# Improving the Area Efficiency of Heterogeneous FPGAs with Shadow Clusters

by

Peter Andrew Jamieson

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

Improving the Area Efficiency of Heterogeneous FPGAs with Shadow Clusters

Peter Andrew Jamieson

Doctor of Philosophy, 2007
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

# Abstract

Field Programmable Gate Arrays (FPGAs) serve the microchip market for designs that need to be created quickly, in small volume, or that need to be updated in the field. FPGAs have not taken over the market for large capacity, high-volume Application-Specific Integrated Circuits (ASICs) since the FPGA cost is too high. This cost is mainly due to the large area gap between FPGAs and ASICs.

One approach to improve the area efficiency of FPGAs is with the inclusion of hard "specific" circuits on the FPGA fabric. These circuits can implement functionality in designs in less silicon area, at a faster speed, and with less power consumption compared to implementing the same functionality in the programmable elements of an FPGA. Common examples include hard multipliers and hard memories.

The fundamental question in FPGA architecture research is determining which hard circuits to include on an FPGA. Every included hard circuit needs to be used and provide a benefit to the range of designs mapped to FPGAs.

In this work, we seek to improve the utilization of hard circuits on FPGAs to make these devices more area efficient. To do this we develop an architecture concept called "shadow clusters" that combines the programmable aspect of an FPGA with a hard circuit such that a hard circuit and its routing resources can always be used. In the best case we measured, it improves the FPGA area efficiency by 12.5%.

We combine shadow clusters with less popular hard circuits, such as a crossbar, that haven't been included on industrial FPGAs because designs in the target market do not sufficiently target these hard circuits to justify their inclusion. Shadow clusters significantly change the area impact of these hard circuits making these FPGAs more area efficient and more likely to be included on industrial FPGAs.

We, also, explore the algorithms in the flow that maps designs to FPGAs with hard circuits. The goal is to efficiently map designs to FPGAs with hard circuits and to maximize their utilization where these algorithms are designed to flexibly and efficiently target a wide range of hard circuits on FPGAs.

# Acknowledgements

> Don't put your trust in revolutions. They always come around again. That's why they're called revolutions.
>
> Terry Pratchett

The three most important people in getting me this far are my supervisor, Jonathan Rose, my father, Andrew Jamieson, and my mother, Georgie Jamieson. Each has contributed significantly to this work through their support, mentor-ship, and advice. In the case of Jonathan Rose, I'm thrilled to have worked with him, and the lessons I have learned over the past few years have truly transformed me and my understanding of not only the small world of FPGAs, but the world itself (and arguably even my Ultimate career).

If I had the chance to choose my parents and family (which includes my sisters Patricia and Jennifer), I can't think of a better choice than what came to be. We have all walked through life together and even though geography currently separates us, your constant support kept me going through the ups and downs.

To my good friends, the past few years have been spectacular because of you. We could argue, and I'm sure we would, that our discussions, debates, and adventures are where the true growth of an individual occurs. There are a number of names to mention as I have met so many people and in no specific order: Kirk, Ian, Inian, Navid, Lexi, Shawn, Jason, Lisa, Lesley, Guy, Norm, Marcius, Taylor, Kapil, Sasha, Peyton, Pam, Dave, Rahil, Ajay, Rob, Dion, Kevin, Alexis, Alison, Scott, Warren, Mike, Ali, Mark, Wei, Aaron, Josh, Paul, Frank, Andrew, and the list continues.

The sport Ultimate has made Toronto a special place for me. To the teams I've played with and the success we've had from winning local tournaments in the winter to taking a University National Championship with the University of Toronto, to travelling vast

distances throughout the North East and beyond to throw plastic - the experience was amazing and kept me in Toronto longer than I would have ever guessed.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

> They say a little knowledge is a dangerous thing, but it's not one half so bad as a lot of ignorance.
>
> Terry Pratchett

## 1.1 Motivation

FPGAs are a widely used implementation medium for digital circuits that are able to implement virtually any digital design. An increasing number of applications employ FPGAs both to avoid dealing with the fabrication process intricacies of designing Application-Specific Integrated Circuits (ASICs) and the high costs associated with manufacturing these chips. A designer can, in seconds, program and test their digital design on a single FPGA without having to handle low-level electrical design layout issues such as Optical Proximity Effect (OPE) [Lie03], power leakage [Bor99], or crosstalk [BVOP03]. To a lesser extent FPGAs are popular because they can be updated in the field, used to quickly prototype designs, and purchased as needed instead of keeping a large inventory of ASICs.

FPGAs have failed to take over the market for large capacity, high-volume ASICs for one main reason: their high cost due to their large silicon area. Figure 1.1 illustrates this point [BFRV92]; the graph shows the unit manufacturing cost of using either FPGAs or ASICs to implement a digital design. The dotted line represents the constant unit cost of an ASIC as volume increases (along the x-axis) and the solid line represents the unit cost of an FPGA as volume increases. ASICs use less area to implement a design, but have a high initial cost that decreases as this cost is amortized over a larger number of units. The value on the x-axis of the crossing point for the two lines is the

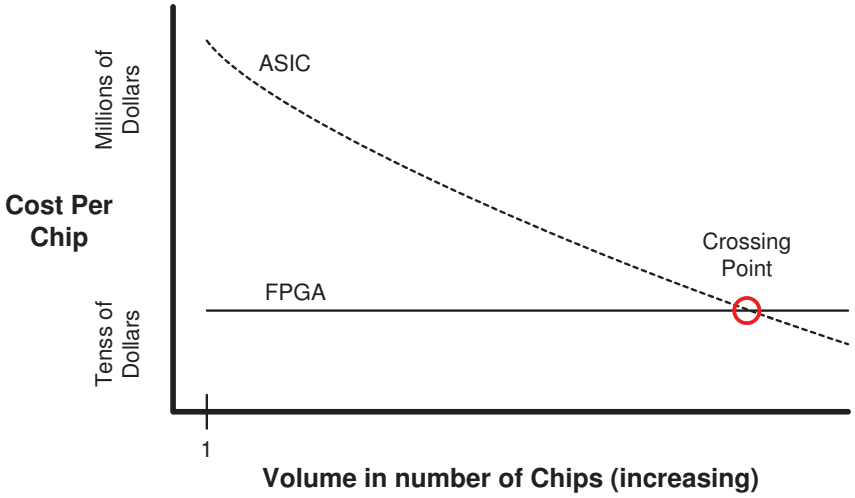


Figure 1.1: Per chip price of FPGAs and ASICs versus volume [BFRV92]

volume of chips at which the unit cost to implement the design on FPGAs or ASICs is equal. For FPGAs to become more broadly applicable in the higher volume markets, this crossing point must move to the right by making each FPGA cheaper.

FPGA researchers attempt to make FPGAs more capable by improving their area efficiency resulting in less used silicon and a lower cost per FPGA. To do this, they try to create FPGA architectures that more efficiently implement the range of designs that are mapped to this technology.

Recently, the inclusion of what we call hard "specific-purpose" circuits, such as hard multipliers, adders, and memory blocks [Lat04, Act02, Qui03, Alt04d, Xil06], improve the area efficiency of designs mapped to these FPGAs. This circuitry is added to FPGAs so that specific operations in a design can be implemented more area efficiently, operate faster, and reduce power consumption.

The goal of this dissertation is to improve these kinds of hard circuits (either the hard circuit itself or how it is mapped) to make FPGAs more area efficient and ultimately reduce their cost, increasing the availability and accessibility of FPGAs to all.

## 1.2 Modern FPGAs and the Central Question in FPGA Architecture

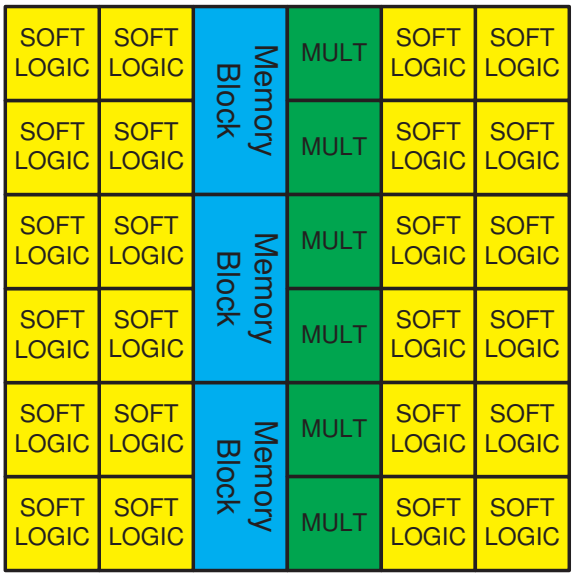


Figure 1.2: Representation of a heterogeneous FPGA

Modern FPGAs, such as the one pictured in Figure 1.2, include hard specific-purpose circuits such as hard multipliers and memories. User's designs map to these hard circuits to help reduce the area consumed (among improving other criteria such as the speed and power consumption of their designs). Instead of using hard circuits, parts of a design could be mapped to the soft programmable logic, but these implementations use significantly more area compared to using hard circuits.

Recently, research at the University of Toronto empirically measured the area, speed, and power benefit of hard circuits. Kuon and Rose [KR06] measured the gap between FPGAs and ASICs with respect to the area, speed, and power consumption of the same design. They showed that for their designs the area of an FPGA without hard circuits is 35 times larger than an ASIC, but this gap is reduced to approximately 21 times when using an FPGA that contained hard multipliers and memories. The smallest area gap (between FPGAs with hard circuits and ASICs) in this work was a filter design

that includes both memory and multipliers; here, the design was only 10 times larger on an FPGA with hard circuits compared to the ASIC implementation.

Clearly, hard circuits can reduce the area gap in a significant way when used, and a common (but naive) thought is to add more hard circuits to FPGAs to further improve them. This approach might benefit designs that target these FPGAs if all the hard circuits are used, but if these hard circuits are not used, then they are wasted silicon making the area problem worse! Even more, this waste includes the area-expensive programmable routing that connects into and out from each hard circuit.

Indeed, the central question of FPGA architecture is when to include more hard specific-purpose circuits on an FPGA [Ros04].

The simple solution is to add a hard circuit when it provides a benefit to most of the designs that target these FPGAs. More formally, to add a hard circuit to an architecture it should satisfy two key criteria:

1. The hard circuit provides an area, speed, or power benefit to implement a part of a design compared to implementing that part in the soft programmable logic.

2. There are a sufficient number of designs, from the FPGA's target market, that will use each of the added hard circuits.

If either one of these criteria is not satisfied, then the hard circuit does not make an FPGA any better. Clearly, if criteria #1 is not satisfied, then the hard circuit provides no benefit for a mapped design, and if criteria #2 is not satisfied, then the additional unusable area by the majority of designs (that don't use the hard circuit) is too costly compared to the benefits gained by a few designs.

The second criteria has an additional nuance. Not only do many of the designs need to use a hard circuit, but also depending on the number of hard circuits available on the FPGA, it is important for all these hard circuits to be used by each design. If any of these hard circuits are not used, then a designer is paying for silicon that they do not use. In the best case, designers would like to map their designs to an FPGA that provides the exact number of resources that matches what their design uses.

These two criteria suggest that the benefit of adding a hard circuit to an FPGA can be scientifically measured if certain aspects of the market, such as the demand for hard

circuits, are known. In this dissertation, we will present a scientific methodology to measure the net area benefit of a hard circuit for a set of designs that target FPGAs.

It is unreasonable, however, to expect that an FPGA will contain the exact number of hard circuits to satisfy the needs of all designs that would use it. For this reason, one of the goals when creating an FPGA with hard circuits is to maximize the utilization of added hard circuits.

In this dissertation, the main theme is to improve the area efficiency of FPGAs by finding more ways, either with CAD for FPGAs or FPGA architecture, to increase the utilization of hard circuits by designs that target these chips.

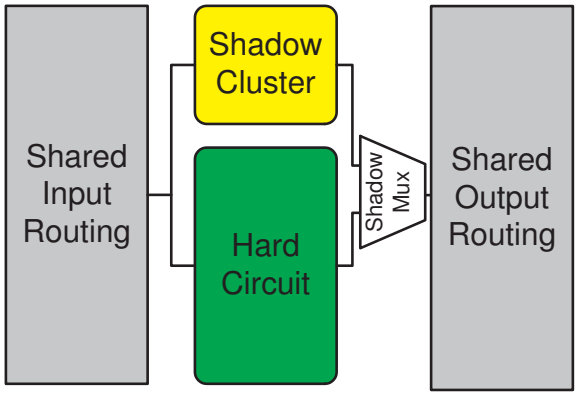


Figure 1.3: Illustration of shadow cluster concept

Our most successful approach to increase the utilization of hard circuits is with an architectural concept called "shadow clusters" in which a standard FPGA logic "cluster" (typically consisting of a group of programmable logic elements and flip-flops) exists within the same logical grouping as a hard circuit such as a multiplier (as illustrated in Figure 1.3). If the hard circuit is not used by the mapped design, simple fabric-programmable multiplexers "swap" in the shadow cluster, which can be used just like the regular soft programmable logic.

The benefit of shadow clusters is that even though the area dedicated for a hard circuit remains unchanged and we are actually increasing area by combining soft programmable logic with the hard circuit, this additional circuitry is always usable by any logic function. This means that the expensive programmable routing that surrounds

the hard circuit (consuming up to 90% of a hard circuit's area) can always be used.

The shadow cluster concept can be applied to almost any hard circuit. In Chapter 4, we show that shadow clusters improve the overall area efficiency of FPGAs with hard multipliers. This work is published in [JR06], and to our knowledge, represents a new way of improving FPGA area efficiency.

We, also, combine shadow clusters with hard circuits that previously have not been added to FPGAs, mainly due to their low demand from designs targeting FPGAs. Here, the metric that we seek to improve is the "frequency" that the need for these hard circuits must appear in the FPGA's target designs for the inclusion of the hard circuit to appear to be area neutral compared to an FPGA with only soft programmable logic. As an exemplar, we architect and include hard crossbars combined with shadow clusters in Chapter 5 and measure how shadow clusters change the area-neutrality metric for an FPGA with hard crossbars compared to an FPGA with only soft programmable logic.

In this dissertation, we examine how to efficiently map designs to FPGAs with hard circuits in an effort to maximize hard circuit utilization. In Chapter 6, we present a front-end Register Transfer Level (RTL) algorithms that are designed to flexibly and efficiently target a wide range of hard circuits on FPGAs, and a tool that we have built using these algorithms achieves close to parity when mapping designs to FPGAs with hard multipliers and memories compared to an industrial tool that performs this same mapping.

This tool is designed to efficiently target modern FPGAs and interfaces with commercial FPGA CAD flows. To study the quality of this tool, we compare the area and speed results of designs mapped by this tool against an industrial strength tool. This work is published in [JR05a], and to our knowledge is the first open source Verilog HDL tool targeting FPGA CAD flows.

## 1.3 Thesis Organization

The organization of this dissertation is as follows: Chapter 2 reviews background material including the basic structure of an FPGA, definitions for FPGAs including ones

with hard circuits, and CAD for targeting FPGAs that include hard circuits. We also review previous work that attempts to maximize the utilization of hard circuits.

Next, in Chapter 3, we introduce some new terminology and architecture concepts that will be used throughout this dissertation. We introduce a scientific methodology to measure the area benefit of including hard circuits on an FPGA, and use this methodology to measure the area benefit of including hard multipliers on an FPGA. In addition, we present details about the benchmarks that we have collected and use in most of the chapters in this dissertation to measure the quality of our ideas.

The research contributions described in the previous section are presented in detail in chapters 4, 5, and 6. Each chapter includes an introduction, discussion, measurement methodology, results, and summary. Even though each of these chapters is relatively self-contained, each chapter contributes to the general theme of this dissertation - improving utilization of hard circuits to improve the overall area efficiency of FPGAs.

In Chapter 4, we introduce our shadow cluster concept and apply this concept to hard multipliers. We then measure the area-efficiency improvement of FPGAs with multipliers combined with shadow clusters compared to FPGAs with only hard multipliers.

In Chapter 5, we present the hard crossbars as a representative of low-demand circuits that are added to FPGAs. We then proceed to measure the frequency at which hard crossbars combined with shadow clusters are area neutral with an FPGA that has no hard crossbars.

In Chapter 6, we describe a front-end RTL elaboration tool and how it efficiently maps designs to the hard circuits available on an FPGA. We then measure how this tool compares to an existing industrial tool that performs this same mapping. We, also, study two types of hard crossbar architectures, single bit and bus based, to determine how these architectures affect our results.

Chapter 7 concludes the dissertation with some final remarks and a review of the contributions. We provide some possible future avenues to extend the research.

Finally, there are three appendices that provide additional information about this work. Appendix A describes an automatic sizing tool used in this research. This includes how the tool works, what external software is used, and the quality of our

sizing estimations. Appendix B provides additional details about the benchmarks used in this work. Finally, appendix C provides raw data from our experiments. This is included in the electronic version of this thesis and represents the data collected and summarized throughout this dissertation.

# 2 Background and Related Work

> The difference between [the hard way] and the easy way is that the hard way works.
>
> Terry Pratchett

## 2.1 Introduction

In this chapter, we review previous work and terminology related to FPGAs with a focus on those with hard circuits. We start with a description of basic FPGAs that consist only of soft programmable logic, and then we describe and provide examples of FPGAs with hard circuits. Next, we review an FPGA CAD flow used to map designs to these FPGAs, and finally, we review previous work that attempts to improve the utilization of hard circuits.

## 2.2 FPGA Basics

The simplest type of FPGA consists of logic blocks and routing that each can be programmed to implement a digital design - we call this the soft logic fabric or soft programmable logic. A homogeneous FPGA consists of only this soft programmable fabric and input and output pins that connect to off chip signals.

The soft logic fabric is built using what is commonly called a *tile*. These tiles are replicated and connected together creating an array of tiles (which is illustrated in portions labeled "soft logic" in Figure 2.1). Each tile consists of programmable units surrounded by programmable routing that allows internal connections and external connections to other tiles.

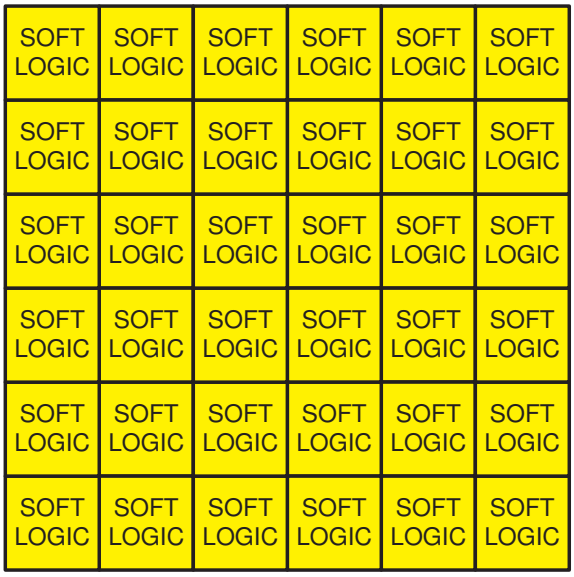


Figure 2.1: Representation of a tiled homogeneous FPGA

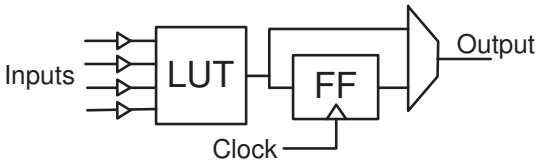


Figure 2.2: Sample BLE built of a LUT and flip-flop

The basic programmable unit is a Basic Logic Element (BLE), which is often some form of Lookup Table (LUT) together with a flip-flop [SMS02] as seen in Figure 2.2. Other examples of BLEs include NAND gates [Ple90], transistors [MC92], and multiplexers [Act96, BCC+91]. Each tile consists of several BLEs and programmable routing, forming what is commonly called a cluster, and we call the tiles that make up the soft programmable fabric, *soft logic cluster tiles*. The programmable routing is used to connect to both internal and external BLEs.

Paths between BLEs are set by programming switches that connect inputs and outputs of BLEs with wire segments (in what are called connection blocks) and wire segments to other wire segments (in what are called switch blocks). There are many

ways to design a programmable switch. The most popular programmable switch is composed of static memory cells, pass transistor multiplexers, and a level-restoring buffer. The memory cells are connected to the control inputs of the multiplexer and determine which input will be connected to the output of the multiplexer. These switches form part of a programmable routing architecture that is designed to connect two different points such that signals move in one predetermined direction (from the output of a routing multiplexer to the input of another multiplexer). This type of routing architecture is called direct drive (also known as uni-directional) [LL04].

The architecture of the soft logic fabric is described by many parameters. These parameters include the number of BLEs per cluster (N), the input size of the LUTs (K) in a BLE, the number of programmable routing tracks in the vertical and horizontal channels connecting tiles (W), the input connectivity to the BLEs in a soft logic cluster ($F_{Cin}$), the output connectivity from the BLEs to the routing tracks ($F_{Cout}$), and the switch block flexibility connecting routing tracks with each other ($F_s$) among several other parameters [BRM99].

Figure 2.3 shows a schematic of a soft logic cluster tile including examples for some of the parameters listed above. This figure includes all the programmable routing in the tile connecting in and out of the BLE. Note the large number of multiplexers and buffers needed to programmably connect the routing tracks to the BLE. These routing circuits are the main reason why programmable routing takes up most of the area in FPGAs.

## 2.3 Modern Heterogeneous FPGAs

We call an FPGA that combines hard circuits with the soft programmable fabric a heterogeneous FPGA, where a hard circuit is formally defined as:

**Definition:** A hard circuit is a specific circuit included on an FPGA to perform specific logic functions, *which could also be implemented using only the soft programmable fabric.* ⋆

Hard circuits are included on the FPGA since they implement a logic function either in less area, greater speed, consuming less power, or a mixture of all three compared

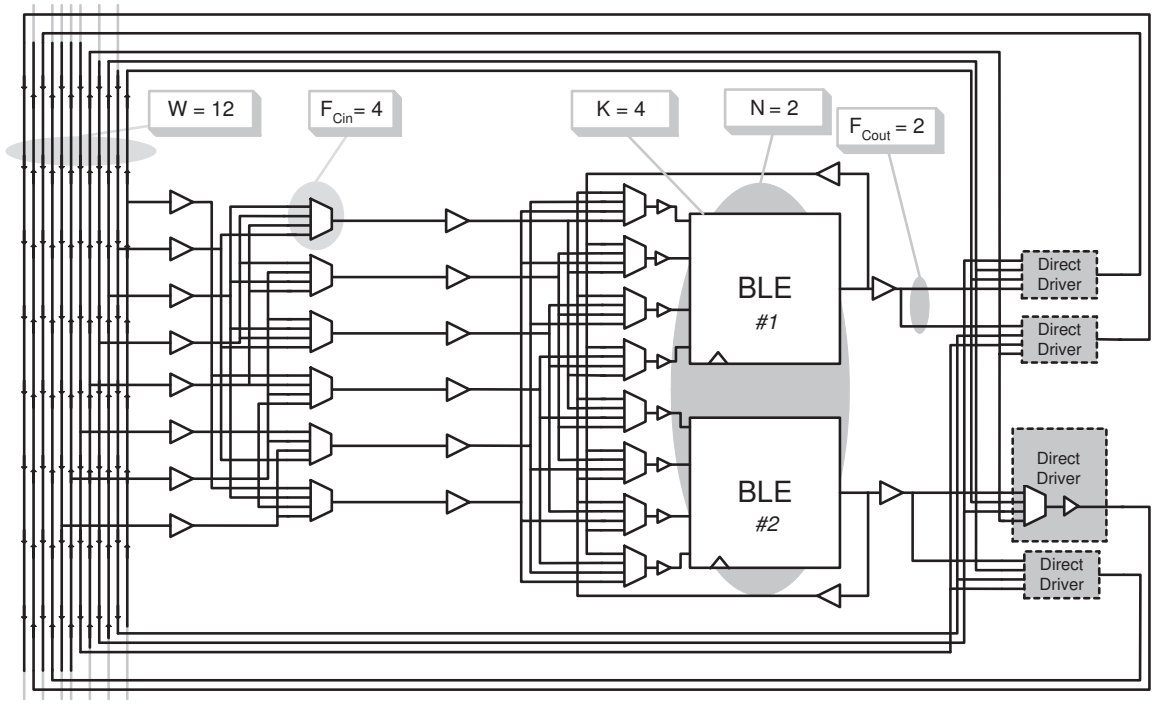


Figure 2.3: Soft logic cluster tile including architecture parameters

to implementing the same logic function in the soft logic fabric. This definition for a hard circuit does not include some other circuit structures on an FPGA such as the specialized input and outputs or phase locked loops, which are also included on modern FPGAs like the Stratix [Alt03] and Virtex [Xil05] FPGAs.

Given this definition, there are two common ways hard circuits are included on FPGAs. The first is to include them into all the tiles in an FPGA. Here, all the tiles are the same, and the FPGA remains a homogeneous array of tiles, and we call this type of heterogeneity, *soft fabric heterogeneity*. For example, flip-flops are commonly paired with LUT based FPGAs making designs mapped to these FPGAs faster, more area efficient, and consuming less power in, arguably, all cases compared to implementing the flip flop in the soft logic fabric [RFCL89, RFLC90].

The second way of including a hard circuit on an FPGA is by including a hard circuit as a differentiated tile. In this case, an FPGA no longer has a homogeneous set of tiles, and we call this kind of heterogeneity, *tile-based heterogeneity*. An example of

a heterogeneous tile is a multiplier [Alt03, Xil05, Qui03], which is a large circuit that can be included on an FPGA as a unique tile.

In the following sections, we discuss these two types of heterogeneity in more detail including examples of these hard circuits in modern FPGAs.

### 2.3.1 Soft Fabric Heterogeneity

The flip-flop is commonly included in FPGA tiles and is an example of soft fabric heterogeneity. Other examples of hard circuits within homogeneous tiles include fine-grained memory [Xil03, Wil97], connection chains [HBO+93, Xil89], and adder configurability [HBO+93], which we will now review.

Many designs use memory for a number of purposes including the storage of incoming, outgoing, and temporary data. Applications benefit from on chip memory since the paths between memory and logic are short compared to paths that connect to off chip memory. One example of memories embedded within the homogeneous tile is Xilinx's Virtex-II and Virtex-4 FPGAs [Xil03, Xil05], which provide memories by adding circuitry to the LUT so it can also be configured as a 16-bit memory. These memory configurable LUTs, LUT_RAMs, allow memory usage in the design to map almost anywhere on an FPGA increasing the chance that memory will be placed close to circuits it connects to.

Another example of soft fabric heterogeneity that provides quick connections between neighboring BLEs is called, connection chains. Connection chains exist as carry chains [HBO+93, HHF98] for building faster arithmetic operations and register chains for building shift registers [Alt03, Alt04d, Xil03, Xil05, Xil06, Lat07a, Lat07b]. Connection chains could be considered part of the routing fabric, but we classify connection chains as hard circuits since they are extra circuitry intended to speed up specific functions instead of using the standard programmable routing.

Arithmetic capabilities are also incorporated in a soft logic cluster tile. For example, in some FPGAs [Alt03, Xil03, Xil05] a 4-input LUT is configurable to act either as a LUT or a 1-bit adder. Without adder capabilities, it takes two 4-input LUTs to implement a 1-bit adder; one LUT produces the sum and one LUT produces the carry. Virtex-5, Stratix II, and Stratix III FPGAs also have adder capabilities in their BLE.

Adder configurability combined with carry chains allows FPGAs to add and subtract with lower area and greater speed compared with a soft logic implementation.

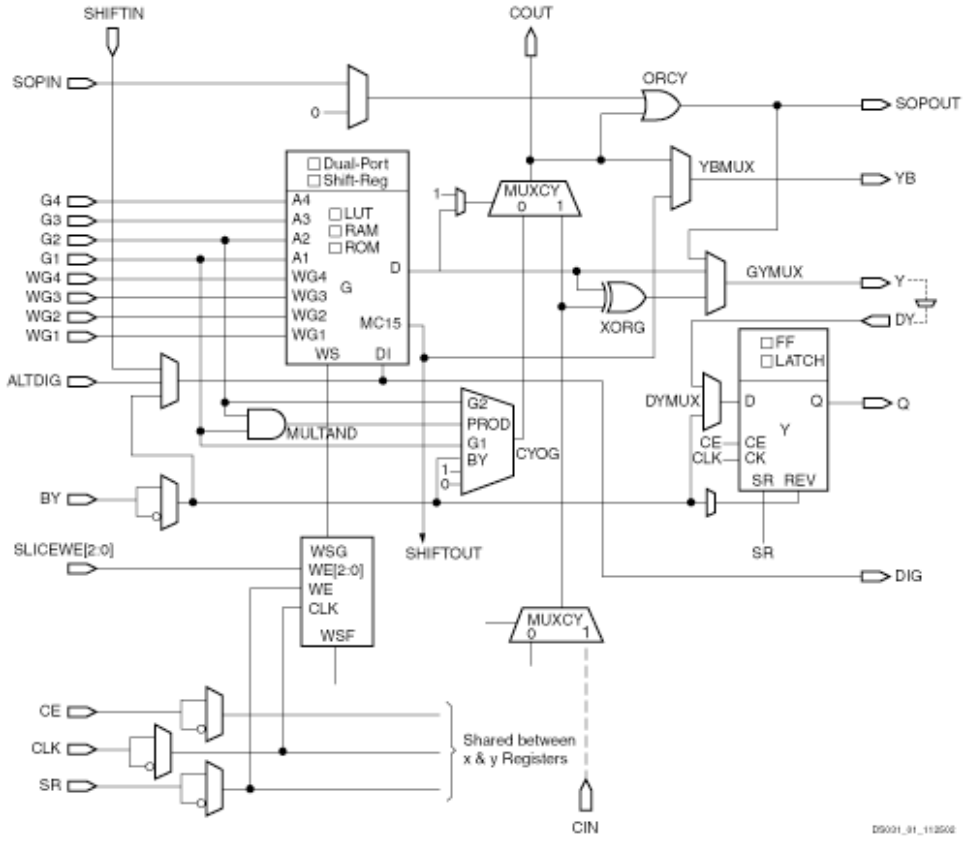


Figure 2.4: Block diagram of a Logic Element in Xilinx's Virtex-II Pro [Xil03]

Figure 2.4 shows a picture of a BLE in the Virtex-II Pro chip [Xil03]. This BLE has many examples of soft fabric heterogeneity including a carry chain labelled by *CIN* and *COUT*, a LUT_RAM pictured in the rectangle in the upper left quadrant, a flip-flop or latch, and a connection chain for a shift register labelled *shiftin* and *shiftout*.

