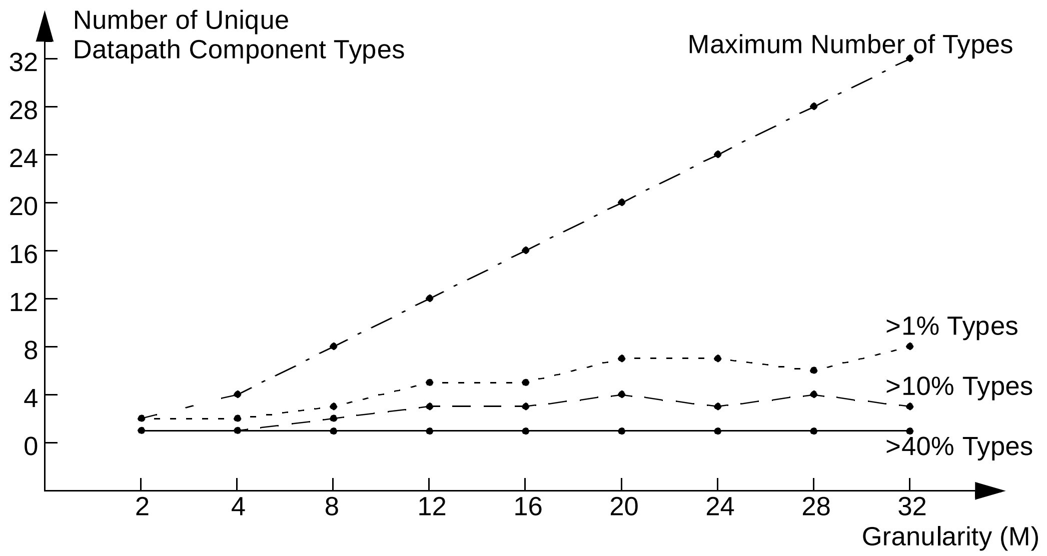


Figure 7.4: Datapath Component Types Containing a Minimum % of BLEs

7.4.1.2 Maximum Width Datapath Components and Irregular Logic

The percentage of BLEs in these maximum width datapath components are plotted in Figure 7.5, where the X-axis represents the granularity values and the Y-axis represents the percentage values. Note that most of the percentage values are much larger than 40%. For M = 2 to 8, the number of BLEs that exist in maximum width datapath components is between 82% and 91%. For the rest of the granularity values, excluding M = 20, the percentage values remain at the 50% to 60% range. Finally, using data from Table 7.1, Figure 7.6 plots the percentage of BLEs in irregular logic versus the granularity of the architecture. It shows that over all granularity values, less than 15% of the total number of BLEs are in irregular logic.

Note that there is a quite large variation for the percentage of BLEs captured in the maximum width datapath components — 91% for M = 2 versus 41% for M = 20. To explain this variance, it is instructive to measure the inherent regularity distribution — the distribution of

BLEs among the various widths of datapath components in the original circuit specifications — of these circuits.

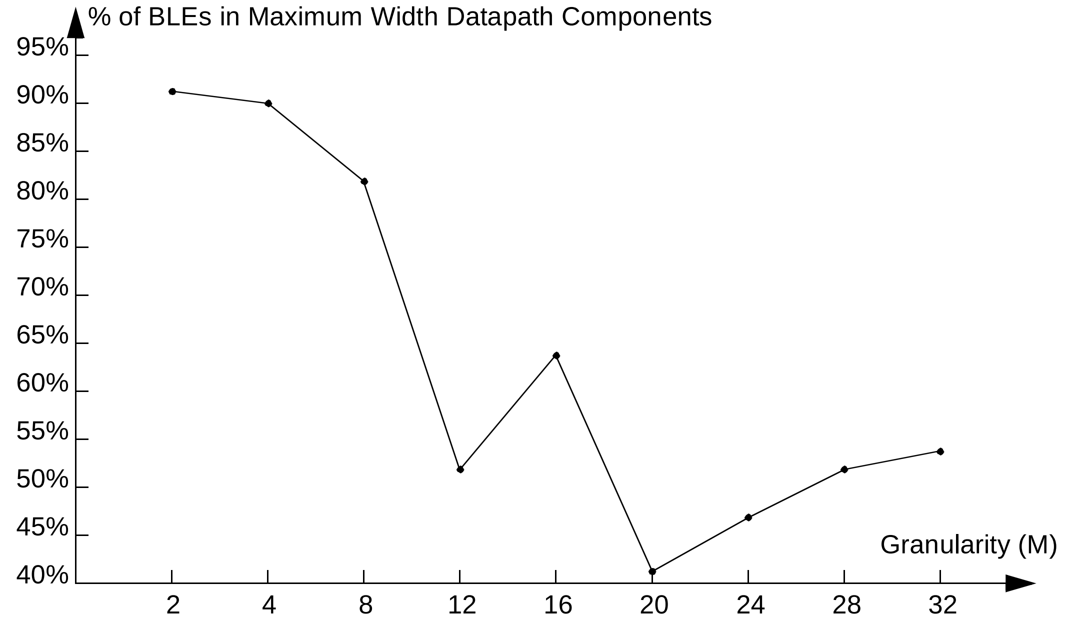


Figure 7.5: % of BLEs in Maximum Width Datapath Components

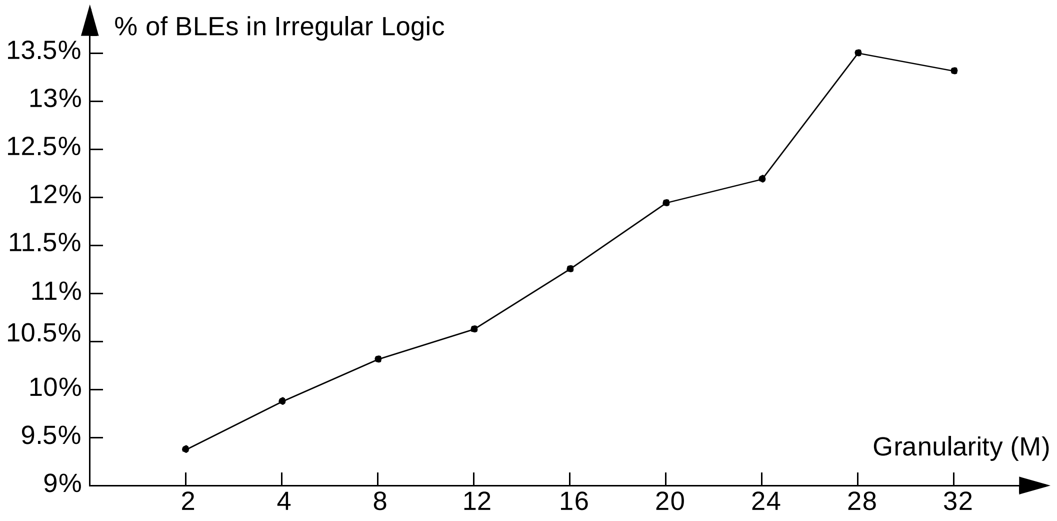


Figure 7.6: % of BLEs in Irregular Logic vs. M

7.4.1.3 Inherent Regularity Distribution

Since the widest datapath in the benchmark circuits is 32 bits wide, the distribution of BLEs among datapath components at M = 32 closely resembles the inherent regularity distribution of these circuits. The detailed distribution of BLEs for M = 32 are restated in Table 7.2. It shows that, for M = 32, 13% of BLEs are in irregular logic. Furthermore, for the remaining datapath component widths, 83% of BLEs are in the six largest categories, including M = 32, 8, 4, 28, 16, and 30, of datapath components. The rest of the datapath components collectively only contain a fraction (around 4%) of the total number of BLEs.

Ranking	Datapath Comp. Width	Percentage of BLEs in Dp. Comp.	Ranking	Datapath Comp. Width	Percentage of BLEs in Dp. Comp.
1	32	54%	**17**	2	0.049%
2	8	14%	**18**	3	0.037%
3	1	13%	**19**	5	0%
4	4	5.9%	**20**	9	0%
5	28	4.1%	**21**	10	0%
6	16	2.7%	**22**	11	0%
7	30	2.0%	**23**	13	0%
8	20	1.1%	**24**	14	0%
9	29	0.71%	**25**	17	0%
10	23	0.57%	**26**	18	0%
11	24	0.44%	**27**	19	0%
12	27	0.33%	**28**	21	0%
13	15	0.18%	**29**	22	0%
14	6	0.15%	**30**	25	0%
15	7	0.086%	**31**	26	0%
16	12	0.074%	**32**	31	0%

Table 7.2: Distribution of BLEs for M = 32

More interestingly, 32, 8, 4, 28, 16, and 30 are all evenly divisible by two, and except 30, all these numbers are evenly divisible by four. This characteristic explains why such a large amount (91%) of BLEs are in 2-bit wide datapath components when M is equal to 2 and equally large amount (90%) of BLEs are in 4-bit wide datapath components when M is equal to 4. It also helps to explain the distribution of BLEs for other granularity values. For example, when M is equal to 8, the largest two groups of datapath components are 8-bit wide and 4-bit wide since 8, 16, and 32 are all evenly divisible by eight and four, and the reminder of 28 divided by 8 is equal to four.

7.4.1.4 Architectural Conclusions

These distribution characteristics of BLEs among datapath components suggest that by using architectures with a granularity value of two or four, one can maximize the number of BLEs captured in the widest available datapath components in these architectures. Furthermore, at these granularity values, logic regularity is concentrated in only one width. Other granularity values have more complex distributions of BLEs among datapath components. This characteristic contributes to the high percentage of maximum-width buses captured by the CAD flow when M is equal to 2 or 4 as shown in Section 7.4.2. Capturing more maximum-width buses is especially desirable for MB-FPGA since the CMS routing tracks are the most efficient when they are used to route maximum-width buses. Note that the best granularity values for capturing datapath regularity is likely to increase proportionally as the maximum width of the datapath applications is increased.

7.4.2 Effect of Granularity on Net Regularity

After packing, net regularity, which is defined as the amount of nets that exist in datapath, can be measures by counting the number of nets that share common source and sink super-clusters with other nets. More precisely, the net regularity is measured by classifying all two-terminal connections that contain one super-cluster output pin and one super-cluster input pin in a circuit into a variety of buses. Each two-terminal connection is called a inter-super-cluster two-terminal connection; and each bus is defined to be a group of two-terminal connections that connect two or three super-clusters together with the following five properties:

1. One super-cluster is the source of all the two-terminal connections in the group; and the remaining super-cluster(s) are the sink(s).
2. Each two-terminal connection in the group has a unique source cluster that is different from the source clusters of all the other two-terminal connections in the group.
3. Each two-terminal connection in the group has a unique sink cluster that is different from the sink clusters of all the other two-terminal connections in the group.
4. All two-terminal connections in the group must have the same amount of shift as defined shortly below in Section 7.4.2.1.
5. The bus contains as much two-terminal connections as it is allowed by property 1 to 4.

The number of two-terminal connections in a bus is called the width of the bus; and an example bus is shown in Figure 7.7. There are three super-clusters in the figure — each with a granularity value of four. These super-clusters are connected together by three two-terminal connections to form a single 3-bit wide bus.

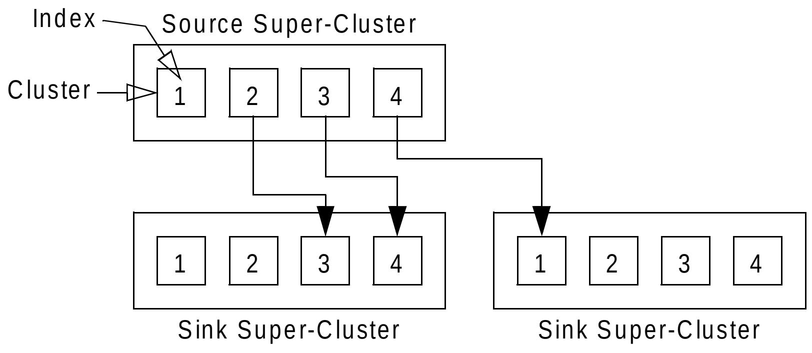


Figure 7.7: A 2-bit wide bus with one-bit shift for M = 4

7.4.2.1 Shift Definition

Buses of the same width are further classified by the amount of shift that each of their two-terminal connections possesses. To determine the amount of shift, each cluster in a super-cluster is first assigned with a unique index number from 1 to M, where M is the granularity of the architecture. The shift of a two-terminal connection is defined to be the difference, d, between the index of the sink cluster and the index of the source cluster if the d is positive. If the d is negative, the shift is defined to be the d + M. Each two-terminal connection shown in Figure 7.7 has a shift value of one, so the bus shown in the figure also has a shift value of one. By definition, the maximum possible width of a bus is M — the granularity of the MB-FPGA architecture; the maximum amount of shift that a bus can have is M - 1; and finally one-bit wide buses represent irregular nets.

The measurement of net regularity as defined above can be used to understand two of the most important questions with regards to MB-FPGA:

1. How many inter-super-cluster two-terminal connections can be grouped into buses and consequently can be efficiently routed through the CMS routing tracks of MB-FPGA?
2. Should the MB-FPGA connection blocks have the capability of shifting buses by 1 to M - 1 bits?

Note that the second question arises from the architectural differences between the MB-FPGA and the DP-FPGA [Cher96] architecture. Recall that, as described in Chapter 2, the DP-FPGA architecture contains special hardware resources called shift blocks, which can perform arithmetic shift operations while routing a group of two-terminal connections. On the other hand, the routing architecture of MB-FPGA, described in Chapter 3, does not contain any hardware support for shift operations. The data presented in this section measure the effect of excluding these dedicated shift operations from MB-FPGA and provides some insights on the actual effectiveness of these dedicated shift hardware resources.

7.4.2.2 Net Regularity Results

Table 7.3 shows the percentage of inter-super-cluster two-terminal connections that exist in each type of buses for the granularity value of 12. The percentage value is calculated by first summing the total number of inter-super-cluster two-terminal connections in a given type of buses over the 15 benchmark circuits. The sum is then divided by the total number of inter-super-cluster two-terminal connections that exist in these benchmark circuits. Each row of the table corresponds to a fixed bus width; while each column of the table corresponds to a fixed amount of shift. Each entry lists the percentage of two-terminal connections that are in the buses with the corresponding width and shift. Again values that are less than 1% are shaded in light gray. The key conclusion to be drawn from this table is that a majority of the bus types contains only a very small amount of inter-super-cluster two-terminal connections; and the same is true for all the other granularity values other than M = 12, whose data are presented in detail in Appendix A.

As shown by the data presented in these tables, M has several important effects on the types and the widths of buses obtained by the CAD flow. In particular, the effect of M on irregular two-terminal connections and the most populous bus types are analyzed below. The analysis is followed by several architectural conclusions.

7.4.2.3 Effect of M on Irregular Two-Terminal Connections

Figure 7.8 plots the percentage of irregular inter-super-cluster two-terminal connections as a function of granularity. The X-axis is the granularity; and the Y-axis is the number of irregular two-terminal connections as the percentage of the total number of inter-super-cluster

Bus Width	Shift											
	0	1	2	3	4	5	6	7	8	9	10	11
1	2.1%	1.7%	1.8%	2.0%	2.2%	2.4%	2.5%	2.5%	2.4%	2.4%	2.4%	2.4%
2	0.80%	1.2%	1.2%	0.96%	0.82%	0.79%	0.98%	0.77%	0.92%	0.79%	0.82%	0.65%
3	0.27%	0.48%	0.29%	0.27%	0.29%	0.28%	0.22%	0.13%	0.27%	0.30%	0.18%	0.21%
4	3.0%	0.066%	0.053%	0.066%	1.7%	0.092%	0.16%	0.16%	1.7%	0.013%	0.066%	0.12%
5	0.050%	0.017%	0.033%	0.033%	0.38%	0.017%	0.0%	0.066%	0.083%	0.017%	0.0%	0.017%
6	0.28%	0.0%	0.079%	0.0%	0.32%	0.020%	0.20%	0.0%	0.020%	0.040%	0.040%	0.020%
7	0.069%	0.21%	0.0%	0.0%	0.046%	0.12%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
8	14%	0.11%	0.0%	0.0%	0.69%	0.0%	0.0%	0.0%	1.1%	0.026%	0.053%	0.18%
9	0.62%	0.059%	0.030%	0.030%	0.089%	0.059%	0.0%	0.030%	0.030%	0.059%	0.030%	0.089%
10	0.43%	0.0%	0.0%	0.0%	0.0%	0.033%	0.066%	0.033%	0.033%	0.066%	0.36%	0.033%
11	0.33%	0.11%	0.036%	0.036%	0.036%	0.073%	0.0%	0.0%	0.036%	0.036%	0.22%	0.40%
12	27%	1.3%	0.24%	0.12%	0.75%	0.12%	0.36%	0.040%	0.75%	0.040%	0.48%	0.36%

Table 7.3: % of Inter-Super-Cluster Two-Terminal Connections Contained in Each Type of Buses for M = 12

two-terminal connections in the benchmark circuits. As shown by the plot, this percentage value remains quite constant across a wide range of granularity values. In particular, the irregular two-terminal connections always consist of 20% to 27% of the total number of two-terminal connections.