Another way of adding soft fabric heterogeneity to an FPGA is using more than one kind of BLE. For example, each tile could include both 3-LUTs and 4-LUTs, where by our definition, the 4-LUTs are considered hard circuits since a 4-LUT can be implemented using 3-LUTs. A mixture of BLEs makes it possible for a circuit to map to a faster and smaller FPGA implementation. He *et. al.* [HR93] studied both how to

map designs to a mixture of LUT sizes. Their results show that a mixture of two sizes of LUTs can lead to area-efficient design mappings compared with architectures with only one LUT size.

### 2.3.2 Tile-based Heterogeneity

Hard circuits are also included on the FPGA as differentiated tiles. Examples of industrial and academic hard circuit tiles include memories, arithmetic blocks, and processors.

A common heterogeneous tile included on an FPGA is the block memory. Wilton *et. al.* [Wil97, WRV97, WRV96, WRV99, Wil99, WKH99, Nga94] were the first to publish results and evaluate embedded memories on an FPGA. This work showed that embedded memories are beneficial to systems on chip designs, and embedded memories speed up these designs by making paths between memory and logic shorter compared to connecting to off chip memories. Wilton *et. al.* also discussed the need for flexibility in the memory structure since various designs use memories with different bit capacities and data widths. Finally, Wilton showed how to map circuits to configurable memories as well as the effectiveness of configurable embedded memories in improving the speed and area of an FPGA.

Many industrial FPGAs include configurable memories including Actel's ProASIC [Act02], Altera's Stratix I [Alt03], Stratix II [Alt04d] and Stratix III [Alt06], QuickLogic's Eclipse [Qui03], Lattice Semiconductor Corporation's ispXPGA [Lat04], ECP [Lat07a], and ECP2 [Lat07b], and Xilinx's Virtex-II [Xil03], Virtex-4 [Xil05] and Virtex-5 [Xil06].

Beyond embedded memories many FPGAs now include hard circuits that help improve the speed and area of computationally-intensive designs. Xilinx includes multipliers in their Virtex-II family [Xil03] among other of their FPGAs. These multipliers, in the Virtex-II, are 18-bit by 18-bit signed multipliers. Lattice ECP [Lat07a] and ECP2 [Lat07b] include 18x18 hard multipliers. QuickLogic's EclipsePlus [Qui03] FPGAs and Altera's Stratix FPGAs [Alt03, Alt04d] include DSP blocks that can perform operations such as multiply and multiply-accumulate.

Altera's DSP block (Stratix I, Stratix II, and Stratix III) can be configured as a multiplier, multiply accumulator, and multiply summing unit. This circuit is an ex-

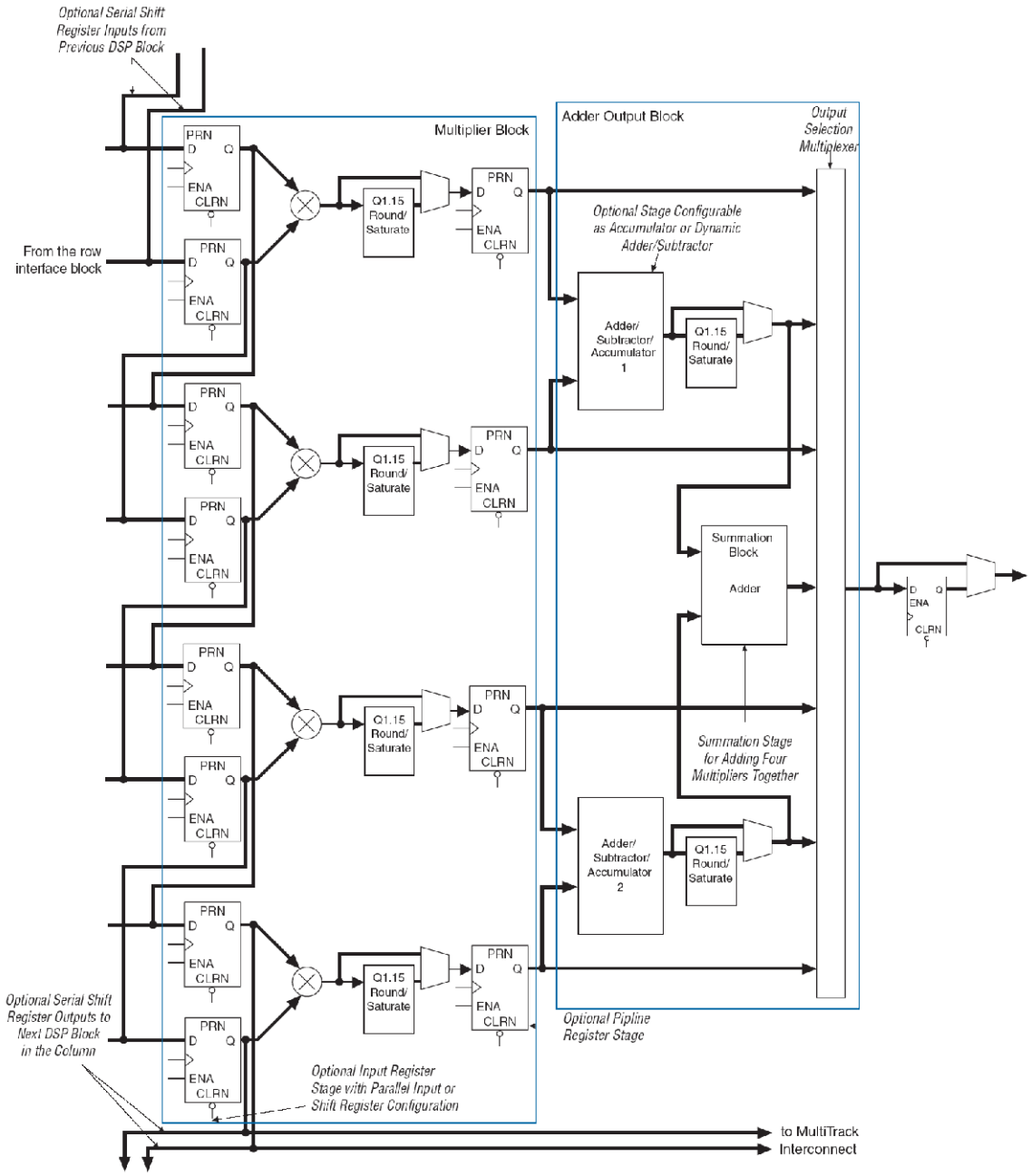


Figure 2.5: High level view of the 18-bit DSP block on the Stratix I [Alt03]

ample of what we call a functionally flexible hard circuit as it can be configured to implement a range of functionality and bit widths so that more designs can make use of this hard circuit. Figure 2.5 shows a representation of the DSP block available on Stratix I FPGAs; with each operation, different bit widths can be specified [Alt03]. For example, this DSP block can be configured to implement either four 9-bit by 9-bit multipliers, two 18-bit by 18-bit multipliers, or one 36-bit by 36-bit multiplier.

Table 2.1: Soft and Hard implementation area and speed results for a 9x9 multiplier

| | Soft Implementation | | Hard Implementation | |
|---|---|---|---|---|
| | Area in Equivalent Cluster Tiles | Max operating frequency | Area in Equivalent Cluster Tiles | Max operating frequency |
| 9x9 Multiplier | 17 | 150 MHz | 2 | 308 MHz |

Table 2.1 shows the potential benefit of mapping 9x9 multiplier to hard circuits on a Stratix I FPGA (while calculating the real benefit requires that we map the entire design to an FPGA with multipliers). The 9x9 multiplier runs at 308 MHz and uses 2 equivalent Stratix I soft logic cluster tiles compared to a soft implementation of the multiplier which uses 17 soft logic cluster tiles and runs at approximately half the speed.

Xilinx's Xtreme DSP [Xil05], introduced in the Virtex-4 FPGA family and included in Virtex-5, is another example of a functionally flexible hard circuit. The Xtreme DSP block's main functionality is an 18 by 18 multiplier, but the Xtreme DSP can also implement multiply accumulation, addition/subtraction, and some bus based multiplexing. Also, the tile includes an adder that allows four of these tiles to be combined to implement a 36 by 36 multiplier without using any additional soft logic. In the Virtex-5, the Xtreme DSP block also includes a bitwise logical mode [Xil06] that can be used instead of the multiplier to perform bitwise logical operations.

Beauchamp [BHUH06] *et. al.* showed the benefit of including a floating point multiplier to an FPGA to improve the speed and implementation area when targeting designs that include floating point operations. This work, however, does not address the effect of including the floating point unit when not all the benchmarks use it, which is one of the key elements presented in this work.

Another group of researchers, Ho *et. al.* [HLL+06], investigated floating point units included on an FPGA using a methodology where they use existing industrial FPGAs and tools to evaluate the speed of designs mapped to their floating point FPGAs. The benefit of their work is the results are generated using industrial tools and architectures, but similar to the previous work the only benchmarks tested using their methodology used the floating point multipliers. To include hard circuits on an existing industrial FPGA, they determine the size and speed of the hard circuit externally, and then map that unit to the FPGA fabric by reserving a rectangular group of soft logic cluster tiles that uses the equivalent area of the hard circuit. To simulate the speed of the hard circuit, the carry chains in these reserved tiles are used since carry chains have the smallest granularity of path delay as each additional BLE hooked up to the carry chain adds a small delay to the overall carry chain path delay.

Another type of hard circuit that has been included on an FPGA is a processor. Embedded memories and multipliers are somewhat functionally flexible, but a processor has both a specific function (though that functionality is programmable itself) and a fixed bit size that may or may not suit designs. Also, the processor is a large circuit compared to multiplier tiles and most memory tiles, making it even more important that designs use the processor when mapped to these FPGAs. Industrial FPGA manufacturers deal with this rigidity of including a processor by providing sub-families that either include or don't include a hard processor.

Examples of a processor on an FPGA include Altera's Excalibur [Alt04f] and Xilinx's Virtex-II Pro, Virtex-4, and Virtex-5 [Xil03, Xil05, Xil06]. The Excalibur can include 200 MHz, 166 MHz, or 133 MHz ARM processors [ARM01] that is embedded in the APEX FPGA architecture [Alt04a]. Xilinx's Virtex-II Pro and Virtex-4 have up to four embedded PowerPCs. This trend will continue in the Virtex-5 FPGA family, and the FX family will include up to 2 PowerPCs when it is released [Roo06]. Altera, however, has not included a hard processor on any of its Stratix FPGAs.

We have now reviewed several examples of hard circuits that are included as differentiated tiles and into the soft logic fabric. In all cases, these inclusions are meant to improve overall speed, area consumption, and power consumption of a design mapped to these FPGAs. Next, we will discuss a typical CAD flow that maps designs to these

heterogeneous FPGAs.

## 2.4 CAD for Heterogeneous FPGAs

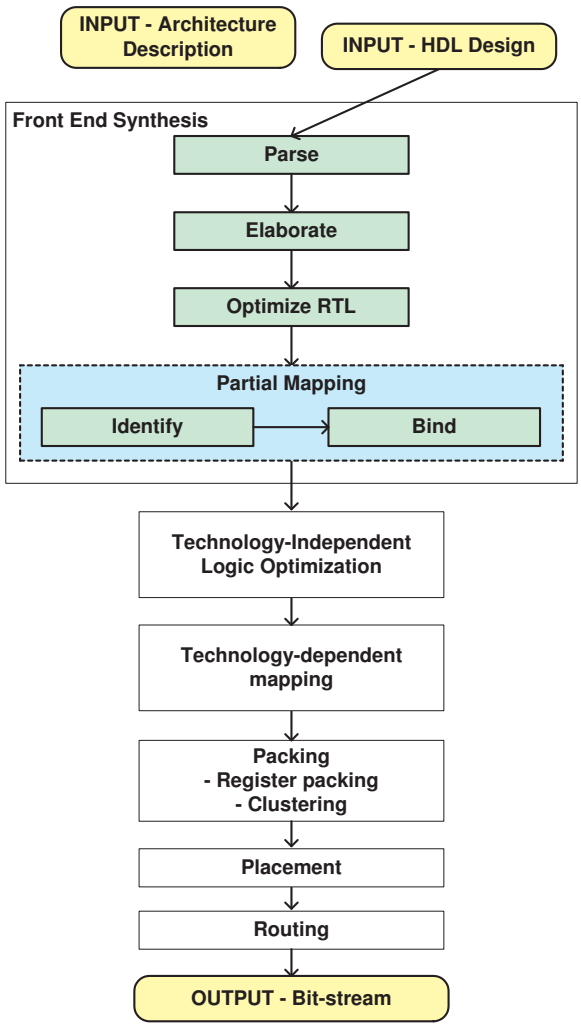


Figure 2.6: CAD flow for converting a design to a bit stream

Heterogeneous FPGA architectures, which include a variety of hard circuits, can be programmed to implement a wide range of designs. FPGA designers use CAD tools to convert their high-level designs into a bit-stream that are then loaded onto these

FPGAs to implement the design. Modern practice has broken CAD process into a series of smaller tasks as illustrated in Figure 2.6. This section describes how a typical CAD flow processes designs with a focus on how the hard circuits are handled.

**INPUT - Verilog HDL Design**

```
module small (a, b, c, out);
        input[5:0] a, b, c;
        output [5:0]out;

        assign out = ({2'b00,a[2:0]} * b) + (b & ~c));
endmodule
```

Figure 2.7: An example Verilog HDL design

Throughout this discussion of heterogeneous FPGA CAD flow we will show how a particular example design (given in Figure 2.7) would be mapped to an FPGA. As we describe each stage of the CAD flow we will show an illustration representing what each stage might do to this example.

The goal of the entire process is to map a design to an FPGA. During each stage, the general goals are to achieve this mapping such that the final design is operating as fast as possible, using the least amount of silicon, and consuming the least amount of power on the FPGA.

The initial input to an FPGA CAD flow is a digital design and an architecture description of the FPGA that is being targeted. A common form for digital designs is a high-level RTL design that describes the transfer of signals between registers and the logical operations performed on these signals during transfers. RTL designs are commonly created using formalized languages that describe electronic circuits. These languages are called Hardware Description Languages (HDLs), and the two most popular standardized HDLs are *Verilog HDL* [Ope93] and *VHDL* [IEE87].

In the first stage, an elaborator transforms the HDL design into a netlist, which consists of input and output ports, primitive logic gates, specific functions such as arithmetic operations, and memory units including registers and memory blocks. Elab-

oration is responsible for preserving as much information from the original HDL design, meaning that instead of converting constructs such as an arithmetic operation into logic gates, the arithmetic operation is preserved. Sometimes, elaboration does not preserve high-level information and instead maps HDL constructs into other forms. For example, control flow statements, including loop and decision statements, might be converted into logical implementations. The elaborator may also convert storage statements into registers or larger memory units. The main task for the elaboration tool is to convert high-level HDL designs to a netlist while preserving the useful high-level design information (for later optimization).

Figure 2.8 shows how the initial input is passed through the first few stages of the front-end synthesis tool. A parser reads in the design, an elaborator creates a netlist representation of the design, and a simple RTL optimizer improves the quality of the netlist. In this example, the original design consisting of a multiplication, an addition, and some logic is converted into a netlist by the elaboration stage. The RTL optimizer, in this example, eliminates some of the most significant bits of the multiplier since these bits are a constant of zero and are not needed.

After elaborating the original HDL design, a stage converts the high-level netlist to a lower-level netlist that more closely conforms to both the FPGAs' hard circuits and soft logic fabric; we call this stage partial mapping since pieces of the design are bound to hard circuits available on the FPGA. The partial mapper uses the library of predefined components available on an FPGA (which is included in the input architecture description) and determines whether parts of the high-level netlist should map to hard circuits or programmable logic. Partial mapping performs this mapping in two steps. First, an identification stage determines the function of each part of the circuit, and second, partial mapping binds functions in the design to hard circuits available on the FPGA or primitive logic gates. After partial mapping, all circuits in the netlist bind either to hard circuits on the FPGA or to logic gates that still must be mapped to the FPGA's soft logic fabric.

To our knowledge, existing CAD flows perform the identification stage of partial mapping in three different ways. The first method involves the designer explicitly instantiating an element from a library of standardized logic functions. An example

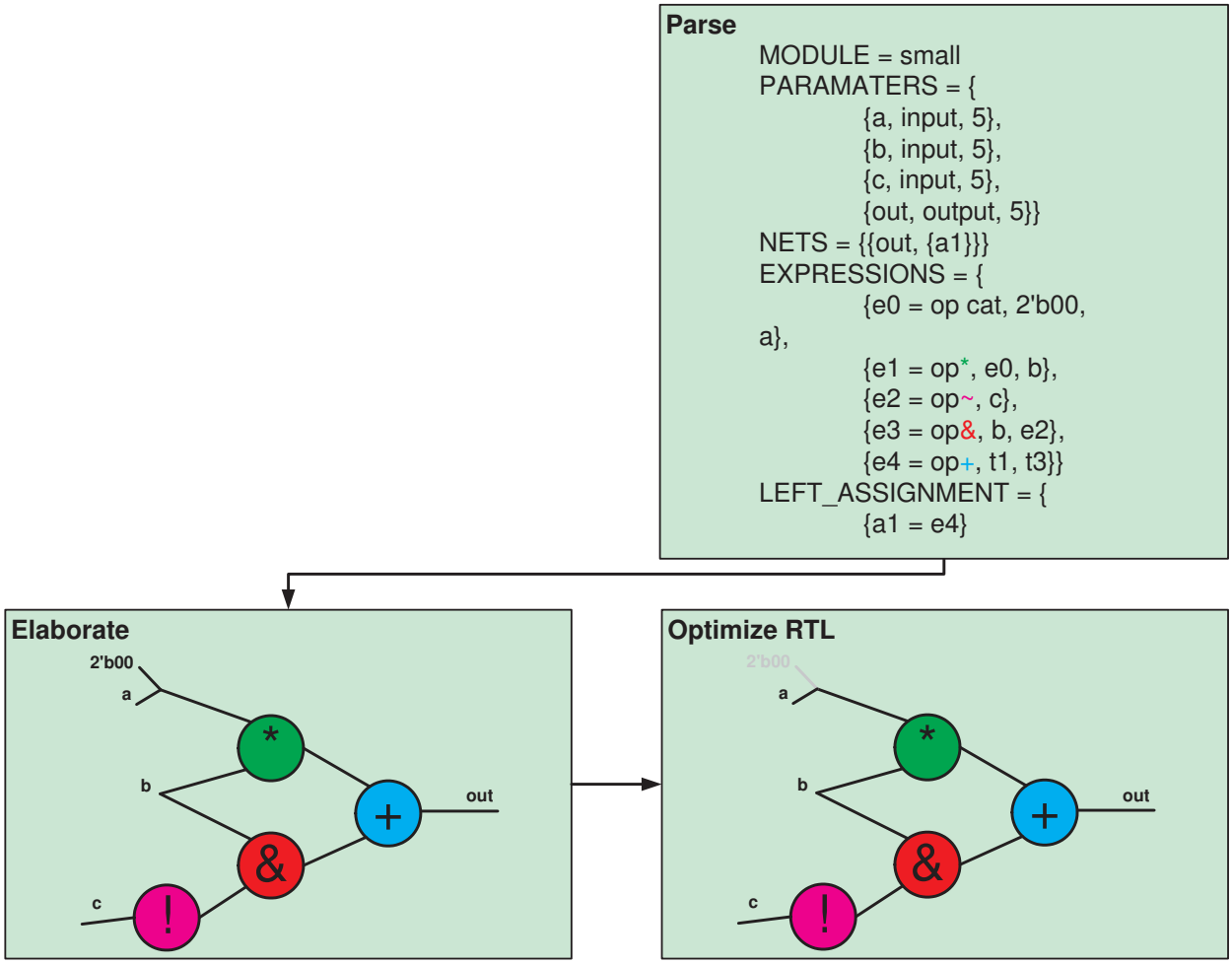


Figure 2.8: Initial HDL design and Elaboration CAD flow Stages.

of this kind of library is the Library Parametrized Modules (LPMs) [LPM93]. LPMs provide designers with simple methods to instantiate complex logic functions, and LPMs also benefit designs since manufacturers optimize LPM instantiations for their specific FPGA architectures.

In the second method, designers follow specified rules for writing HDL descriptions of complex circuits so that the partial mapper can easily identify what circuits the designer intends to use. For example, some synthesis tools specify how to write HDL statements so that flip-flops are identified by the partial mapper [Syn03]. The third

method of identifying functions is an open-ended approach in which the synthesis tool uses matching techniques to extract complex circuits.

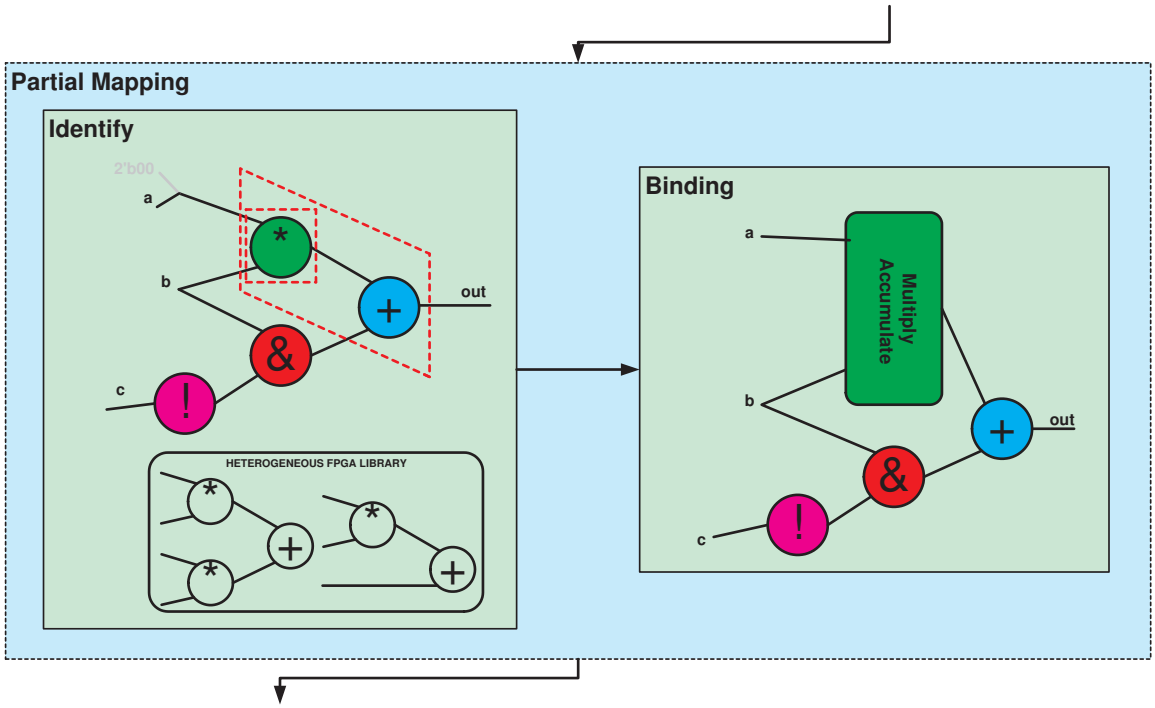


Figure 2.9: The partial mapping stage of front-end synthesis

Figure 2.9 shows the two stages of partial mapping. In the example, the identification stage finds that either the multiplier or the multiplier combined with the adder can be mapped to a hard circuit on the FPGA. In this stage, there is a heterogeneous FPGA library that describes which hard circuits are available on the FPGA and what these hard circuits can be used to implement. The binding stage shows that only the multiplier is mapped to a hard circuit on the FPGA, and this netlist will be passed onto the next stages of the CAD flow.

Once the netlist more closely resembles circuits available on the FPGA, any remaining logic gates must be converted to the FPGA's soft logic fabric. Before mapping logic gates into the soft logic fabric, a technology-independent logic optimizer transforms logic functions within the netlist to improve speed, area, or power implementation

of the overall design [BHSV90, Mic94, BL90, DMNSV87, Bry86]. Next, technology-dependent mapping converts the logic gates into a BLE implementation for the targeted FPGA [CD94, CX00, YW94, CCD+92, FRV91]. After these two steps, a netlist exists in a form that closely matches the circuits available on the FPGA.

For cluster based FPGA architectures, clustering or packing algorithms combine different FPGA elements together. For example, in many FPGAs a LUT and a flip-flop are joined in the tile to form a BLE, and a packing algorithm combines connected LUTs and flip-flops in the netlist into BLEs. Similarly, in some FPGAs, BLEs are grouped together to form clusters. Clustering algorithms are responsible for grouping BLEs together into clusters [SMS02, MBR99].

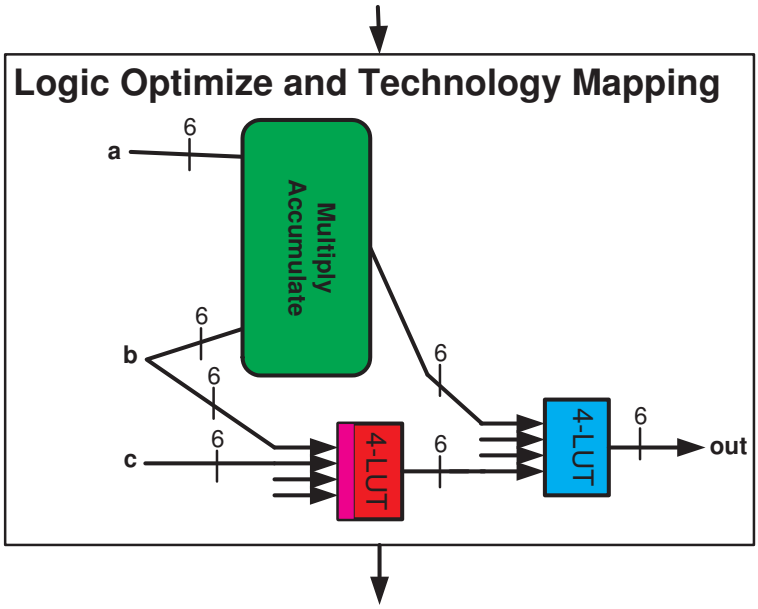


Figure 2.10: Technology mapping stage of the CAD flow

Figure 2.10 shows how our example has been mapped into a hard circuit and 6 two LUTs pairings. In this example, signal *c* goes through an inverter before it is input into an AND gate along with signal *b*. In the figure, these two operations have been combined into one 4-LUT. Our example does not show any technology-independent logic optimization or clustering and packing results.

At this stage, the netlist consists entirely of circuits available on the FPGA. Placement and routing algorithms now map this netlist onto an FPGA. Once clustered and packed, the placer chooses the physical location of both the clusters and hard circuits on the FPGA. These choices are made to minimize overall wire length and maximize speed [SRRJ00, DK85, HK97, KSJA91, SDJ91, ACH+97, KGV83]. Finally, the router chooses wire segments and activates the programmable switches as connection paths between placed elements. The goal of routing is to achieve connectivity while optimizing speed [CTZW94, BRV92, LW95, LB93, BRV90, SBR98]. With the successful completion of placement and routing, the final output of the CAD flow is a configuration bit-stream that specifies the programming of the FPGA.

Figure 2.11 shows the result of our example design going through the placement and routing stages. The final output is a bit-stream that will be loaded onto the FPGA to implement the original design.

The focus in this dissertation with respect to a heterogeneous CAD flow is the front-end synthesis including the elaboration and partial mapping. There are a number of commercial front-end synthesis tools that target heterogeneous FPGAs. Both Altera and Xilinx incorporate front-end synthesis into their FPGA CAD flow tools called Quartus [Alt04c] and ISE [Xil04] respectively. Other popular front-end synthesis tools for FPGAs include Synplify [Syn03], Blast FPGA [Mag05], LeonardoSpectrum [Men01], and Design Compiler FPGA [Syn04].

To our knowledge, there are no available open source front-end synthesis tools, but some HDL parsers do exist. For example, Icarus [Wil07] is a front-end Verilog parser, which does have a simple back-end implementation that targets Xilinx's "xnf" format for describing netlists. This back-end implementation is incomplete and can only map to the soft logic fabric on Xilinx FPGAs. We, however, use the front-end of Icarus as the parsing portion of our own front-end synthesis tool that is presented in Chapter 6.

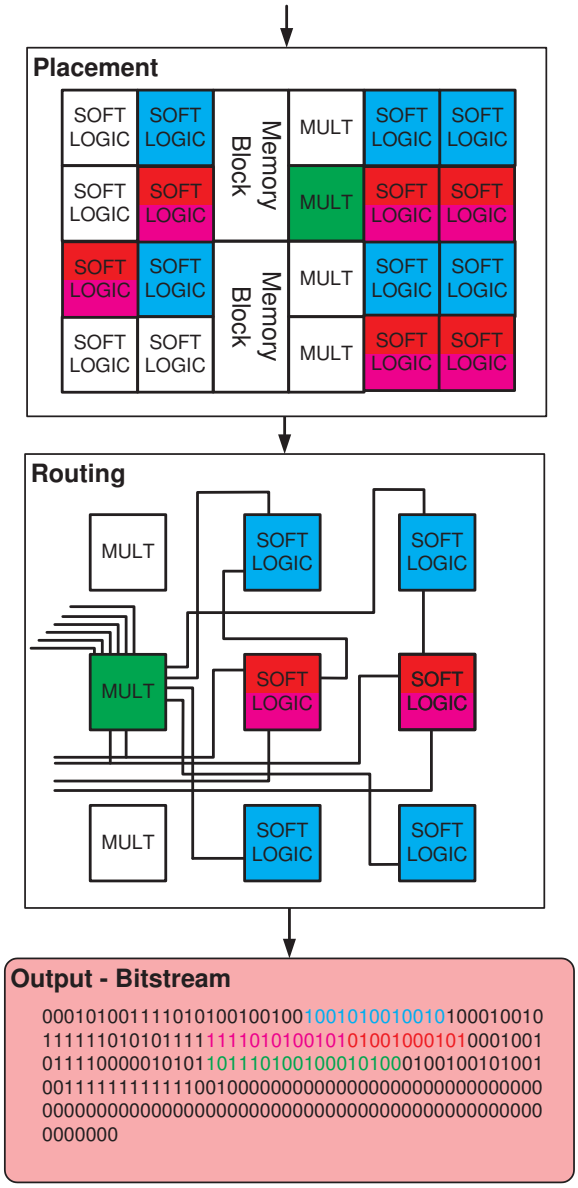


Figure 2.11: The placement and routing stages and the final output bit-stream

## 2.5 Improving Heterogeneous FPGAs

### 2.5.1 Improving the Utilization of Hard Circuits

Now we review previous work that attempts to make heterogeneous FPGAs better by maximizing the utilization of hard circuits. Again, the general theme of this disser-

tation is increasing the utilization of hard circuits to improve the area efficiency of heterogeneous FPGAs. This entails:

1. Improving the flexibility of the hard circuit so it can be used more often

2. Using CAD tools to effectively target hard circuits on the FPGA

In this section, we review previous research in these two areas.

The first approach is to combine different functionality in hard circuits to make them more useful, so they can be used by a larger range of design functionality. This increased flexibility costs more in silicon area, but makes a hard circuit more useful potentially improving the overall area efficiency of an FPGA.

Both Altera's DSP block [Alt03] and Xilinx's Xtreme DSP block [Xil05], previously reviewed in section 2.3.2, are examples of improving the flexibility of a hard circuit so more functions can target these multi-purpose circuits. We called this flexibility functional flexibility.

In each new generation of Xilinx FPGAs, the flexibility has been increased in the hard multiplier. The Virtex-II included a simple 18 by 18 multiplier. The Virtex-4 FPGAs improve on this simple multiplier and introduce the Xtreme DSP, which can flexibly implement a range of multiplication operations and even bus multiplexing. From the Virtex-4 to Virtex-5, more flexibility was introduced to include bitwise logical mode that can be used for logic operations when the multiplier is unused.

In the second approach, there exist CAD techniques to utilize hard circuits in an FPGA to implement functionality not normally intended for these circuits. For example, memories can implement large logic functions by acting as large LUTs [Wil02, CX00]. Hard memories can also implement multiplication operations, and these implementations perform particularly well when the width of the multiplicand is small [Kna99]. The hard multipliers in an FPGA can also implement barrel shifters with significant area and speed savings [Gig04].

We have done work in this area with a CAD technique in which we introduce a method to map multiplexers to unused hard multipliers on an FPGA during front-end synthesis. We show that this technique can improve the utilization of hard multipliers

while not significantly decreasing the speed of designs that already contain multipliers. This work is published in [JR05b], and is not included in this dissertation.

Similar to this work, Morris *et. al.* [MCC05] presented a novel method to use the multipliers on a Stratix II to implement ROM memories. In this work, the address selection of a ROM is mapped to hard multipliers instead of using the soft logic fabric. This work is similar to our multiplexer to multiplier transformation since memory addressing is a multiplexing operation in which stored data is passed through multiplexers, which are controlled by the memory address.

In each of these cases, the approach is to develop CAD tools that take advantage of unused hard circuits and transform design functionality to use these hard circuits.

The previous two techniques attempt to use existing hard circuits on the FPGA so that they are not wasted. Xilinx takes another approach to improve the utilization of their hard circuits so that they are not wasted. Instead of providing one family of FPGAs with a set number of hard circuits, they provide 3 sub-families each with a different number of hard circuits. This allows designers to pick an FPGA with the number of hard circuits that more closely matches what their design uses.

Table 2.2: Virtex 4 sub-families and their intended target designs

| Virtex-4 Family | Target Designs |
|---|---|
| SX | Ultra-high performance signal processing |
| FX | Embedded processing and serial connectivity |
| LX | High-performance logic |

Table 2.2 shows the three sub-families of Virtex-4 FPGAs in column one (in order of decreasing number of available multipliers per chip) and their intended target designs in column two. The SX Virtex-4 FPGAs, for example, would be used for designs that demand many multipliers, while the FX and LX FPGAs are meant for designs that mostly use the soft logic fabric and other resources (including hard processors in the case of the FX).

Both the Virtex-5 and Stratix III follow this trend of diversifying an FPGA family based on supplying a range of FPGA resources to try to match designer's specific needs with what is available on the chip [Xil06, Alt06]. The drawback of this approach is that

these FPGA companies now have to create many FPGAs, which can take significant engineering resources and time. Additionally, these companies need to maintain an even larger inventory of FPGAs to ensure that each sub-family of FPGA is available to their customers.

### 2.5.2 Creating Heterogeneous FPGAs

The work by Smith *et. al.* [Smi06] considers the creation of heterogeneous FPGAs to implement a set of digital designs. Instead of an architect specifying the FPGA and experimenting with that FPGA, their methodology creates an algorithm to select the best circuit structures and number of these structures to include on an FPGA to implement the original set of designs. This algorithm has constraints on the area of the FPGA and tries to optimize the architecture for the fastest operating frequencies.

This flow begins with inputs consisting of a set of digital designs, a set of possible circuit structures that can make up an FPGA, and an area constraint for the maximum size of the FPGA. The final output is a heterogeneous FPGA that can implement the original set of input designs and an estimate of the size and operating frequency of this output FPGA. An additional input includes estimates of the speed and area to implement each part of the input designs from which an optimization engine selects the best circuit structures to implement all parts of each design (assuming that circuit structures will be shared since only one design will be mapped to an FPGA at a time) while satisfying the global area constraint.

The specific set of circuit structures that is used to create the FPGAs include basic soft logic, multipliers, and memories of fixed granularity. The input designs are described in terms of these circuit structures, and each piece can be mapped to a combination of these circuit structures. The design inputs include timing and area estimates, generated by an industrial CAD tool, for each possible implementation of a piece of the design.

The goal of the methodology is to select what circuit structures to include on the FPGA to map all pieces in the designs and to determine the overall speed and area of this FPGA that will implement all the designs. This FPGA must satisfy the global area constraint while being created to maximize operating frequency. They point out that

designing FPGAs using the traditional empirical flow is limited by both the quality of the synthesis algorithms in the CAD flow needed to map a design to the architecture and by the lack of design space exploration. Their work attempts to remove these limitations by using formal optimization methods and estimates of speed and area to decide, during architecture generation, the best mix of circuit structures to be included on an FPGA.

The constraints for the optimizer are expressed as linear constraint equations that can be passed to integer linear problem solvers. These constraints are input to an integer linear program solver as the optimization engine that selects the circuit structures to include on the FPGA. With an additional constraint on the ratio in which circuit structures must appear on the FPGA in relation to one another, the flow can model an industrial FPGA that can be compared to other generated architectures.

To study the tradeoff between area and speed for FPGAs created by their methodology they change the global area constraint. Each architecture is compared to an optimal architecture in terms of speed where this optimal architecture has no constraint on area and is assumed to produce the fastest operating FPGA. They measure the speed and area benefits of hard multipliers and memories for a set of 6 DSP benchmark circuits claiming hard circuits (multipliers and memories) make their designs 1.6 times faster and 7 times smaller compared to an architecture that only has soft logic.

An FPGA, similar to Xilinx's Virtex-II [Xil03], is created by using a constraint on the ratio of the number of hard circuits to the number of soft logic cluster tiles. Their results show that the Virtex-II is composed of a very different set of circuit structures compared to the optimal architectures the flow generates without this additional constraint. This is not surprising since the target set of benchmarks is focused on DSP applications as opposed to a larger group of designs that the Virtex-II needs to serve.

Our architectural work in Chapter 3 and Chapter 4 uses a measurement methodology in which we measure the area efficiency of an FPGA architecture where the ratio of soft logic cluster tiles to hard circuits is varied to find the best ratio. Our method cannot handle two types of hard circuits, and we study architectures containing either hard multipliers or hard crossbars (we do not consider memory). Our goal is to not only find the smallest heterogeneous FPGA, but also to improve the hard circuits themselves by

increasing their utilization. It is possible to combine our ideas with theirs, but the empirical measurement methodology that we use is sufficient for our needs.

## 2.6 Summary

In this chapter, we reviewed the state of the art of homogeneous and heterogeneous FPGAs. This includes a description of a homogeneous FPGA, definitions for different types of heterogeneity in FPGAs, and a review of a typical CAD flow to map designs to them. We also reviewed previous work that attempts to increase the utilization of hard circuits.

In the next chapter, we describe how to include hard circuits as differentiated tiles in the fabric of an experimental FPGA, and we provide details about the benchmarks among other parts of the experimental setup that will be used throughout this work.

# 3 Design of Heterogeneous FPGAs and Measuring their Area Efficiency

> Too many people, when listing all the perils to be found in the search for lost treasure or ancient wisdom, had forgotten to put at the top of the list the man who arrived just before you.
>
> Terry Pratchett

## 3.1 Introduction

The focus of this research is to improve the area efficiency of heterogeneous FPGAs. In this chapter, we introduce a methodology that measures the implementation area of a set of benchmarks mapped to an FPGA with and without hard circuits so we can compare different architectures.

In this measurement methodology, a benchmark is mapped to an architecture described by its FPGA architectural parameters and the resulting area of each FPGA is measured. We introduce a new architectural parameter that describes the economical supply of hard circuits, and a similar economical concept of demand that characterizes a benchmark's use of hard circuits. Both supply and demand of hard circuits are useful when discussing how a hard circuit may benefit target markets and the FPGAs that would serve these markets.

At the end of this chapter, we demonstrate the use of our measurement methodology with experiments that measures the area efficiency of FPGAs with hard multipliers compared to FPGAs without hard multipliers.

## 3.2 Design of Heterogeneous FPGAs

In this section, we define the supply ratio and the demand ratio and discuss how these parameters affect the mapping to experimental FPGAs. We also examine existing commercial FPGAs and their supply ratio, and outline a few principles on how hard circuit tiles are integrated with an FPGA's soft logic and programmable routing.

### Supply Ratio

A heterogeneous tile-based FPGA consists of soft logic cluster tiles as defined in Section 2.2 and hard circuit tiles among other tiles such as I/O pads. These FPGAs are created by designing each type of tile so that they can be abutted to one another to form the heterogeneous array.

We call the ratio of the number of hard circuit tiles to the number of soft logic cluster tiles on an FPGA, the *supply ratio*, $R_S$. The supply ratio is useful in determining the number of hard circuits based on the number of soft logic cluster tiles in the FPGA.

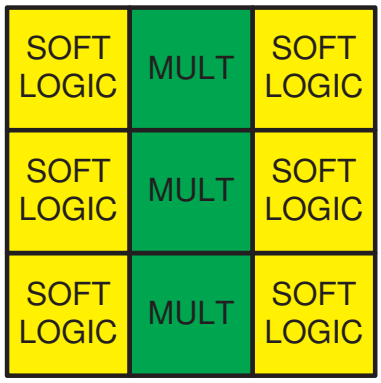


Figure 3.1: An FPGA with a supply ratio equal to 1:2

For example, Figure 3.1 shows a small FPGA with 3 multiplier tiles and 6 soft logic cluster tiles. This FPGA has a supply ratio equal to 1:2, meaning there is 1 multiplier tile for every 2 soft logic cluster tiles.

### Demand Ratio

We can also describe a given benchmark in terms of its demand for hard circuits. We define the demand ratio, $R_D$, as the number of hard circuit tiles to the number of soft logic cluster tiles that a digital benchmark requires when implemented on an FPGA, if all circuits capable of being implemented in the hard circuit tile actually are.

When discussing either ratio (supply or demand) we will say that a certain architecture or benchmark has a greater supply or demand compared to some other architecture or benchmark. This means that there are more multipliers available on the architecture with greater supply or desired by the circuit with greater demand. For example, an architecture with a supply ratio of 1:8 has a greater supply of multipliers compared to an architecture with a supply ratio of 1:10.

Throughout this dissertation, we represent the supply or demand ratio in terms of one hard circuit tile to a fractional number of soft logic cluster tiles. For example, we would represent a supply ratio of 2:3 as 1:1.5 in this work. For cases when there are no hard circuits supplied or demanded we define these ratios as one to infinity, which in symbol form is 1:$\infty$. This representation is not perfect, and there are several ways to represent these ratios such as fractional or decimal form. We find that representing the ratio with the colon is the most intuitive way to display either ratio.

A benchmark suite, which is a collection of benchmarks, can be described in terms of its average demand ratio. The average demand ratio is calculated by arithmetically averaging the demand ratios of each benchmark in the suite. The average demand ratio is calculated by using the decimal equivalent of the demand ratio for each benchmark and arithmetically averaging all of these values. For those benchmarks which demand no multipliers, we convert their demand ratio of 1:$\infty$ to zero for this average demand ratio calculation. This method is used any time we calculate the average demand ratio.

### Mapping to floating FPGA sizes

The supply ratio is used when following the common practice in FPGA architecture research [RFCL89]; to allow the FPGA size to change to accommodate a mapped benchmark. As the FPGA size changes all the architectural parameters are maintained,

and when an FPGA with hard circuits is changing in size, the supply ratio is kept constant. If we wanted to increase the size of the FPGA in Figure 3.1 to accommodate a larger benchmark with 4 multipliers, then the FPGA increases in size to 4 hard multiplier tiles and 8 soft logic cluster tiles to maintain the supply ratio of 1:2.

We use a ratio instead of a fixed constant of supplied hard circuits because using the supply ratio and fine grained fitting of benchmarks to FPGAs allows us to create an FPGA for each benchmark that closely matches the needs of the benchmark while providing a fractionally equal amount of hard circuit resources for each of these mappings. This avoids large quantization errors with fixed size FPGAs that have a set number of hard circuits since this may hide the affects of a CAD technique or an architecture improvement, which we are measuring.

### Commercial FPGAs Supply Ratio

We now use supply ratio to look at existing commercial FPGAs and trends within these FPGAs with respect to hard multipliers included in their fabric.

We measure the supply ratio for existing commercial FPGAs; Table 3.1 provides the average hard multiplier supply ratios for several FPGA families where each ratio is calculated by geometrically averaging the supply ratios for each FPGA in the family. For all the FPGA families, other than the Virtex-4 LX and Virtex-5 LX family, the supply ratio remains relatively constant over each member of the family. This is shown in the third and fourth column of Table 3.1 where we show the upper and lower limit of the supply ratio. These limits show the biggest and smallest supply ratios in the family.

For each FPGA, the supply ratios are normalized to a cluster size of 10 BLEs (each BLE consists of a 4-LUT and a flip-flop) and a size 18x18 multiplier. Normalizing the Stratix I [Alt03], VirtexII [Xil03], Cyclone II [Alt07a], Cyclone III [Alt07b], Virtex-4 [Xil05], Spartan-3A [Xil07], and Lattice [Lat07a, Lat07b] FPGAs are simple since each of these FPGAs is built using 4-LUT BLEs, and we divide the total number of BLEs in each FPGA by ten. All the FPGAs in Table 3.1 include 18x18 multipliers except the Virtex-5 [Xil06]. In the case of the Virtex-5, the extreme DSP block is a 25x18 hard multiplier, and we do not normalize to an 18x18 when calculating the

Table 3.1: Average Multiplier Supply Ratios for Industrial FPGAs

| FPGA FPGA | Average Supply Ratio | Upper limit of supply ratios | Lower limit of supply ratios |
|---|---|---|---|
| Stratix I | 1:66 | 1:79 | 1:44 |
| Stratix II | 1:45 | 1:53 | 1:32 |
| Stratix III SL | 1:42 | 1:58 | 1:21 |
| Stratix III SE | 1:12 | 1:12 | 1:12 |
| Cyclone II | 1:53 | 1:95 | 1:35 |
| Cyclone III | 1:33 | 1:44 | 1:22 |
| VirtexII | 1:23 | 1:26 | 1:23 |
| Virtex-4 SX | 1:15 | 1:18 | 1:11 |
| Virtex-4 FX | 1:60 | 1:87 | 1:38 |
| Virtex-4 LX | 1:104 | 1:208 | 1:43 |
| Virtex-5 SX | 1:19 | 1:20 | 1:16 |
| Virtex-5 LX | 1:166 | 1:194 | 1:108 |
| Spartan-3A DSP | 1:44 | 1:45 | 1:42 |
| Lattice ECP | 1:59 | 1:102 | 1:38 |
| Lattice ECP2 | 1:66 | 1:100 | 1:50 |

supply ratio.

The Stratix II [Alt04d], Stratix III [Alt06], and the Virtex-5 do not contain 4-LUTs and to normalize their supply ratio takes one additional step. For the Stratix II FPGAs, we convert to Altera's latest BLE, which is called an Adaptive Logic Module (ALM) and can implement a 6-LUT among other combinations of 4-LUTs and 5-LUTs, to 4-LUT BLEs by multiplying the number of ALMs in the FPGA by 2.5 [Alt04e] a fact that Altera gives. Similarly, for the Virtex-5 FPGAs that use 6-LUT BLEs, we convert each 6-LUT into equivalent number of 4-LUTs by multiplying the number of 6-LUTs in these FPGAs by 1.8 [Xil06]. Once these architectures are described in terms of 4-LUTs, we divide the total number of equivalent 4-LUTs by ten.

One trend to notice in Table 3.1 is that newer generations of FPGAs tend to have an increased supply of hard multipliers. For example, the Stratix II includes 50% more multipliers compared to the previous generation, Stratix I FPGAs. It is hard to predict whether this trend will continue especially since both Altera and Xilinx are now

breaking their FPGA families into sub-families, each of which has a different multiplier supply ratio.

### 3.2.1 Routing Architecture for Hard Circuits

The supply ratio describes the number of hard circuit tiles and soft logic cluster tiles on a heterogeneous FPGA, but it does not describe how to build the programmable routing to connect to them. In this section, we describe how we architect an FPGA with hard circuit tiles based on both our intuition and observations made in industrial FPGA architectures.

A homogeneous soft logic fabric FPGA with a given soft logic cluster size (N) and LUT size (K) in each tile will require a specific number of tracks per channel (W) to route most benchmark circuits. The parameters N, K, and the routing architecture parameters can be used to calculate the number of pins that could be routed from the BLEs in a cluster to the programmable routing, which we call the *pin demand*. Pin demand includes both the input pins entering and output pins emanating from the cluster.

The number of tracks needed in an architecture (W) is a function of pin demand and the other routing architecture parameters described in section 2.2. This number is usually determined experimentally by an FPGA architect [LAB+05].

To include a hard circuit with higher pin demand than the soft logic tile means an architecture needs more tracks per channel. If the hard circuit is implemented on the FPGA in one logical tile then the number of tracks in the channel (W) would need to increase. The reason for this can be explained using an equation that determines the width of a channel for designs first described by El Gamal [Gam81] and later applied to the channel width and routability in FPGA's by Brown *et. al.* [BFRV92]. The form of the equation is:

$$W = \frac{\lambda \bar{R}}{2} \tag{3.1}$$

In this equation, $\lambda$ is the total number of connections to a tile, which we call pin demand, and $\bar{R}$ is the average connection length between tiles for the set of designs mapped to this FPGA. To include a hard circuit that has more input and output

connections than $\lambda$ for the other tiles on an FPGA means that W will need to be increased if we want designs mapped to this FPGA to be routable.

Alternatively, the hard circuit could be stretched over multiple logical tiles distributing the number of incoming and outgoing connections to keep the total per tile pin demand less than $\lambda$. We choose this solution, and any hard circuits implemented on the FPGA will have a per tile pin demand that approximately matches the pin demand of the soft logic cluster tiles. For example, in an FPGA with a soft logic cluster tile with pin demand equal to 32 (22 input pins plus 10 output pins), if we include a hard circuit that has a total pin demand of 62 then the hard circuit will be implemented over two tiles, and each tile will have a pin demand of approximately 31.

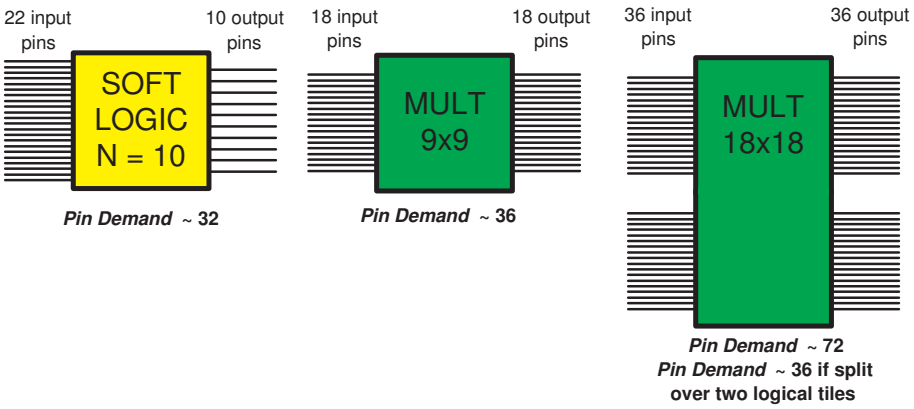


Figure 3.2: Pin distribution for hard circuit tiles

Figure 3.2 shows the pin demand of a soft logic cluster tile, a 9x9 multiplier, and an 18x18 multiplier. The pin demand of the 9x9 multiplier and the soft logic cluster tile is roughly the same, meaning that they could share the same global routing architecture without an increase in track count. The 18x18 multiplier, however, needs to be implemented in the equivalent of 2 soft logic cluster tiles so that the pin demand for each half of the multiplier is roughly the same as the pin demand of the soft logic cluster tile.

This does not mean that the 18x18 multiplier will use the equivalent area of two soft logic cluster tiles. Instead, the multiplier when stretched over two tiles will have

different area dimensions but the same pin demand. For example, tiles in Stratix FPGAs [Alt03] have equal vertical dimensions and have roughly the same pin demand, but the DSP blocks in this architecture have a larger horizontal dimension compared to the soft logic cluster tiles. Stretching a tile is in terms of equivalent logical tiles and not area equivalent tiles.

## 3.3 Measurement Methodology

To determine the area effectiveness of a hard circuit we employ an empirical approach: measure the area consumed by a set of benchmarks after mapping them into FPGAs with and without hard circuits. The following sections describe the mapping of benchmarks into a heterogeneous FPGA and then the determination of its area. Subsequent sections describe the set of benchmarks and the architectural parameters of the soft logic fabric that we use in this chapter and the rest of this dissertation.

### 3.3.1 Benchmark Circuit Mapping Flow

Figure 3.3 gives the experimental flow we use to measure the area for implementing benchmarks on an FPGA architecture. The first step is to map a benchmark design to the different FPGA tiles available on the FPGA, and then, calculate the area of the FPGA based on the tiles used. We discuss the calculation of the area in the next section.

A benchmark is modeled as requiring a number of soft logic cluster tiles and multiplier tiles. A mapping step is required because the FPGA's multiplier supply ratio (defined in Section 3.2) will rarely be the same as the benchmark's multiplier demand ratio. Depending on the supply ratio, the appropriate size of FPGA is determined.

There are two ways to approach the mapping of hard circuits in a benchmark to an FPGA. In the first algorithm, we initially set the number of hard circuits on the FPGA equal to the number present in the benchmark. We then size the FPGA by finding the amount of soft logic cluster tiles needed. An alternative is to start with one hard circuit and then increase the number of hard circuits until the benchmark fits

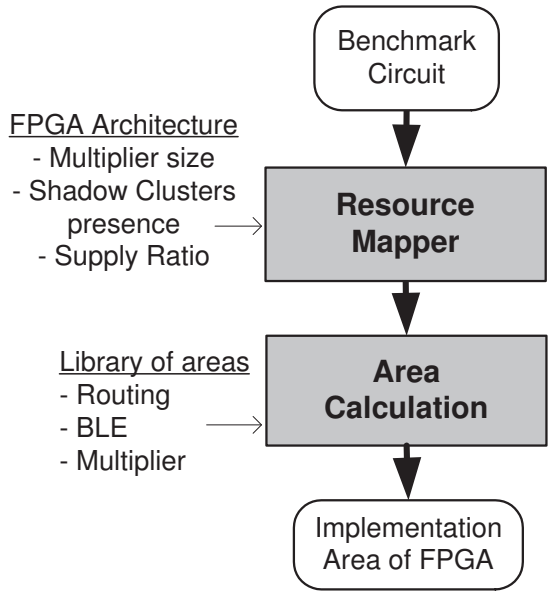


Figure 3.3: The measurement methodology

the FPGA. In this approach, it is necessary to map hard circuits in the design to hard circuits or convert them to soft logic implementations. In both cases, the supply ratio is maintained as the FPGA is sized.

The main difference between the two algorithms is the second algorithm might result in a slower circuit, because if there are less hard circuits on the FPGA compared to what the benchmark would use, then hard circuits in the benchmark that are converted to soft logic might become critical paths and decrease the maximum operating frequency of the benchmark. In this dissertation, we use both algorithms choosing the first if we believe the hard circuit is used both for its area and speed benefits.

Algorithm 1 and Algorithm 2 show the steps taken for each algorithm.

Each algorithm is built with a simple incremental search method to find the number of tiles needed to map a benchmark. These algorithms can be accelerated by using better search methods instead of simply incrementing the number of hard circuits by one, and we actually use a binary search method for this purpose.

Table 3.2 summarizes the differences between the two choices of mapping algorithms. Column two describes the initial conditions and column three shows if the algorithm

```
Input: Design (#hard circuits, #soft logic cluster tiles), Supply Ratio, Size and
       Architecture of Hard Circuit
Output: Number of each type of tiles in FPGA
numHardCircuits = numHardCircuitsIn(Design);
notMapped = TRUE;
while notMapped do
    softLogicAvailableOnFPGA = numHardCircuits * supplyRatio;
    totalSoftLogicNeededByDesign = (soft logic in design);
    if softLogicAvailableOnFPGA > totalSoftLogicNeededByDesign then
        notMapped = FALSE;
    end
    else
        numHardCircuits++;
    end
end
```

**Algorithm 1**: Our first algorithm for mapping a design to an FPGA

will possibly convert some of the hard circuits to soft logic implementations.

Once the number of tiles of each type is known and the area of each tile is known, the total area for each benchmark can be calculated.

### 3.3.2 Transistor and Cell Area Estimation of Tiles

The area calculations require the relative area between regular multiplier tiles and regular soft logic cluster tiles. These sizes are determined in a 90nm CMOS process that was available to us [STM05, Mic07]. Two methods to estimate size were used as follows:

1. For all the FPGA-specific components (programmable routing in the multiplier tile and soft logic cluster tiles as well as the LUT-based logic) we carefully sized a transistor-level circuit using an automated transistor sizing approach. This method is described in detail in Appendix A.

2. The multiplier size was estimated by mapping multiplier designs through a standard cell design flow. The area of the multiplier portion of the multiplier tile is determined by using a cell-based approach with commercial standard cells in

**Input**: Design (#hard circuits, #soft logic cluster tiles), Supply Ratio, Size and Architecture of Hard Circuit
**Output**: Number of each type of tile in FPGA
numHardCircuits = 1;
notMapped = TRUE;
**while** *notMapped* **do**
  softLogicAvailableOnFPGA = numHardCircuits * supplyRatio;
  **A** - Map hard circuits in the benchmark to available hard circuits on the FPGA;
  totalSoftLogicNeededByDesign = (soft logic in design) + (design's circuits not mapped to hard circuits);
  **if** *softLogicAvailableOnFPGA* > *totalSoftLogicNeededByDesign* **then**
    notMapped = FALSE;
  **end**
  **else**
    numHardCircuits++;
  **end**
**end**

**Algorithm 2**: Another algorithm for mapping a design to an FPGA

Table 3.2: Summary of the two mapping algorithm choices.

| | Initial Conditions | Will hard circuits be converted to a soft implementation |
|---|---|---|
| Algorithm 1 | Each hard circuit in benchmark is mapped to a hard circuit on FPGA | No |
| Algorithm 2 | Start with one hard circuit | Yes |

the 90nm process [STM05]. A multiplier is described in Verilog and synthesized using Synopsys Design Compiler V-2004.06-SP1.

The area of the multiplier tile includes the cell-based multiplier and the programmable routing area. The programmable routing is built to match the input and output pin

demand of the multiplier. For example, a 9x9 multiplier has 18 input pins, so the input programmable routing area is determined for 18 inputs. This area is obtained using the results generated by the same sizing tool described in Appendix A.

## 3.4 Benchmarks

In this chapter along with each of the following chapters of this dissertation, one common set of benchmarks is used for many of our experiments, and we review the makeup of these benchmarks. Additional details are provided in Appendix B including the number and size of multipliers, the number of I/O pins, and the number of memory bits in each benchmark.

We have collected 32 *real* benchmarks applications coded in the Verilog HDL [Ope93] and gathered from various sources including: The Opencores organization [ope07], SCU-RTL [scu98], Texas-97 [tex97], and the Benchmarks for Placement 2001 [Pla01].