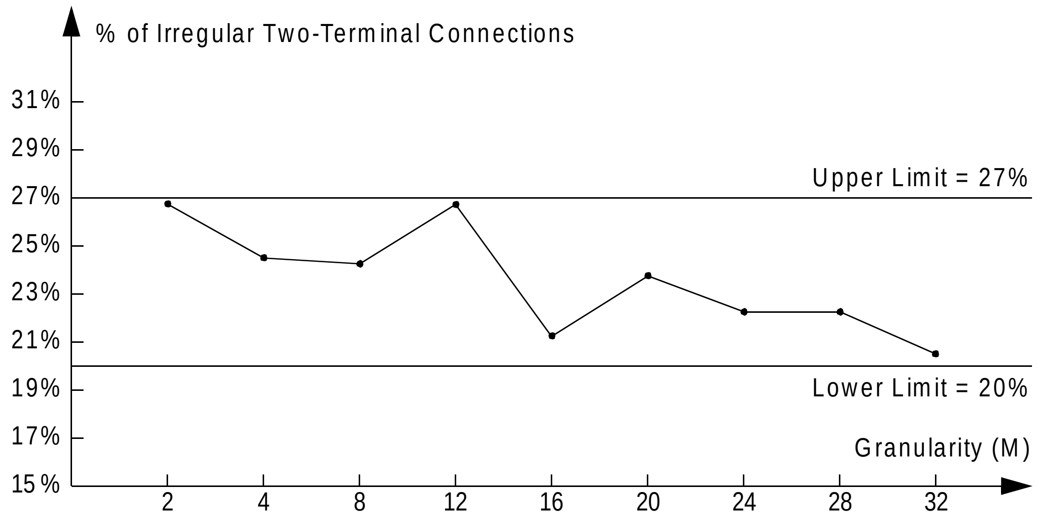


Figure 7.8: % of Irregular Two-Terminal Connections vs. M

7.4.2.4 Effect of M on the Most Populous Bus Types

Figure 7.9 plots the percentage of two-terminal connections in the three bus types that contain the most amount of inter-super-cluster two-terminal connections as a function of granularity, M. The top curve represents the bus type that contains the most amount of two-terminal connections; and the bottom curve represents the bus type that contains the third most amount of two-terminal connections. The exact types of these buses are labeled beside each data point by a pair of integers indicating the width and the shift of the bus. The figure shows that, for all granularity values, the most populous bus type (the bus type that contains the most amount of inter-super-cluster two-terminal connections) is always M-bit wide and non-shifting. The percentage of two-terminal connections in these M-bit wide non-shifting buses decreases quite significantly as M is increased from 2 to 20. The number recovers slightly as M is further increased to 32. The second most populous bus type usually is significantly smaller than the most populous type. It usually is also non-shifting. The majority of the remainder bus types usually contain a very small percentage of the total number of two-terminal connections each. For example, when M is equal to 12, 12-bit wide non-shifting buses are the most populous bus type and consist of 27% of the total number of two-terminal connections. The second most populous bus type is 8-bit wide non-shifting buses, which consist of 14% of the total number of two-terminal connections. The third most populous bus type contains just slightly less than 3% of the total number of two-terminal connections.

7.4.2.5 Architectural Conclusions

The above observations can guide us to choose the appropriate types of routing resources for various CMS architectures. First of all, if there is only one type of connection between the CMS routing tracks and the super-clusters in the architecture, the best choice is M-bit wide non-shifting connections. Secondly, it is likely that regardless of granularity, 20% to 27% of

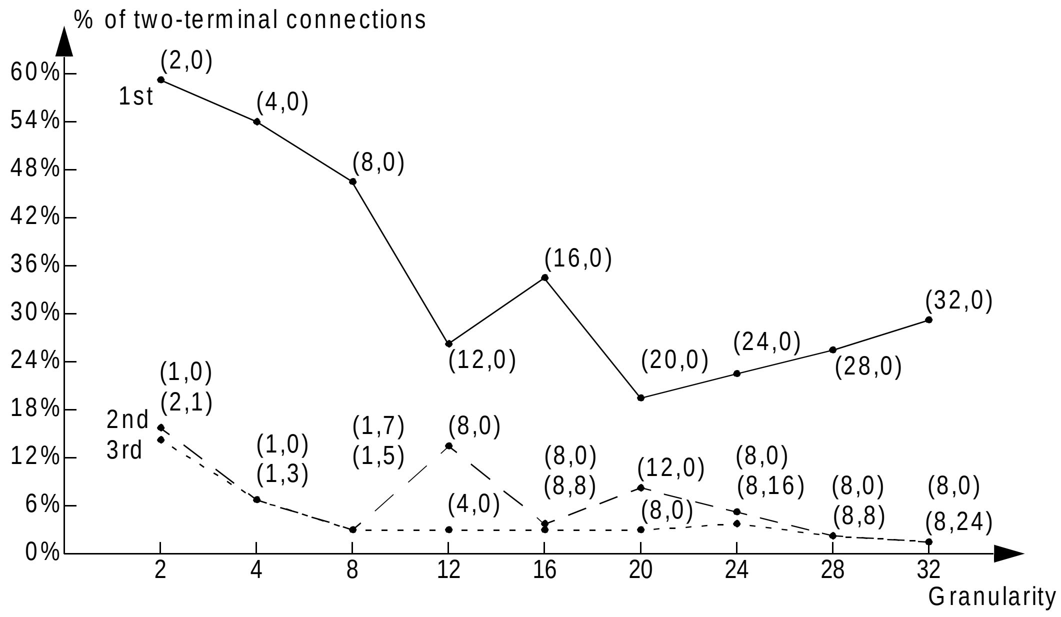


Figure 7.9: The Most Populous Bus Types for Each Granularity

the routing resources should be fine-grain. Furthermore, since the MB-FPGA architecture only has one type of CMS routing tracks — CMS routing tracks of width M, many small buses are routed through fine-grain routing resources. This further increases the demand for fine-grain routing resources. Finally, due to the small amount of M-bit wide and near M-bit wide shifting buses, M-bit wide shift blocks, like those used in the original DP-FPGA architecture [Cher96], will not be cost effective for datapath architectures.

7.5 Summary and Conclusions

This chapter has explored the relationship between the granularity of the MB-FPGA architecture and the regularity preserved by the datapath-oriented CAD flow. The principle conclusions are that regardless of granularity, a majority of BLEs reside in datapath components with only a small number of datapath widths. Furthermore, a great number of BLEs exist in the widest possible datapath components. The inherent regularity along with the net regularity indicates that the granularity values of 2 and 4 are the best for area. Finally, regardless of granularity, it is likely that 20% to 27% of the MB-FPGA routing tracks should be fine-grain; and M-bit wide shift blocks will not be cost effective for the MB-FPGA architecture.

8 The Area Efficiency of MB-FPGA

8.1 Introduction

In this chapter, an experimental approach is used to investigate the effect of the CMS routing capacity of the MB-FPGA architecture on its area efficiency. *CMS routing capacity* is a measure of the routing capability of the CMS routing tracks, and is a function of the granularity of the architecture and the quantity of CMS routing tracks in each routing channel. *Area efficiency* is defined to be the maximum amount of logic that can be implemented in a given amount of area. The experiments consist of implementing a set of circuits on several variants of the MB-FPGA architecture that have different granularity values and varying amounts of CMS routing tracks. The area efficiency is measured over this spectrum of architectures, using the synthesis, packing, and routing algorithms described in Chapter 4, Chapter 5, and Chapter 6, respectively. The effect of these architectures on the resulting speed of the circuits is also measured.

As discussed in Chapter 1, a carefully designed CMS routing architecture can significantly improve the area-efficiency of datapath-oriented FPGAs for arithmetic-intensive applications. During the design process, it is important to match the CMS routing capacity with the routing demands of typical arithmetic-intensive applications. High capacity results in an FPGA that can accommodate all coarse-grain routing demands in CMS routing tracks, but the area efficiency suffers if CMS routing tracks are also used to route fine-grain nets or buses of lesser width. Low capacity, on the other hand, forces wide buses onto routing tracks of lower granularity values, resulting in unrealized area savings.

The datapath-oriented FPGA architecture that will be studied in this chapter is the MB-FPGA model that was introduced in Chapter 3. Figure 8.1 reproduces the overview of the MB-

FPGA architecture, for ease of reference. The architecture shown in the figure has two fine-grain routing tracks and four CMS routing tracks per routing channel. The four CMS tracks are grouped into a single routing bus so the *granularity* of the architecture is four. Recall that the MB-FPGA design requires the number of clusters in each super-cluster to be equal to the number of CMS routing tracks in a routing bus; consequently, there are four clusters in each of the super-clusters shown in the figure. The CMS routing capacity of the MB-FPGA architecture can be altered by changing the number of CMS routing tracks in each routing channel or simultaneously changing the width of the routing buses and the number of clusters in each super-cluster. The relative capacity of CMS routing as compared to fine-grain routing can also be changed by increasing or decreasing the number of fine-grain routing tracks in each routing channel.

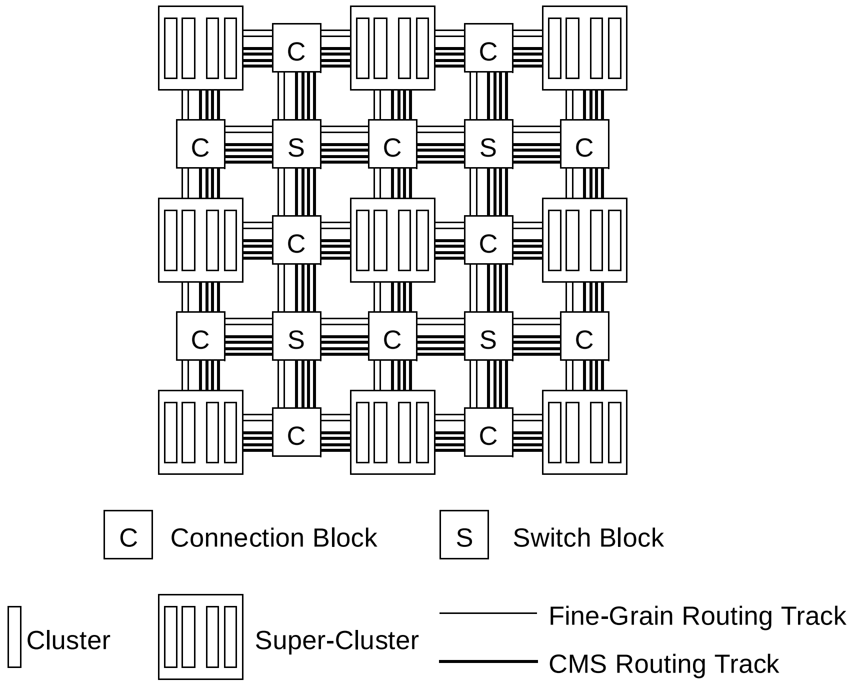


Figure 8.1: The MB-FPGA Architecture

The specific questions concerning the CMS routing resources of MB-FPGA that are answered in this chapter include:

1. What is the effect of the granularity of the MB-FPGA architecture on its area efficiency?
2. What is the effect of the amount of CMS routing tracks on the area efficiency of the MB-FPGA architecture?
3. How does the MB-FPGA architecture compare against conventional FPGA architectures?

Section 8.5.1 shows that the granularity values of 2 and 4 are the best for area. Section 8.5.2 shows that, in order to achieve good area results, nearly half of the routing tracks in each routing channel should be CMS tracks for a wide range of granularity values. Finally, Section 8.5.3 shows that, comparing to the conventional architecture [Betz99a] described in Chapter 2, the MB-FPGA architecture is nearly 10% more area efficient for implementing datapath circuits.

This chapter is organized as follows: Section 8.2 reviews the MB-FPGA architecture and lists architectural parameters involved in this study; Section 8.3 describes the experiment procedure; the limitations of this work are discussed in Section 8.4; Section 8.5 presents experimental results and explanations; and concluding remarks appear in Section 8.6.

8.2 MB-FPGA Architectural Assumptions

The set of potential FPGA architectures is an extremely large design space. For example, 22 architectural parameters are needed to completely describe the MB-FPGA architecture presented in Chapter 3. Conventional architectures can be characterized using fewer design parameters since they only contain fine-grain resources. Nevertheless, fourteen architectural parameters are still needed the to completely characterize the conventional FPGA architecture described in Chapter 2. In addition to the architectural parameters, one also has to specify the size of each type of transistor in order to get meaningful area and speed measurements.

This combination of parameters creates a design space that is too large to be explored completely. The study conducted in this chapter uses a more intelligent exploration strategy where many of these parameters are set to be know good values from previous FPGA studies. Care is also taken in the parameter selection process to ensure a fair comparison between the MB-FPGA architecture and the conventional FPGA architecture.

In the remainder of this section, the architectural parameters involved in this study are first summarized and defined. Then the selected values of each parameter are presented and justified. Finally, the transistor sizing issues are discussed in detail.

8.2.1 A Summary of Architectural Parameters

The 22 architectural parameters that completely define the MB-FPGA architecture are listed in Table 8.1. The first column shows the classification of the architectural parameters; and column 2 lists the symbols that will be used in this chapter to reference each of these parameters. As shown by the table, the parameters can be classified into five categories. The first category, routing capacity parameters, contains three members, M, W_f, and W_c. These parameters characterize the routing capacity of the fine-grain and the CMS routing tracks in each routing channel of the MB-FPGA architecture. The second category, super-cluster parameters, contains four members, K, N, I, and T_p. Along with M, these parameters completely define the structure of the MB-FPGA super-clusters. The third and the fourth category, connection block and switch block parameters, contain seven members each. They define the complete structures of the connection blocks and the switch blocks, respectively. Finally the pad_ratio parameter defines the I/O block characteristics of the MB-FPGA architecture.

Class.	Symb.	Conv. FPGA	Definition
Routing Capacity Parameters	M	no	the granularity of the architecture
	W_f	yes	the number of fine-grain routing tracks in a routing channel
	W_c	no	the number of CMS routing tracks in a routing channel

Table 8.1: MB-FPGA Architectural Parameters

Class.	Symb.	Conv. FPGA	Definition
Super-Cluster Parameters	K	yes	LUT size — the number of inputs that a LUT has
	N	yes	the number of BLEs per cluster
	I	yes	the number of cluster inputs per cluster
	T_p	yes	the topology of the physical placement of super-cluster inputs and outputs
Connection Block Parameters	Fc_if	yes	the number of fine-grain routing tracks that a super-cluster input connects to as a percentage of W_f
	Fc_ic	no	the number of routing buses that a super-cluster input bus connects to as a percentage of W_c/M
	Fc_of	yes	the number of fine-grain routing tracks that a super-cluster output connects to as a percentage of W_f
	Fc_oc	no	the number of routing buses that a super-cluster output bus connects to as a percentage of W_c/M
	Fc_pf	yes	the number of fine-grain routing tracks that an I/O block input/output pin connects to as a percentage of W_f
	Fc_pc	no	the number of routing buses that an pad-input/pad-output bus connects to as a percentage of W_c/M
	T_I	yes	the topology of the physical placement of isolation buffers
Switch Block Parameters	T_s	yes	the switch block topology
	Fs_f	yes	the fine-grain flexibility of the switch blocks — the number of fine-grain tracks that each fine-grain track connects to in a switch block
	Fs_c	no	the CMS flexibility of the switch blocks — the number of routing buses that a routing bus connects to in a switch block
	L_f	yes	the length of a fine-grain routing track
	L_c	no	the length of a CMS routing track
	S_f	yes	the type of fine-grain routing switches that connect fine-grain routing tracks to each other in a switch block
	S_c	no	the type of CMS routing switches that connect routing buses to each other in a switch block
I/O Block Parameter	pad_ratio	yes	MB-FPGA: the number of I/O blocks residing on one side of a super-cluster conventional FPGA: the number of I/O blocks residing on one side of a cluster

Table 8.1: MB-FPGA Architectural Parameters

The 14 architectural parameters that describe the conventional FPGA architecture presented in Chapter 2 is a subset of the 22 parameters listed in Table 8.1. Column 3 of the table indicates if a parameter listed in Table 8.1 also can be used to describe the conventional FPGA architecture. The definition of each parameter is given in column 4. These definitions are self explanatory when Chapter 2 and Chapter 3 are referenced for the conventional FPGA and the MB-FPGA architectures, respectively.

8.2.2 Parameter Values

The values used for each parameter are listed in Table 8.2, where the classification and the symbol of each parameter are listed in column 1 and 2, respectively. As shown by column 3, all 22 parameters are involved in the investigation of question 1 and 2. M, W_f, and W_c are the variables of the investigation. K is set to be 4 since it has been shown that 4-LUTs are one of the most efficient LUT sizes [Rose90] [Ahme00] and this LUT size is also used in many commercial FPGAs [Alte02] [Xili02]. N and I are set to be 4 and 10 respectively since this combination was shown to be one of the most efficient by [Betz97b] [Betz98] [Betz99a] and is used in many previous FPGA studies [Ahme00] [Betz99a] [Betz99b] [Lemi01] [Marq99] [Marq00a] [Marq00b] [Sank99]. T_p and T_I are discussed in detail in Section 8.2.2.1 and Section 8.2.2.2, respectively.