Some of our benchmarks come from applications developed locally at the University of Toronto that we converted from VHDL to Verilog. These benchmarks include, Raytrace [FR03] (separated into 4 benchmarks called rayTraceA, rayTraceB, rayTraceC, and rayTraceD), Stereo Vision [DRM03] (separated into 4 benchmarks called stereoVisionA, stereoVisionB, stereoVisionC, and stereoVisionD), and Molecular Dynamic system [AKE+04]. Both the Raytrace and Stereo Vision applications were implemented on an experimental FPGA prototyping board that used 4 separate FPGAs, and this is the reason each of these applications is broken into four benchmarks.

Table 3.3 shows the number of Stratix BLEs (mapped by Quartus 5.0 to the Stratix I FPGA) and the number and size range of multipliers in each benchmark. In some cases, the benchmark name has the suffix "_no_mem"; this means that the memory was removed from the benchmark so that we can use these benchmarks in academic CAD flows that do not support memories.

Table 3.3: Benchmarks Details

| Benchmark | BLEs | Multipliers | Size Range Multipliers |
|---|---|---|---|
| fft | 2374 | 32 | 8x8 |
| iirA | 289 | 5 | 8x8 to 10x10 |
| iirB | 297 | 5 | 8x16 to 16x16 |
| firA | 84 | 4 | 8x8 |
| firB | 1598 | 25 | 16x16 |
| firC | 998 | 17 | 8x8 |
| diffeqA | 221 | 5 | 32x32 |
| diffeqB | 512 | 5 | 32x32 |
| stereoVisionA | 17765 | 152 | 8x8 |
| stereoVisionB | 35554 | 528 | 4x7 to 16x9 |
| stereoVisionB_no_mem | 34279 | 528 | 4x7 to 16x9 |
| rayTraceA | 2622 | 18 | 7x8 to 16x16 |
| rayTraceA_no_mem | 2118 | 18 | 7x8 to 16x16 |
| rayTraceB | 25056 | 31 | 16x16 to 29x33 |
| rayTraceB_no_mem | 21557 | 31 | 16x16 to 29x33 |
| oc45_cpu | 2191 | 1 | 16x16 |
| reedSolDecoderA | 1151 | 13 | 4x4 |
| reedSolDecoderB | 1799 | 9 | 8x8 |
| moleculeDynamics | 10542 | 19 | 38x38 to 43x50 |
| cordicA | 591 | 0 | - |
| cordicB | 2830 | 0 | - |
| MACA | 2864 | 0 | - |
| MACB | 9828 | 0 | - |
| crc33_d264 | 102 | 0 | - |
| desArea | 1481 | 0 | - |
| desPerf | 4592 | 0 | - |
| stereoVisionC | 12433 | 0 | - |
| stereoVisionC_no_mem | 7281 | 0 | - |
| stereoVisionD | 170 | 0 | - |
| rayTraceC | 766 | 0 | - |
| rayTraceC_no_mem | 546 | 0 | - |
| rayTraceD | 1807 | 0 | - |

## 3.5 Experimental FPGA

Each of the FPGAs used in this work must be specified in terms of their soft fabric architectural parameters. We will define experimental FPGAs and use them in this chapter, Chapter 4, and Chapter 5 to evaluate our new architectural concepts.

The architecture of the soft logic fabric includes many parameters, including the number of BLEs per cluster (N), the input size of a LUT (K) in a BLE, the number of routing tracks per channel (W), the input connectivity to the BLEs in a soft logic cluster ($F_{Cin}$), the output connectivity from the BLEs to the routing tracks ($F_{Cout}$), logical wire length in terms of the number of soft logic clusters spanned (L), and the switch block flexibility connecting routing tracks with each other ($F_s$).

For the soft logic fabric we use in this dissertation, we select two sets of parameters chosen to be close to the typical parameters of modern FPGAs. The parameters we use in this dissertation are given in Table 3.4.

Table 3.4: FPGA Architectural Parameters for Two Experimental Architectures

| Parameter | W | N | K | $F_{cin}$ | $F_{cout}$ | $F_s$ | L |
|---|---|---|---|---|---|---|---|
| Architecture 1 | 180 | 10 | 4 | 0.17 | 0.1 | 3 | 4 |
| Architecture 2 | 188 | 8 | 6 | 0.17 | 0.1 | 3 | 4 |

Architecture 1 (from Table 3.4) is used in the majority of our experiments, but we provide Architecture 2 because later in this dissertation we will measure how area-efficiently FPGAs implement crossbars in the soft logic fabric. Architecture 2 implements crossbars more efficiently than Architecture 1 because Architecture 2 has 6-LUTs that implement crossbars of size greater than 6 inputs more area efficiently than 4-LUTs. Also, Architecture 2 is similar to modern FPGAs such as Stratix II [Alt04d], Stratix III [Alt06], and Virtex-5 [Xil06] and their 6-LUT based BLEs.

In Table 3.4, the track count (W) is chosen by averaging the horizontal and vertical channel widths in Stratix I and II FPGAs. The cluster size (N) and LUT size (K) were selected for Architecture 1 and 2 to match the values in Stratix I [Alt03] and Stratix II [Alt04d] clusters respectively. These parameters fall into the range provided in [AR00] where their work shows that the number of BLEs in a cluster should be

between 3 to 10 and the size of the LUT should be between 4 to 6 to build FPGAs with low area-delay products.

The routing parameters are selected based on the ranges given in Lemieux's work in [LL01] and Ahmed's work in [AR00]. To determine these ranges they run experiments on routing parameters to find values that result in the lowest area-delay for a set of benchmarks mapped to these FPGAs. For example, parameter $F_C$ is studied in Lemieux's work and for a cluster size of 6 they state that a reasonable $F_C$ is equal to 0.366. The routing architecture used is direct-drive [LL04, Ye06] as discussed in Chapter 2 Section 2.2.

Finally, the experimental FPGAs in this dissertation use a wire segment length of 4 meaning each wire segment travels either in a horizontal or vertical direction for a distance of 4 clusters (see [BRM99] for details on different segment lengths and the impact on FPGAs).

## 3.6 Measuring the Benefit of Hard Multipliers

In our first experiment, we want to measure the area benefit of including a hard multiplier on an FPGA expecting that since all the industrial FPGAs include hard multipliers that there is an area benefit. We use the above measurement methodology, benchmarks, and experimental FPGAs to measure the area effect of including a hard multiplier on an FPGA compared to an FPGA with purely soft logic fabric.

The benchmark suites are mapped to an FPGA with hard multipliers and an FPGA without hard multipliers to compare their area. The benchmarks after logic synthesis need only be represented as the number and size of multipliers and the number of BLEs (which determines the number of soft logic cluster tiles) in each benchmark. To obtain the number of BLEs, 27 of our real benchmarks without memories (given in Figure 3.3) are passed through Altera's Quartus tool [Alt04c] Version 5.0. We determine the number of multipliers in each circuit by manually counting and identifying multipliers.

For this experiment, Algorithm 1 (summarized in Table 3.2) is used when mapping the hard circuits to the FPGA algorithm 1. Each benchmark is mapped to an FPGA

with the soft logic parameters of Architecture 1 (shown in Table 3.4) and the FPGA with hard multipliers will have a hard multiplier supply ratio of 1:8. Here, we are matching the supply ratio to the average demand ratio of our benchmark suite. This choice may show the hard multipliers in its best light. Each hard multiplier included on an FPGA is a simple 18x18 hard multiplier (meaning it is not functionally flexible and cannot be broken up to implement smaller multipliers). The area for soft logic cluster tiles and multiplier tiles is estimated as described in Section 3.3.2.

Table 3.5 shows the results for all 27 benchmarks with an average demand ratio of 1:8 mapped to both an FPGA with and without hard 18x18 multipliers. The table first gives the benchmark name and the calculated demand ratio based on the number of multipliers and soft logic cluster tiles of size 10 from Table 3.3. The next column gives the area of the FPGA without hard multipliers required to implement the benchmark, assuming, as discussed above, that the FPGA can grow to accommodate the size of the benchmark. The next column gives the area required for an FPGA with hard multipliers, and the final column gives the ratio between the "with" and "without" hard multipliers area.

When a benchmark does not use any multipliers there is an area penalty, and this penalty is related both to the size of the hard multiplier and the supply ratio. For example, desArea needs no multipliers and when mapped to the FPGA with hard multipliers is 1.5 times bigger than when it is mapped to the soft logic FPGA. Conversely, those benchmarks that contain multipliers have a reduced area implementation when mapped to FPGAs with hard multipliers. For example, moleculeDynamics is 1.2 times smaller when mapped to the FPGA with hard multipliers than an FPGA without.

Overall, the benchmark suite is implemented 1.024 times smaller on an FPGA with hard multipliers compared to the FPGA without hard multipliers. This result is obtained by geometrically averaging the area-efficiency metric for each benchmark in the suite.

This result is a small area benefit compared to what the hard multiplier could potentially improve FPGA area efficiency. The potential benefit hard multipliers could have is based on the area improvement multipliers provide and the multipliers tile area compared to the soft logic cluster tile. An 18x18 multiplier is implemented in 47 soft

Table 3.5: Results for individual benchmarks on FPGAs without hard multipliers and with hard multipliers (supply ratio equal to 1:8)

| Benchmark Name | Demand Ratio (N = 10) | FPGA Area Without Hard Multipliers ($10^4$ $um^2$) | FPGA Area with Hard Multipliers ($10^4$ $um^2$) | Area Efficiency Metric |
|---|---|---|---|---|
| fft | 1:7.4 | 482 | 464 | 1.038 |
| iirA | 1:5.8 | 75 | 72 | 1.037 |
| iirB | 1:5.9 | 160 | 73 | 2.205 |
| firA | 1:2.1 | 45 | 58 | 0.780 |
| firB 25 | 1:6.4 | 807 | 362 | 2.224 |
| firC 17 | 1.5.9 | 237 | 246 | 0.963 |
| diffeqA | 1:1.1 | 502 | 290 | 1.732 |
| diffeqB | 1:2.6 | 523 | 290 | 1.804 |
| stereoVisionA | 1:11.7 | 2762 | 2204 | 1.253 |
| stereoVisionB_no_mem | 1:6.5 | 11478 | 7657 | 1.499 |
| rayTraceA_no_mem | 1:11.8 | 489 | 261 | 1.876 |
| rayTraceB_no_mem | 1:13.1 | 4635 | 2393 | 1.937 |
| oc45_cpu | 1:219 | 184 | 203 | 0.924 |
| reedSolDecoderA | 1:8.9 | 114 | 188 | 0.608 |
| reedSolDecoderB | 1:20 | 218 | 174 | 1.234 |
| moleculeDynamics | 1:3.5 | 5460 | 4409 | 1.239 |
| cordicA | 1:$\infty$ | 44 | 58 | 0.755 |
| cordicB | 1:$\infty$ | 207 | 261 | 0.791 |
| MACA | 1:$\infty$ | 210 | 261 | 0.802 |
| MACB | 1:$\infty$ | 718 | 899 | 0.798 |
| crc33_d264 | 1:$\infty$ | 8 | 15 | 0.554 |
| desArea | 1:$\infty$ | 109 | 145 | 0.750 |
| desPerf | 1:$\infty$ | 336 | 421 | 0.798 |
| stereoVisionC_no_mem | 1:$\infty$ | 532 | 667 | 0.798 |
| stereoVisionD | 1:$\infty$ | 12 | 29 | 0.428 |
| rayTraceC_no_mem | 1:$\infty$ | 56 | 73 | 0.775 |
| rayTraceD | 1:$\infty$ | 130 | 174 | 0.759 |
| Average | 1:8 | | | 1.024 |

logic cluster tiles with 10 4-LUTs, and the hard 18x18 multiplier is about 4 times the size of this soft logic cluster tile. Therefore, the hard 18x18 multipliers could improve the area efficiency of the FPGA by 11.75 times.

However, the conditions for this improvement are that the benchmarks only contain 18x18 multipliers. The small benefit of 1.024 times that we get for our real benchmark collection is due to several factors such as the presence of soft logic not used for multiplication, the mismatch of design multiplier size and the hard multipliers available on the FPGA, and the nature of waste due to a fixed supply ratio. The area benefit is also dependent on both the average demand ratio and distribution of demand in the benchmark suite and the relationship between demand and supply on the FPGA mapped to. We will see more of this economic interaction in the following experiments and the next chapter.

### 3.6.1 Measuring the Benefit of Different Hard Multiplier Architectures

In this experiment, we explore how the choice of a hard circuit's architecture affect the area-efficiency results. Specifically, we measure the area-efficiency effect of a hard multiplier architecture where the architecture choices are the bit-width and functional flexibility of the hard multiplier.

Earlier, we discussed how the functional flexibility of a hard circuit means that the hard circuit can be utilized more often by targeting designs reducing wasted tiles. Similarly, the bit-width of a hard circuit affects utilization since designs will use different sized hard circuits and may under utilize a hard circuit resulting in inefficient use of a tile. Here, we study how both flexibility of a hard multiplier and the bit-widths it can implement affects the area efficiency of an FPGA that includes these hard circuits.

Table 3.6: Hard Multiplier Architectures

| Multiplier Name | Maximum Input Bit-width | Implements these Multipliers | Relative Size to 9x9 |
|---|---|---|---|
| 9x9 | 9 | one 9x9 | 1.0 |
| 18x18 | 18 | one 18x18 | 1.73 |
| Dynamic-18x18 | 18 | one 18x18 or two 9x9 | 2.24 |
| Dynamic-36x36 | 36 | one 36x36, two 18x18, or four 9x9 | 4.08 |

In this experiment, we will map our collection of benchmarks to FPGAs with four different types of multiplier architectures listed in Table 3.6. Column 1 of Table 3.6 shows the name of the hard multiplier, column 2 shows the maximum input bit-width of one of the operands, column 3 shows the multipliers that can be implemented in the hard circuit through its internal configurability, and column 4 shows the relative area difference between a single tile of a 9x9 multiplier compared to a single tile of the other 3 hard multipliers. The hard multipliers, Dynamic-18x18 and Dynamic-36x36, can only implement one size of multiplication per hard circuit determined at programming time; for example, one Dynamic-36x36 cannot implement one 18x18 and two 9x9 multipliers.

The difference between the 18x18 hard multiplier and the Dynamic-18x18 multiplier includes both the flexibility that the Dynamic-18x18 hard multiplier has and the physical size of the hard circuit. Both Dynamic-18x18 and Dynamic-36x36 can be configured to implement different size multipliers more efficiently, but this flexibility does cost area. The flexibility means that the Dynamic-18x18 hard multiplier is composed of 9x9 multipliers and programmable switches to select which size of multipliers the hard multiplier will implement, and this flexibility is less area efficient than implementing a 18x18 hard multiplier that is not configurable. The Dynamic-18x18 hard multiplier tile is 2.24 times larger than the 9x9 hard multiplier tile as opposed to the 18x18 hard multiplier tile, which is only 1.73 times bigger than the 9x9 hard multiplier tile.

For our experiment, each FPGA with hard multipliers has a supply ratio of 1:8. We divide the area of a benchmark mapped to a purely soft FPGA by the area of the same benchmark mapped to an experimental FPGA containing hard multipliers to get an area-efficiency metric. When this ratio is greater than one it shows that the experimental FPGA is smaller. We summarize the results for a benchmark suite using a geometric average of the area-efficiency metrics for all the benchmarks in our collection and show the results for the benchmark suite mapped to each experimental architecture.

Table 3.7 shows the results for each benchmark mapped to each of the experimental FPGAs compared to a purely soft FPGA. Column 1 shows the benchmark, and columns 2, 3, 4, and 5 shows the ratio of the area for a benchmark mapped to a soft FPGA divided by the area of the same benchmark mapped to architectures with the 9x9,

Table 3.7: Results for area efficiency of hard multiplier architectures

| Benchmark Name | 9x9 Area-efficiency metric | 18x18 Area-efficiency metric | Dynamic-18x18 Area-efficiency metric | Dynamic-36x36 Area-efficiency metric |
|---|---|---|---|---|
| fft | 2.175 | 1.038 | 2.051 | 1.976 |
| iirA | 0.987 | 1.037 | 0.931 | 0.897 |
| iirB | 1.155 | 2.205 | 1.089 | 1.049 |
| firA | 1.634 | 0.780 | 1.541 | 1.485 |
| firB | 1.165 | 2.225 | 1.099 | 1.059 |
| firC | 2.016 | 0.962 | 1.901 | 1.832 |
| diffeqA | 0.907 | 1.732 | 0.855 | 1.648 |
| diffeqB | 0.945 | 1.805 | 0.891 | 1.717 |
| stereoVisionA | 1.789 | 1.253 | 1.687 | 1.625 |
| stereoVisionB_no_mem | 1.291 | 1.499 | 1.218 | 1.173 |
| rayTraceA_no_mem | 1.572 | 1.876 | 1.483 | 1.429 |
| rayTraceB_no_mem | 1.014 | 1.937 | 0.957 | 1.748 |
| oc45_cpu | 0.934 | 0.924 | 0.913 | 0.879 |
| reedSolDecoderA | 1.104 | 0.608 | 1.041 | 1.003 |
| reedSolDecoderB | 1.371 | 1.254 | 1.293 | 1.246 |
| moleculeDynamics | 0.649 | 1.239 | 0.612 | 1.179 |
| cordicA | 0.791 | 0.755 | 0.746 | 0.719 |
| cordicB | 0.829 | 0.791 | 0.782 | 0.729 |
| MACA | 0.841 | 0.803 | 0.793 | 0.764 |
| MACB | 0.843 | 0.798 | 0.795 | 0.742 |
| crc33_d264 | 0.580 | 0.554 | 0.547 | 0.512 |
| desArea | 0.827 | 0.750 | 0.780 | 0.751 |
| desPerf | 0.836 | 0.798 | 0.789 | 0.760 |
| stereoVisionC_no_mem | 0.835 | 0.798 | 0.788 | 0.759 |
| stereoVisionD | 0.597 | 0.428 | 0.564 | 0.543 |
| rayTraceC_no_mem | 0.812 | 0.775 | 0.766 | 0.716 |
| rayTraceD | 0.830 | 0.759 | 0.783 | 0.754 |
| Average | 1.020 | 1.024 | 0.964 | 1.022 |

18x18, Dynamic-18x18, and Dynamic-36x36 multipliers respectively. In the final row of Table 3.7, the geometrically average ratios are shown for each architecture.

The best hard multiplier architecture for the benchmark suite is a simple 18x18 multiplier that implements the benchmark suite 1.024 times smaller than an FPGA with no hard multipliers. In comparison, the simple 9x9 multiplier improves area efficiency over a purely soft design, but is not as beneficial as the 18x18 since only 6 of the 19 benchmarks contain multipliers that all fit in a 9x9 multiplier; in most other benchmarks, the mapped multipliers use both the hard 9x9 multiplier and soft logic cluster tiles, which can be more costly compared to using a hard 18x18 multiplier.

The functionally flexible multipliers, Dynamic-18x18 and Dynamic-36x36, have additional bit-width flexibility that comes at an area cost to implement this flexibility.

Our collection of benchmarks do not have enough variety of multiplier use to gain a significant benefit from these functionally flexible hard multipliers. The small benefit that some of the benchmarks get is offset by those benchmarks that cannot use the flexibility and incur the additional area cost.

Note that these results show the best multiplier architecture is a simple 18x18 hard multiplier for our collection of benchmarks. This does not mean that a simple 18x18 hard multiplier is the best type of hard multiplier to include on an FPGA, and instead, this depends on the multiplier bit-width demand of the benchmarks. In this experiment, the overall area benefit for any hard multiplier architecture is still small, but in the next section we will see how the supply ratio is a major factor for the area benefit of a hard circuit.

### 3.6.2 Best Architecture

The area benefit for hard multipliers in the two previous experiments is very small where the FPGA with hard multipliers only implements the benchmark suite, in the best case, in 1.024 times less area than a purely soft FPGA. One of the reasons for this is in these experiments the supply ratio is set for the FPGA to match the average demand ratio of the benchmark suite. Instead, the supply ratio can be varied to find the best ratio. In the following experiment, we will determine how the supply ratio affects the area efficiency of an FPGA that includes hard multipliers.

To find the supply ratio that results in the smallest FPGA with hard multipliers, we will vary the supply ratio and measure the area efficiency of these FPGAs to find the most area efficient one. We will use FPGAs containing simple 18x18 hard multipliers. For each experimental FPGA we will vary the supply ratio and compare the benchmarks mapped to this experimental architecture to an architecture without multipliers. This non-multiplier FPGA will be called the base, and the area-efficiency metric is the area of the experimental multiplier FPGA to map a benchmark divided by the area of the base FPGA. We geometrically average the area-efficiency metrics for all the benchmarks in a suite on each experimental architecture using the result to evaluate the quality of that architecture.

Figure 3.4 plots the results of this experiment with bars marking the standard de-

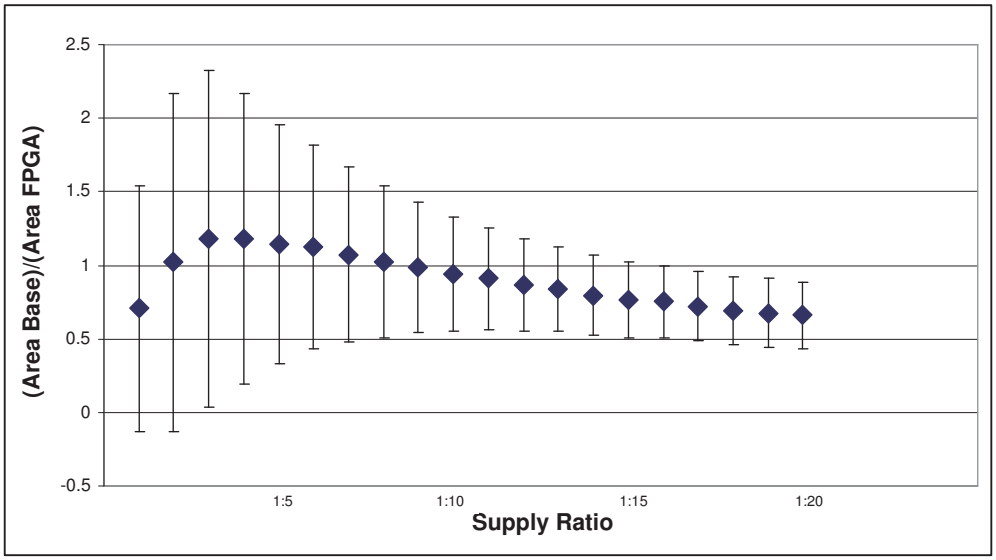


Figure 3.4: Area efficiency for varying supply ratios on an architecture with hard multipliers

viation for each point. The supply ratio of the architecture is on the x-axis and the area-efficiency metric of the experimental architecture is on the y-axis. Each data point in this graph, represented by a diamond, shows the area-efficiency metric of an architecture with hard multipliers and a set supply ratio compared to the base architecture. A value greater than 1 means that the architecture specified by the supply ratio on the x-axis has better area efficiency than the base.

Table 3.8 shows each of the data points in Figure 3.4. Column 1 contains the supply ratio of the architecture, and column 2 contains the geometrically averaged area-efficiency metric for the benchmark suite mapped to this FPGA.

The data in Figure 3.4 shows that for our collection of benchmarks the most area-efficient architecture has a supply ratio of 1:3 and results in an area-efficiency metric of 1.18. The main reason the best supply ratio, 1:3, is significantly larger than a supply ratio of 1:8, is due to the variance in the demand ratio distribution in the benchmarks. What this means is that there are benchmarks in the suite with a demand closer to 1:3 which will gain significant area benefits when mapped to an FPGA with a higher supply ratio outweighing the area increase for the lower demand benchmarks. For

Table 3.8: Results for the points in Figure 3.4

| Supply ratio | Area-efficiency metric |
|---|---|
| 1:1 | 0.71 |
| 1:2 | 1.02 |
| 1:3 | 1.18 |
| 1:4 | 1.18 |
| 1:5 | 1.14 |
| 1:6 | 1.12 |
| 1:7 | 1.07 |
| 1:8 | 1.02 |
| 1:9 | 0.99 |
| 1:10 | 0.94 |
| 1:11 | 0.91 |
| 1:12 | 0.86 |
| 1:13 | 0.84 |
| 1:14 | 0.80 |
| 1:15 | 0.77 |
| 1:16 | 0.75 |
| 1:17 | 0.72 |
| 1:18 | 0.69 |
| 1:19 | 0.68 |
| 1:20 | 0.66 |

example, moleculeDynamics is 1.2 times smaller on an FPGA with multipliers and a supply ratio of 1:8 compared to the soft FPGA, but the same benchmark is 4.9 times smaller on an FPGA with multipliers and a supply ratio of 1:3. Similarly, a low demand benchmark, stereoVisionC_no_mem, is 1.3 times larger on the FPGA with supply ratio of 1:8 and is only 1.7 times larger on an FPGA with supply ratio of 1:3. In this case, the benchmarks that get an area benefit outweigh the benchmarks that lose resulting in an overall area efficiency improvement for a supply ratio of 1:3 compared to 1:8.

## 3.7 Summary

In this chapter, we introduced a scientific methodology for measuring the area benefit of hard circuits in FPGAs. This included the introduction of a new FPGA architectural parameter, called the supply ratio and a benchmark characteristic called demand ratio. We described the benchmarks and experimental FPGA architectures that will be used throughout this dissertation.

Finally, we demonstrated how this methodology measures the area benefit of including a hard circuit on an FPGA. For these experiments, we compared the implementation area of an FPGA with and without hard multipliers when implementing our benchmark suite. We also showed how to determine what is the best hard multiplier architecture for our suite of benchmarks, and how large an effect supply ratio has on the area efficiency of a hard circuit included on an FPGA. In our last experiment, an FPGA with a supply ratio of 1:3 for hard multipliers resulted in the benchmark suite being implemented 1.2 times smaller than a purely soft logic FPGA compared to an FPGA with a supply ratio of 1:8 which resulted in the benchmark suite being implemented only 1.02 times smaller than a purely soft logic FPGA.

It is clear that a hard circuit like the hard multiplier provides an area benefit for FPGAs that include them even if not all the benchmarks targeting the FPGA use these hard circuits. In the next chapter, we focus on the theme of this dissertation and introduce an architectural concept that further improves the area efficiency of heterogeneous FPGAs by improving the utilization of these hard circuits.

# 4 Enhancing Area Efficiency of FPGAs using Hard Circuits and Shadow Clusters

Sometimes, the best answer is a more interesting question

Terry Pratchett

## 4.1 Introduction

We have discussed how hard circuits can reduce the area gap between FPGAs and ASICs if these circuits are actually used. If the hard circuits are not used, however, then the silicon is wasted including the expensive programmable routing that surrounds them. The central question of FPGA architecture is when to include hard circuits on the FPGA, and how to make them as flexible as possible so that most application circuits can use them.

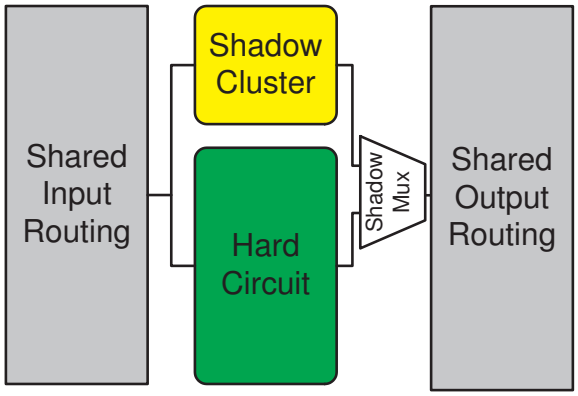


Figure 4.1: Illustration of shadow cluster concept

In this chapter, we propose a new architectural concept, called *shadow clusters*, that is a way to create this flexibility and is illustrated in Figure 4.1. A shadow cluster is a standard FPGA logic "cluster" that is combined with a hard circuit, such as a multiplier, in the same logical tile. In the event that the hard circuit cannot be used by an application circuit, simple fabric-programmable multiplexers "swap" in the shadow cluster, which can be used just like the regular soft logic fabric. While this may seem like a waste of area itself (because the shadow cluster goes unused when the hard circuit is used), the additional area is very small: well over 70% of the area of the soft logic of an FPGA is dedicated to the routing that would be shared between these two structures, and more importantly, this routing can always be used in an FPGA with shadow clusters.

We study this architectural concept by modeling the area of FPGAs with and without shadow clusters and mapping benchmark suites (characterized by their demand for soft and hard logic) to these FPGAs. The efficacy of this concept turns on the correct measurement of FPGA area, and so we carefully size the transistors in the FPGA and the hard circuits.

Two other key concepts are needed to correctly implement the idea: architecting the soft logic cluster tile and hard circuit tile that includes a shadow cluster so that their *pin demand* for programmable routing is about the same, and making the correct choice of how to map application circuits into hard or soft logic. These topics were addressed in Chapter 3, and will be revisited in this chapter as they apply to shadow clusters.

To concretize our measurements we focus on multiplier-based hard circuits, which are now common in modern FPGAs, but the concepts will be applicable to many kinds of circuits where the programmable routing consumes a significant amount of the hard circuit tile area.

## 4.2 Shadow Clusters

Our goal is to improve the area efficiency of an FPGA with hard circuits by reducing the penalty in area for unused hard circuits. The proposed technique leverages the fact

that the largest area cost in FPGAs is in the programmable routing. The idea is to add soft logic, which we call a *shadow cluster*, in combination *with* the hard circuit, so that either the hard circuit or the soft logic will be used ensuring that the expensive programmable routing is used. In this section, we discuss the architectural design of a hard circuit combined with a shadow cluster.