For the remaining parameters, both Fc_ic and Fc_if are set to be 0.50; and Fc_oc and Fc_of are set to be 0.25. These values were found to generate good area results by the study done in [Betz99a] for fine-grain routing resources. Due to the lack of studies on Fc_pf, both Fc_pc and Fc_pf are set to be 1.00 as it is done in [Betz99a] for fine-grain routing tracks. To minimize the impact of not using the best values for Fc_pc and Fc_pf, the area of the input and output connection blocks of the I/O blocks are excluded from the total area count. This exclu-

Class.	Param.	Val. for Q1 & Q2	Val. for Q3	
			MB-FPGA	Conv. FPGA
Routing Capacity Parameters	M	variable	4	n.a.
	W_f	variable	variable	variable
	W_c	variable	variable	n.a.
Super-Cluster Parameters	K	4	4	4
	N	4	4	4
	I	10	10	10
	T_p	see Sec. 8.2.2.1	see Sec. 8.2.2.1	see Sec. 8.2.2.1
Connection Block Parameters	Fc_if	0.5	best	best
	Fc_ic	0.5	equal to Fc_if	n.a.
	Fc_of	0.25	best	best
	Fc_oc	0.25	equal to Fc_oc	n.a.
	Fc_pf	1.0	best	best
	Fc_pc	1.0	equal to Fc_pf	n.a.
	T_I	see Sec. 8.2.2.2	see Sec. 8.2.2.2	see Sec. 8.2.2.2
Switch Block Parameters	T_s	disjoint	disjoint	disjoint
	Fs_f	3	3	3
	Fs_c	3	3	n.a.
	L_f	2	best	best
	L_c	2	equal to L_f	n.a.
	S_f	bi-directional buffered	bi-directional buffered	bi-directional buffered
	S_c	bi-directional buffered	bi-directional buffered	n.a.
I/O Block Parameters	pad_ratio	core-limited	core-limited	core-limited

Table 8.2: Values for Architectural Parameters

sion simulates the real-life design practices of keeping both datapath and its control logic on the same chip in order to reduce the total number of chip-level I/Os.

The disjoint switch block topology [Hsei90] with Fs_f and Fs_c set to be three is used for both the fine-grain track connections and the CMS routing bus connections since this is one of the most efficient and widely used topologies for conventional FPGAs. A fully buffered global routing architecture is also assumed — all switches in the switch blocks are buffered switches — since buffered switches are widely used in many current commercial FPGAs [Lewi03]

[Alte02] [Xili02]. The track length is measured in terms of the number of super-clusters that a routing track passes before it is interrupted by a switch. It is set to be two for both the CMS and the fine-grain routing tracks. The track length of two along with the cluster size of four were found to generate good area results in [Betz99a] for conventional FPGAs. Finally, the pad_ratio is assumed to be sufficiently high so all benchmarks are core-limited — the minimum area required to implemented each benchmark is bounded by the area of the logic and routing resources needed instead of the area required to implement the I/O blocks of the benchmark.

The investigation of question 3 requires the definition of two sets of independent architectural parameters. One set describes the MB-FPGA architecture and is shown in column 4 of Table 8.2. The other set describes the conventional FPGA architecture and is shown in column 5. For MB-FPGA, W_f and W_c are the variables of the investigation. M is set to be 4 since it is shown to be one of the most area efficient granularity values by the results of question 1. K is again set to be 4; and again all benchmarks are assumed to be core-limited.

The rest of the parameters can be classified into two groups — one group describes the topological features of the architecture and the other group consists of single numerical values. The topological parameters, including T_p, T_I, T_s, Fs_f, Fs_c, S_f, and S_c, are set to be the same values as the ones used in the investigation of question 1 and 2. Two of the numerical parameters, N and I, describe the super-cluster structure; and they are also set to be the same values as the ones used to address question 1 and 2.

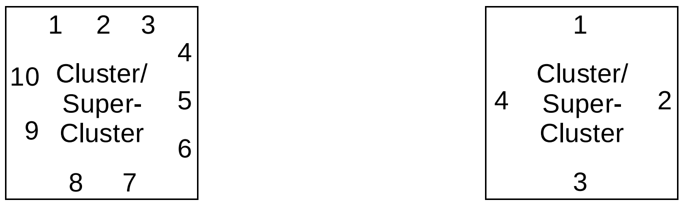
The remaining parameters describe the MB-FPGA routing architecture. As shown in Table 8.2, it is assumed that the architectural parameters that describe the CMS routing resources are always equal to their corresponding parameters that describe the fine-grain routing resources. Since the routing resources usually consume the majority of FPGA area, these fine-grain parameters, including Fc_of, Fc_if, Fc_pf, and L_f, are systematically searched to ensure that the best possible area results are obtained for the MB-FPGA architecture. These experimentally determined values will be presented in detail in Section 8.5.3.

To fairly compare the area efficiency of the conventional architecture with the MB-FPGA architecture, the corresponding numerical parameters of the conventional architecture, including Fc_of, Fc_if, Fc_pf, and L_f, are also systematically searched to find a set of values that generate the best area. These experimentally determined values will be presented in detail in Section 8.5.3. Finally, for the conventional architecture, all benchmark circuits are assumed to be core-limited; and all other parameters to are set to be the most area efficient values based on the results of the previous studies, which are summarized for question 1 and 2.

8.2.2.1 Physical Placement of Super-Cluster Inputs and Outputs

In Table 8.2, T_p represents the distribution topology of the input and output pins for an MB-FPGA super-cluster or a conventional FPGA cluster. For the conventional FPGA, the cluster inputs and outputs are assumed to be uniformly distributed around the perimeter of each logic cluster similar to the distribution topology used in [Betz99a] [Betz01]. This distribution topology takes the advantage of the logical equivalency among the cluster inputs or outputs [Betz99a]. An example of the distribution topology is shown in Figure 8.2. Here each number represents either a cluster input or a cluster output.

The MB-FPGA uses a similar distribution topology for the super-cluster inputs and outputs. However, instead of uniformly distributing cluster inputs or cluster outputs, the input buses or output buses are uniformly distributed. For the MB-FPGA architecture, each number in Figure 8.2 represents an input/output bus instead of an individual cluster input/output.



(a) Physical Placement of 10 Cluster Inputs or Super-Cluster Input Buses

(b) Physical Placement of 4 Cluster Outputs or Super-Cluster Output Buses

Figure 8.2: T_p for FPGA Architectures with N = 4 and I = 10

Again this uniform distribution topology takes the advantage of the logical equivalency among input buses or output buses when carry chains are not used.

8.2.2.2 Physical Placement of Isolation Buffers

In Table 8.2, T_I represents the physical placement of the isolation buffers in either the MB-FPGA architecture or the conventional FPGA architecture. Recall that the function of the isolation buffers is to electrically isolate the routing tracks from the input connection blocks. For the conventional FPGA architecture, each routing track has one isolation buffer for each cluster position that it passes [Betz99a]. An example is shown in Figure 8.3 where an X indicates the presence of an isolation buffer. In the figure, the conventional FPGA consists of 16 clusters, 5 horizontal routing channels, and 5 vertical routing channels. There is one routing track in each routing channel; and these tracks are labeled x1 to x5 and y1 to y5. In total, there are 40 isolation buffers in the figure. In general, the total number of isolation buffers, C, in a conventional architecture can be determined by the following formula:

$$C = W \times (X \times (Y + 1) + (X + 1) \times Y),$$

where W is the number of routing tracks in each routing channel. X is the number of rows of clusters; and Y is the number of columns of clusters.

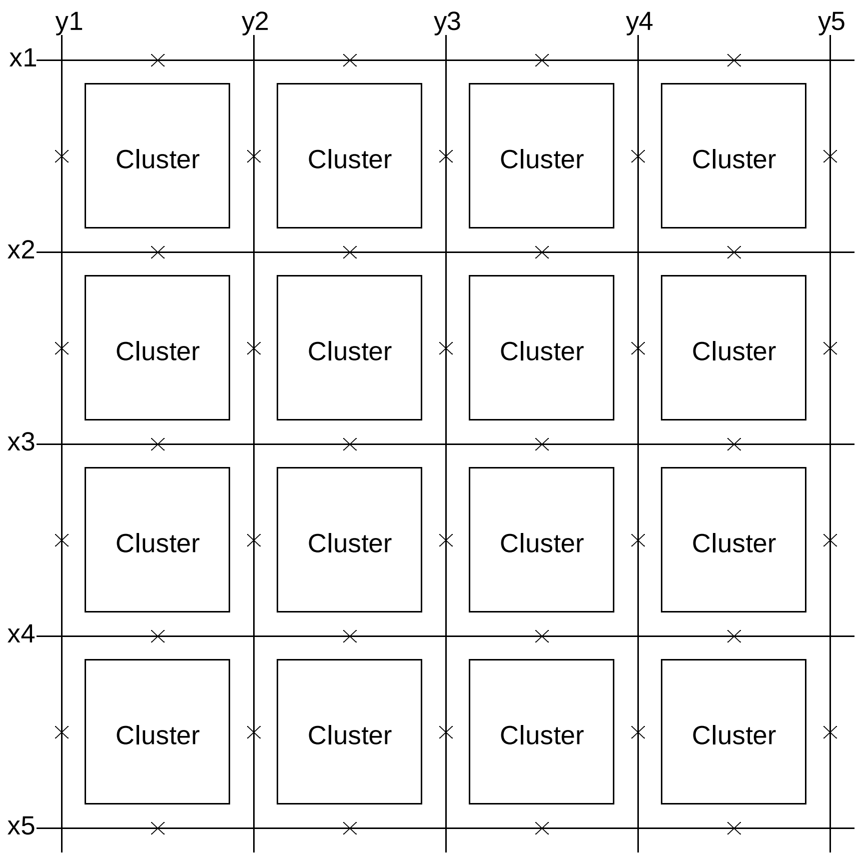


Figure 8.3: Isolation Buffer Topology for Conventional FPGA

For the MB-FPGA architecture, electrically, it is also sufficient to place only one isolation buffer for every super-cluster position that a routing track passes. However, this topology gives the MB-FPGA architecture an unfair area advantage since it needs only half of the isolation buffers as compared with an equivalent conventional FPGA architecture. This unfairness is illustrated by Figure 8.4, which is a transformation of Figure 8.3. The conventional architecture in Figure 8.3 is transformed into the MB-FPGA architecture shown in Figure 8.4 by rearranging the routing tracks and the clusters. In Figure 8.4, every four clusters are grouped into a super-cluster. As shown, only 20 isolation buffers are needed for the new architecture. In general, the total number of isolation buffers, C’, needed in the transformed MB-FPGA architecture is determined by the formula:

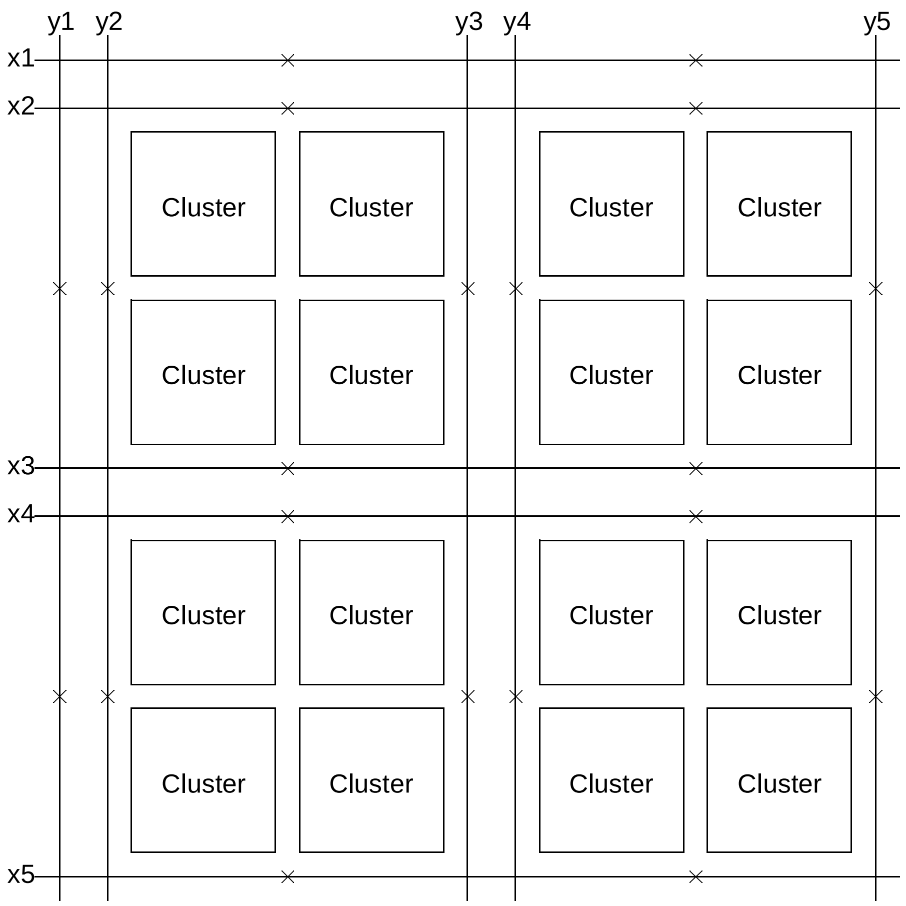


Figure 8.4: Equivalent MB-FPGA Architecture

$$C' = W \times \left(\frac{X \times (Y+1)}{\lceil \sqrt{M} \rceil} + \frac{(X+1) \times Y}{\lceil \sqrt{M} \rceil} \right) = \frac{C}{\lceil \sqrt{M} \rceil}.$$

Since isolation buffers do not influence the overall routing capacity of the FPGAs, this reduction in isolation buffers would unfairly advantage the MB-FPGA architecture in area measurements. Throughout this study, extra isolation buffers are added to the MB-FPGA architecture to cancel this unfair advantage. For the MB-FPGA architecture shown in Figure 8.4, two isolation buffers, instead of one, are counted for every super-cluster position that a track passes. Note that the adjusted isolation buffer placement slightly disadvantages the MB-FPGA architecture if all routing channels in Figure 8.4 contain two routing tracks.

8.2.3 Transistor Sizing

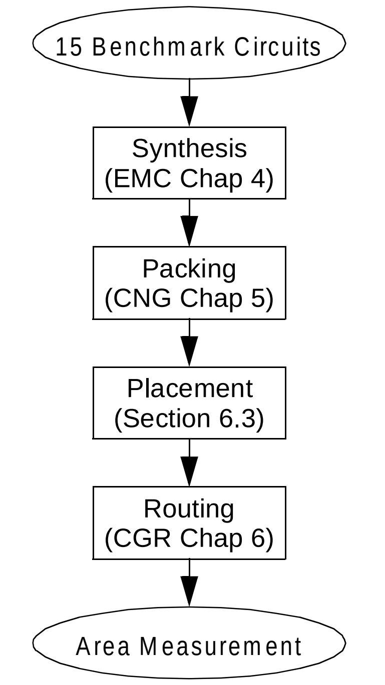
Each type of transistor used in the experiments is sized according to the methodology laid out in [Betz99a] based on the TSMC $0.18\mu m$ process. Both the MB-FPGA architecture and the conventional FPGA architecture use the same transistor sizes for the same transistor types.

In particular, for configuration memory, each of the six transistors in an SRAM cell is sized to be minimum width; and each tri-state buffer in a buffered switch is sized to have a drive strength that is five times of the minimum drive strength. Note that the difference between the SRAM size and the tri-state buffer size is one of the major factors in determining the effectiveness of configuration memory sharing in increasing area efficiency. Unlike the current study, none of the previous studies [Cher96] [Leij03] have detailed enough models to take this sizing effect into account.

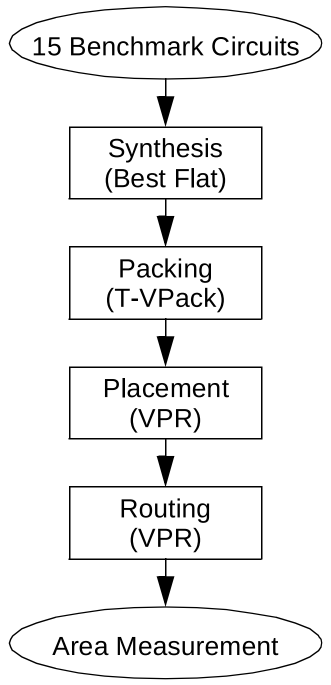
8.3 Experimental Procedure

This section describes the experimental procedure that is used to investigate the MB-FPGA routing architecture. The CAD flow used throughout this investigation is shown in Figure 8.5(a). Its input consists of fifteen benchmark circuits from the Pico-Java processor from SUN Microsystems [Sun99]. The benchmark set covers all major datapath components of the processor. These circuits are synthesized into LUTs using the EMC datapath-oriented synthesis process described in Chapter 4. This synthesis process preserves the regularity of datapath circuits while attempting to minimize area.

The synthesized circuits are then packed into super-clusters using the CNG datapath-oriented packing algorithm as described in Chapter 5. The packing tool tries to pack adjacent bit-



(a) CAD Flow for the MB-FPGA Architecture



(b) CAD Flow for A Conventional FPGA Architecture

Figure 8.5: CAD Flows

slices into a series of super-clusters. The packer also utilizes the super-cluster level carry connections to minimize the delay of carry chains. The packed circuits are then placed using a placement algorithm modified from the VPR placer [Betz99a] as described in Section 6.2. The algorithm moves super-clusters as a basic unit if they contain grouped bit-slices. Otherwise, non-datapath clusters are optimized individually. The placed circuits are then routed using the CGR datapath-oriented router described in Chapter 6, which is modified to efficiently use the CMS routing resources. Using a set of specially designed cost functions, the router tries to balance the use of the fine-grain routing resources with the use of the CMS routing resources based on congestion and the goal of timing optimization.

The area results used to address question 1, 2, and 3 are measured at the end of the CAD flow. There are two options for averaging the area results across the fifteen benchmark circuits. They are the geometric averaging method, which weights each circuit equally regardless of its size, and the arithmetic averaging method, which gives proportionally more weight to larger circuits. For this study, the arithmetic averaging method is used, so the results contain a higher percentage of contribution from the larger benchmark circuits in the benchmark suite.

Figure 8.5(b) shows the flow used for the conventional FPGA architecture, whose area results are used in comparison with the MB-FPGA architecture to address question 3. For this flow the best available flat synthesis results of Chapter 4 is used instead of the regularity preserving datapath synthesis. The T-VPack algorithm is used for packing; and the VPR tools [Betz99a] are used for placement and routing.

8.4 Limitations of this work

This section discusses the effects of the architectural assumptions and the experimental procedure on the accuracy and the implication of the results that are presented later in this chapter.

The models that have been used for the MB-FPGA and the conventional FPGA architectures are highly realistic. As a result, each model contains a large amount of architectural parameters. Because of this high degree of parameterization, it is impossible to obtain the best architectures through a full exploration of the design spaces. Instead, the values of many architectural parameters are selected based on previous studies on conventional FPGA architectures as practical time limits preclude complete explorations. For the MB-FPGA architecture, these are only "best-guessed" values since the MB-FPGA contains several significant differences from the previous conventional FPGA architectures.

The benchmarks used in this study are all regular datapath circuits. The effectiveness of the architecture in implementing irregular control logic is not examined. Furthermore, since the benchmarks are mainly 32-bit wide datapath circuits, the effectiveness of the MB-FPGA architecture in implementing wider datapath circuits can only be inferred from these results.

Finally, as all other empirical studies, the accuracy of the results that are presented in this chapter depends on the quality of the CAD tools employed in the investigation — the results reflect what is achievable by the current state-of-the-art tools; and future results might vary with the development of the CAD technology.

8.5 Experimental Results

The experimental results that are presented here are based on the fifteen benchmark circuits used in the verification experiments from Chapter 4 through Chapter 6. Various aspects of these circuits were described in Table 4.1, Table 5.1, and Table 6.1. Furthermore, their regularity was quantified and analyzed in detail in Chapter 7. In this section, these circuits are used to investigate the effect of M and W_c on the area efficiency of MB-FPGA. They are also used to compare the area efficiency of the MB-FPGA architecture against the conventional FPGA architecture.

8.5.1 Effect of Granularity on Area Efficiency

While Chapter 7 indirectly investigates the architectural implications of granularity through its effect on logic and net regularity, the direct effect of granularity on the area efficiency of MB-FPGA is examined here. Note that this examination is based on the set of parameters listed in column three of Table 8.2; and in order to separate out the effect of granularity on coarse-grain logic from its effect on coarse-grain routing, MB-FPGA architectures containing only fine-grain routing tracks are first investigated. Then CMS routing tracks are added to these architectures to fully explore the effect of granularity on MB-FPGA.

8.5.1.1 MB-FPGA Architectures with No CMS Routing Tracks

Figure 8.6 plots the average area required to implement the benchmark circuits versus M for MB-FPGA architectures that contain no CMS routing tracks. The X-axis represents the granularity values. The Y-axis represents the average area; and it is measured using the equivalent minimum-width transistor area model as described in [Betz99a]. As shown, the granularity values of 2 and 4 consume the least area. As the granularity value is increased beyond 4, the average area required increases significantly.

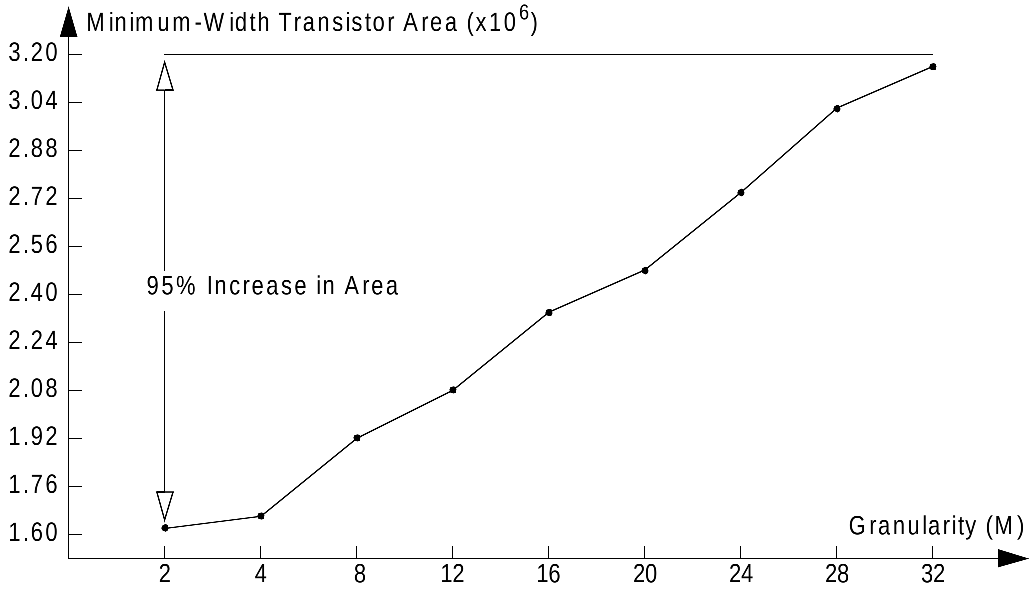


Figure 8.6: Total Area vs. M with No CMS Routing Tracks

The primary cause of this increase is due to the large demand for routing created by increased super-cluster capacities, which is a consequence of the high granularity values. As the granularity value is increased, the logic capacity of the super-clusters increases proportionally. For example, for a 32-bit wide architecture, there are 128 look-up tables, 320 inputs and 128 outputs per super-cluster. Such large super-clusters need to be served by very wide routing channels; and in wide routing channels, each super-cluster input is connected to many routing tracks. This results in a very large input connection block. Similarly, the size of the output connection blocks also increases significantly as the routing channels get wider. It is observed, through our experiments, that this area increase cannot be countered by simply reducing the values of Fc_if and/or Fc_of. (For example, no further area savings can be achieved by reducing Fc_of to less than 0.25, which is the most area efficient Fc_of value for N = 4 [Betz99a].) As a result, the area consumed by the global routing resources is significantly increased as M is increased.

Note that the cause of this area increase is very different from a similar scenario described in [Betz99a]. In [Betz99a], for the conventional FPGA architecture, the area that required to implement a set of benchmark circuits increases as the cluster size, N, is increased. The cause of this area inflation, however, is due to the quadratic increase in area consumed by the local routing resources inside the clusters.

In the current experiment, the local routing resource area consumed by each super-cluster increases only linearly with respect to the granularity value, M, since N and I are fixed at 4 and 10, respectively. As a result, the logic area — the total area consumed by the super-clusters — remains relatively constant. Figure 8.7 plots this logic area versus M averaged over the benchmarks. The figure shows that the average logic area increases only by 13% as M is increased from 2 to 32. In contrast, in Figure 8.6, the average total area (the logic area plus the global routing area) is increased by 95% over the same range of granularity values.

8.5.1.2 MB-FPGA Architectures with CMS Routing Tracks

Figure 8.8 plots the average area required to implement each benchmark circuit versus M for MB-FPGA architectures with CMS routing tracks. The X-axis is M; and the Y-axis is the minimum-width transistor area. The plot contains two curves. The top curve represents the MB-FPGA architectures with no CMS routing tracks and is the same curve that was shown in Figure 8.6. The bottom curve represents the most area efficient MB-FPGA architectures that

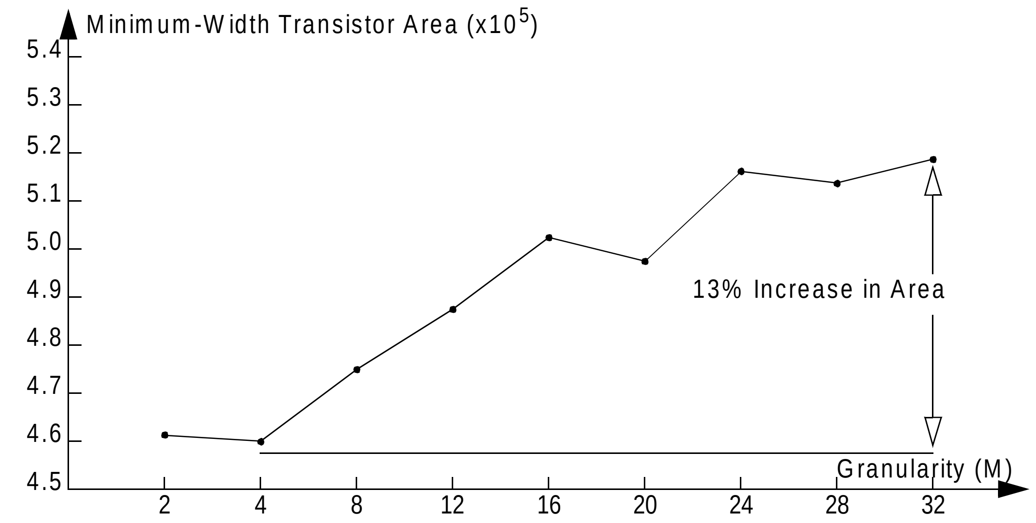


Figure 8.7: Logic Area vs. M with No CMS Routing Tracks

contain CMS routing tracks. The percentage of CMS routing tracks in each routing channel, which is calculated by the formula $\frac{W_c}{W_c + W_f}$, is also labeled beside each data point on the bottom curve.

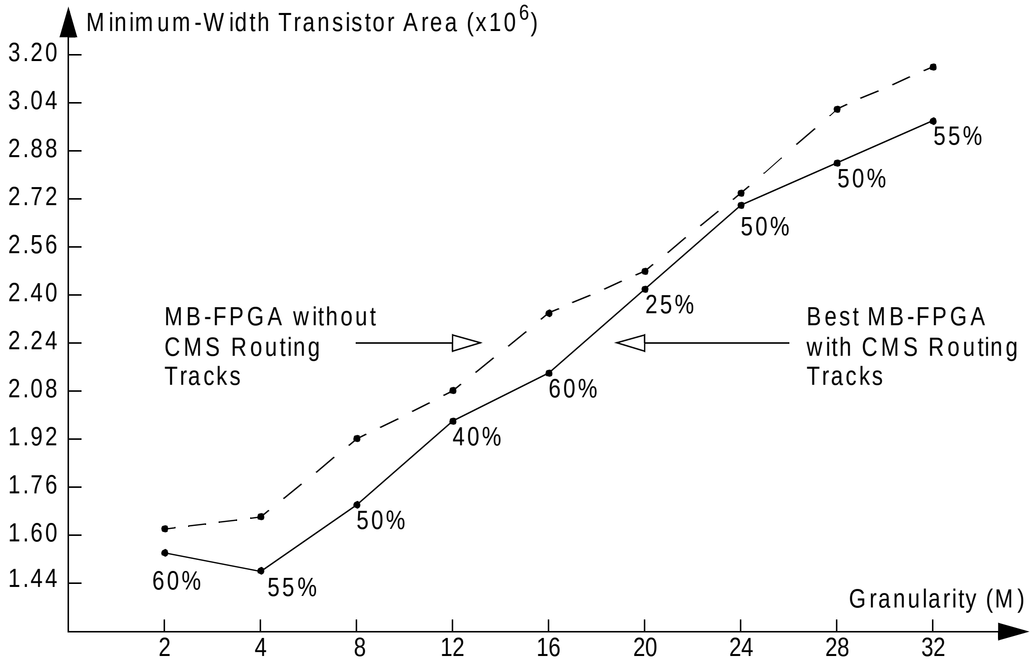


Figure 8.8: Area vs. M with CMS Routing Tracks

As shown, for all granularity values, CMS routing tracks can be effectively used to increase the area efficiency of MB-FPGA. For example, when M is equal to 2, the best architecture with CMS routing tracks is 5.6% smaller than the architecture with no CMS routing tracks. When M is equal to 4, the best architecture with CMS routing tracks is 11% smaller than the architecture with no CMS routing tracks. Overall, the most area efficient MB-FPGA architecture, shown in Figure 8.8, has a granularity value of 4 and contains a CMS track count that is equal to 55% of the total number of tracks in each routing channel.

8.5.2 Effect of Proportion of CMS Tracks on Area Efficiency

Figure 8.9 plots the average area consumed by each benchmark circuits against the percentage of CMS routing tracks in each routing channel. The X-axis represents the percentage of CMS routing tracks per channel. The Y-axis represents the area. There are 9 curves in the figure; and each curve represents an MB-FPGA architecture with a fixed granularity value of M. An X marks the location of the minimum area on each curve. As shown, except M = 20, the most area efficient proportion of CMS routing tracks remains relatively constant at between 40% to 60% range for all granularity values. (Note that for M = 20, less CMS routing tracks are needed since, as discussed in Chapter 7, relative small amount of logic and net regularity are captured by the CAD flow at this granularity.)

This relative consistency of the best proportion of CMS routing tracks can be explained by the percentage, P, of inter-super-cluster two-terminal connections that can be grouped into M-bit wide buses after packing. These connections can be most efficiently routed through the CMS routing tracks. For each granularity value in the graph, P is marked by an O. The graph shows that the most area efficient percentage of CMS routing tracks is around P + 10% for smaller granularity values of 2, 4, and 8, and around P + 30% for larger granularity values of 16, 24, 28, and 32. Interestingly, at smaller granularity values, more two-terminal connections

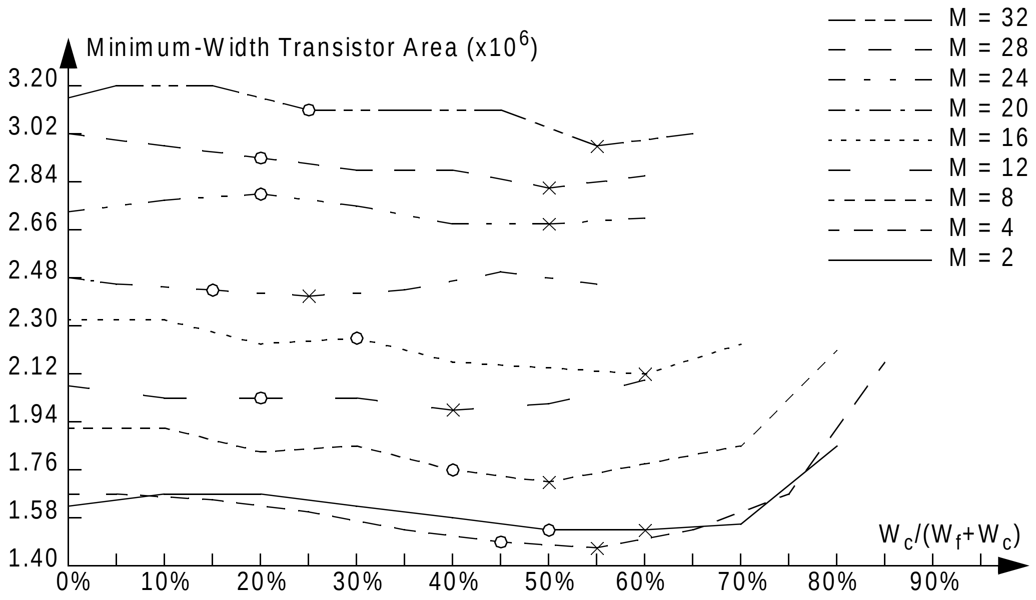


Figure 8.9: Area vs. Proportion of CMS Tracks

are captured in buses by the CAD flow, so P already is a quite large number. For larger granularity values, however, fewer connections are captured in buses; and P is relatively small.

The main reason for the most area efficient percentage value to deviate more from P at larger granularities is that, at high granularity values, the configuration memory area is amortized among a greater number of CMS routing tracks. As a result, the cost per track for the CMS routing tracks decreases as the granularity value is increased. This reduction in cost warrants the increased use of CMS routing tracks relative to P.

8.5.3 MB-FPGA Versus Conventional FPGA

This section compares the MB-FPGA architecture against the conventional FPGA architecture by measuring the average area required to implement the benchmark circuits on each of the architectures. Two sets of experiments were performed. The first set of experiments determines the most area efficient values for the numerical parameters describing the routing architectures. Using these values, the second set of experiments measures the area, as well as the performance, of the benchmark circuits implemented on the two architectures. Each set of these experiments is described in turn.