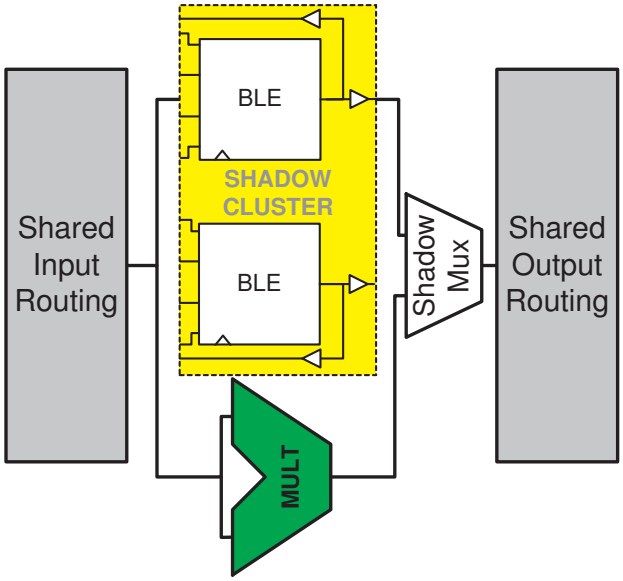


Figure 4.2: Multiplier combined with a shadow cluster in a tile

Figure 4.2 illustrates the shadow cluster concept with a tile containing a shadow cluster and a multiplier. In this figure, the logic block's programmable input routing drives both the BLEs of the shadow cluster and the multiplier, and a multiplexer selects which output to employ, under the usual programmable control. Only one of the multiplier and shadow cluster will be active after programming (i.e. for a given programming configuration file). The input and output routing for both hard circuits and soft logic cluster tiles are almost identical since both are designed to flexibly route to specific pins, and this allows us to combine the two circuits together without having to make significant changes to either circuit or the programmable routing.

There is an architectural question to deal with here: the size of the shadow cluster to place within the multiplier tile. As discussed in Chapter 3, it makes sense to match

the pin demand of a hard circuit to the pin demand of the soft logic cluster tile. If the hard circuit's pin demand is significantly greater than the soft logic cluster tiles, Section 3.2.1 argues the need to stretch the hard circuit over multiple logic tiles so that each logical tile has a pin demand that is roughly the same as the soft logic cluster tiles.

For example, an FPGA architecture with ten 4-input LUTs (such as that in Table 3.4) typically has an input pin demand of 22 [BRM99] and an output pin demand of 10 (usually the same as N) for a total pin demand of 32. To match this soft logic cluster tile's pin demand to that of an $n$ by $n$ multiplier suggests setting $n$ to 8, because an 8 by 8 multiplier (which has 16 inputs and 16 outputs for a total of 32 pins) exactly matches the soft logic cluster tile's pin demand.

Slightly larger multipliers, such as a 9 by 9 with a total pin demand of 36, could be added to the FPGA fabric without affecting the global routing architecture. This is because routing is architected to route all the designs that target them, which typically exert wildly varying demand for tracks. This tolerance of wide variation will also tolerate small perturbations in the total pin demand.

In this chapter, we will use an 18x18 multiplier as our example multiplier. This common size is used in both the Stratix II [Alt04d] and Virtex-4 [Xil05] architectures. Since this multiplier has 36 input pins and 36 output pins, and our soft logic cluster tile has 22 inputs pins and 10 output pins, the multiplier is "stretched" over two logical tiles to approximately match the pin demand.

This same argument, in reverse form, tells us that the shadow cluster should be the same size (number of BLEs) as the soft logic cluster tile. So, each of the two tiles together implementing the 18x18 multiplier will each have a 10 BLE shadow cluster.

Figure 4.3 shows a 4x4 array and a column of 18x18 multipliers combined with shadow clusters. Note how the 18x18 multiplier is stretched over two tiles.

As an aside, we believe that the extra multiplexing and area required to implement a shadow cluster will incur a minor speed penalty. As can be seen in Figure 4.2, there is no extra active path on the input side (and so only a minor capacitive load increase may occur), and only a 2:1 multiplexer is added on the output side, which is just one of many such multiplexers. This additional 2:1 multiplexer results in approximately a

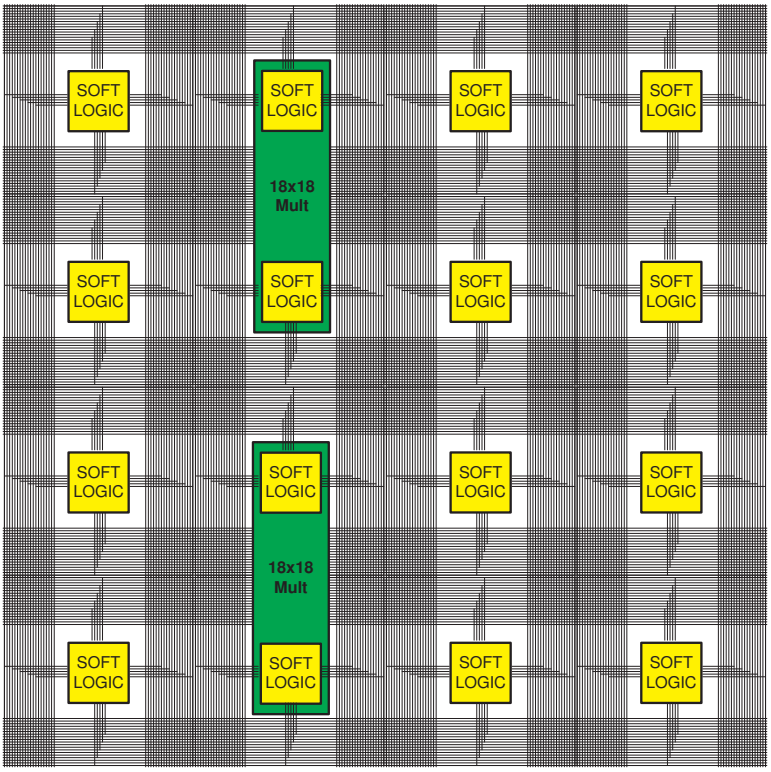


Figure 4.3: A 4 by 4 array with "stretched" 18x18 hard multipliers.

1% critical path delay increase for a path through the multiplier. These results were obtained using our automatic sizing tool (which we will discuss later). It is also possible to incorporate the 2:1 mux into the final output switch resulting in an area increase for additional control bits, but at no cost in speed.

Similarly, there is some concern that additional power will be consumed when the shadow cluster is being used instead of the multiplier. Here, the shared inputs drive both the shadow cluster and the multiplier, and dynamic power may be dissipated in the unused but electrically driven multiplier or shadow cluster, and static power will be consumed by the electrically active unused multiplier or shadow cluster. This loss, due to shared inputs, can be dealt with using simple power saving techniques such as gating [MDM+95], but is not addressed in this work.

## 4.3 Shadow Cluster Benefit

The key benefit with the shadow cluster concept is that the waste of unused programmable routing is mitigated in hard circuit tiles since a shadow cluster always allows the programmable routing designed for hard circuits to be used. Mapped designs that do not use all the available hard circuits on the FPGA for their primary purpose will benefit from shadow clusters. In these cases, a circuit's demand ratio is lower than the supply ratio for hard circuits on an FPGA. In this section, we will describe and show simple illustrations of the cases when shadow clusters do and do not provide a benefit.

In the cases when a benchmark's demand ratio is equal to or is greater than the supply ratio of the FPGA, all the hard circuits are used for their primary purpose. FPGAs that include shadow clusters will be less area efficient since in each hard circuit tile the area for the shadow cluster is not used by the design.

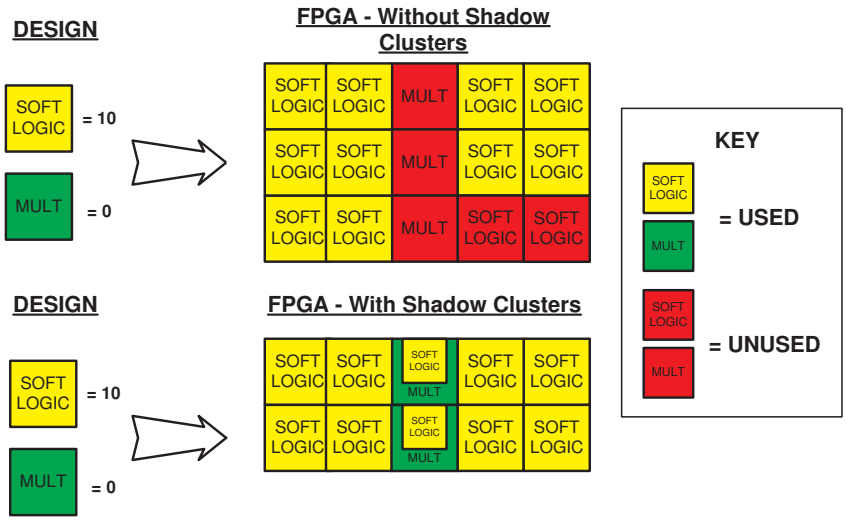


Figure 4.4: A design mapped to a shadowed and non-shadowed FPGA with supply ratio of 1:4.

Figure 4.4 shows an example when shadow clusters potentially provide a benefit depending on the details of the area cost of the programmable routing and shadow clusters. Here, the benchmark's demand ratio ($R_D$ = 1:$\infty$) is less than the FPGA's

supply ratio ($R_S$ = 1:4). This means that an FPGA with shadow clusters has hard circuit tiles that are not being used for their primary purpose and can, instead, be used to implement logic in the design. In the figure, the shadow cluster FPGA (the FPGA on the bottom of Figure 4.4) is more area efficient compared to the FPGA without shadow clusters and results in a smaller implementation.

The overall benefit in area efficiency of FPGAs with shadow clusters depends on how often these two cases occur in the benchmarks that target FPGAs and the area cost of shadow clusters. Before analyzing this benefit, in the next section, we describe modifications to the measurement methodology, described in Chapter 3, as it relates to shadow clusters.

## 4.4 Measurement Methodology

To determine the effectiveness of the shadow cluster concept we employ an empirical approach: we measure the area consumed by a set of benchmarks after mapping into a heterogeneous FPGA with hard multipliers, with and without shadow clusters. The following sections describe the mapping of benchmarks into the heterogeneous FPGA and then the determination of the FPGA's area. The subsequent section describes the set of benchmarks we employ.

### 4.4.1 Circuit Mapping Flow

Figure 3.3 in Chapter 3 shows the experimental flow used to measure the area for implementing benchmarks on an FPGA architecture. The first step in this flow maps a benchmark design to the different FPGA tiles available on the FPGA, and the second step calculates the area of the FPGA based on the tiles used. We discuss the calculation of the area in the next section.

A benchmark is modeled as requiring a number of soft logic cluster tiles and multiplier tiles. Depending on the supply ratio, the appropriate size of an FPGA is determined, and whenever the FPGA size must be increased, it is done by adding hard tiles and soft logic clusters tiles in this ratio.

Note that the multiplier hard circuit we employ in this work is a simple 18x18 multiplier. It is not decomposable into smaller multipliers such as the DSP block in Stratix FPGAs [Alt03], and so no special mapping is needed. This multiplier architecture choice is based on our results in Chapter 3, Section 3.6.1.

The mapping of a benchmark to an FPGA is determined by setting the number of multipliers equal to the number present in the benchmark, and then determining the number of soft logic clusters based on the supply ratio. This is mapping algorithm 1 (see Table 3.2), which we previously discussed in Chapter 3. This mapping algorithm is used when circuit speed is one of the key factors for employing a hard circuit, which is often the case for designs with multipliers.

In the case when the architecture has shadow clusters, the mapping algorithm takes this into account, by having each multiplier converted to use as a cluster if that benefits the final area.

Once the number of tiles of each type is known, and the area of each tile is known the total area for each benchmark can be calculated. The next section describes how the area of each tile type is determined.

### 4.4.2 Transistor and Cell Area Estimation of Tiles

The area-efficiency calculation requires the relative area between regular multiplier tiles, multiplier tiles with shadow clusters, and regular soft logic cluster tiles. As described in Chapter 3, Section 3.3.2 these sizes were determined in a 90nm CMOS process that was available to us [STM05, Mic07]

The area estimations for tiles are obtained using the results generated by the same sizing tool described in Appendix A. The area of a multiplier tile that contains a shadow cluster will contain, in addition, the area for the logic, programmable SRAM bits, and extra multiplexers needed to add the shadow cluster.

#### Relative Tile Area

Table 4.1 shows the area profile of different tiles (the soft logic cluster tile, the pure hard multiplier tile, and the shadowed multiplier tile) on a percentage basis. The final

Table 4.1: Percentage area within a tile and relative area

| Tile Type | BLEs | Mult. | Routing | Relative Size per Tile |
|---|---|---|---|---|
| Cluster (N=10) | 13% | - | 87% | 1.0 |
| Multiplier 18x18 (1 of 2 tiles) | - | 55% | 45% | 1.9 |
| Multiplier + Shadow 18x18 (1 of 2 tiles) | 6% | 52% | 42% | 2.0 |

column shows the size of each tile relative to the 10 by 4-LUT soft logic cluster tile. Note that the 18x18 multiplier is stretched over two tiles to match its pin demand with that of the soft logic cluster tile, and 1.9x represents how much one half of the multiplier is larger compared to the area of a soft logic cluster tile. This profile was obtained using the FPGA architecture parameters for Architecture 1 given in Table 3.4.

Notice for the soft logic cluster tile that the programmable routing takes over 80% of the area! Also, the shadow cluster increases the area of the hard multiplier tile by only 5% relative to the soft logic cluster tile. For the shadow concept to be useful, it must earn more than this back through an architectural gain when mapping a set of benchmarks to the FPGA.

In discussions with FPGA architects from both Xilinx and Altera [Roo06, Lew06], it appears these relative areas are reasonable compared to older industrial numbers. However, these architects claim that modern BLEs can account for as much as 40% of the soft logic cluster tile area. These BLEs include all the soft-fabric heterogeneity that exists in modern FPGAs. In the results section of this chapter, we perform an experiment that explores the effect of the relative size of logic and routing.

## 4.5 Benchmarks

The benchmarks need only be represented as the number and size of multipliers and the number of BLEs (which determines the number of soft logic cluster tiles) in each design. To obtain the number of BLEs, we pass 27 of our real benchmarks without memories (as described in Figure 3.3) through Altera's Quartus tool [Alt04c] Version

5.0. We determine the number of multipliers in each circuit by manually counting and identifying multipliers.

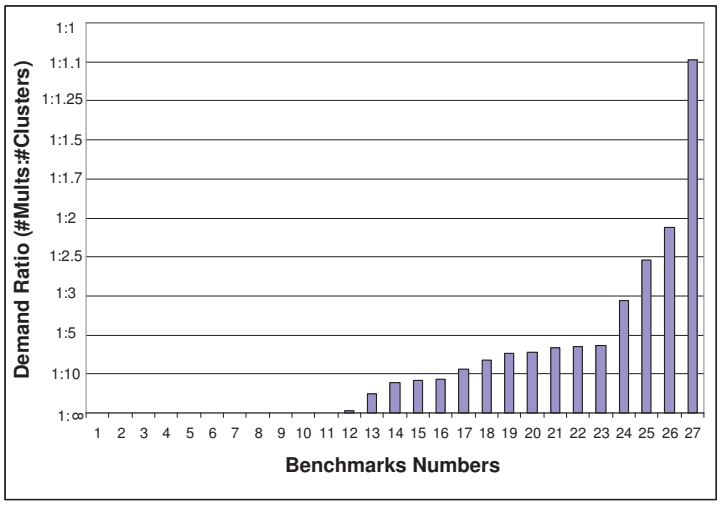


Figure 4.5: Demand ratios from our benchmarks

Figure 4.5 shows the demand ratios of the real benchmarks ordered from least to greatest. The ratio is calculated based on a soft logic cluster size of N=10 and multiplier size of 18 by 18. In these real benchmarks, there are multipliers that are both smaller and larger than 18x18 multipliers. To normalize the demand ratio in terms of 18x18 multipliers, we map every multiplier in the design to equivalent soft logic cluster tiles and total how many soft logic cluster tiles is needed to implement all the multipliers in a design. Next, we divide this total by the number of soft logic cluster tiles needed to implement an 18x18 multiplier giving us a number, which represents the normalized number of 18x18 multipliers in a design.

We choose this method to normalize multipliers since it is flexible enough to normalize a range of multiplier sizes such as a 9x15. Note that this method is dependent on the CAD tool used to map these multipliers circuits, but we use the same tool for all designs. Possible alternatives to multiplier normalization are transistor counts of a hard instantiation of these multipliers (which is very similar to our approach and is, again, dependent on the CAD tools) or using a scaling approach assuming that one 4x4 multiplier is 4 times as small as an 8x8 multiplier. The latter approach is captured

in our approach and becomes difficult to use when normalizing multipliers that are not bit width multiples of two.

Figure 4.5 shows that 12 of 27 of these benchmarks have no multipliers ($R_d = 1{:}\infty$), which we believe is realistic because only a subset of applications require multiplication. The remainder of the benchmarks have demand ratios ranging between roughly 1:20 to 1:1. The arithmetic average of demand ratio for all the circuits is approximately $R_d$ = 1:8, which is significantly larger than any of the supply ratios for existing industrial FPGAs (as shown in Table 3.1). The average demand is greater than even the Virtex-4 SX [Xil05] which has the greatest supply ratio of industrial FPGAs providing 1 multiplier tile for every 15 soft logic cluster tiles (with a cluster size of 10). This suggests that the characteristics of our collection of benchmarks is different compared to the benchmarks that were used to architect industrial FPGAs. We address this issue by synthetically generating a range of benchmarks with differing demand characteristics.

### 4.5.1 Synthetic Benchmarks

Synthetic benchmarks are used to analyze the effect of different statistics of demand on the shadow cluster architectural concept. These benchmark suites are drawn from a distribution either based on the demand ratios shown in Figure 4.5, which is our most realistic model to draw from, or a distribution that we have created (to experiment with demand ratio variance). We do this to postulate the effect of different application domains.

Table 4.2: Details of real and synthetic benchmark suites

| Name | Num. Bmarks | Avg. Demand | Variance | BLE Range | Mult. Range |
|---|---|---|---|---|---|
| B8 | 27 | 1:8 | 1:41 | 10542 to 34379 | 0 to 528 |
| SB45 | 250 | 1:45 | 1:83 | 10000 to 25000 | 0 to 145 |
| SB15_V0 | 250 | 1:15 | 0 | 10000 to 25000 | 0 to 350 |
| SB15_V1 | 250 | 1:15 | 1:1111 | 10000 to 25000 | 0 to 350 |
| SB15_V2 | 250 | 1:15 | 1:333 | 10000 to 25000 | 0 to 350 |
| SB15_V3 | 250 | 1:15 | 1:128 | 10000 to 25000 | 0 to 350 |
| SB15_V4 | 10 | 1:15 | 1:131 | 10000 to 25000 | 0 to 350 |

Table 4.2 describes the benchmark suites used in the experiments in this chapter.

Within the table, we report the average demand ratio of the benchmark suite, the variance among the benchmark's demand ratios, the range of BLEs, and the range of multipliers.

The benchmark suite, B8, consists of our original benchmarks in which small benchmarks are artificially inflated in size (by multiplying the number of multipliers and BLEs by a constant) to eliminate quantization noise in the experiments. The benchmark suites SB45 and SB15_V2 are synthetically created using a distribution based on our original benchmarks (shown in Figure 4.5). To create the synthetic benchmarks from this existing distribution, we simply stretch the distribution based on the desired number of benchmarks and then either amplify or dampen the demand ratio for each benchmark to get the desired average demand ratio.

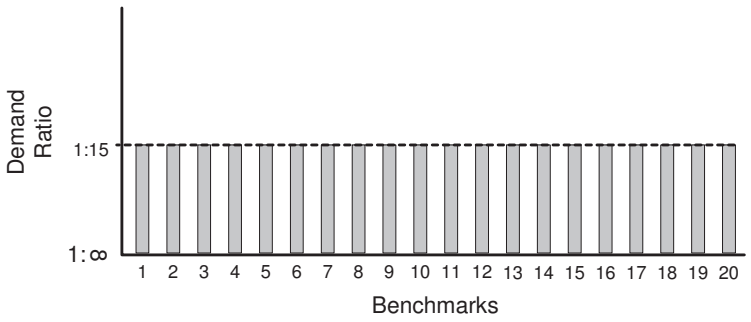


Figure 4.6: Demand ratio distribution for SB15_V0.

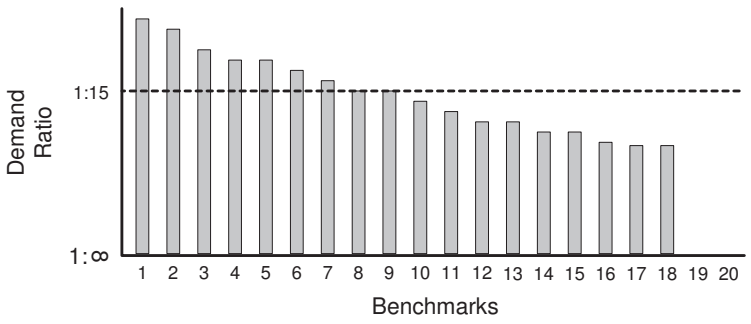


Figure 4.7: Example of demand Ratio distribution for SB15_V1.

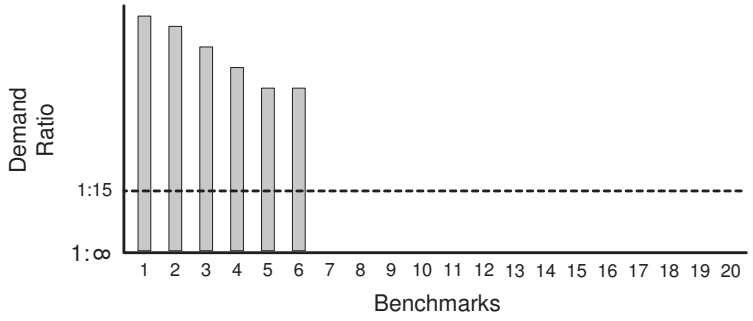


Figure 4.8: Demand Ratio distribution for SB15_V3.

The remaining benchmark suites all have an average demand ratio of 1:15 (the largest supply ratio present in the industrial FPGAs), but these benchmarks were drawn from different distribution curves to change the variance, where greater variance (a larger demand ratio for variance) means that the benchmark's demand ratios are more widely distributed from the average: SB15_V0 has no variance and is drawn from a distribution like the one illustrated in Figure 4.6. SB15_V1 is drawn from the distribution in Figure 4.7 where this distribution represents the case when the market has a large number of benchmarks that use a similar number of hard circuits. SB15_V3 is drawn from the distribution in Figure 4.8, which represents the case where a small number of benchmarks use many hard circuits.

## 4.6 Results: Effect of Shadow Cluster on FPGA Area Efficiency

In this section, we measure the effectiveness (in terms of area efficiency) of adding shadow clusters to heterogeneous FPGAs with multiplier tiles, under various scenarios of FPGA architectures (as expressed by the supply ratio), and demand for multipliers (as expressed by the demand ratio statistics of a particular benchmark suite).

We begin by exploring FPGAs that have supply ratios of modern FPGAs ranging from 1:15 at the high end and 1:104 at the low end as shown in Table 3.4. We will measure the effectiveness of adding shadow clusters to these FPGAs under two demand

scenarios: the first assumes that the average demand ratio is the same as the supply ratio (under the assumption that this is how industrial FPGA architects target their applications). The second (in Section 4.6.2 below) will use demands that are plausible, but different from those predicted by the commercial architects.

### 4.6.1 Avg. Demand Ratio Equal to Commercial Supply Ratio

In the first scenario, the average demand ratio of the benchmarks is set to be the same as the supply ratio of the FPGA. Consider the supply ratio of the Virtex-4 SX, which is 1:15 [Xil05]. Table 4.3 shows the results for a suite of 10 benchmarks (suite SB15_V4 from Table 4.2) that have an average demand ratio of 1:15.

Table 4.3: Results for individual benchmarks on shadowed and non-shadowed FPGAs with supply ratio equal to 1:15

| Benchmark Name | # Soft Logic Clusters | # Multipliers (18x18) | Demand Ratio | Area no Shadow Clusters ($10^5$ $um^2$) | Area with Shadow Clusters ($10^5$ $um^2$) | Area-Efficiency Metric |
|---|---|---|---|---|---|---|
| SB15_V4_1 | 1849 | 0 | 1:$\infty$ | 170 | 152 | 1.12 |
| SB15_V4_2 | 1904 | 0 | 1:$\infty$ | 174 | 156 | 1.12 |
| SB15_V4_3 | 1420 | 0 | 1:$\infty$ | 131 | 117 | 1.12 |
| SB15_V4_4 | 1042 | 0 | 1:$\infty$ | 96 | 87 | 1.11 |
| SB15_V4_5 | 1638 | 19 | 1:97.6 | 147 | 135 | 1.09 |
| SB15_V4_6 | 1925 | 89 | 1:21.6 | 175 | 171 | 1.02 |
| SB15_V4_7 | 1523 | 141 | 1:10.8 | 204 | 207 | 0.99 |
| SB15_V4_8 | 1304 | 121 | 1:10.8 | 175 | 177 | 0.99 |
| SB15_V4_9 | 1528 | 284 | 1:5.4 | 411 | 417 | 0.99 |
| SB15_V4_10 | 1502 | 349 | 1:4.3 | 506 | 513 | 0.99 |
| Average | | | 1:15 | | | 1.05 |

The table first gives the benchmark name, the number of soft logic cluster tiles in the benchmark, the number of 18x18 multipliers it requires, and the calculated demand ratio. The next column gives the area of the FPGA without shadow clusters required to implement the benchmark, assuming, as discussed previously, that the FPGA can grow to accommodate the size of the benchmark. The next column gives the area required for an FPGA with shadow clusters and the final column gives the calculation of the area "without" shadow clusters divided by the area "with" shadow clusters where a value greater than 1 means a shadow cluster architecture is smaller.

Table 4.3 illustrates when shadow clusters give a benefit - if the demand ratio is less than the supply ratio of the architecture, they allow the multiplier tiles' routing to be

used and result in an area-efficiency gain: the first 6 circuits in Table 4.3 gain 2% to 12% area efficiency. These benchmarks all use the available shadow clusters to decrease the overall area needed to implement them.

When the supply and demand ratios are equal or the demand ratio is greater than the supply then the shadow clusters cause a loss of efficiency because the shadow logic goes wasted. In each case where we lose area efficiency, the ratio is the same (0.986). The reason the ratio is the same is that each benchmark is mapped to shadow and non-shadow clustered FPGAs with enough hard multipliers for each multiplier in the benchmark. This means that both the shadow and non-shadow cluster architectures will have the same number of multipliers and soft logic clusters due to the supply ratio. In the shadow cluster architecture, each multiplier wastes space for each shadow cluster in a multiplier, and this waste is proportional to the supply ratio.

Overall, for benchmark suite (SB15_V4), the shadow cluster FPGA has a 1.054 area-efficiency metric meaning it is more area efficient than the non-shadow clustered FPGA. Note that the shadow cluster concepts wins, if slightly, under the scenario in which the average demand ratio *matches* the supply ratio, and therefore shows promise. The win arises because there is variance about the demand, which is clearly true for any realistic set of designs targeting FPGAs. The area improvement is a function of the supply ratio (which determines the amount of potential waste with unused multipliers), the average demand ratio (as discussed above) and the variance of the demand. In the next section, we explore the effect of varying supply ratio and demand ratio, keeping the variance constant.

### 4.6.2 Effect of Differing Average Demand Ratios

In this experiment, we explore the effect of the average demand ratio when mapped to an FPGA with fixed supply ratio to observe how the area benefit of shadow clusters changes. As the average demand increases, shadow clusters will not provide as great a benefit.

Table 4.4 illustrates the results of a series of experiments, each one like that given in Table 4.3, but with different supply ratio architectures and average demand ratios for the synthetic benchmarks. Each number in the table is the geometric average of

Table 4.4: Area efficiency of different benchmarks on architectures with different supply ratios

| | Area-efficiency Metric | | |
|---|---|---|---|
| $R_s$ | $R_d$=1:8 (B8) | $R_d$=1:15 (SB15_V2) | $R_d$=1:45 (SB45) |
| 1:15 | 1.043 | 1.047 | 1.083 |
| 1:23 | 1.027 | 1.025 | 1.046 |
| 1:45 | 1.013 | 1.012 | 1.017 |
| 1:60 | 1.011 | 1.009 | 1.011 |
| 1:66 | 1.007 | 1.007 | 1.010 |
| 1:104 | 1.004 | 1.004 | 1.006 |

the area ratios (without shadow clusters/with shadow clusters) across all the circuits in a given benchmark suite. Recall that Table 4.2 shows the characteristics of different suites, with different demand ratios. For example, the Table 4.4 entry with $R_s$ = 1:15 tested under a average demand ratio of 1:45 has an area-efficiency metric of 1.083, showing that a shadow cluster architecture is about 8.3% more area efficient than a non-shadowed architecture.

It is remarkable to note that every ratio in Table 4.4 is greater than one, indicating that shadow clusters *never* reduce area efficiency under all these scenarios! This suggests that the shadow cluster concept has merit.

These experiments compare FPGAs with fixed supply ratios with and without shadow clusters. Of more interest is to determine the best shadow clustered architecture (across all supply ratios) against the best non-shadowed cluster architecture (across all supply ratios), which follows below.

### 4.6.3 Best Shadowed and Non-Shadowed Architectures

In this section, we measure the area efficiency of FPGAs with supply ratios that vary for both shadowed and non-shadowed architectures. We will compare a suite of architectures with different supply ratios against a non-shadowed architecture with a fixed supply ratio of 1:15. We will refer to this 1:15 non-shadowed architecture as

the base_non_15 architecture. Its purpose is to provide normalization, which allows comparison across different architectures.