8.5.3.1 Parameter Results

The first set of experiments systematically searches four numerical parameters, including Fc_of, Fc_if, Fc_pf, and L_f. A divide-and-conquer approach is used to reduce the number of searches required to explore this four dimensional design space. First, the most area efficient value of Fc_of is determined. Then the best values for Fc_if and Fc_pf are determined using an iterative approach. Finally, the most area efficient value of L_f is determined experimentally. To further reduce the number of searches, only the MB-FPGA architectures that contain no CMS routing tracks are considered in most of the experiments. It is assumed that these results are generally true across all MB-FPGA architectures.

Fc_of

As it is described in [Betz99a], for the conventional FPGA architecture, the most area efficient values for Fc_of is equal to $\frac{1}{N}$. Using the same set of experiments, it is found that the same formula applies to the MB-FPGA architecture containing no CMS routing tracks.

Fc_if and Fc_pf

An iterative approach is used to determine the most area efficient values for Fc_if and Fc_pf using the MB-FPGA architecture without CMS routing tracks. For iteration 1, the Fc_pf is set to be 1.00. Figure 8.10 plots the average area consumed by the routing resources against Fc_if in implementing the benchmark circuits. The X-axis in the figure represents Fc_if. The Y-axis represents the area. As shown, the best area is generated when Fc_if is equal to 0.5.

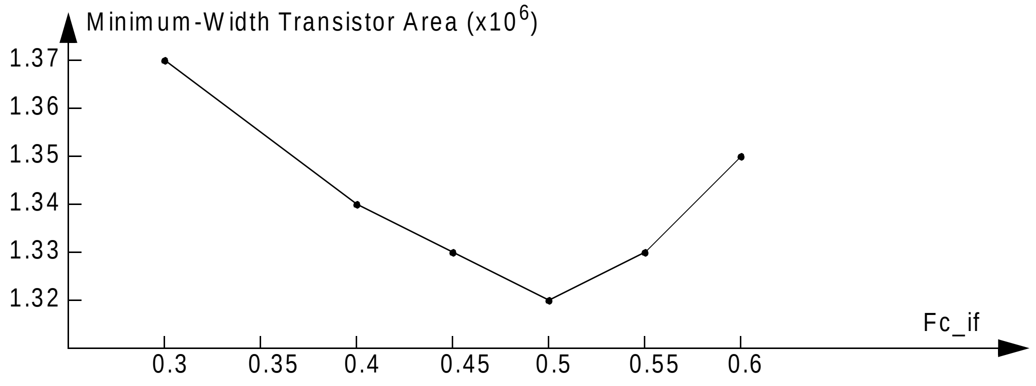


Figure 8.10: Iteration 1: Routing Area vs. Fc_if for Fc_pf = 1.00

In the second iteration, Fc_if is set to be 0.5, which is the most area efficient value in iteration 1. Fc_pf is plotted against the routing area as shown in Figure 8.11. As shown, the best area is achieved when Fc_pf is equal to 0.2. Finally in the third iteration, Fc_pf is set to be 0.2; and Figure 8.12 plots the average routing area against Fc_if. The best area is achieved when Fc_if is equal to 0.4. Subsequent iterations confirm that the most area efficient values for Fc_pf and Fc_if is equal to 0.2 and 0.4, respectively. Finally, using the same set of experiments, the most area efficient values of Fc_pf and Fc_if are determined to be 0.5 and 0.4, respectively, for the conventional FPGA architecture.

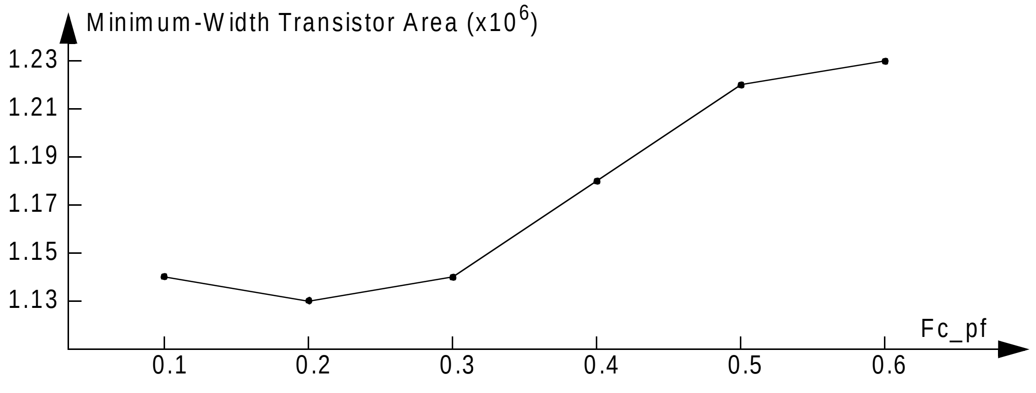


Figure 8.11: Iteration 2: Routing Area vs. Fc_pf for Fc_if = 0.5

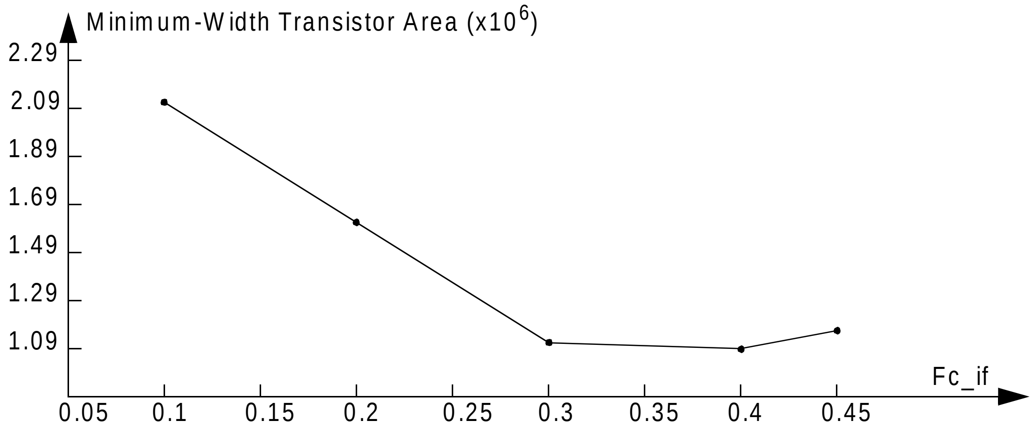


Figure 8.12: Iteration 3: Routing Area vs. Fc_if for Fc_pf = 0.2

L_f

Figure 8.13 is a plot of the average area required to implement the benchmark circuits versus the track length, L_f. Here, the track length is measured in terms of the logical track length, which is equal to the number of super-clusters that a track spans. It is assumed that 50% of the tracks in the MB-FPGA architecture are CMS; and L_c is always equal to L_f. The X-axis in the figure represents L_f, which ranges from 1 to 16. The Y-axis represents the area. There are 4 curves in the figure. Each curve represents an unique cluster size, N, including 2, 4, 8, and 10. For these cluster sizes, I is set to be 4, 10, 18, and 22, respectively. These values of I are shown to generate good area results for their corresponding cluster sizes [Betz99a]. As shown, the cluster size of 4 and the track length of 2 are the best architectural choice for the MB-FPGA architecture. Furthermore, the track length of 2 is always the most area efficient across all cluster sizes. Finally, using the same experiment, the best L_f value for the conventional FPGA architecture is determined also to be 2.

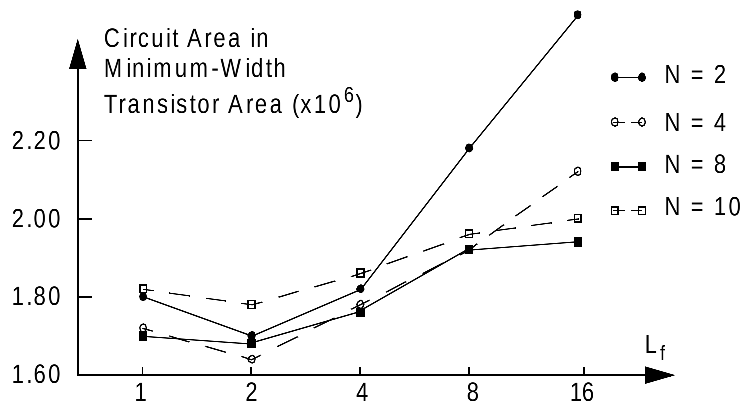


Figure 8.13: Area vs. Logical Track Length

8.5.3.2 Area and Performance Results

After obtaining the best possible architectural parameters for MB-FPGA, a set of experiments were performed by repeatedly invoking the CGR router over a range of values for W_c and W_f. For each invocation a fixed value of W_c is first chosen from the range of 0 to 80 in increments of 4. Then the router is instructed to search for the minimum value of W_f that is needed to successfully route each of the benchmark circuits. The resulting MB-FPGA architectures are then classified into eight groups based on the percentage of total tracks that are CMS tracks. The percentile ranges are (0%, 0%], (0%, 10%], (10%, 20%], (20%, 30%], (30%, 40%], (40%, 50%], (50%, 60%], and (60%, 70%]. Within each region, the minimum area obtainable by each circuit is first recorded. These minimum area values are then averaged across the fifteen benchmark circuits. The area again is measured in terms of the number of equivalent minimum-width transistor area as described in [Betz99a]. The arithmetic average of the area values is then plotted against each percentile range.

Figure 8.14 is a graph of the total area versus the percentage of routing tracks that are CMS tracks in the MB-FPGA routing architecture. The figure shows that when there are only a small percentage of CMS routing tracks, the implementation area of circuits on MB-FPGA

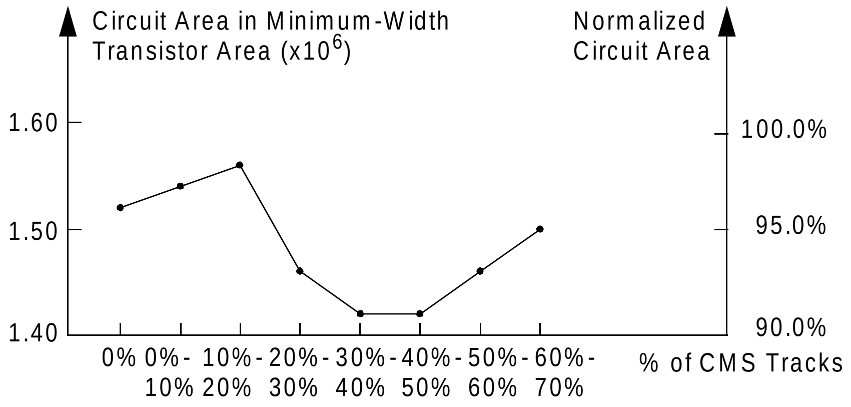


Figure 8.14: Area vs. Percentage of CMS Tracks

actually increases with the increased number of CMS routing tracks. There are two main causes of this initial increase in area. First when there are few CMS routing tracks, not all logic block input pins can be connected to all logic block output pins through CMS routing. This limitation dramatically reduces the usefulness of the CMS routing resources, hence resulting in increased area. A secondary cause is that as the CMS routing tracks are added to the routing fabric, routing resources are differentiated into two types. This differentiation reduces the routing flexibility and also accounts for the rise in area.

As the number of CMS routing tracks is increased to the 20% range, enough logic block pins can be connected to each other through the CMS routing tracks; and the benefit of CMS routing tracks starts to outweigh the decreased flexibility in routing. As a result, the total area required decreases until it reaches the minimum when CMS routing tracks account for between 40% to 50% of the total number of routing tracks. When the number of CMS tracks is further increased, the number of CMS routing tracks provided by the architecture starts to exceed the number of CMS routing tracks required by the circuits. The router then is forced to use CMS routing tracks to implement fine-grain routing. This reduces the efficiency of the MB-FPGA architecture past the 50% point.

Overall, the best area is achieved when CMS routing tracks account for 40% to 50% of the total number of routing tracks, where the benchmark circuits use 6% less area comparing to architectures with no CMS routing tracks. It is interesting to note that even though 90% of LUTs in the benchmark circuits belong to four-bit wide datapath components, only 40% to 50% of CMS routing tracks are needed. It is because many datapath components are not only connected by buses but also by a substantial amount of non-bus control signals, indicating that even highly regular circuits need many fine-grain routing tracks.

The right hand axis of Figure 8.14 also shows the area data normalized against the area of the best conventional FPGA architecture. All MB-FPGA architectures performed better than the best conventional architecture, where the 100% point represents the area of the conventional architecture when implementing the same circuits. Even with no CMS routing tracks, the MB-FPGA architecture is 3.6% smaller due to the more efficient datapath-oriented placement and routing. Overall the best MB-FPGA architecture is 9.6% smaller than the best standard architecture.

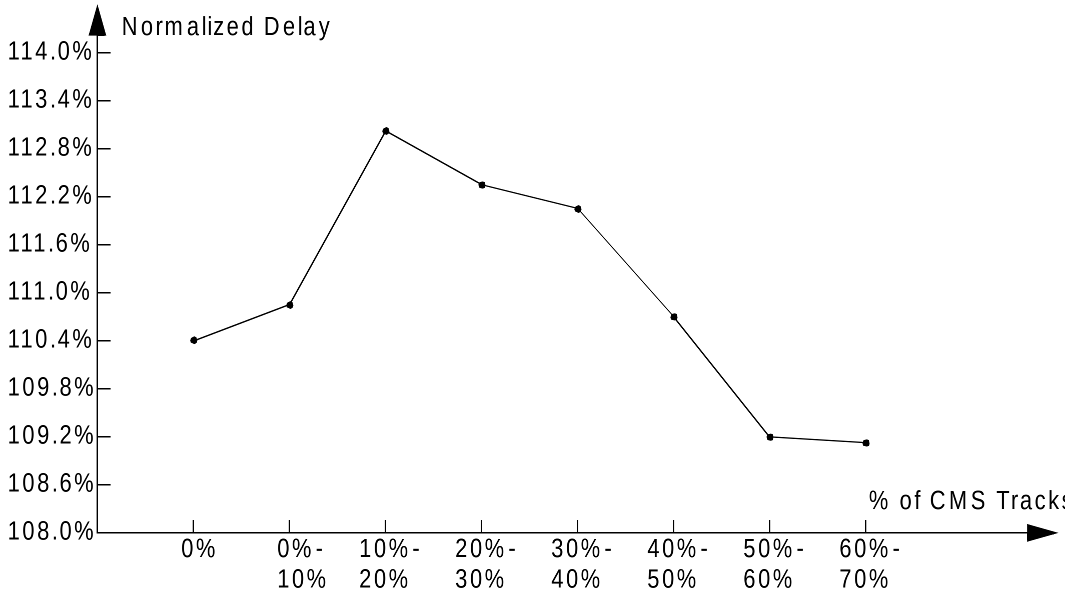


Figure 8.15: Normalized Delay vs. Percentage of CMS Tracks

Finally, Figure 8.15 plots the geometric average of circuit delay against the proportion of CMS routing tracks. The delay is normalized against the delay of the same circuits implemented on the conventional FPGA architecture. Note that, here, the delay of the carry chain is assumed to be the same as the delay of the local routing network. As shown the MB-FPGA implementation is around 9% to 13% slower than the conventional FPGA implementation. However, if the carry chains are much faster than the local routing network, this speed penalty might be significantly reduced if not completely eliminated.

8.6 Summary and Conclusions

This chapter has explored the relationship between the capacity of CMS routing resources and the area efficiency of MB-FPGA. The principle conclusions are that the granularity value of 4 has the best area result. Furthermore, to achieve the best area results, 40% to 50% of the total number of routing tracks should be CMS routing tracks despite the fact that, in the benchmark circuits, over 90% of LUTs are in regular datapath components. Finally, for cluster size of four, the best MB-FPGA architecture is 9.6% smaller than the best conventional architecture. The best architecture, however, has a potential speed penalty of 9.2%. Note that the area saving is much less than the 50% savings predicated by the DP-FPGA study [Cher96].

The results suggest that in order for the configuration memory sharing technique to be an effective methodology in achieving significant area savings, the configuration memory area in the target FPGA architectures must be significantly greater than the area of the switch that they control. The case of using configuration memory sharing architectures for applications with significantly more regularity and wider datapath than the benchmarks used in this study is more compelling but remains to be proven in a future study.

9 Conclusions

9.1 Thesis Summary

The main focus of this thesis has been the study of datapath-oriented FPGA architectures with regard to the effect of multi-bit logic and CMS routing resources on their area efficiency. The study's purpose has been to determine the most appropriate amount of CMS routing resources that can be used in FPGAs in order to minimize the implementation area of real datapath circuits under real modern CAD flows.

To this end, the first major in-depth study is conducted on the amount of datapath regularity that can be actually translated into area savings through configuration memory sharing. The study found that when detailed implementation issues are taken into account, the actual achievable area savings can be significant less than the previous estimations. The configuration memory sharing FPGA architecture, the MB-FPGA architecture, used in this study is only about 10% more area efficient than a comparable conventional and widely studied FPGA architecture in implementing datapath circuits. Furthermore, this increase in area efficiency has a potential speed penalty of around 10%.

In particular, the studies conducted in this thesis found that when transistors are properly sized, the SRAM size relative to the tri-state buffer size in a buffered switch is not significant enough to generate significant area savings as predicated by previous studies, which do not have detailed enough models to take the transistor sizing effect into account. Furthermore, the study on datapath regularity found that net regularity does not necessarily correspond to logic regularity. For example, for the benchmarks used in this study, 90% of the logic belongs to 4-bit wide datapath components; only around 50% of the nets, however, belongs to 4-bit wide buses.

The results suggest that in order for the configuration memory sharing technique to be an effective methodology in achieving significant area savings the configuration memory area in the target FPGA architectures must be significantly greater than the area of the switch that they control. Note that for both the conventional FPGA and the MB-FPGA architecture, SRAM composed of minimum width transistors are found to be sufficient. This SRAM size as compared to the size of a reasonably sized tri-state buffer is not significant enough to generate good area savings.

The case of using configuration memory sharing architectures for applications with significantly more regularity and wider datapath than the benchmarks used in this study is more compelling but remains to be proven in a future study. Finally, the study found that the configuration memory-sharing scheme still can be used to significantly reduce the amount of configuration memory used to control an FPGA and remains to be a promising technology in applications where such an effect is desired.

The research has been carried out using an experimental approach. For the experimental study, a new FPGA architecture called the Multi-Bit FPGA (MB-FPGA) has been developed. The architecture contains both multi-bit logic and CMS routing resources. It is also highly parameterized and closely resembles the cluster-based architectures that have been widely used in many academic studies and state-of-the-art commercial FPGAs. This close resemblance along with high parameterization enables a direct comparison between the area efficiency of the MB-FPGA architecture and the area efficiency of a conventional FPGA architecture that was widely used in many previous studies.

During the study, new types of synthesis, packing, and routing algorithms, specially designed to preserve and utilize datapath regularity, have been used to measure the effect of CMS routing resources on the area efficiency of the MB-FPGA architecture. The results of the work in this thesis provide new insights into the design of FPGA architectures that utilize multi-bit logic and CMS routing resources. The thesis also provides an in-depth study on the design and the construction of datapath-oriented CAD tools and proposes a method of systematically characterizing and quantifying datapath regularity.

9.2 Thesis Contributions

This dissertation makes the following detailed contributions:

In Chapter 3, the MB-FPGA architecture was described. It is the first datapath-oriented FPGA architecture that contains a completely specified global and detailed routing architecture. The MB-FPGA organizes its logic resources into super-clusters, which can be used to capture the intra-bit-slice connections into inter-super-cluster buses. These buses are then routed through the global routing resources, which employs configuration memory sharing in order to reduce the area overhead of transporting buses across the architecture. The global routing resources also contain a mixture of fine-grain and CMS routing tracks in each routing channel, resulting in a homogenous architecture that is capable of implementing both large datapath circuits and small non-datapath circuits.

In Chapter 4, the Enhanced Module Compaction (EMC) datapath-oriented synthesis algorithm was described. It is the first published algorithm that preserves user-specified regularity information in datapath circuits while achieving an area-efficiency that is comparable to the conventional flat synthesis techniques. EMC employs two word-level optimization techniques and several bit-slice I/O optimizations to enhance the area efficiency of the original module compaction algorithm. Furthermore, unlike the original module compaction algorithm, the EMC algorithm does not rely on the results of any placement tools. EMC has been used to obtain excellent synthesis results, especially in terms of area-efficiency and datapath regularity preservation, for realistic datapath circuits implemented on FPGAs.

The Coarse-grain Node Graph (CNG) packing algorithm for MB-FPGA was developed in Chapter 5. It is the first published FPGA packing algorithm that preserves the regularity of datapath circuits throughout the packing process. CNG employs specially designed cost functions that take into account datapath regularity as well as performance and area efficiency during packing. CNG has been used to obtain excellent packing results for realistic datapath circuits implemented on the MB-FPGA architecture.

The Coarse-Grain Resource (CGR) routing algorithm for MB-FPGA was described in Chapter 6. It is the first published FPGA routing algorithm that accommodates CMS routing resources, which share configuration memory, as well as conventional fine-grain routing resources. The accommodation is achieved by a set of specially designed cost functions that balance the demands on each type of routing resources with their availability. The cost functions also allow the optimization of FPGA routing area and routing delay. CGR has been used to obtain excellent routing results for realistic datapath circuits implemented on the MB-FPGA architecture.

Chapter 7 determines appropriate values for several MB-FPGA architectural parameters through an analytical approach. It formally characterizes and measures datapath regularity in terms of two parameters — the logic regularity and the net regularity. Using the regularity information, the granularity values of 2 and 4 are determined to be good values for the efficient implementation of datapath circuits on MB-FPGA. It also shows that, for good area efficiency, the number of fine-grain routing tracks should be at least 20% to 27% of the total number of routing tracks. Finally, it shows that the M-bit wide shift blocks employed in the DP-FPGA architecture are likely to be inefficient in area.

Chapter 8 gives the results of an experimental study of the effects of CMS routing resources on the area efficiency of the MB-FPGA architecture. This study is the first of its kind for FPGAs that contain CMS routing resources. The principle conclusions reached are that for the best area efficiency, 40% to 50% of the routing tracks in the MB-FPGA architecture should be CMS routing tracks. This is true even for highly regular datapath circuits that have over 90% of LUTs in regular datapath components. Also it shows that the granularity value of 4 gives the best area results for the MB-FPGA architecture when implementing 32-bit wide datapath circuits. Finally, the results show that the best MB-FPGA architecture is 9.6% smaller than the best conventional FPGA architecture. This area saving is much smaller than the savings predicated by the DP-FPGA study; and the best architecture has a worst case speed penalty of 9.2%.

9.3 Suggestions for Future Research

The MB-FPGA architecture can be used as a research vehicle to investigate into the various questions regarding multi-bit logic and CMS routing resources, including:

1. Can configuration memory sharing be effectively used in super-clusters to further increase the area efficiency of the MB-FPGA architecture? Currently there is no sharing of configuration memory inside the super-clusters. However, since, as described in Chapter 3, in order to capture intra-bit-slice connections into inter-super-cluster buses, identical LUT configurations are often kept inside a single super-clusters and each of these identical LUTs are implemented in unique clusters, configuration memory sharing across clusters might be able to reduce the implementation area of the super-clusters. The sharing need not to be implemented for all logic and local routing resources; instead, it might be most beneficial to only share configuration memory for only a percentage of these resources. Research needs to be done to find the best proportion of configuration memory sharing resources in a super-cluster. The CNG packing algorithm also has to be modified accordingly to accommodate this architectural change.

2. Is there any benefit to adding programmable connections between the CMS routing tracks and the fine-grain routing tracks inside the MB-FPGA switch blocks? Currently, it is assumed that inside each MB-FPGA switch block, there is no connectivity between a CMS track and a fine-grain track. Adding such connectivity, however, might increase routing flexibility by allowing buses to be dispersed into individual signals and vice versa — individual signals to be grouped into buses. The benefit of such added functionality is unclear and warrants further research.
3. Will there be any benefit in having different track lengths for the CMS routing tracks and the fine-grain routing tracks? Buses and individual signals might have different routing requirements. Having different track lengths for these two types of tracks might improve the overall performance of the MB-FPGA architecture and warrants further research.

Several improvements can also be made to the MB-FPGA CAD tools including:

1. Fully automate the two word-level optimizations described in Chapter 4.
2. For the datapath-oriented routing algorithm, future research should improve upon the timing performance of the router. The goal should be to achieve consistently good routing delays regardless of the number of CMS routing tracks in each routing channel.

Finally, according to [Tuan03], configuration memory consumes significant amount (around 38%) of the total FPGA leakage power. Configuration memory sharing allows the programmable resources to share configuration memory; therefore, reduces the total amount of configuration memory required to implement datapath circuits. It is possible that datapath-oriented architectures like the MB-FPGA architecture will have much lower leakage power consumption than conventional FPGAs — making it more suitable for implementing mobile applications, which often are arithmetic and datapath intensive. This area warrants further research.

Appendix A: Net Regularity Distribution

This appendix lists the net regularity distribution for the granularity values of 2, 4, 8, 12, 16, 20, 24, 28, and 32. It supplements the discussion presented in Section 7.5.2. Each section lists the net regularity distribution for a specific granularity value. Each row of the table corresponds to a fixed bus width. Each column of the table corresponds to a fixed amount of shift. Each entry of the table lists the percentage of two-terminal connections that are in the buses with the corresponding width and shift. Entries that are less than 1% are shaded in light gray.

A.1 MB-FPGA Architectural Granularity = 2

Bus Width	Shift	
	0	**1**
1	16%	11%
2	59%	14%

Table A.1: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 2

A.2 MB-FPGA Architectural Granularity = 4

Bus Width	Shift			
	0	**1**	**2**	**3**
1	6.9%	5.5%	5.7%	6.5%
2	2.0%	3.4%	3.0%	2.3%
3	0.79%	1.2%	1.2%	0.80%
4	54%	2.8%	2.2%	1.5%

Table A.2: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 4

A.3 MB-FPGA Architectural Granularity = 8

Bus Width	Shift							
	0	1	2	3	4	5	6	7
1	3.0%	2.7%	2.9%	2.8%	3.1%	3.2%	3.1%	3.3%
2	1.5%	1.6%	1.3%	1.2%	1.2%	1.1%	1.4%	1.1%
3	0.48%	0.68%	0.48%	0.54%	0.50%	0.58%	0.39%	0.56%
4	2.3%	0.29%	0.35%	0.41%	1.5%	0.16%	0.20%	0.16%
5	0.083%	0.033%	0.050%	0.13%	0.33%	0.083%	0.050%	0.017%
6	0.54%	0.020%	0.20%	0.0%	0.080%	0.10%	0.34%	0.060%
7	0.23%	0.61%	0.023%	0.023%	0.070%	0.047%	0.070%	0.14%
8	47%	1.6%	0.59%	0.13%	2.1%	0.19%	0.80%	0.75%

Table A.3: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 8

A.4 MB-FPGA Architectural Granularity = 12

Bus Width	Shift											
	0	1	2	3	4	5	6	7	8	9	10	11
1	2.1%	1.7%	1.8%	2.0%	2.2%	2.4%	2.5%	2.5%	2.4%	2.4%	2.4%	2.4%
2	0.80%	1.2%	1.2%	0.96%	0.82%	0.79%	0.98%	0.77%	0.92%	0.79%	0.82%	0.65%
3	0.27%	0.48%	0.29%	0.27%	0.29%	0.28%	0.22%	0.13%	0.27%	0.30%	0.18%	0.21%
4	3.0%	0.066%	0.053%	0.066%	1.7%	0.092%	0.16%	0.16%	1.7%	0.013%	0.066%	0.12%
5	0.050%	0.017%	0.033%	0.033%	0.38%	0.017%	0.0%	0.066%	0.083%	0.017%	0.0%	0.017%
6	0.28%	0.0%	0.079%	0.0%	0.32%	0.020%	0.20%	0.0%	0.020%	0.040%	0.040%	0.020%
7	0.069%	0.21%	0.0%	0.0%	0.046%	0.12%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
8	14%	0.11%	0.0%	0.0%	0.69%	0.0%	0.0%	0.0%	1.1%	0.026%	0.053%	0.18%
9	0.62%	0.059%	0.030%	0.030%	0.089%	0.059%	0.0%	0.030%	0.030%	0.059%	0.030%	0.089%
10	0.43%	0.0%	0.0%	0.0%	0.0%	0.033%	0.066%	0.033%	0.033%	0.066%	0.36%	0.033%
11	0.33%	0.11%	0.036%	0.036%	0.036%	0.073%	0.0%	0.0%	0.036%	0.036%	0.22%	0.40%
12	27%	1.3%	0.24%	0.12%	0.75%	0.12%	0.36%	0.040%	0.75%	0.040%	0.48%	0.36%

Table A.4: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 12

A.5 MB-FPGA Architectural Granularity = 16

Bus Width	Shift							
	0	1	2	3	4	5	6	7
1	1.2%	1.1%	1.3%	1.3%	1.4%	1.4%	1.4%	1.4%
2	0.82%	0.87%	0.70%	0.67%	0.64%	0.69%	0.66%	0.73%
3	0.40%	0.41%	0.35%	0.41%	0.29%	0.32%	0.35%	0.39%
4	1.0%	0.19%	0.22%	0.20%	0.70%	0.19%	0.16%	0.079%
5	0.083%	0.12%	0.033%	0.050%	0.26%	0.033%	0.017%	0.050%
6	0.10%	0.020%	0.10%	0.0%	0.040%	0.060%	0.0%	0.020%
7	0.046%	0.0%	0.0%	0.0%	0.046%	0.046%	0.023%	0.0%
8	4.4%	0.0%	0.0%	0.053%	0.42%	0.0%	0.0%	0.0%
9	0.089%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.033%	0.0%	0.0%	0.0%	0.033%	0.0%	0.10%	0.0%
11	0.073%	0.15%	0.0%	0.0%	0.0%	0.036%	0.0%	0.0%
12	0.60%	0.0%	0.040%	0.0%	0.32%	0.0%	0.0%	0.0%
13	0.0%	0.0%	0.0%	0.043%	0.0%	0.0%	0.0%	0.0%
14	0.97%	0.046%	0.14%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.25%	0.89%	0.0%	0.0%	0.0%	0.050%	0.0%	0.0%
16	34%	0.79%	0.21%	0.11%	0.32%	0.11%	0.21%	0.11%

Table A.5: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 16 – Part 1 of 2

Bus Width	Shift							
	8	9	10	11	12	13	14	15
1	1.4%	1.4%	1.4%	1.4%	1.3%	1.3%	1.3%	1.4%
2	0.73%	0.58%	0.62%	0.67%	0.77%	0.75%	0.81%	0.72%
3	0.25%	0.34%	0.32%	0.36%	0.34%	0.43%	0.28%	0.29%
4	0.52%	0.079%	0.11%	0.12%	0.87%	0.15%	0.19%	0.21%
5	0.083%	0.033%	0.050%	0.066%	0.066%	0.050%	0.050%	0.050%
6	0.020%	0.020%	0.079%	0.020%	0.060%	0.060%	0.020%	0.0%
7	0.0%	0.046%	0.0%	0.0%	0.046%	0.0%	0.0%	0.0%
8	2.2%	0.026%	0.11%	0.026%	0.13%	0.0%	0.0%	0.053%
9	0.45%	0.030%	0.0%	0.0%	0.0%	0.0%	0.060%	0.0%
10	0.10%	0.033%	0.033%	0.0%	0.0%	0.0%	0.0%	0.0%
11	0.11%	0.036%	0.073%	0.0%	0.073%	0.0%	0.0%	0.0%
12	0.040%	0.0%	0.0%	0.0%	0.20%	0.0%	0.040%	0.040%
13	0.0%	0.043%	0.0%	0.0%	0.043%	0.086%	0.086%	0.086%
14	0.0%	0.0%	0.0%	0.0%	0.0%	0.046%	0.51%	0.046%
15	0.050%	0.10%	0.0%	0.0%	0.050%	0.0%	0.10%	0.40%
16	2.8%	0.0%	0.32%	0.0%	0.64%	0.0%	0.26%	0.42%

Table A.6: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 16 – Part 2 of 2

A.6 MB-FPGA Architectural Granularity = 20

Bus Width	Shift									
	0	1	2	3	4	5	6	7	8	9
1	1.1%	1.1%	1.2%	1.1%	1.2%	1.2%	1.3%	1.2%	1.2%	1.3%
2	0.54%	0.63%	0.58%	0.54%	0.50%	0.69%	0.65%	0.72%	0.70%	0.77%
3	0.36%	0.27%	0.29%	0.30%	0.29%	0.25%	0.26%	0.17%	0.24%	0.20%
4	1.6%	0.13%	0.17%	0.13%	1.0%	0.067%	0.11%	0.094%	0.52%	0.054%
5	0.067%	0.033%	0.017%	0.033%	0.28%	0.050%	0.0%	0.084%	0.10%	0.0%
6	0.10%	0.080%	0.040%	0.020%	0.12%	0.0%	0.020%	0.0%	0.080%	0.0%
7	0.094%	0.023%	0.0%	0.0%	0.070%	0.070%	0.047%	0.023%	0.023%	0.023%
8	3.0%	0.0%	0.080%	0.080%	0.62%	0.0%	0.027%	0.0%	2.5%	0.054%
9	0.060%	0.0%	0.0%	0.030%	0.060%	0.0%	0.0%	0.030%	0.33%	0.060%
10	0.23%	0.0%	0.067%	0.0%	0.0%	0.0%	0.033%	0.0%	0.13%	0.0%
11	0.074%	0.51%	0.0%	0.0%	0.0%	0.037%	0.0%	0.0%	0.037%	0.037%
12	8.6%	0.12%	0.040%	0.040%	0.24%	0.0%	0.0%	0.0%	0.24%	0.040%
13	0.74%	0.0%	0.0%	0.0%	0.0%	0.043%	0.0%	0.087%	0.0%	0.0%
14	0.14%	0.0%	0.0%	0.0%	0.0%	0.0%	0.047%	0.0%	0.0%	0.0%
15	0.10%	0.15%	0.0%	0.0%	0.0%	0.0%	0.0%	0.050%	0.0%	0.0%
16	1.9%	0.054%	0.0%	0.0%	0.054%	0.0%	0.0%	0.0%	0.11%	0.0%
17	0.0%	0.0%	0.0%	0.057%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
18	0.24%	0.0%	0.12%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
19	0.19%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
20	19%	0.54%	0.13%	0.13%	0.40%	0.067%	0.13%	0.0%	0.20%	0.0%