This experiment represents an architectural exploration in which the supply ratio is the FPGA architectural parameter that is explored.

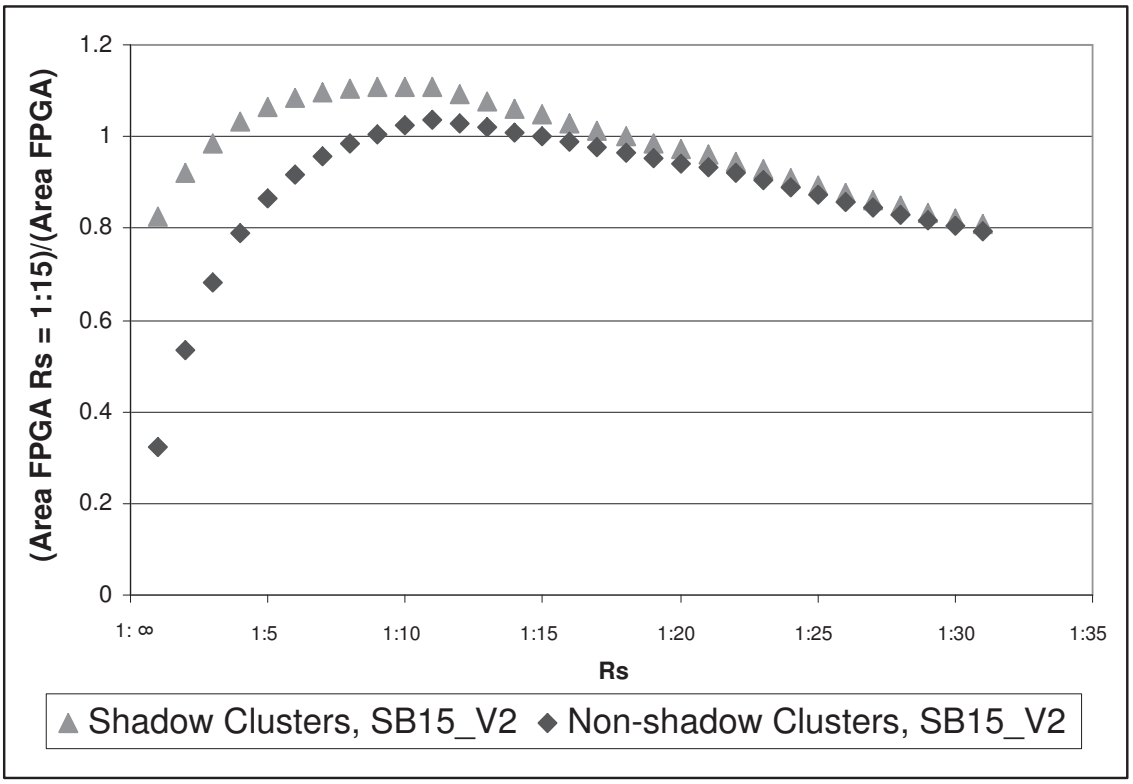


Figure 4.9: Area efficiency for varying supply ratios

Figure 4.9 shows data points in two curves where supply ratio of the architecture is on the x-axis and the area-efficiency metric (defined as the area of the base_non_15 divided by the area of the experimental FPGA) of the experimental architecture is on the y-axis. The data points marked by triangles compares shadowed architectures against base_non_15. Each triangle point represents an experiment, and shows the average area required to implement the benchmark suite, SB15_V2, on the base_non_15 architecture divided by the average implementation area on a shadow architecture with a specific supply ratio (given on the x-axis). A value greater than 1 means that the architecture specified by the x-axis has better area efficiency than base_non_15. We

do not include bars on this graph to represent the variance, but in Appendix C in Table C.47 we provide these values.

Similarly, the data points marked by diamonds compares non-shadowed architectures against base_non_15. Each diamond point represents an experiment comparing the average area required to implement the same benchmark suite on base_non_15 divided by a non-shadowed architecture with a specific supply ratio.

The triangle data of Figure 4.9 shows that the shadow cluster architectures with supply ratios ranging from 1:4 through 1:16 have better area efficiency than base_non_15 since the area-efficiency metric is greater than one.

Of these architectures, the one with a supply ratio equal to 1:9 is, on average, the most area-efficient shadow cluster architecture. The diamond data points, representing non-shadow clustered architectures with a supply ratio equal to 1:11 implements the benchmarks in the smallest area. When we compare the best shadow architecture and the best non-shadow architecture the shadow architecture is 7.2% smaller showing the benefit of including shadow clusters with the hard circuits. The next section will look at this area win with shadow clusters in more detail.

It is interesting to note how the shadowing concept changes the best supply ratio, in shadowed architectures vs. non-shadowed, and how shadowing appears to support a greater supply of multipliers.

For both shadow and non-shadow architectures, the supply ratio of the best architecture is greater than the average demand ratio of the benchmark suite. It might be expected that for a non-shadow architecture the supply ratio would match the average demand ratio of the benchmark suite. This would be true for some distributions where the variance is zero, but this is not the case in the above experiment (more details about variance in the demand distribution will be looked at later in this chapter).

The main reason that the best supply ratio is higher than the average demand ratio is the same as the explanation in Chapter 3, Section 3.6.2 where we showed that the benefit some benchmarks in the suite get for an increased supply of hard multipliers outweighs the loss for circuits that don't use them. Including shadow clusters in the architecture means that circuits that don't use the hard multipliers have an even lower loss factor and the higher supply ratio is beneficial to those circuits that can use the

additional hard multipliers.

### 4.6.4 Effect of Demand Ratios

In this section, we study how the distribution of demand ratios in our benchmark suites affects the area efficiency of FPGAs with shadow clusters. We measure the area efficiency of benchmark suites with different average demand ratios mapped to shadow and non-shadow FPGAs with varying supply ratios.

Table 4.5: Smallest implementation architecture for benchmark suites

| | Non-shadow Cluster Best Supply Ratio | Shadow Cluster Best Supply Ratio | Non-shadow vs Shadow |
|---|---|---|---|
| B8 | 1:11 | 1:9 | 1.125 |
| SB15_V2 | 1:11 | 1:10 | 1.072 |
| SB45 | 1:28 | 1:19 | 1.046 |

Table 4.5 shows a summary of the results in which we record the supply ratio of shadow and non-shadow architectures which result in, on average, the smallest area implementation for each benchmark suite. Column 1 names the benchmark suites; Columns 2 and 3 show the supply ratio at which the non-shadow cluster architecture and shadow architecture achieve the smallest implementation area, and Column 4 shows an area ratio where the area of the smallest non-shadow cluster architecture is divided by the area of the smallest shadow cluster architecture.

For our original benchmark suite (B8), the smallest shadow cluster architecture is 12.5% smaller than the smallest non-shadow architecture. The benchmark suite SB45 implemented on the smallest shadow cluster architecture is 4.6% smaller than the smallest non-shadow cluster architecture. This decrease in benefit is because as the benchmarks average demand decreases the non-shadowed and shadowed architecture have a best supply ratio that is also lower. As supply decreases, the benefit of shadow clusters also decreases since there are fewer multipliers and consequently shadow clusters to benefit from.

For each benchmark suite, the smallest shadow cluster architecture is more area

efficient than the smallest non-shadow cluster architecture, and in general, shadow cluster architectures improve implementation area for each benchmark suite.

### 4.6.5 Demand Ratio Variance within Benchmark Suites

Next, we measure the effectiveness of shadow clusters for benchmark suites with the same average demand ratio but different variances.

Table 4.6: Smallest implementation architecture for different demand variances

| Variance | Non-shadow Cluster Best Supply Ratio | Shadow Cluster Best Supply Ratio | Non-shadow vs Shadow |
|---|---|---|---|
| SB15_V0 | 1:15 | 1:15 | 0.960 |
| SB15_V1 | 1:10 | 1:10 | 1.045 |
| SB15_V2 | 1:13 | 1:11 | 1.072 |
| SB15_V3 | 1:11 | 1:9 | 1.114 |

Table 4.6 shows a summary of our results, similar to the previous table, in which we report supply ratios of the architectures that on average result in the smallest implementation of benchmark suites with average demand ratio equal to 1:15 and different variances. In the second row of the table, benchmark suite SB15_V0 has no variance, and as we continue down the rows in the table, variance increases, meaning the demand ratios of each benchmark within the benchmark suite are more widely distributed from the average. In general, as variance within the benchmark suite increases (as we go down the table) the area improvement for shadow cluster architectures compared to non-shadow cluster architecture increases.

This happens because greater variance means there are more circuits with demand ratios further from the mean. Therefore, there are more circuits with demand ratios lower than supply ratio, including benchmarks with no demand for multipliers that will benefit from the presence of shadow clusters. In addition, increased variance means there will be more circuits with demand ratio greater than supply, but as explained earlier, in this case, the small area increase for wasted shadow clusters is dominated by those circuits that decrease area.

For SB15_V0 with a variance of 0 (meaning that all benchmarks have a demand ratio equal to 1:15) we can see that the shadow cluster architecture results in a larger implementation area. This 1.4% larger area for the shadow cluster architecture represents the area cost for adding shadow clusters to an architecture with a 1:15 supply ratio.

### 4.6.6 Effect of a Larger BLE

Recall that, in Section 4.4.2 our discussion with two commercial FPGA architects and their comments that our area ratio between BLEs and programmable routing in a soft logic cluster tile and a shadow cluster is unrealistic compared to their modern architectures. In this section, we explore the effect of a larger BLE in comparison to the routing area and how the area benefit of shadow clusters is affected. This experiment and the next one provide a quick analysis of how the area-efficiency benefit changes if the area ratio between the BLE and the programmable routing in a tile changes.

To do this, we artificially increase the relative size of a BLE to the programmable routing in both the soft logic cluster tile and the shadow cluster. The original size of a BLE in our experimental architecture is 13% of the tile. In this experiment, we will increase percentage of the tile area dedicated to the BLE from 15% to 50% in increments of 5%.

We will map benchmark suite SB15_V2 to an FPGA with and without shadow clusters, and these FPGA's all have a supply ratio of 1:15. This setup is used since it represents an industrial FPGA supply ratio and a benchmark suite that would target such an FPGA.

Table 4.7 shows the new relative percentage area of the BLE in the soft logic cluster tile in column one, the relative percentage area of the BLE in the hard multiplier in column 2, and the geometrically averaged area-efficiency metric in column 3 (area of the non-shadow architecture divided by the area of the experimental architecture). Note that as we increase the size of the BLE we increase its size in both the soft logic cluster tile and the shadow cluster.

From Table 4.7 we can see that even if the BLE takes up half the area of the soft logic cluster tile (more than that of modern FPGAs), the shadow cluster concept still improves the area efficiency of an FPGA and still provides an overall area win. The

Table 4.7: Effect of increasing relative percentage area of the BLE in the soft logic cluster tile and the multiplier tile

| BLE percentage Area of soft logic cluster tile | BLE percentage Area of hard multiplier tile | (Area of purely soft FPGA) / (Area of shadow cluster FPGA) |
|---|---|---|
| 15% | 7% | 1.041 |
| 20% | 10% | 1.038 |
| 25% | 13% | 1.034 |
| 30% | 16% | 1.030 |
| 35% | 19% | 1.027 |
| 40% | 23% | 1.023 |
| 45% | 27% | 1.019 |
| 50% | 31% | 1.015 |

benefit, however, has decreased from a 4.2% area-efficiency improvement to 1.5%.

### 4.6.7 Effect of a Larger BLE only in the Soft Logic Cluster Tile

In the previous experiment, the results are somewhat disappointing since the area benefit of shadow clusters decreases as the area of the BLE increases compared to the programmable routing. It is, however, possible to conceive of an architecture in which the soft logic cluster tile consists of a modern and larger BLE as in the previous experiment, but the shadow clusters can be architected as small simple BLEs consisting of a LUT and a flip-flop. One assumption when considering an architecture like this is that the CAD tools targeting it have the ability to map designs to a heterogeneous soft fabric in which BLEs have different functionality.

We measure the area effect of increasing the size of the BLE in only the soft logic cluster tiles while keeping the size of the shadow cluster constant. We follow a similar method as the previous experiment and only increase the relative size of the BLE to the programmable routing in the soft logic cluster tile. This means we will keep a simple BLE that consists of only a LUT and flip-flop for the shadow clusters included with hard multipliers.

In this experiment, we assume that CAD tools can map simple logic to shadow

clusters with 100% logic utilization. This assumption is optimistic. Again, we will map SB15_V2 to an FPGA ($R_s$ = 1:15) with and without shadow clusters since it represents an industrial FPGA and its targeting market.

Table 4.8: Effect of increasing relative percentage area of only the BLE in the soft logic cluster tile

| BLE percentage Area of soft logic cluster tile | BLE percentage Area of hard multiplier tile | (Area of purely soft FPGA) / (Area of shadow cluster FPGA) |
|---|---|---|
| 15% | 6% | 1.043 |
| 20% | 6% | 1.044 |
| 25% | 6% | 1.045 |
| 30% | 6% | 1.046 |
| 35% | 6% | 1.048 |
| 40% | 6% | 1.049 |
| 45% | 6% | 1.051 |
| 50% | 6% | 1.052 |

Table 4.8 has the same structure as in the previous section. Column 1 contains the relative percentage area of the BLE in the soft logic cluster tile, column 2 contains the relative percentage area of the BLE in the hard multiplier, and column 3 contains the geometrically averaged area-efficiency metric (comparing the Non-shadow architecture to the shadow architecture).

Here, the results are better for including simple shadow clusters. The reason for this is as the soft logic cluster tile is more complicated and using a simple shadow cluster is a more area-efficient structure to implement simple logic in a design. These results suggest that it might be interesting to look at modern FPGAs and determine if a heterogeneous soft fabric that has a mixture of complex and simple soft logic cluster tiles might also improve FPGA area efficiency.

## 4.7 Summary

In this chapter, we presented an architectural concept called shadow clusters that when employed by existing hard circuits improves the overall area efficiency of the FPGA. The reason this concept works is that it tackles the main question in FPGA architecture by making hard circuits more flexible, and therefore, more likely to be used. The main saving, using this technique, is the programmable routing used to connect to the hard circuit.

We measured the effectiveness of shadow clusters showing that, under realistic scenarios, they always improve the area efficiency of existing industrial FPGAs that include simple 18x18 hard multipliers. Additionally, our results show that shadow clusters improve area efficiency to various degrees as a function of the supply ratio and its relation to the average demand ratio and variance of demand ratios within a benchmark suite. Our best results show that a shadow cluster architecture is 12.5% smaller than a non-shadow architecture when we map our real benchmark suite to these architectures and allow the supply ratio to vary.

In the next chapter, we will use shadow clusters to include new hard circuits and measure how this concept changes the frequency at which hard circuits need to appear in targeting benchmark suites.

# 5 Increasing FPGA Area Efficiency of Lower-Demand Hard Circuits

> The way to deal with an impossible task was to chop it down into a number of merely very difficult tasks, and break each one of them into a group of horribly hard tasks, and each of them into tricky jobs, and each of them...
>
> Terry Pratchett

## 5.1 Introduction

One of the main benefits of hard circuits in FPGAs is that they reduce area when used by a design targeting heterogeneous FPGAs. However, when a design does not target these hard circuits there is an area penalty for not only the unused logic needed to implement the hard circuit, but the programmable routing needed to connect to it. Only those hard circuits that are frequently used by designs and that provide a large area (or other) benefit should be included on the FPGA fabric.

In this chapter, we investigate how shadow clusters change the needed "frequency" of a hard circuit appearing in a design such that an FPGA with this new hard circuit is area neutral compared to an FPGA without the hard circuit. This study shows how hard circuits that are not currently included on FPGAs (due to low demand) might now practically be included if the hard circuit is combined with a shadow cluster.

Shadow clusters, described in Chapter 4 and reprised in Figure 4.1, improve the area efficiency of FPGAs as demonstrated in the last chapter. By incorporating a shadow cluster with a hard circuit, the high cost for wasted routing is mitigated since either

the hard circuit or the shadow cluster can be used by a design. From the perspective of a design that does not use a hard circuit, an FPGA with shadow clusters appears to have more expensive soft logic, while designs that use hard circuits gain most of the hard circuit's area, speed, and power benefits.

In this chapter, we show that shadow clusters make it more practical to include hard circuits with lower demand in the target market by reducing the penalty incurred in those application circuits that cannot make use of the hard circuit. This concept is illustrated by focusing on crossbar hard circuits, which up to now, have been considered to have a demand too low for inclusion on FPGAs. Both a single bit crossbar and bus-based crossbar, the later having a greater area benefit than the first when fully utilized, are included on FPGAs in this study.

The metric that we seek to improve (by decreasing it) is the "frequency" that the need for hard crossbars must appear in the FPGA's target application suite for the inclusion of the hard crossbar to appear to be area neutral. This kind of change could have significant impact on the architecture "argument" that goes on inside FPGA companies on which hard circuits to include on the device.

To appropriately prepare for this discussion we will also architect the features and parameters of a hard crossbar, including varying the bit-width of the hard crossbar and using either a bus-based crossbar or a single bit crossbar.

The remainder of this chapter is organized as follows: In section 5.2, we develop the architecture of the hard crossbars to be used including bus-based crossbars. In section 5.3, we describe the changes to the experimental methodology that is used to measure the area efficiency of our architectures including the synthetic benchmarks used to model different FPGA target markets, and section 5.5 presents the results of these experiments and analysis.

## 5.2 Architecting Hard Crossbars for FPGAs

We are using hard crossbars as an exemplar of a circuit currently not included on FPGAs likely due to the fact that there are not a sufficient number of designs in the FPGAs target market that would benefit from its inclusion. Crossbars, also known as

crosspoint switches, are circuits commonly used in communication applications as well as other digital circuits such as the interconnect for a multiprocessor system [ZJC+06]. FPGAs are used to implement crossbars in their soft logic with published works in both academia [BL03, KN02] and industry [Alt04b, Alt02, Xil00, YAF+03].

Jones and Wilton [JW04, JW03] published two patents on adding a bus-based crossbar to a programmable device. The structure of their bus-based crossbar is very similar to the one that we will present in this chapter. Their bus-based crossbar design is included on the FPGA as tiles in the programmable fabric hooked up to the programmable routing. These crossbars are built using pass transistor multiplexers. This is the same as the crossbars presented in this chapter, but the Jones and Wilton crossbar is cascading, meaning larger crossbars (than the bit width of the hard crossbar) can be built by connecting smaller units together. The bus-based crossbar presented in this chapter can be used to build larger crossbars, but will require the use of the standard programmable routing and soft logic to implement the larger crossbar.

In this section, we will describe the design of the hard crossbars we include on FPGAs. This description includes the range of crossbar sizes that can be implemented on each crossbar, the number of logical tiles the crossbar is stretched across, and the crossbar architecture including a single bit crossbar and a bus-based crossbar.

### 5.2.1 Definition of Crossbar Terms included on a FPGA

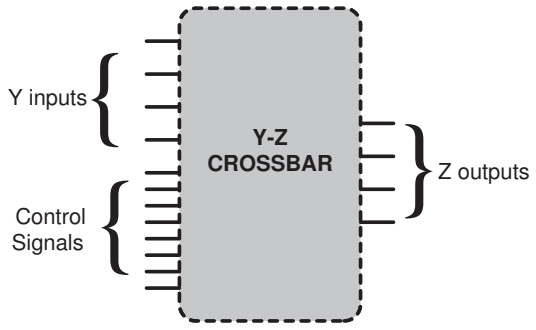


Figure 5.1: A full-way crossbar

Figure 5.1 shows a crossbar in which Y inputs pins can be routed to Z output pins.

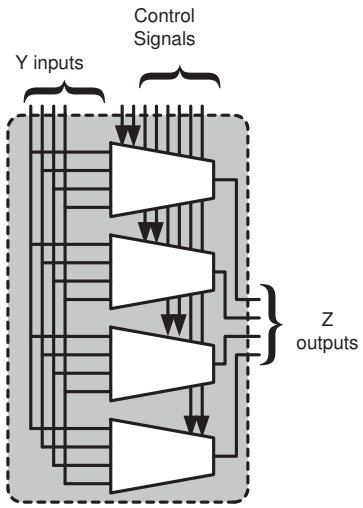


Figure 5.2: 4-4 full-way crossbar implemented with multiplexers

We will consider a full-way crossbar that can be dynamically controlled to broadcast one input to all outputs, unicast each input to a unique output, or a mixture of both. This type of crossbar can be implemented as Z multiplexers where each multiplexer is a Y to 1 multiplexer. Figure 5.2 shows a 4-4 crossbar implemented with multiplexers.

Table 5.1: Crossbars included on an FPGA

| Crossbar Name | Max Size | Num. 16-16 Crossbars | Num. 32-32 Crossbars | Num. 64-64 Crossbars |
|---|---|---|---|---|
| 16-16 | 16 | 1 | NA | NA |
| 32-32 | 32 | 2 | 1 | NA |
| 64-64 | 64 | 4 | 2 | 1 |

We will consider three different crossbars on an FPGA; Table 5.1 summarizes these three crossbars. Column 1 shows the type and name we will use to identify the crossbar; column 2 shows the maximum size crossbar that the hard circuit can implement using just the hard circuit. Columns 3, 4, and 5 show how each hard circuit can potentially be configured to implement different sized crossbars. For example, the 64-64 crossbar can be configured to implement either four 16-16 crossbars, two 32-32 crossbars, or one 64-64 crossbar.

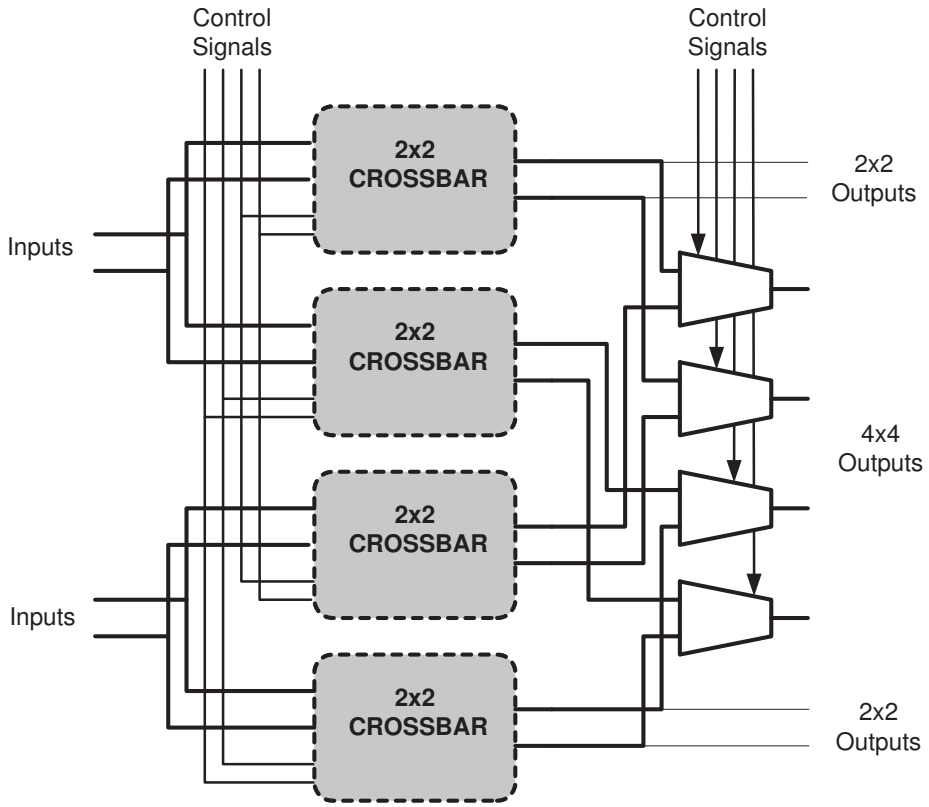


Figure 5.3: 4-4 crossbar implemented with 2-2 crossbars

To build a flexible hard crossbar like the 32-32 and 64-64 in Table 5.1, the design includes some shared inputs and additional multiplexers. Figure 5.3 shows an example of a 4-4 crossbar that can also be used to implement two 2-2 crossbars. In this figure, it takes four 2-2 crossbars to implement one 4-4 crossbar, and if this crossbar is in 4-4 mode then the first four control signals control selection in each of the 2-2 crossbars, and the last 4 control signals control the multiplexers at the output of the 2-2 crossbars. If this structure is used to implement two 2-2 crossbars then the top and bottom 2-2 crossbars implement these operations, and this mode only uses the first four control signals.

When this circuit is included on an FPGA the 2-2 outputs and 4-4 outputs would be combined together so that the final outputs, which drive into the programmable

routing, are shared. Either this could be done by using an additional stage of 2 to 1 multiplexers that would be programmably controlled to select the correct mode, or the second stage of control signals controlling the multiplexers for the 4-4 crossbar could be used to select between 2-2 mode and 4-4 mode by inputting constants in the 2-2 mode.

This sharing construction principle is used for both the 32-32 and 64-64 crossbar noting that some active area (making up the multiplexers) will be wasted depending on the mode of the crossbar.

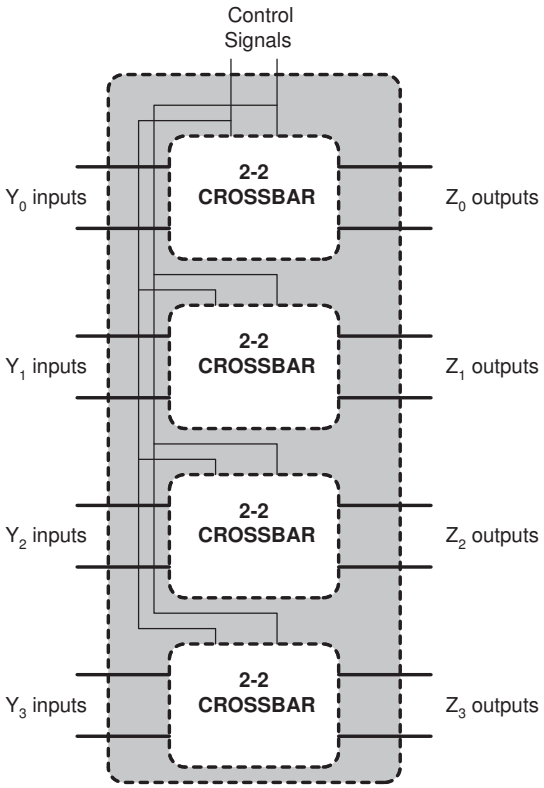


Figure 5.4: A bus-based crossbar consisting of four 2-2 crossbars

The most straightforward crossbar is one that allows individual control of each data bit. We call this a a *single bit* crossbar. As an alternative, more than one data bit can be controlled by a single set of control signals, which we call a *bus-based* crossbar. Figure 5.4 shows the structure of bus-based crossbar with a bus size of 4. In this

figure, $Y_0$ can be routed to $Z_0$ outputs sharing the control signals with the other 2-2 crossbars. Since the control signals are shared in the bus-based crossbar this reduces its pin demand and increases the area-benefit of these crossbars if the circuit is highly utilized.

### 5.2.2 Hard Crossbar Pin Demand and Number of Tiles

Given these hard crossbars to include on an FPGA and using the architecture parameters for Architecture 1 and Architecture 2 described in Table 3.4 for our experimental FPGAs, we now determine how many logical tiles the hard crossbars will be stretched over by dividing the total number of pins used by the crossbar divided by the pin demand of the soft logic cluster tile. This concept was addressed in section 4.2. The total number of pins needed to implement a Y-Z crossbar equals:

$$totalpins = Y + Z + (Z\lceil log_2 Y \rceil) \tag{5.1}$$

In this equation, Y and Z represent the input and output pins. The final term represents the number of pins needed for the control signals to select paths through the crossbar.

The total number of pins needed to implement X Y-Z crossbars where X represents the bit-width of a bus-based crossbar equals:

$$totalpins = X * (Y + Z) + (Z\lceil log_2 Y \rceil) \tag{5.2}$$

The major pin cost in the single bit crossbar is the number of control pins, and this is due to the choice that crossbars are full-way needing many control pins to make each possible switch pattern. It is possible to implement a crossbar with less flexibility and consequently less control pins, but without detailed knowledge on how targeting FPGA designs use crossbars, we take a worst case approach and use fully flexible crossbars. Some of the pin cost is reduced in the bus-based crossbar since the control signals are shared between all the crossbars in the hard circuit.

Table 5.2 shows the number of tiles each of the single bit hard crossbars will be stretched over for the two architectures as described in Table 3.4. Column 1 shows the

Table 5.2: Tiles per single bit crossbar

| Crossbar Name (Y-Z) | Pins in Crossbar | Architecture 1 Soft Logic Pin Demand | Architecture 1 Number Tiles | Architecture 2 Soft Logic Pin Demand | Architecture 2 Number Tiles |
|---|---|---|---|---|---|
| 16-16 | 96 | 32 | 3 | 35 | 3 |
| 32-32 | 224 | 32 | 7 | 35 | 7 |
| 64-64 | 512 | 32 | 16 | 35 | 15 |

name of the crossbar, and column 2 shows the total number of pins in each crossbar. Columns 3 and 5 show the soft logic cluster tile pin demand for Architecture 1 and Architecture 2 respectively, and columns 4 and 6 show how many logical tiles are needed to implement one hard crossbar for Architecture 1 and Architecture 2.