Table A.7: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 20 – Part 1 of 2

Bus Width	Shift									
	10	11	12	13	14	15	16	17	18	19
1	1.3%	1.2%	1.2%	1.2%	1.2%	1.2%	1.2%	1.1%	1.1%	1.2%
2	0.66%	0.68%	0.61%	0.56%	0.60%	0.64%	0.53%	0.58%	0.58%	0.47%
3	0.23%	0.16%	0.25%	0.29%	0.22%	0.26%	0.28%	0.21%	0.22%	0.29%
4	0.013%	0.094%	0.72%	0.040%	0.067%	0.027%	1.0%	0.16%	0.094%	0.12%
5	0.0%	0.067%	0.10%	0.033%	0.0%	0.084%	0.050%	0.0%	0.033%	0.017%
6	0.060%	0.0%	0.10%	0.0%	0.060%	0.0%	0.020%	0.020%	0.020%	0.040%
7	0.0%	0.0%	0.023%	0.070%	0.0%	0.023%	0.0%	0.0%	0.047%	0.047%
8	0.0%	0.080%	2.6%	0.054%	0.054%	0.0%	0.35%	0.027%	0.0%	0.0%
9	0.0%	0.0%	0.24%	0.0%	0.0%	0.030%	0.090%	0.0%	0.030%	0.030%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.033%	0.033%
11	0.0%	0.0%	0.037%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
12	0.0%	0.0%	0.12%	0.0%	0.12%	0.080%	0.040%	0.0%	0.080%	0.12%
13	0.0%	0.0%	0.087%	0.043%	0.0%	0.0%	0.0%	0.043%	0.043%	0.0%
14	0.0%	0.0%	0.047%	0.047%	0.28%	0.0%	0.047%	0.0%	0.0%	0.047%
15	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.10%
16	0.0%	0.0%	0.21%	0.054%	0.0%	0.0%	0.21%	0.0%	0.054%	0.0%
17	0.0%	0.0%	0.0%	0.057%	0.0%	0.0%	0.057%	0.057%	0.0%	0.0%
18	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.48%	0.0%
19	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.064%	0.064%	0.19%	0.44%
20	0.067%	0.0%	0.60%	0.0%	0.0%	0.0%	0.067%	0.0%	0.0%	0.13%

Table A.8: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 20 – Part 2 of 2

A.7 MB-FPGA Architectural Granularity = 24

Bus Width	Shift											
	0	1	2	3	4	5	6	7	8	9	10	11
1	0.78%	0.81%	0.89%	0.90%	0.89%	0.94%	1.0%	0.99%	0.94%	0.98%	0.91%	0.96%
2	0.61%	0.60%	0.55%	0.52%	0.53%	0.45%	0.50%	0.46%	0.49%	0.35%	0.48%	0.46%
3	0.30%	0.38%	0.29%	0.23%	0.21%	0.25%	0.19%	0.28%	0.25%	0.33%	0.24%	0.24%
4	0.49%	0.082%	0.15%	0.11%	0.44%	0.15%	0.22%	0.14%	0.52%	0.15%	0.14%	0.10%
5	0.10%	0.051%	0.068%	0.085%	0.12%	0.085%	0.068%	0.0%	0.017%	0.034%	0.068%	0.017%
6	0.12%	0.0%	0.041%	0.020%	0.020%	0.020%	0.0%	0.020%	0.16%	0.020%	0.061%	0.0%
7	0.071%	0.071%	0.0%	0.024%	0.024%	0.0%	0.0%	0.024%	0.024%	0.19%	0.0%	0.0%
8	5.6%	0.11%	0.0%	0.027%	0.16%	0.027%	0.0%	0.082%	3.3%	0.11%	0.0%	0.0%
9	0.55%	0.031%	0.0%	0.0%	0.061%	0.0%	0.0%	0.0%	0.37%	0.061%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.10%	0.034%	0.034%	0.0%
11	0.11%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
12	0.12%	0.0%	0.0%	0.0%	0.041%	0.0%	0.0%	0.0%	0.082%	0.0%	0.0%	0.0%
13	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
14	0.048%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.048%	0.0%
15	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
16	2.0%	0.16%	0.0%	0.0%	0.0%	0.0%	0.0%	0.11%	1.5%	0.0%	0.0%	0.0%
17	0.058%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.058%	0.058%	0.0%	0.0%
18	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.12%	0.0%	0.0%	0.0%	0.0%	0.0%
19	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.065%
20	0.48%	0.0%	0.0%	0.0%	0.34%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
21	0.0%	0.0%	0.0%	0.14%	0.0%	0.0%	0.0%	0.071%	0.0%	0.0%	0.0%	0.0%
22	0.37%	0.0%	0.15%	0.0%	0.0%	0.075%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
23	0.23%	0.55%	0.078%	0.0%	0.0%	0.0%	0.0%	0.0%	0.078%	0.0%	0.0%	0.0%
24	23%	0.24%	0.0%	0.0%	0.082%	0.0%	0.082%	0.0%	0.082%	0.0%	0.0%	0.0%

Table A.9: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 24 – Part 1 of 2

Bus Width	Shift											
	12	13	14	15	16	17	18	19	20	21	22	23
1	0.89%	1.0%	0.92%	0.99%	0.93%	0.95%	0.92%	0.94%	0.95%	0.94%	0.93%	0.95%
2	0.55%	0.46%	0.50%	0.48%	0.55%	0.55%	0.57%	0.55%	0.48%	0.54%	0.54%	0.51%
3	0.16%	0.22%	0.25%	0.23%	0.23%	0.18%	0.19%	0.22%	0.33%	0.24%	0.22%	0.30%
4	0.52%	0.15%	0.15%	0.041%	0.34%	0.068%	0.12%	0.11%	0.56%	0.18%	0.19%	0.15%
5	0.085%	0.0%	0.068%	0.034%	0.017%	0.034%	0.017%	0.085%	0.034%	0.017%	0.017%	0.017%
6	0.0%	0.0%	0.041%	0.061%	0.10%	0.020%	0.10%	0.0%	0.041%	0.041%	0.020%	0.0%
7	0.048%	0.024%	0.024%	0.0%	0.0%	0.048%	0.024%	0.0%	0.024%	0.0%	0.0%	0.0%
8	0.22%	0.027%	0.054%	0.16%	3.7%	0.082%	0.054%	0.0%	0.22%	0.0%	0.027%	0.027%
9	0.0%	0.0%	0.0%	0.15%	0.21%	0.0%	0.0%	0.031%	0.0%	0.0%	0.031%	0.0%
10	0.034%	0.0%	0.0%	0.0%	0.10%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
11	0.0%	0.0%	0.0%	0.0%	0.037%	0.0%	0.0%	0.0%	0.0%	0.0%	0.075%	0.0%
12	0.41%	0.0%	0.0%	0.0%	0.082%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
13	0.13%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.044%	0.0%	0.0%
14	0.0%	0.048%	0.0%	0.0%	0.048%	0.0%	0.048%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.10%
16	0.16%	0.0%	0.0%	0.054%	0.82%	0.0%	0.16%	0.0%	0.0%	0.0%	0.0%	0.0%
17	0.0%	0.0%	0.0%	0.0%	0.17%	0.17%	0.12%	0.0%	0.0%	0.058%	0.0%	0.0%
18	0.0%	0.0%	0.0%	0.0%	0.0%	0.061%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
19	0.0%	0.0%	0.0%	0.0%	0.065%	0.0%	0.0%	0.0%	0.0%	0.065%	0.0%	0.0%
20	0.0%	0.0%	0.0%	0.0%	0.068%	0.0%	0.0%	0.0%	0.20%	0.0%	0.0%	0.0%
21	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.90%	0.0%
23	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.078%	0.63%
24	0.0%	0.0%	0.0%	0.0%	0.49%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.082%

Table A.10: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 24 – Part 2 of 2

A.8 MB-FPGA Architectural Granularity = 28

Bus Width	Shift											
	0	1	2	3	4	5	6	7	8	9	10	11
1	0.69%	0.69%	0.75%	0.72%	0.75%	0.81%	0.80%	0.83%	0.82%	0.84%	0.84%	0.85%
2	0.44%	0.36%	0.38%	0.43%	0.39%	0.48%	0.40%	0.39%	0.33%	0.36%	0.42%	0.46%
3	0.25%	0.36%	0.26%	0.30%	0.19%	0.14%	0.24%	0.24%	0.21%	0.16%	0.15%	0.20%
4	1.8%	0.19%	0.19%	0.16%	0.82%	0.13%	0.12%	0.16%	0.92%	0.19%	0.073%	0.15%
5	0.26%	0.073%	0.073%	0.13%	0.16%	0.16%	0.055%	0.091%	0.11%	0.018%	0.055%	0.018%
6	0.15%	0.13%	0.088%	0.0%	0.066%	0.022%	0.022%	0.0%	0.066%	0.022%	0.0%	0.022%
7	0.0%	0.0%	0.0%	0.0%	0.026%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
8	2.4%	0.0%	0.0%	0.029%	0.15%	0.0%	0.0%	0.0%	2.0%	0.029%	0.0%	0.0%
9	0.10%	0.0%	0.033%	0.0%	0.033%	0.0%	0.0%	0.0%	0.033%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.036%	0.0%	0.036%	0.0%	0.0%	0.0%
11	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
12	0.13%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.044%	0.0%	0.0%	0.0%
13	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
14	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.11%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
16	1.6%	0.18%	0.0%	0.058%	0.0%	0.0%	0.0%	0.12%	0.23%	0.0%	0.0%	0.0%
17	0.062%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
18	0.066%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.066%	0.0%
19	0.069%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
20	0.51%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.22%	0.0%	0.0%	0.0%
21	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%	0.0%	0.08%	0.24%	0.0%	0.0%	0.0%	0.0%	0.0%
23	0.25%	0.0%	0.084%	0.084%	0.17%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
24	0.35%	0.0%	0.0%	0.0%	0.088%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
25	0.0%	0.0%	0.0%	0.091%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
26	0.38%	0.0%	0.095%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
27	0.20%	0.59%	0.10%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
28	25%	0.41%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table A.11: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 28 – Part 1 of 3

Bus Width	Shift											
	12	13	14	15	16	17	18	19	20	21	22	23
1	0.81%	0.90%	0.89%	0.86%	0.82%	0.85%	0.80%	0.78%	0.69%	0.78%	0.83%	0.84%
2	0.41%	0.45%	0.41%	0.46%	0.50%	0.40%	0.46%	0.31%	0.36%	0.35%	0.32%	0.35%
3	0.18%	0.20%	0.25%	0.25%	0.16%	0.19%	0.16%	0.21%	0.26%	0.32%	0.33%	0.22%
4	0.73%	0.058%	0.13%	0.088%	0.80%	0.058%	0.088%	0.23%	0.72%	0.19%	0.18%	0.36%
5	0.055%	0.091%	0.055%	0.11%	0.055%	0.055%	0.091%	0.073%	0.15%	0.091%	0.018%	0.091%
6	0.022%	0.0%	0.0%	0.0%	0.022%	0.066%	0.0%	0.022%	0.022%	0.0%	0.022%	0.022%
7	0.0%	0.0%	0.0%	0.0%	0.051%	0.051%	0.026%	0.0%	0.051%	0.026%	0.0%	0.0%
8	0.85%	0.0%	0.0%	0.0%	1.1%	0.0%	0.0%	0.0%	1.6%	0.029%	0.029%	0.0%
9	0.16%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.20%	0.0%	0.0%	0.0%
10	0.036%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.073%	0.0%	0.0%	0.0%
11	0.080%	0.0%	0.0%	0.040%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
12	0.13%	0.0%	0.0%	0.0%	0.18%	0.0%	0.0%	0.0%	0.044%	0.0%	0.0%	0.0%
13	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
14	0.10%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
16	0.41%	0.0%	0.0%	0.0%	0.12%	0.0%	0.0%	0.0%	0.12%	0.058%	0.058%	0.0%
17	0.0%	0.0%	0.0%	0.0%	0.062%	0.062%	0.0%	0.0%	0.062%	0.062%	0.062%	0.0%
18	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.066%	0.0%	0.0%
19	0.069%	0.0%	0.0%	0.0%	0.069%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
20	0.15%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.29%	0.0%	0.15%	0.0%
21	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.15%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
23	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
24	0.088%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
25	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
26	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
27	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
28	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table A.12: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 28 – Part 2 of 3

Bus Width	Shift			
	24	25	26	27
1	0.74%	0.87%	0.80%	0.72%
2	0.44%	0.32%	0.45%	0.42%
3	0.31%	0.30%	0.19%	0.22%
4	1.1%	0.26%	0.23%	0.19%
5	0.13%	0.018%	0.091%	0.073%
6	0.044%	0.0%	0.0%	0.022%
7	0.0%	0.026%	0.0%	0.026%
8	0.15%	0.029%	0.0%	0.0%
9	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%
11	0.0%	0.0%	0.0%	0.0%
12	0.044%	0.0%	0.0%	0.0%
13	0.0%	0.047%	0.0%	0.0%
14	0.0%	0.0%	0.0%	0.0%
15	0.0%	0.0%	0.0%	0.055%
16	0.0%	0.058%	0.0%	0.0%
17	0.0%	0.0%	0.0%	0.0%
18	0.0%	0.0%	0.0%	0.0%
19	0.0%	0.069%	0.0%	0.069%
20	0.0%	0.0%	0.0%	0.0%
21	0.0%	0.0%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%
23	0.25%	0.0%	0.17%	0.0%
24	0.088%	0.0%	0.0%	0.0%
25	0.0%	0.0%	0.18%	0.0%
26	0.0%	0.0%	0.76%	0.19%
27	0.0%	0.0%	0.0%	0.39%
28	0.0%	0.0%	0.0%	0.0%

Table A.13: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 28 – Part 3 of 3

A.9 MB-FPGA Architectural Granularity = 32

Bus Width	Shift											
	0	1	2	3	4	5	6	7	8	9	10	11
1	0.53%	0.55%	0.57%	0.63%	0.59%	0.60%	0.65%	0.64%	0.61%	0.69%	0.62%	0.66%
2	0.42%	0.45%	0.39%	0.35%	0.36%	0.42%	0.30%	0.31%	0.34%	0.31%	0.39%	0.40%
3	0.21%	0.17%	0.21%	0.18%	0.22%	0.18%	0.23%	0.18%	0.17%	0.21%	0.21%	0.16%
4	0.47%	0.15%	0.19%	0.20%	0.44%	0.20%	0.20%	0.24%	0.31%	0.13%	0.16%	0.11%
5	0.084%	0.10%	0.034%	0.067%	0.084%	0.017%	0.017%	0.051%	0.034%	0.034%	0.017%	0.067%
6	0.040%	0.040%	0.0%	0.020%	0.040%	0.061%	0.020%	0.0%	0.061%	0.040%	0.020%	0.0%
7	0.047%	0.024%	0.0%	0.0%	0.047%	0.0%	0.047%	0.047%	0.024%	0.0%	0.0%	0.0%
8	1.6%	0.0%	0.054%	0.054%	0.13%	0.027%	0.027%	0.027%	0.92%	0.027%	0.0%	0.027%
9	0.12%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.21%	0.0%	0.0%	0.0%
10	0.03%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.10%	0.0%	0.0%	0.0%
11	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.037%	0.0%	0.0%	0.0%
12	0.20%	0.0%	0.0%	0.0%	0.040%	0.0%	0.0%	0.0%	0.040%	0.0%	0.040%	0.0%
13	0.044%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
14	0.047%	0.0%	0.047%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.10%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
16	0.97%	0.054%	0.0%	0.0%	0.0%	0.0%	0.0%	0.054%	0.65%	0.0%	0.11%	0.0%
17	0.11%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
18	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
19	0.13%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
20	0.40%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
21	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%	0.0%	0.074%	0.0%	0.0%	0.0%	0.0%	0.074%	0.0%
23	0.16%	0.0%	0.0%	0.0%	0.078%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
24	0.16%	0.0%	0.081%	0.081%	0.0%	0.0%	0.0%	0.0%	0.40%	0.0%	0.0%	0.0%
25	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.168%	0.0%	0.0%	0.0%
26	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.18%	0.0%	0.0%	0.0%	0.0%	0.0%
27	0.091%	0.18%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
28	1.4%	0.094%	0.0%	0.0%	0.47%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
29	0.0%	0.0%	0.0%	0.098%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
30	1.5%	0.10%	0.30%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
31	0.31%	0.42%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
32	29%	0.11%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.11%	0.0%	0.0%	0.0%

Table A.14: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 32 – Part 1 of 3