Table 5.3: Tiles per bus-based crossbar

| Bus Size (X) | Crossbar Name (Y-Z) | Pins in Crossbar | Architecture 1 Soft Logic Pin Demand | Architecture 1 Number Tiles | Architecture 2 Soft Logic Pin Demand | Architecture 2 Number Tiles |
|---|---|---|---|---|---|---|
| 4 | 16-16 | 192 | 32 | 6 | 35 | 6 |
| 8 | 16-16 | 320 | 32 | 10 | 35 | 10 |
| 12 | 16-16 | 448 | 32 | 14 | 35 | 13 |
| 16 | 16-16 | 576 | 32 | 18 | 35 | 17 |
| 4 | 32-32 | 416 | 32 | 13 | 35 | 12 |
| 8 | 32-32 | 672 | 32 | 21 | 35 | 20 |
| 12 | 32-32 | 928 | 32 | 29 | 35 | 27 |
| 16 | 32-32 | 1184 | 32 | 37 | 35 | 34 |
| 4 | 64-64 | 896 | 32 | 28 | 35 | 26 |
| 8 | 64-64 | 672 | 32 | 44 | 35 | 41 |
| 12 | 64-64 | 1920 | 32 | 60 | 35 | 55 |
| 16 | 64-64 | 2432 | 32 | 76 | 35 | 70 |

Table 5.3 shows the number of tiles each of the hard bus-based crossbars will be stretched over for the two architectures (as described in Table 3.4). Column 1 shows the bit-width of the crossbar, column 2 shows the bit-width of each crossbar in the hard circuit, and column 3 shows the total number of pins in each crossbar. Columns 4 and 6 show the soft logic cluster tile pin demand for Architecture 1 and Architecture 2 respectively, and columns 5 and 7 show how many logical tiles are needed to implement one hard crossbar for Architecture 1 and Architecture 2.

From these values we can see that a bus-based crossbar will implement many more crossbars in less tiles. For example, bit-width 4, 16-16, hard bus-based crossbar in Table 5.3 uses 192 less pins and 6 less logical tiles to implement a bus-based crossbar compared to using 4 single bit hard crossbars from Table 5.2. This means the bus-based crossbar will provide a greater area-efficiency improvement when implementing designs that use bus-based crossbars, but this depends on how much of the bus is utilized. If a design maps to hard bus-based crossbars of bit-width 2 and the FPGA includes bus-based crossbars of bit-width 4 then part of the hard bus-based crossbar is wasted.

### 5.2.3 Hard Crossbar Benefit over Soft Logic Implementation

With a description of the architectures and sizes of the hard crossbars that we will include on an FPGA we can make preliminary calculations as to what area benefit using a hard crossbar will have over implementing a crossbar in soft logic cluster tiles. This benefit is calculated by taking the number of soft logic cluster tiles needed to implement the crossbars in a design multiplied by the size of the soft logic cluster tile divided by the number of tiles a hard crossbar is stretched over multiplied by the size of the hard crossbar tile.

Hard crossbar tiles and soft logic cluster tiles use approximately the same area since the dominating area component in both tiles is the programmable routing (both a LUT and crossbar are essentially a few pass transistors that implement multiplexers). In this case, we can simplify the calculations and simply divide the number of soft logic cluster tiles needed to implement a crossbar by the number of tiles in a hard crossbar.

To calculate the number of soft logic cluster tiles needed to implement a crossbar in a design, we use Altera's Quartus CAD tool [Alt04c] to map crossbars to the soft logic of a Stratix I FPGA [Alt03] (similar to the architecture parameters we use in this chapter).

Table 5.4 shows different sized crossbars in a design and what benefit these crossbars will have when implemented on a 16-16, 32-32, and 64-64 hard single bit crossbar. For example, Table 5.4 shows that a 32-32 crossbar in a design uses 7 hard logical tiles and 77 soft logic cluster tiles. In this example, using a hard crossbar to implement the crossbar in a design will save a total 70 soft logic cluster tiles.

Table 5.4: Relative benefit of hard crossbars over soft crossbars

| Design Crossbar Size (Y-Z) | Soft Cost in clusters (N=10) | Hard Crossbar 16-16 tiles | Gain Factor | 32-32 tiles | Gain Factor | 64-64 tiles | Gain Factor |
|---|---|---|---|---|---|---|---|
| 8-8 | 4 | 3 | 1.33 | 7 | 0.57 | 16 | 0.24 |
| 16-16 | 18 | 3 | 6.00 | 7 | 2.57 | 16 | 1.13 |
| 24-24 | 41 | - | - | 7 | 5.86 | 16 | 2.56 |
| 32-32 | 77 | - | - | 7 | 11 | 16 | 4.81 |
| 48-48 | 154 | - | - | - | - | 16 | 9.63 |
| 64-64 | 308 | - | - | - | - | 16 | 19.3 |

This example, and the others in the table, shows that a hard crossbar can provide a significant area benefit over implementing the crossbar in a design in an FPGA's soft logic; however, a proper measurement needs to be performed over a suite of realistic applications, as we describe in the next section. This is needed since those applications that do not contain crossbars will pay an area penalty for including hard crossbars on an FPGA.

Table 5.5: Relative benefit of 4-bit bus-based 32-32 hard crossbar over soft crossbar implementation

| Design Crossbar Size (Y-Z) | Bus Utilization | Hard Crossbar Soft Cost in clusters (N=10) | 16-16 | Gain Factor |
|---|---|---|---|---|
| 32-32 | 1 | 77 | 17 | 4.53 |
| 32-32 | 2 | 154 | 17 | 9.06 |
| 32-32 | 3 | 231 | 17 | 13.59 |
| 32-32 | 4 | 308 | 17 | 18.11 |

Table 5.5 shows the gains of a 4-bit bus-based, 32-32 hard crossbar implementing 32-32 crossbars in a design with different bus width utilization compared to implementing those same crossbars in soft logic. Column 1 and column 2 show the size of the crossbars in the design and how many bits of the bus these circuits use. Column 3 and 4 shows how many soft logic cluster tiles and how many hard bus-based crossbar tiles it takes

to implement each bus-based crossbar respectively. Column 5 shows the gain factor of the hard bus-based crossbar included on an FPGA over implementing a design's bus-based crossbar in soft logic.

Table 5.6: A comparison between the gain factors of a single bit and bus-based 32-32 hard crossbar

| | Bus-Based | | Single Bit |
|---|---|---|---|
| Design Crossbar Size (Y-Z) | Bus Utilization | Gain Factor | Gain Factor |
| 32-32 | 1 | 4.53 | 11 |
| 32-32 | 2 | 9.06 | 11 |
| 32-32 | 3 | 13.59 | 11 |
| 32-32 | 4 | 18.11 | 11 |

Table 5.6 compares a 32-32 hard single bit crossbar's gain factor (which is equal to 11) to the results in Table 5.5. We can see that the bus-based crossbar provides an area benefit when more than 50% of the bits in the bus-based crossbar are used. When the bus-based crossbar is fully utilized there is a significant area benefit, but these gains will only be seen for an FPGA architecture when all of the hard bus-based crossbars are fully utilized by the designs representing the FPGA's target market, and this, we believe, is highly unlikely.

## 5.3 Measurement Methodology

As in the previous two chapters, our goal is to measure the area effectiveness of hard crossbars. The measurement methodology is the same as first described in Chapter 3, and involves a step to map a benchmark designs to the different FPGA tiles available on the FPGA, and then, calculating the area of the FPGA based on the tiles used.

In the experiments in this chapter we use both soft Architecture 1 and Architecture 2 described in Table 3.4 in Chapter 3. As described earlier (in Section 3.5), Architecture 2 implements crossbars in less area than Architecture 1 because Architecture 2 includes 6-LUTs that implement crossbars more area-efficiently than 4-LUTs. The inclusion of

Architecture 2 allows us to fairly evaluate hard crossbars combined with shadow clusters because of the more efficient modern soft logic fabric for implementing crossbars.

### 5.3.1 Mapping Benchmarks to Architectures

To measure the area consumed by a design, we map a benchmark to tiles available on the FPGA. We will map benchmarks to three types of FPGA architectures: without hard crossbars, with hard crossbars, and with hard crossbars including shadow clusters.

The benchmarks to be mapped to these architectures are modelled as requiring a number of soft logic cluster tiles and crossbars. The mapping step assigns crossbars to either hard crossbar tiles, soft logic cluster tiles, or a mixture of both. This is necessary since an FPGA may either not have enough or any hard crossbars, or the design crossbars may be of a size larger than the hard crossbars can implement. In the case that the design crossbar is larger than the hard crossbar, a combination of hard crossbar tiles and soft logic cluster tiles can be used to implement the design crossbar in less area than soft logic cluster tiles alone.

In the same fashion as Chapter 3, we will follow the usual practice in FPGA architecture research [RFCL89], and allow the size of the FPGA to be matched to the size of each benchmark, while maintaining the key FPGA architectural parameters.

The number and type of tiles required for a benchmark on a particular FPGA architecture is determined by increasing the number of soft logic cluster tiles and hard crossbar tiles until the benchmark design fits the FPGA. This is done by incrementally increasing the number of hard crossbar tiles, mapping the crossbars in the design to either available hard crossbars or soft logic cluster tiles, and determining if there is enough soft logic cluster tiles (calculated with the supply ratio) for the design.

When mapping the tiles in the design to an FPGA that includes hard crossbars we take a different approach compared to Chapter 3 and 4. Instead of mapping every crossbar in the design to a hard crossbar on the FPGA (where multipliers were used in Chapters 3 and 4), we take the other approach in which crossbars can be mapped to either hard crossbars or the soft logic cluster tiles. The reason for this is we do not consider speed to be a major factor in the mapping process as we do when mapping multipliers.

To do this mapping we use algorithm 3 described earlier in Chapter 3 and replicated here.

**Input**: Design (#hard circuits, #soft logic cluster tiles), Supply Ratio, Size and Bus-based or Single bit hard crossbar
**Output**: Number of each type of tile in FPGA
numHardCircuits = 1;
notMapped = TRUE;
**while** *notMapped* **do**
  softLogicAvailableOnFPGA = numHardCircuits * supplyRatio;
  **A** - Map hard circuits in the benchmark to available hard circuits on the FPGA;
  totalSoftLogicNeededByDesign = (soft logic in design) + (design's circuits not mapped to hard circuits);
  **if** *softLogicAvailableOnFPGA* > *totalSoftLogicNeededByDesign* **then**
    notMapped = FALSE;
  **end**
  **else**
    numHardCircuits++;
  **end**
**end**

**Algorithm 3**: The algorithm we use for mapping a design with crossbars to an FPGA with hard crossbars

The actual mapping of crossbars (denoted as step **A** within Algorithm 3) to hard crossbars on the FPGA is done in the following way. Given the set of crossbars in the design and a table similar to Table 5.4, which is extended for all crossbar sizes found in the designs, we rank each design crossbar in order of the largest gain factor to least. Table 5.4 describes the area benefits for each design crossbar depending on its size, the size of the hard crossbar, and the architecture. Once the crossbars in the designs are ranked in order of benefit, we start mapping the crossbars starting with the crossbar that gets the greatest area benefit.

Our crossbar allocation also takes into account crossbars that are bigger than a hard crossbar available on an FPGA and a number of smaller crossbars in a design that can all be mapped into one hard crossbar (such as the 64-64 hard crossbar that can implement four 16-16 crossbars). In both cases, the gain factor is calculated based on

the situation. In the case of a large design crossbar, the gain factor takes into account the need for multiple hard crossbars and soft logic. In the case of multiple small design crossbars fitting into one hard crossbar, the gain factors for each crossbar are totalled together.

After all the hard crossbars have been used on the FPGA, the remaining unmapped crossbars in the design are mapped to soft logic cluster tiles using a simple lookup from a table generated by mapping crossbars of all sizes to soft logic implementations on a Stratix FPGA [Alt03]. In the case where the architecture has shadow clusters, then the mapping algorithm uses the shadow clusters when a hard crossbar is not being used.

After this mapping, the number of each type of tile is known, and the area of the FPGA is calculated by multiplying the tile requirements by each tiles area. The following section discusses how we determine the area of each tile.

### 5.3.2 Transistor Area Estimation of Tiles

Similar to Chapter 3 and 4, the relative area of the soft logic cluster tile and the crossbar tile are determined using a 90nm CMOS process [STM05, Mic07] and an automated transistor sizing method, as discussed in Appendix A. We size the programmable routing, the LUT-based logic, and the crossbar tiles.

For the hard crossbar, multiplexers are implemented using pass transistors, and any wires that connect across one logical tile or the connections between the crossbars used to build larger crossbars (such as the crossbar in Figure 5.3) use a level restoring buffer. These circuits are included in our automatic sizing method. Additionally, the 32-32 and 64-64 crossbars are implemented using 16-16 crossbars that are combined together to form a functionally flexible hard circuit.

Table 5.7 shows the area profile of different tiles (including the soft logic tile, the hard crossbar tile, and the shadowed hard crossbar tile) on a percentage basis for our experimental architecture. The final columns show the size of each tile relative to the soft logic cluster size N=10. For each crossbar, the values represent only one of the logical tiles used to make the entire crossbar. For example, the 16-16 hard single bit crossbar is stretched over 3 logic tiles and the size comparison only compares one of these three tiles.

Table 5.7: Percentage area within a tile and relative area for Architecture 1

| Tile Type | BLEs | Crossbar | Routing | Relative Size soft logic cluster tile (N=10) vs. |
|---|---|---|---|---|
| Cluster (N=10) | 18% | - | 82% | 1.00 |
| Single bit Crossbar 16-16 (1 of 3 tiles) | - | 2% | 98% | 0.97 |
| Single bit Crossbar 16-16 + shadow cluster (1 of 3 tiles) | 15 % | 1% | 84% | 1.15 |
| Single bit Crossbar 32-32 (1 of 7 tiles) | - | 4% | 96% | 1.09 |
| Single bit Crossbar 32-32 + shadow cluster (1 of 7 tiles) | 15% | 3% | 82% | 1.24 |
| Single bit Crossbar 64-64 (1 of 16 tiles) | - | 6% | 94% | 1.10 |
| Single bit Crossbar 64-64 + shadow cluster (1 of 16 tiles) | 15% | 4% | 81% | 1.27 |

Table 5.8 shows the size of each tile relative to the soft logic cluster size N=10 for all the hard bus-based crossbars we explore in our experimental FPGAs. The hard bus-based crossbars tile's area slightly increases in size as bit-width increases. This per tile area increase is due to two factors. First, the buffers that drive the shared control signals grow in size since both the physical wire distance and capacitive load are increasing. Second, as the bus bit-width increases more transistors are packed into each logical tile that makes up the hard bus-based crossbar.

The most interesting thing about hard crossbars is they are relatively cheap to build in silicon (a crossbar tile without shadow clusters is roughly the same size as a soft logic cluster tile), and yet, their soft logic cost, seen in Table 5.4, is high. The low cost of implementing hard crossbars is because the high pin demand of the hard crossbar means it is spread over multiple logical tiles. The high cost to implement a crossbar in soft logic cluster tiles is due to the FPGA's inefficiency in implementing multiplexers.

Table 5.8: Relative tile area for hard crossbars compared to a soft logic cluster tile

| Tile Type | Bus bit-width | Relative Size per N=10 | Relative Size per N=10 with Shadow Cluster |
|---|---|---|---|
| Cluster (N=10) | 1 | 1.0 | - |
| 16-16 (1 of 3) | 1 | 0.97 | 1.15 |
| 16-16 (1 of 6) | 4 | 1.03 | 1.15 |
| 16-16 (1 of 10) | 8 | 1.05 | 1.18 |
| 16-16 (1 of 14) | 12 | 1.08 | 1.20 |
| 16-16 (1 of 18) | 16 | 1.10 | 1.21 |
| 32-32 (1 of 7) | 1 | 1.07 | 1.24 |
| 32-32 (1 of 13) | 4 | 1.17 | 1.29 |
| 32-32 (1 of 21) | 8 | 1.19 | 1.32 |
| 32-32 (1 of 29) | 12 | 1.20 | 1.34 |
| 32-32 (1 of 37) | 16 | 1.22 | 1.35 |
| 64-64 (1 of 16) | 1 | 1.10 | 1.27 |
| 64-64 (1 of 28) | 4 | 1.21 | 1.35 |
| 64-64 (1 of 44) | 8 | 1.23 | 1.36 |
| 64-64 (1 of 60) | 12 | 1.25 | 1.37 |
| 64-64 (1 of 76) | 16 | 1.27 | 1.39 |

## 5.4 Benchmarks

We now discuss the model used to describe benchmark applications that include crossbars. Our measurements only require the number of soft logic tiles required in a circuit, and the number and type of crossbars in a design including how many bus bits they use if they are bus-based crossbars. This allows us to model the benchmarks with just these numbers, but leaves us with the problem of validating whether any of these numbers realistically represent actual FPGA markets. Part of this problem is solved by the way we posed the question in the introduction - we wanted to show the effect on the demand of crossbars in the target market required for the area-efficiency measurement to break even. To answer this, we vary the benchmark crossbar demand, and so our results give that demand as an output rather than an input. Within each benchmark, however, there are different possibilities for how the circuit can demand the crossbar - they could be small or large, and therefore have a specific internal distribution of demand that needs to be realistic as well.

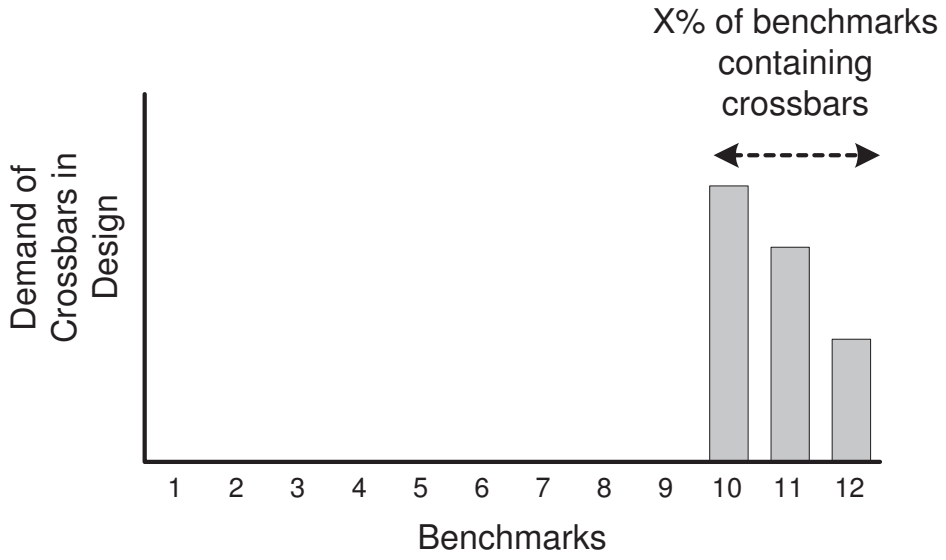


Figure 5.5: Example distribution crossbar demand in synthetic benchmarks

Figure 5.5 shows the general form of benchmark distributions we generate to represent the benchmarks targeting FPGAs. It is based on our observations of real benchmarks that suggest that only subset of circuits will have non-zero demand for crossbars. There is support for this observation in the fact that no widely used commercial FPGA yet contains crossbars of the nature we have described. The two key parameters of the distribution are the percentage of benchmarks containing crossbars and the average demand ratio for crossbars within those benchmarks. Creating our benchmarks in this fashion allows us to change the percentage of benchmarks containing crossbars so that we can model a range of benchmark distributions.

Table 5.9: Examples of synthetic benchmark suites with crossbars

| Name | Num. Bmarks | Percent with Crossbars | Avg. Demand of Bmarks with Crossbars | Avg. Demand | BLE Range | Crossbar Range |
|---|---|---|---|---|---|---|
| SB_5 | 100 | 5% | 1:15 | 1:300 | 10000 to 25000 | 0 to 350 |
| SB_10 | 100 | 10% | 1:15 | 1:150 | 10000 to 25000 | 0 to 350 |
| SB_15 | 100 | 15% | 1:15 | 1:100 | 10000 to 25000 | 0 to 350 |
| SB_20 | 100 | 20% | 1:15 | 1:75 | 10000 to 25000 | 0 to 350 |

Table 5.9 shows some examples of synthetic benchmark suites used in our experiments. Within this table, we report the benchmark name, the number of benchmarks, the percentage of benchmarks containing crossbars, the average demand ratio of benchmarks containing crossbars, the benchmark suite's average demand ratio, the range of BLEs per benchmark, and the range of the number of crossbars per benchmark.

In all benchmarks, the size of the design crossbars are either all 16-16, 32-32, or 64-64. Regardless of the size of the crossbars, the demand ratio remains the same per benchmark since demand ratio is normalized to 64-64 hard crossbars and a soft logic cluster size of 10 LUTs per cluster. For example, a design with 2400 soft logic cluster tiles and a demand ratio equal to 1:15 will include either 160 16-16 hard crossbars, 40 32-32 hard crossbars, or 10 64-64 hard crossbars.

Each individual benchmark that contains crossbars has a demand ratio between 1:1 (representing a design similar to a digital router with a primary function to route packets to destinations) to 1:227 (representing a design that needs very few crossbars such as a multi processor system that needs a network to communicate with each processor in the system). The average demand ratio for each benchmark suite depends on the percentage of benchmarks containing crossbars, and for the benchmarks that do contain hard crossbars the average demand ratio for these benchmarks is set to 1:15. For example, if 10% of the benchmarks in a benchmark suite (with 100 benchmarks) have crossbars in them, these benchmarks will have an average demand of roughly 1:15, but the entire benchmark suite will have an average demand ratio of 1:150 (as seen in Table 5.9) since the other 90% of the benchmarks in the suite have a demand ratio of $1{:}\infty$.

The last parameter that we vary within our synthetic benchmarks is the bus utilization of the benchmarks that use crossbars. We do not create a benchmark suite in which each design has different bus bit-width demands, and instead, for all benchmarks in the suite and for all crossbars used each bus-based crossbar has the same bus bit-width. For example, one of the benchmark suites will consist of 12% of the benchmarks that use hard bus-based crossbars of bit-width 5.

Our approach is to create a range of benchmark suites based on demand for crossbars, bus utilization for bus-based crossbars, and crossbar size to represent a range of possible

markets that would target FPGAs. In this way, we can make observations of the area efficiency of different FPGA architectures serving markets with different demand characteristics. Given the characteristics of a target market, we can then state what are the FPGA architectures that result in the most area-efficient implementation of the designs within each market. This approach is not the most desirable approach (which would use real benchmarks), but generating a wide range of target markets based on synthetic benchmarks is a reasonable method to at least observe architectural trends based on possible market characteristics.

In summary, we have created benchmark suites with similar characteristics as those described in Table 5.9. These benchmark suites range from 1% of benchmarks containing crossbars all the way to 100% of benchmarks containing crossbars. We create benchmark suites that contain 16-16, 32-32, and 64-64 crossbars and bus utilization ranging from bus bit-widths of 1 to 16. For example, one benchmark suite includes 17% of the benchmarks with 3 bit 64-64 crossbars. There are a total of 4800 benchmark suites each with 100 benchmarks, and these suites are used to determine what demand for crossbars results in a break-even area point for implementation area when comparing hard crossbar architectures to purely soft fabric FPGAs under a variety of market characteristics.

## 5.5 Results

We now measure the relative area efficiency of FPGAs with hard crossbars, and FPGAs with hard crossbars combined with shadow clusters compared to purely soft FPGAs to determine the area effectiveness of hard crossbars. As described above, this is done by mapping suites of benchmark circuits into each type of FPGA (which grows in a consistent architectural manner to accommodate each benchmark) and then measuring the area of the resulting FPGAs.

The benchmark suites will be mapped to the soft logic architectures described by the parameters in Table 3.4. We also map these benchmarks to FPGAs that contain 16-16, 32-32, or 64-64 hard crossbars (as shown in Table 5.1) that in some cases will be bus-based and include shadow clusters.

We will use the area for of the pure soft logic FPGA as the basis for comparison for the crossbar-based architectures - normalizing by dividing the soft logic-only area by the area for the other architectures. Thus, if this area ratio is greater than one it means that the experimental architecture is using less area than the pure soft logic FPGA. Finally, we will geometrically average all these area ratios for a set of benchmarks implemented on a particular architecture, and the average represents how well the experimental architecture compares to a soft logic FPGA when implementing that particular benchmark suite.

In each experiment, for each heterogeneous FPGA with crossbars, we vary the supply ratio (the number of hard crossbar tiles to the number of soft logic cluster tiles) and select the supply ratio that implements the benchmark suite in the smallest average area. In Chapter 4, section 4.6.1, we showed how the best supply ratio differs between shadow and non-shadow multiplier architectures.

One of the metrics that we seek for each experimental FPGA architecture is the "frequency" that the need for hard crossbars must appear in the benchmark suite for the inclusion of the hard crossbar to appear to be area neutral. This "frequency" is determined by the area break-even point where this break-even point is determined as follows. We will map the benchmark SB_1 (which has 1% of the benchmarks that use crossbars) to the architecture under study, where the mapping includes varying the supply ratio for hard crossbars and finding the best supply ratio as described above. If the geometrically averaged area ratio is greater than one than the break-even point is 1% for this particular architecture. Otherwise, we now map SB_2 to the architecture and repeat the process. This is continued until we find the percentage of benchmarks containing hard crossbars at which the experimental and soft logic FPGAs are area neutral.

### 5.5.1 Effectiveness of Hard Crossbars with and without Shadow Clusters

In our first experiment, we will look at how a shadow cluster changes the area efficiency of an FPGA that includes hard crossbars to determine how this changes the argument

for including hard crossbars on FPGAs. We will measure the area effectiveness of an FPGA with hard single bit crossbars and an FPGA with hard single bit crossbars combined with shadow clusters. The required demand is determined by the "frequency" of the use of crossbars in the designs which results in an area-neutral architecture compared to a purely soft FPGA as described above.

Table 5.10: Area break-even demand points for architectures including hard single bit crossbars

| | | Crossbar Architecture | | Crossbar+Shadow Architecture | |
|---|---|---|---|---|---|
| Hard Crossbar Type | Design Crossbar Size | Percent of Benchmarks with Crossbars | Average Demand Ratio | Percent of Benchmarks with Crossbars | Average Demand Ratio |
| 16-16 | 16-16 | 18% | 1:83 | 3% | 1:500 |
| 16-16 | 32-32 | 10% | 1:150 | 2% | 1:750 |
| 16-16 | 64-64 | 18% | 1:83 | 3% | 1:500 |
| 32-32 | 16-16 | 32% | 1:47 | 9% | 1:167 |
| 32-32 | 32-32 | 12% | 1:125 | 3% | 1:500 |
| 32-32 | 64-64 | 6% | 1:300 | 2% | 1:750 |
| 64-64 | 16-16 | 49% | 1:30 | 12% | 1:125 |
| 64-64 | 32-32 | 15% | 1:100 | 5% | 1:300 |
| 64-64 | 64-64 | 8% | 1:188 | 2% | 1:750 |

Table 5.10 shows the percentage of benchmarks containing crossbars at which the average implementation area is the same (meaning the area-efficiency metric is greater than or equal to 1.0) for both a soft logic FPGA and an FPGA with hard crossbars (with or without shadow clusters). Column 1 shows the type of hard crossbar included on the FPGA and column 2 shows the size of the crossbar in the benchmarks within each benchmark suite. Columns 3 and 4 show the area break-even point and average demand ratio of the benchmark suite that is area neutral for an FPGA that includes hard crossbars. Similarly, Columns 5 and 6 show the area break-even point and average demand ratio of the benchmark suite that is area neutral for an FPGA that includes hard crossbars combined with shadow clusters.

For example, for an FPGA with 16-16 hard crossbars and no shadow clusters implementing benchmarks with 16-16 crossbars, 18% of the benchmarks must contain crossbars (with an average demand ratio of 1:83) for that FPGA to have the same area

efficiency as the pure soft logic FPGA.

The shadow cluster architectures always "break even" with significantly less demand for crossbars than those without shadow clusters. This result is similar to the previous chapter for hard multipliers. For example, the 16-16 shadowed architecture implementing benchmarks with 16-16 crossbars requires only 3% of the benchmarks to demand crossbars and an average demand ratio of 1:215 compared to 18% and 1:83 average demand ratio for the same architecture without shadow clusters.

These results show that shadow clusters make it far more practical to include lower-demand circuits on FPGAs, and have potential to alter the architecture argument in FPGA companies in a substantial way.

One other observation from these results is that it appears the best type of hard crossbar to implement a crossbar in a design is not necessarily the same size as the crossbars included on the FPGA. A 16-16 hard crossbar included on an FPGA implements a benchmark with 32-32 crossbars at a lower area break-even point than a hard 32-32 hard crossbar (where the frequency values are 10% versus 12% respectively). The break-even metric, however, is not the most suitable way to compare hard crossbar architectures, and instead, we will use area-efficiency values to compare architectures with hard crossbars to a purely soft FPGA fabric. In this experiment, we set the average demand ratio of the benchmarks to 1:25 and pick the FPGA architecture supply ratio that results in the most area-efficient architecture.

Table 5.11: Area efficiency for architectures including hard single bit crossbars

| Hard Crossbar Type | Design Crossbar Size | Area-efficiency Metric | Best Supply Ratio |
|---|---|---|---|
| 16-16 | 16-16 | 1.43 | 1:2 |
| 16-16 | 32-32 | 1.56 | 1:5 |
| 16-16 | 64-64 | 1.05 | 1:30 |
| 32-32 | 16-16 | 1.27 | 1:3 |
| 32-32 | 32-32 | 1.50 | 1:4 |
| 32-32 | 64-64 | 1.53 | 1:8 |
| 64-64 | 16-16 | 1.20 | 1:3 |
| 64-64 | 32-32 | 1.41 | 1:5 |
| 64-64 | 64-64 | 1.51 | 1:7 |

Table 5.11 shows these results. Column 1 shows the type of hard crossbar included on the FPGA and column 2 shows the size of the crossbar used by the benchmarks within each benchmark suite. Column 3 shows the area efficiency of a hard crossbar architecture compared to a purely soft architecture, and column 4 shows the supply ratio of the FPGA at which this best area-efficiency result is achieved.