Bus Width	Shift											
	12	13	14	15	16	17	18	19	20	21	22	23
1	0.69%	0.70%	0.68%	0.70%	0.68%	0.71%	0.64%	0.64%	0.63%	0.70%	0.69%	0.63%
2	0.38%	0.36%	0.36%	0.37%	0.34%	0.41%	0.43%	0.37%	0.36%	0.32%	0.33%	0.41%
3	0.18%	0.11%	0.17%	0.19%	0.17%	0.12%	0.16%	0.15%	0.23%	0.14%	0.18%	0.14%
4	0.36%	0.13%	0.12%	0.081%	0.26%	0.081%	0.094%	0.094%	0.30%	0.12%	0.081%	0.081%
5	0.067%	0.017%	0.0%	0.034%	0.017%	0.034%	0.017%	0.034%	0.10%	0.034%	0.051%	0.067%
6	0.0%	0.020%	0.0%	0.0%	0.040%	0.0%	0.0%	0.020%	0.040%	0.040%	0.0%	0.061%
7	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
8	0.027%	0.0%	0.0%	0.0%	1.5%	0.0%	0.0%	0.0%	0.27%	0.0%	0.0%	0.0%
9	0.0%	0.0%	0.0%	0.0%	0.061%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
11	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.037%	0.0%	0.0%
12	0.040%	0.0%	0.0%	0.0%	0.040%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
13	0.0%	0.0%	0.0%	0.0%	0.044%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
14	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.0%	0.0%	0.0%	0.051%	0.051%	0.051%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
16	0.16%	0.0%	0.0%	0.0%	1.08%	0.054%	0.054%	0.054%	0.0%	0.0%	0.0%	0.054%
17	0.0%	0.0%	0.0%	0.057%	0.11%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
18	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
19	0.0%	0.0%	0.0%	0.0%	0.13%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
20	0.27%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
21	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.071%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
23	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
24	0.0%	0.0%	0.0%	0.0%	0.40%	0.0%	0.0%	0.0%	0.081%	0.0%	0.0%	0.0%
25	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
26	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
27	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.091%	0.0%	0.0%	0.0%
28	0.28%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.094%	0.0%	0.0%	0.0%
29	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
30	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
31	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
32	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.11%	0.0%	0.0%	0.0%

Table A.15: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 32 – Part 2 of 3

Bus Width	Shift							
	24	25	26	27	28	29	30	31
1	0.62%	0.66%	0.67%	0.65%	0.64%	0.61%	0.58%	0.55%
2	0.34%	0.32%	0.31%	0.35%	0.32%	0.36%	0.40%	0.43%
3	0.20%	0.22%	0.18%	0.20%	0.29%	0.26%	0.19%	0.17%
4	0.40%	0.054%	0.12%	0.067%	0.34%	0.094%	0.15%	0.15%
5	0.084%	0.017%	0.051%	0.034%	0.067%	0.10%	0.051%	0.067%
6	0.020%	0.040%	0.0%	0.0%	0.020%	0.0%	0.061%	0.0%
7	0.0%	0.0%	0.0%	0.0%	0.024%	0.024%	0.0%	0.0%
8	1.6%	0.0%	0.0%	0.0%	0.13%	0.0%	0.0%	0.0%
9	0.12%	0.030%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.034%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
11	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
12	0.0%	0.0%	0.0%	0.0%	0.040%	0.0%	0.0%	0.0%
13	0.0%	0.0%	0.0%	0.0%	0.0%	0.044%	0.0%	0.0%
14	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
15	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.051%
16	0.27%	0.11%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
17	0.0%	0.057%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
18	0.061%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
19	0.0%	0.0%	0.0%	0.0%	0.0%	0.13%	0.0%	0.0%
20	0.067%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
21	0.0%	0.071%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
22	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
23	0.0%	0.0%	0.0%	0.0%	0.078%	0.0%	0.16%	0.0%
24	0.32%	0.081%	0.32%	0.0%	0.081%	0.0%	0.081%	0.0%
25	0.084%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
26	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.088%	0.088%
27	0.0%	0.0%	0.0%	0.0%	0.091%	0.0%	0.091%	0.0%
28	0.0%	0.0%	0.0%	0.0%	0.19%	0.0%	0.0%	0.094%
29	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.10%
30	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.40%	0.0%
31	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.21%
32	0.22%	0.0%	0.0%	0.0%	0.11%	0.0%	0.22%	0.11%

Table A.16: % of Inter-Super-Cluster Connections Contained in Each Type of Buses for M = 32 – Part 3 of 3

References

[Ahme00]
E. Ahmed and J. Rose, “The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, February 2000, pp.3–12.

[Also00]
A. Alsolaim, J. Starzyk J. Becker, and M. Glesner, “Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems,” *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines,* April 2000, pp.205–214.

[Alte02]
Altera Data Sheet, Altera, 2002.

[Betz97a]
V. Betz and J. Rose, “VPR: A New Packing, Placement and Routing Tool for FPGA Research,” *Proceedings of the International Workshop on Field-Programmable Logic and Applications,* 1997, pp.213–222.

[Betz97b]
V. Betz and J. Rose, “Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size,” *Proceedings of the IEEE Custom Integrated Circuits Conference,* 1997, pp.551–554.

[Betz98]
V. Betz and J. Rose, “How Much Logic Should Go in an FPGA Logic Block?”, *IEEE Design and Test Magazine*, Spring 1998, pp.10–15.

[Betz99a]
V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs,* February 1999, Kluwer Academic Publishers.

[Betz99b]
V. Betz and J. Rose, “FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* February 1999, pp.59–68.

[Betz00]
V. Betz and J. Rose, “Automatic Generation of FPGA Routing Architectures from High-Level Descriptions,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* February 2000, pp.175–184.

[Betz01]
V. Betz, J. Rose, and A. Marquardt, “VPR: A Placement and Routing Tool for FPGA Research,” *Software Publication at http://www.eecg.utoronto.ca/~vaughn/vpr/vpr.html,* University of Toronto, 2001.

[Bitt96]

R. Bittner, P. Athanas, and M. Musgrove, “Colt: An Experiment in Wormhole Run-Time Reconfiguration,” *Proceedings of the Conference on High-Speed Computing, Digital Signal Processing, and Filtering Using reconfigurable Logic,* 1996.

[Bozo01]

E. Bozorgzadeh, S. Memik, and M. Sarrafzadeh, “RPack: Routability-Driven Packing for Cluster-Based FPGAs,” *Proceedings of the Conference on Asia-South Pacific Design Automation Conference,* January 2001, pp.629–634.

[Brow92a]

S. Brown, “Routing Algorithms and Architectures for Field-Programmable Gate Arrays,” *Ph.D. Dissertation,* University of Toronto, 1992.

[Brow92b]

S. Brown, J. Rose, and Z. Vranesic, “A Detailed Router for Field-Programmable Gate Arrays,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* May 1992, pp.620–628.

[Call98]

T. Callahan, P. Chong, A. DeHon, and J. Wawrzynek, “Fast Module Mapping and Placement for Datapaths in FPGAs,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 1998, pp.123–132.

[Cart86]

W. Carter, K. Duong, R. Freeman, H. Hsieh, J. Ja, J. Mahoney, L. Ngo, and S. Sze, “A User Programmable Reconfigurable Logic Array,” *Proceedings of the IEEE Custom Integrated Circuits Conference,* 1986, pp.233–235.

[Chan00]

P. Chan and M. Schlag, “New Parallelization and Convergence Results for NC: A Negotiation-Based FPGA Router,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 2000, pp.165–174

[Chen92]

D. Chen and J. Rabaey, “A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths,” *IEEE Journal of Solid-State Circuits,* December 1992, pp.1895–1904.

[Chen03]

D. Chen, J. Cong, and Y. Fan, “Low-Power High-Level Synthesis for FPGA Architectures,” *Proceedings of the International Symposium on Low Power Electronics and Design,* August 2003, pp.134–139.

[Cher94]

D. Cherepacha and D. Lewis, “A Datapath Oriented Architecture for FPGAs,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 1994.

[Cher96]

D. Cherepacha and D. Lewis, “DP-FPGA: An FPGA Architecture Optimized for Datapaths,” *VLSI Design,* 1996, pp.329–343.

[Cher97]
D. Cherepacha, *M.A.Sc. Thesis*, University of Toronto, 1997.

[Cong03]
J. Cong, M. Romesis, and M. Xie, “Optimality and Stability Study of Timing-driven Placement Algorithms,” *Proceedings of the International Conference on Computer Aided Design,* November 2003, pp.472–478.

[Cora96]
M. Corazao, M. Khalaf, M. Potkonjak, and J. Rabaey, “Performance Optimization Using Template Mapping for Datapath-Intensive High-Level Synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 1996, pp.877–888.

[Ebel95]
C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, “Placement and Routing Tools for the Triptych FPGA,” *IEEE Transactions on VLSI Systems,* December 1995, pp.473–482.

[Ebel96]
C. Ebeling, D. Cronquist, P. Franklin, and C. Fisher, “RaPiD A Configurable Computing Architecture for Compute-Intensive Applications,” *Proceedings of the International Workshop on Field-Programmable Logic and Applications, 1996.*

[Elmo48]
W.C. Elmore, “The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifier,” Journal of Applied Physics, 19, pp.55–63, 1948.

[Gold00]
S. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, “PipeRench: a reconfigurable architecture and compiler,” *IEEE Computer,* April 2000, pp.70–77.

[Hama02]
C. Hamacher, Z. Vranesic, and S. Zaky, *Computer Organization,* 5th Edition, 2002, McGraw-Hill.

[Harr02]
I. Harris and R. Tessier, “Testing and Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* November 2002, pp.1337–1343.

[Haus97]
J. Hauser and J. Wawrzynek, “Garp: A MIPS Processor with a Reconfigurable Coprocessor,” *Proceedings of the IEEE Symposium of Field-Programmable Custom Computing Machi*nes, April 1997, pages 24–33.

[Hsei90]
H. Hseih, et al, “Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays,” *Proceedings of the IEEE Custom Integrated Circuits Conference,* 1990, pp.31.2.1–31.2.7.

[Katz94]
R. Katz, *Contemporary Logic Design*, 1994, The Benjamin/Cummings Publishing Company.

[Kirk83]

S. Kirkpatrick, C. Gelatt and M. Vecchi, “Optimization by Simulated Annealing,” *Science,* May 13, 1983, pp.671–680.

[Koch96a]

A. Koch, “Structured Design Implementation — A Strategy for Implementing Regular Datapaths on FPGAs,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 1996, pp.151–157.

[Koch96b]

A. Koch, “Module Compaction in FPGA-based Regular Datapaths,” *Proceedings of the Design Automation Conference,* 1996, pp.471–476.

[Kore02]

I. Koren, *Computer Arithmetic Algorithms*, 2002, A K Peters.

[Kutz00a]

T. Kutzschebauch and L. Stok, “Regularity Driven Logic Synthesis,” *Proceedings of the IEEE/ACM International Conference on Computer Aided Design,* 2000, pp.439–446.

[Kutz00b]

T. Kutzschebauch, “Efficient Logic Optimization Using Regularity Extraction,” *Proceedings of the International Conference on Computer Design,* 2000, pp.487–493.

[Lee61]

C. Lee, “An Algorithm for Path Connections and its Applications,” *IRE Transactions on Electronic Computing,* Vol. 10, 1961, pp.346–365.

[Leij03]

K. Leijten-Nowak and J. van Meerbergen, “An FPGA architecture with enhanced datapath functionality,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 2003, pp.195–204.

[Lemi97]

G. Lemieux, S. Brown, D. Vranesic, “On Two-Step Routing for FPGAs,” *ACM Symposium on Physical Design,* 1997, pp.60–66.

[Lemi01]

G. Lemieux and D. Lewis, “Using Sparse Crossbars within LUT Clusters,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 2001, pp.59–68.

[Lemi02]

G. Lemieux and D. Lewis, “analytical Framework for Switch Block Design,” *Proceedings of the Conference on Field-Programmable Logic and Applications,* 2002, pp.122–131.

[Leve03]

P. Leventis, M. Chan, M. Chan, D. Lewis, B. Nouban, G. Powell, B. Vest, M. Wong, R. Xia, and J. Costello, “Cyclone: A Low-Cost, High-Performance FPGA,” *Proceedings of the IEEE Custom Integrated Circuits Conference,* 2003.

[Lewi03]

D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose, "The Stratix Routing and Logic Architecture," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* February 2003, pp.15–20.

[Li03]

F. Li, D. Chen, L. He, and J. Cong, "Architecture Evaluation for Power-Efficient FPGAs," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, February 2003, pp.175–184.

[Lin03]

J. Lin, A. Jagannathan, and J. Cong, "Placement-Driven Technology Mapping For LUT-Based FPGAs," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, February 2003, pp.121–126.

[Marq99]

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

[Marq00a]

A. Marquardt, V. Betz, and J. Rose, "Timing-Driven Placement for FPGAs," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* February 2000, pp.203–213.

[Marq00b]

A. Marquardt, V. Betz, and J. Rose, "Speed and Area Tradeoffs in Cluster-Based FPGA Architectures," *IEEE Transactions on VLSI Systems,* February 2000, pp.84–93.

[Mars99]

A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 1999, pp.135–143.

[Masu99]

M. Masud and S. Wilton, "A New Switch Block for Segmented FPGAs," *Proceedings of the International Workshop on Field Programmable Logic and Applications,* August 1999, pp.274–281.

[Mirs96]

E. Mirsky and A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines,* April 1996, pp.157–166.

[Nase94]

A. Naseer, M. Balakrishnan, and A. Kumar, "FAST: FPGA Targeted RTL Structure Synthesis Technique," *Proceedings of the International Conference on VLSI Design,* 1994, pp.21–24.

[Nase98]

A. Naseer, M. Balakrishnan, and A. Kumar, “Direct Mapping of RTL Structures onto LUT-Based FPGAs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* July 1998, pp.624–631.

[Okam96]

T. Okamoto and J. Cong, “Buffered Steiner Tree Construction with Wire Sizing for Interconnect Layout Optimization,” *Proceedings of the IEEE/ACM International Conference on Computer Aided Design,* 1996, pp.44–49.

[Quar03]

Quartus II Manual, Altera Corp., 2003.

[Rose90]

J. Rose, R. Francis, D. Lewis, and P. Chow, “Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency,” *IEEE Journal of Solid-State Circuits*, October 1990, pp.1217–1225.

[Sank99]

Y. Sankar and J. Rose, “Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* February 1999, pp.157–166.

[Sech85]

C. Sechen and A. Sangiovanni-Vincentelli, “The TimberWolf Placement and Routing Package,” *IEEE Journal of Solid-State Circuits,* April 1985, pp.510–522.

[Sech86]

C. Sechen and A. Sangiovanni-Vincentelli, “TimberWolf3.2: A New Standard Cell Placement and Global Routing Package,” *Proceedings of the Design Automation Conference,* 1986, pp.432–439.

[Sech87]

C. Sechen and K. Lee, “An Improved Simulated Annealing Algorithm for Row-Based Placement,” *Proceedings of the IEEE/ACM International Conference on Computer Aided Design,* 1987, pp.478–481.

[Sun95]

W. Sun and C. Sechen, “Efficient and Effective Placement for Very Large Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* March 1995, pp.349–359.

[Sun99]

Pico-Java Processor Design Documentation, 1999, Sun Microsystems.

[Swar95]

W. Swartz and C. Sechen, “Timing Driven Placement for Large Standard Cell Circuits,” *Proceedings of the Design Automation Conference,* 1995, pp.211–215.

[Swar98]

J. Swartz, V. Betz and J. Rose, ``A Fast Routability-Driven Router for FPGAs," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* 1998, pp.140–149.

[Syno99]
Synopsys Design Compiler Manual, Synopsys Inc., 1999.

[Synp03]
Synplify Pro Manual, Synplicity Inc., 2003.

[Taka98]
T. Miyamori and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines,* April 1998.

[Tess02]
R. Tessier, "Fast Placement Approaches for FPGAs," *ACM Transactions on Design Automation of Electronic Systems,* April 2002, pp.284–305.

[Tuan03]
T. Tuan and B. Lai, "Leakage Power Analysis of a 90nm FPGA," *Proceedings of the IEEE Custom Integrated Circuit Conference,* November 2003, pp.57–60.

[Varg99]
G. Varghese, H. Zhang, and J. Rabaey, "Design of a Low Energy FPGA," *Proceedings of the International Symposium on Low Power Electronics and Design,* 1999, pp.188–193.

[Wain97]
E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring It All to Software: Raw Machines," *IEEE Computer,* September 1997, pp.86–93.

[West92]
N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective,* 2nd Edition, 2002, Addison Wesley Longman.

[Wilt97]
S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories," *Ph.D. Dissertation,* University of Toronto, 1997.

[Xili02]
Xilinx Data sheet, Xilinx Inc., 2002.

[Ye97]
A. Ye, "Microelectronics Bridge Camp Project: A 16-bit CPU," *Technical Report,* University of Toronto, Summer 1997.

[Ye99a]
A. Ye and D. Lewis, "Procedural Texture Mapping on FPGAs," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* February 1999, pp.112–120.

[Ye99b]
A. Ye, "Procedural Texture Mapping on FPGAs," *M.A.Sc. Thesis,* University of Toronto, June 1999.

[Yeun93]
A. Yeung and J. Rabaey, “A Reconfigurable Data Driven Multi-Processor Architecture for Rapid Prototyping of High Throughput DSP Algorithms,” *Proceedings of the HICCS Conference*, January 1993, pp.169–178.