These results show that an FPGA with 16-16 hard crossbar implements 32-32 crossbars with an area-efficiency metric of 1.56. This is more area efficient than an FPGA with 32-32 hard crossbars with its area-efficiency metric of 1.50. To understand why, we look at the cost of implementing a 32-32 crossbar on each FPGA. It takes four 16-16 hard crossbars and 3.2 soft logic cluster tiles for a total of 17.2 tiles to implement a 32-32 crossbar in a design as opposed to 7 32-32 hard crossbar tiles. In the cases where the hard crossbars are used by the benchmarks in the suite, an FPGA with 32-32 hard crossbars are more than twice as small implementing these crossbars. However, this benefit will be offset by the benchmarks that do not use crossbars, since in these cases, the 16-16 hard crossbar tile is physically smaller compared to the 32-32 hard crossbar tile meaning that there is less of an area penalty for the unused smaller crossbars. It turns out that in this case (which is determined by the benchmark demand) the difference between crossbar tile areas is a larger factor than the area benefit for mapping to the more efficient 32-32 hard crossbars.

If we increase the size of the crossbars in the benchmark to 64-64, then the gain factors for larger hard crossbars like the 32-32 and 64-64 are big enough to dominate the increased size of these tiles. In this case, the 16-16 crossbar is not the best hard crossbar choice.

### 5.5.2 Effect of a Better Soft Fabric for Crossbars

Next, we explore the effect of the soft fabric architecture of an FPGA. If the soft fabric is more area efficient at implementing crossbars then the gain in using hard crossbars is reduced. We seek to determine if this reduction in hard crossbar benefit changes the previous analysis, and if shadow clusters still provide an improvement in area efficiency.

We perform the same experiment as above using the soft logic fabric of Architecture 2 (described in Table 3.4), which implements crossbars more efficiently than the soft

logic fabric of Architecture 1. This will help us validate if shadow clusters combined with hard crossbars are still a practical architectural inclusion on these more efficient soft fabric architectures.

Architecture 2 is modeled after the Altera Stratix II [Alt04d] and the Virtex-5 [Xil06] and implements crossbars in the soft logic cluster tiles on average in 28% less area than Architecture 1 (using our tile area measurements). We will measure how this improved soft logic implementation of crossbars changes our measurements of the area benefit of hard crossbars combined with shadow clusters.

We include hard crossbars in our experimental FPGA and evaluate the benefits of these crossbars in the same manner as described in Section 5.2.

To make our results a fair comparison with the previous sections results, we do two conversions to the synthetic benchmarks when mapping them to this architecture to normalize the benchmarks. First, the lookup table to calculate the soft cost of a crossbar is generated using a Stratix II ALUT. Second, in each benchmark the total number of Logic Elements (LEs) required by the benchmark is first divided by 1.25 where 1.25 is Altera's quoted Stratix II improvement in logic utilization over Stratix I chips [Alt04e].

Table 5.12: Area break-even points for a better soft fabric and hard single bit crossbars

| | | Crossbar Architecture | | Crossbar+Shadow Architecture | |
|---|---|---|---|---|---|
| Hard Crossbar Type | Design Crossbar Size | Percent of Benchmarks with Crossbars | Average Demand Ratio | Percent of Benchmarks with Crossbars | Average Demand Ratio |
| 16-16 | 16-16 | 27% | 1:56 | 9% | 1:167 |
| 16-16 | 32-32 | 15% | 1:100 | 3% | 1:500 |
| 16-16 | 64-64 | 23% | 1:65 | 15% | 1:100 |
| 32-32 | 16-16 | 37% | 1:41 | 14% | 1:107 |
| 32-32 | 32-32 | 19% | 1:79 | 7% | 1:214 |
| 32-32 | 64-64 | 11% | 1:136 | 3% | 1:500 |
| 64-64 | 16-16 | 40% | 1:38 | 20% | 1:75 |
| 64-64 | 32-32 | 21% | 1:71 | 9% | 1:167 |
| 64-64 | 64-64 | 12% | 1:125 | 3% | 1:500 |

Table 5.12 shows the same information as Table 5.10 except the benchmark suites

are mapped to FPGAs with Architecture 2's soft logic parameters. Columns 3 and 5 show the percentage of benchmarks containing crossbars at which the average implementation area is the same for both a soft logic FPGA and an FPGA with hard crossbars (with and without shadow clusters).

In general, we can see that because Architecture 2 implements crossbars more area efficiently in the soft logic that the area benefit of hard crossbars combined with or without shadow clusters is reduced. For example, implementing 32-32 crossbars on an Architecture 1 with 16-16 hard crossbars results in an area break-even point of 10% compared to Architecture 2 with 16-16 hard crossbars resulting in a break-even point of 15%. The break-even point demand increases for Architecture 2 since the purely soft FPGA implements crossbars more area efficiently than Architecture 1.

These results still show that shadow clusters reduce the area penalty for unused hard crossbars, and shadow clusters combined with hard crossbars provide a needed demand to be area-neutral that is lower than that of an architecture without shadow clusters. These results, however, do suggest that as the soft logic fabric improves the inclusion of a hard circuit may become less desirable.

### 5.5.3 Effectiveness of Bus-based Hard Crossbars

In the above work, we focussed on single-bit crossbars. Here, we will compare the hard crossbar architectures by measuring the area efficiency of hard bus-based crossbars and hard single bit hard crossbars. As discussed in section 5.2 hard bus-based crossbars share the control pins between each crossbar in the bus thus reducing the pin demand of the hard circuit and increasing the area benefit if more than one of the bits in the bus are used.

In this experiment, we will fix the benchmark suite to 20% of the benchmarks containing crossbars, which is equivalent to an average demand ratio of 1:75 for the benchmark suite. For each FPGA that includes hard bus-based crossbars with a specified bus size, we will map each our benchmark suite, SB_20, with a specified crossbar bus utilization ranging from 1 bit to 16 bits. When we map each benchmark to the architecture, we pick the supply ratio that results in the most area-efficient architecture. This was first seen in Chapter 3, Section 3.6.2.

We use the area-efficiency metric to compare each of the architectures implementing the benchmark suite. In each case, an area-efficiency metric is calculated as the area used to implement the benchmarks on a purely soft FPGA divided by the area to implement the same benchmarks on an experimental architecture, and an area-efficiency metric greater than one means that the experimental FPGA is smaller than the purely soft FPGA. These area-efficiency metric are geometrically averaged for each benchmark in the benchmark suite.

We use an FPGA with the soft logic architectural parameters for Architecture 1 noting that the general results are similar for both Architecture 1 and Architecture 2 with similar affects as described in the previous experiments. Similarly, we have performed these experiments for hard bus-based crossbars of size 16-16 and 32-32 with similar results as described below.

Table 5.13 shows the area-efficiency metrics for FPGAs with 64-64 hard bus-based crossbars. Column 1 and column 2 shows the size of the hard bus-based crossbars and the bus bit-width. Column 3 and column 4 show the size and bus utilization of the crossbars in the benchmark. Column 5 shows the supply ratio that results in the best area-efficiency metric for the given benchmark suite mapped to this architecture, and column 6 shows the area-efficiency metric.

These results show that hard bus-based crossbars provide an area-efficiency benefit over a hard single bit crossbar depending on the how much of the bus is utilized. The 4-bit hard bus-based crossbar needs to have a bus utilization of 2 or more to be more area-efficient compared to the hard single bit crossbar. Similarly, the 8-bit hard based crossbar with a bus utilization of 3 and the 16-bit hard based crossbar with a bus utilization of 5 are more area-efficient than the hard single bit crossbar.

We can conclude that the hard bus-based crossbar is a more area-efficient architecture for a hard crossbar included on an FPGA if the designs have sufficient bus utilization in the target market.

Table 5.13: Area-efficiency results for hard 64-64 bus-based crossbars

| Crossbar Type | Bus Size on FPGA | Design crossbar size | Bus utilization by benchmark | Best Supply Ratio | Area-Efficiency Metric |
|---|---|---|---|---|---|
| 64-64 | 1 | 64-64 | 1 | 1:14 | 1.075 |
| 64-64 | 4 | 64-64 | 1 | 1:14 | 1.030 |
| 64-64 | 4 | 64-64 | 2 | 1:14 | 1.088 |
| 64-64 | 4 | 64-64 | 3 | 1:18 | 1.119 |
| 64-64 | 4 | 64-64 | 4 | 1:18 | 1.139 |
| 64-64 | 8 | 64-64 | 1 | 1:72 | 0.996 |
| 64-64 | 8 | 64-64 | 2 | 1:13 | 1.047 |
| 64-64 | 8 | 64-64 | 3 | 1:14 | 1.082 |
| 64-64 | 8 | 64-64 | 4 | 1:16 | 1.105 |
| 64-64 | 8 | 64-64 | 5 | 1:18 | 1.121 |
| 64-64 | 8 | 64-64 | 6 | 1:18 | 1.134 |
| 64-64 | 8 | 64-64 | 7 | 1:20 | 1:144 |
| 64-64 | 8 | 64-64 | 8 | 1:20 | 1:153 |
| 64-64 | 16 | 64-64 | 1 | 1:40 | 0.980 |
| 64-64 | 16 | 64-64 | 2 | 1:20 | 1.002 |
| 64-64 | 16 | 64-64 | 3 | 1:15 | 1.033 |
| 64-64 | 16 | 64-64 | 4 | 1:15 | 1.056 |
| 64-64 | 16 | 64-64 | 5 | 1:14 | 1.076 |
| 64-64 | 16 | 64-64 | 6 | 1:15 | 1.091 |
| 64-64 | 16 | 64-64 | 7 | 1:13 | 1.099 |
| 64-64 | 16 | 64-64 | 8 | 1:18 | 1.113 |
| 64-64 | 16 | 64-64 | 9 | 1:20 | 1.201 |
| 64-64 | 16 | 64-64 | 10 | 1:18 | 1.128 |
| 64-64 | 16 | 64-64 | 11 | 1:18 | 1.135 |
| 64-64 | 16 | 64-64 | 12 | 1:20 | 1.141 |
| 64-64 | 16 | 64-64 | 13 | 1:20 | 1.146 |
| 64-64 | 16 | 64-64 | 14 | 1:20 | 1.149 |
| 64-64 | 16 | 64-64 | 15 | 1:20 | 1.154 |
| 64-64 | 16 | 64-64 | 16 | 1:20 | 1.157 |

### 5.5.4 Effectiveness of Bus-based Hard Crossbars with Shadow Clusters

Next, we include shadow clusters with the hard crossbars on an FPGA to measure how shadow clusters change the quality of an architecture that includes either the hard single bit or hard bus-based crossbar.

We perform the same experiment as described in the last section, but here, we combine shadow clusters with the hard crossbars to observe what effect shadow clusters have on the area efficiency of these hard circuits.

Table 5.14: Area-efficiency results for hard 64-64 bus-based crossbars combined with shadow clusters

| Crossbar Type | Bus Size on FPGA | Design crossbar size | Bus utilization by benchmark | Best Supply Ratio | Area-Efficiency Metric |
|---|---|---|---|---|---|
| 64-64 | 1 | 64-64 | 1 | 1:7 | 1.149 |
| 64-64 | 4 | 64-64 | 1 | 1:5 | 1.111 |
| 64-64 | 4 | 64-64 | 2 | 1:7 | 1.161 |
| 64-64 | 4 | 64-64 | 3 | 1:9 | 1.185 |
| 64-64 | 4 | 64-64 | 4 | 1:12 | 1.197 |
| 64-64 | 8 | 64-64 | 1 | 1:4 | 1.071 |
| 64-64 | 8 | 64-64 | 2 | 1:5 | 1.128 |
| 64-64 | 8 | 64-64 | 3 | 1:7 | 1.156 |
| 64-64 | 8 | 64-64 | 4 | 1:8 | 1.174 |
| 64-64 | 8 | 64-64 | 5 | 1:10 | 1.185 |
| 64-64 | 8 | 64-64 | 6 | 1:11 | 1.193 |
| 64-64 | 8 | 64-64 | 7 | 1:13 | 1.200 |
| 64-64 | 8 | 64-64 | 8 | 1:15 | 1.204 |
| 64-64 | 16 | 64-64 | 1 | 1:5 | 1.017 |
| 64-64 | 16 | 64-64 | 2 | 1:5 | 1.078 |
| 64-64 | 16 | 64-64 | 3 | 1:5 | 1.112 |
| 64-64 | 16 | 64-64 | 4 | 1:6 | 1.134 |
| 64-64 | 16 | 64-64 | 5 | 1:7 | 1.149 |
| 64-64 | 16 | 64-64 | 6 | 1:8 | 1.160 |
| 64-64 | 16 | 64-64 | 7 | 1:8 | 1.170 |
| 64-64 | 16 | 64-64 | 8 | 1:9 | 1.177 |
| 64-64 | 16 | 64-64 | 9 | 1:10 | 1.183 |
| 64-64 | 16 | 64-64 | 10 | 1:11 | 1.188 |
| 64-64 | 16 | 64-64 | 11 | 1:12 | 1.191 |
| 64-64 | 16 | 64-64 | 12 | 1:13 | 1.195 |
| 64-64 | 16 | 64-64 | 13 | 1:14 | 1.198 |
| 64-64 | 16 | 64-64 | 14 | 1:14 | 1.200 |
| 64-64 | 16 | 64-64 | 15 | 1:15 | 1.202 |
| 64-64 | 16 | 64-64 | 16 | 1:17 | 1.204 |

Table 5.14 shows the area-efficiency metrics for hard crossbars combined with shadow cluster, and this table has the same structure as Table 5.13.

Overall, the shadow cluster improves the area efficiency of the hard bus-based crossbar compared to including a hard bus-based crossbar on an FPGA, as expected. In each of the 4-bit, 8-bit, and 16-bit hard bus-based crossbars they are more area-efficient than the hard single bit crossbar if the bus utilization is 2, 3, and 5 bits respectively. This bus utilization result is the same result as in the previous experiment.

### 5.5.5 Effectiveness of Bus-based Hard Crossbars with Shadow Clusters in terms of Market Demand

The previous two experiments (crossbars with and without shadow clusters) indicate that a hard bus-based crossbar with sufficient bus utilization is more area efficient compared to a hard single bit crossbar regardless if the hard crossbar contains shadow clusters or not. In this experiment, we will study what market demand is needed to include either a hard single bit crossbar or a hard bus-based crossbar on an FPGA. In this way, we are asking, regardless of area-efficiency improvement of a hard crossbar, what type of hard crossbar architecture is best to include on an FPGA given the demand for crossbars in the target markets?

This experiment follows the same procedure as section 5.5.1 in which we measure the area effectiveness of each architecture determining the "frequency" of the use of crossbars in the designs which results in an area-neutral architecture compared to a purely soft FPGA.

Table 5.15 shows the area break-even points for FPGAs with hard 64-64 hard bus-based crossbars. Column 1 and column 2 shows the size of the hard bus-based crossbars and the bus bit-width. Column 3 and column 4 show the size of the crossbars in the benchmark and the bus utilization. Column 5 and 6 show the percentage of benchmarks containing crossbars and the average demand of the benchmark suite at the break-even point.

If we compare the break-even points of the hard bus-based crossbar architectures to the hard single bit crossbar architectures, then hard bus-based crossbar have a lower market demand when the benchmarks have a demand of at least 50% bus utilization. When the bus utilization is less than 50%, the hard single bit crossbar has a lower

Table 5.15: Area break-even points for hard 64-64 bus-based crossbars

| Crossbar Type | Bus Size on FPGA | Design crossbar size | Bus utilization by benchmark | Percent of Benchmarks with Crossbars | Average Demand Ratio |
|---|---|---|---|---|---|
| 64-64 | 1 | 64-64 | 1 | 8% | 1:188 |
| 64-64 | 4 | 64-64 | 1 | 13% | 1:115 |
| 64-64 | 4 | 64-64 | 2 | 8% | 1:188 |
| 64-64 | 4 | 64-64 | 3 | 5% | 1:300 |
| 64-64 | 4 | 64-64 | 4 | 4% | 1:375 |
| 64-64 | 8 | 64-64 | 1 | 21% | 1:71 |
| 64-64 | 8 | 64-64 | 2 | 12% | 1:125 |
| 64-64 | 8 | 64-64 | 3 | 9% | 1:167 |
| 64-64 | 8 | 64-64 | 4 | 6% | 1:250 |
| 64-64 | 8 | 64-64 | 5 | 5% | 1:300 |
| 64-64 | 8 | 64-64 | 6 | 5% | 1:300 |
| 64-64 | 8 | 64-64 | 7 | 4% | 1:375 |
| 64-64 | 8 | 64-64 | 8 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 1 | 45% | 1:33 |
| 64-64 | 16 | 64-64 | 2 | 19% | 1:79 |
| 64-64 | 16 | 64-64 | 3 | 14% | 1:107 |
| 64-64 | 16 | 64-64 | 4 | 12% | 1:125 |
| 64-64 | 16 | 64-64 | 5 | 10% | 1:150 |
| 64-64 | 16 | 64-64 | 6 | 9% | 1:167 |
| 64-64 | 16 | 64-64 | 7 | 8% | 1:188 |
| 64-64 | 16 | 64-64 | 8 | 6% | 1:250 |
| 64-64 | 16 | 64-64 | 9 | 6% | 1:250 |
| 64-64 | 16 | 64-64 | 10 | 5% | 1:300 |
| 64-64 | 16 | 64-64 | 11 | 5% | 1:300 |
| 64-64 | 16 | 64-64 | 12 | 5% | 1:300 |
| 64-64 | 16 | 64-64 | 13 | 5% | 1:300 |
| 64-64 | 16 | 64-64 | 14 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 15 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 16 | 4% | 1:375 |

required demand.

Next, we include shadow clusters with the hard bus-based crossbars to measure how these circuits change the needed market demand.

Table 5.15 shows the same results as Table 5.16 except these area break-even points are for FPGAs with hard 64-64 hard bus-based crossbars combined with shadow clusters.

Table 5.16: Area break-even points for hard 64-64 bus-based crossbars with shadow clusters

| Crossbar Type | Bus Size on FPGA | Design crossbar size | Bus utilization by benchmark | Percent of Benchmarks with Crossbars | Average Demand Ratio |
|---|---|---|---|---|---|
| 64-64 | 1 | 64-64 | 1 | 3% | 1:500 |
| 64-64 | 4 | 64-64 | 1 | 13% | 1:115 |
| 64-64 | 4 | 64-64 | 2 | 6% | 1:250 |
| 64-64 | 4 | 64-64 | 3 | 4% | 1:300 |
| 64-64 | 4 | 64-64 | 4 | 3% | 1:375 |
| 64-64 | 8 | 64-64 | 1 | 13% | 1:115 |
| 64-64 | 8 | 64-64 | 2 | 6% | 1:250 |
| 64-64 | 8 | 64-64 | 3 | 4% | 1:300 |
| 64-64 | 8 | 64-64 | 4 | 4% | 1:300 |
| 64-64 | 8 | 64-64 | 5 | 4% | 1:300 |
| 64-64 | 8 | 64-64 | 6 | 4% | 1:300 |
| 64-64 | 8 | 64-64 | 7 | 3% | 1:375 |
| 64-64 | 8 | 64-64 | 8 | 3% | 1:375 |
| 64-64 | 16 | 64-64 | 1 | 38% | 1:39 |
| 64-64 | 16 | 64-64 | 2 | 16% | 1:94 |
| 64-64 | 16 | 64-64 | 3 | 11% | 1:136 |
| 64-64 | 16 | 64-64 | 4 | 9% | 1:167 |
| 64-64 | 16 | 64-64 | 5 | 6% | 1:250 |
| 64-64 | 16 | 64-64 | 6 | 6% | 1:250 |
| 64-64 | 16 | 64-64 | 7 | 5% | 1:300 |
| 64-64 | 16 | 64-64 | 8 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 9 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 10 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 11 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 12 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 13 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 14 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 15 | 4% | 1:375 |
| 64-64 | 16 | 64-64 | 16 | 4% | 1:375 |

Shadow clusters combined with the hard crossbars change the break-even points even more significantly. With shadow clusters, the hard bus-based crossbar compared to the hard single bit crossbar is area-beneficial when almost 100% of the bits are used in the bus. This is true for a 4-bit and 8-bit bus-based crossbar, and a 16-bit bus based crossbar is never as good as a hard single bit crossbar combined with shadow clusters.

This is the case for one reason; a shadow cluster allows each tile of the hard crossbars, bus-based or not, to be used. The gain factor area benefits previously observed for hard bus-based crossbars is dominated by the fact that the hard single bit crossbar is small, provides an area benefit, and can always be used when combined with a shadow cluster.

The conclusion here is that if the architect is concerned with the market demand needed to include a hard crossbar on an FPGA, then a hard single-bit crossbar combined with a shadow cluster is the hard circuit that results in the lowest needed market demand for an architecture to be area neutral with a purely soft FPGA and this may be a factor when deciding to include a hard crossbar on an FPGA.

## 5.6 General Equation for Area Efficiency of Shadow Clusters

We now discuss the equations that calculate the area efficiency of an FPGA with and without shadow clusters for a particular benchmark given demand ratio. These equations calculates the area efficiency metric we have been generating using our measurement methodology in the last two chapters. There are two equations needed; the first equation applies when the demand ratio of the benchmark is greater than or equal to the supply ratio of the FPGA:

$$AreaEfficiencyMetric = \frac{Area_{HardCircuit} + \frac{1}{R_s} Area_{SoftLogicTile}}{Area_{HardCircuit+ShadowCluster} + \frac{1}{R_s} Area_{SoftLogicTile}} \quad (5.3)$$

The second equation applies when the demand ratio is less the supply ratio:

$$AreaEfficiencyMetric = \frac{Area_{HardCircuit} + \frac{1}{R_s} Area_{SoftLogicTile}}{Area_{HardCircuit+ShadowCluster} + \frac{1}{(R_s - R_d)} Area_{SoftLogicTile}} \quad (5.4)$$

In both equations, the area of the hard circuit (with or without shadow clusters) is

added to the area of the soft logic cluster tile multiplied by the amount of soft logic needed, and the area of an FPGA without shadow clusters is divided by the area of an FPGA with shadow clusters. These two equations apply when a benchmark is mapped using algorithm 1 first introduced in Chapter 5, Section 3.3.1.

The area of the soft logic cluster tile in the divisor of the second equation, which represents and architecture with shadow clusters and a demand lower than the supply, is multiplied by the supply ratio subtracted by the demand ratio. The reasoning here is that the additional unused hard circuits that are used as shadow clusters will reduce the number of soft logic cluster tiles needed in the architecture, thus changing the observed supply ratio. In a way, some of the hard circuits with shadow clusters are acting as soft logic cluster tiles.

For any arbitrary hard circuit, the area efficiency benefit of such a hard circuit combined with shadow clusters can be calculated using the the above equations for a given benchmark and the area costs for the programmable routing, the shadow cluster, and the logic for the hard circuit. This allows for an upfront analysis of the benefit of shadow clusters for a given hard circuit.

## 5.7 Summary

In this chapter, we introduced a hard crossbar as a hard circuit to include on an FPGA. Hard crossbars have not been included in FPGAs since there are not sufficient number of designs that use hard crossbars. We measured how hard crossbars combined with shadow clusters improve these FPGA's area efficiency such that the frequency of hard crossbars appearing in designs is reduced so that the FPGA is area neutral with a purely soft programmable logic FPGA.

Our measurements show that in all cases, the combination of a shadow cluster and a hard crossbar results in an architecture that needs much fewer of the designs to employ hard crossbars. Our results show that a hard bus-based crossbar will provide a greater area-efficiency benefit over hard single bit crossbars if minimum bus utilization is met in the target market. When an architect is concerned with the demand needed by the market to achieve an area-neutral architecture compared to a purely soft logic FPGA,

then a hard single bit crossbar combined with shadow clusters is the best choice.

The reason shadow clusters are so effective when combined with hard crossbars is that in terms of area this hard circuit is very similar to a soft logic cluster tile. Arguably, the hard crossbar combined with shadow cluster represents soft fabric heterogeneity more than tile-based heterogeneity meaning the tile is always useable as a piece of soft logic with additional specific functionality that comes at a small area increase in the tile.

In the next chapter, we shift our attention from architectural improvement of heterogeneous FPGAs to the CAD tools targeting heterogeneous FPGAs. We will describe a tool that we have created to efficiently target heterogeneous FPGAs at the RTL level.

# 6 A Verilog RTL Front-End Synthesis Tool for Heterogeneous FPGAs

> The trouble with having an open mind, of course, is that people will insist on coming along and trying to put things in it.
>
> Terry Pratchett

## 6.1 Introduction

The previous three chapters in this dissertation have focused on creating heterogeneous FPGAs that are more area efficient by building them with the correct number of hard circuits and increasing the utilization of those hard circuits. In this chapter, the focus changes from heterogeneous FPGA architecture to heterogeneous FPGA CAD flow. Regardless of how well we design the heterogeneous FPGA architecture the quality of our mapped designs will only be as good as the CAD tools that map them to these architectures.

FPGAs with hard circuits increase the need for quality CAD algorithms to target these structures. Traditional "soft" logic mapping into LUTs [CC04, CH00, FRV91, CCD+92] is done after technology-independent logic optimization [CPD96, BL90, Bry86], but the mapping into coarse-grain structures such as multipliers and memories is much more appropriately done at the RTL synthesis level where these structures are more directly recognizable as discussed in Chapter 2, Section 2.4.

The purpose of this chapter is to present the algorithms to flexibly target different hard circuits on an FPGA. These hard circuits are either tile-based heterogeneity or soft fabric heterogeneity, which we previously defined in Chapter 2, Section 2.3. Recall that

*soft fabric* heterogeneity is defined as the presence of hard circuits included in every tile in the programmable fabric and *tile-based* heterogeneity is defined as a differentiated tile(s) included on the FPGA as opposed to general programmable tiles. We will demonstrate that the front-end synthesis tool we have built achieves approximate parity with the quality of industrial-class RTL synthesis tools.

Previous work in targeting heterogeneous FPGAs with a front-end synthesis tool has been the work of industry. Both Altera and Xilinx incorporate front-end synthesis into their FPGA CAD flow tools called Quartus [Alt04c] and ISE [Xil04] respectively. Other popular front-end synthesis tools for FPGAs include Altera's Quartus [Alt04c], Xilinx's ISE [Xil04], Synplify [Syn03], Blast FPGA [Mag05], LeonardoSpectrum [Men01], and Design Compiler FPGA [Syn04]. To our knowledge this work is the first academic effort to build a front-end tool to target heterogeneous FPGAs.

This chapter is organized as follows: Section 6.2 describes the basic flow of the tool. Section 6.3 describes the mapping techniques needed in a high quality tool including the algorithm to flexibly target different hard circuits. In Section 6.5, we present comparison results for our tool versus Altera's Quartus RTL synthesis tool [Alt04c].

## 6.2 Overview of the flow for Front-end Synthesis

In this chapter, we present an HDL synthesis tool that converts a Verilog design into a flattened structural netlist. The input to this flow is a Verilog design [Ope93] and a description of the targeted FPGA including a hard circuit library that describes the functionality of the tile-based heterogeneous hard circuits. The output is a flattened netlist consisting of structures available on the targeted FPGA: either hard circuits or primitive gates. The output netlist can be passed into Quartus' [Alt04c] FPGA CAD flow.

Figure 6.1 shows the major stages of the tool flow. First, a public-domain front-end parser, Icarus [Wil07], parses the Verilog [Ope93] design and generates a hierarchical representation of the design. This hierarchical representation is a tree structure where the leaves of the tree represent computational primitives such as memory, addition, and logic, and nodes in the tree represent a combination of primitives and other nodes

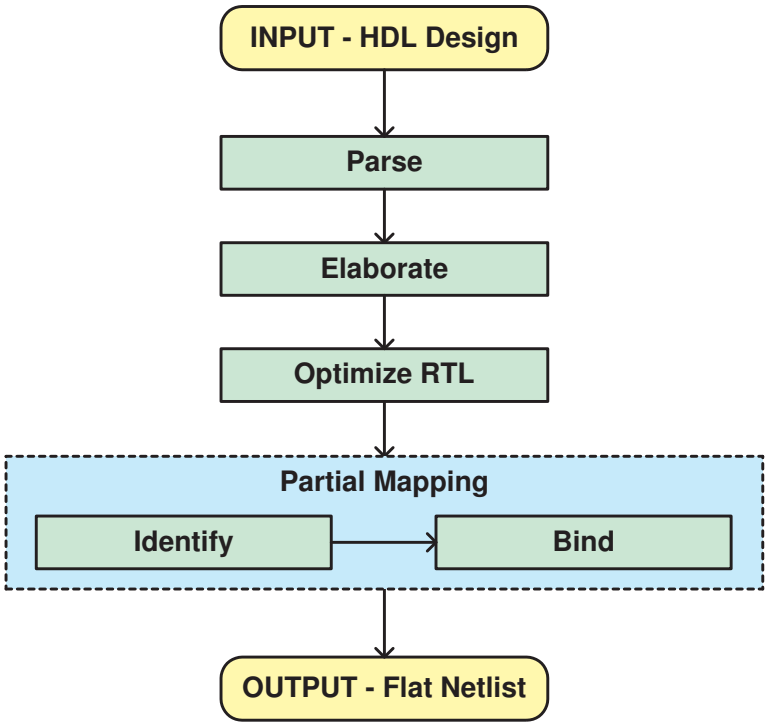


Figure 6.1: General flow to convert HDL design to a logic netlist

in what is called a module in Verilog.

An elaboration stage traverses this hierarchical representation of the design to create a flat netlist that consists of structures including logic blocks, memory blocks, *if* and *case* blocks, arithmetic operations, and registers. We call these netlist structures, primitives.

The next stage in the flow performs simple optimizations on this netlist. These optimizations include examining adders and multipliers for constant inputs to possibly shrink the size of the computation, collapsing multiplexers, and detecting and re-encoding finite state machines to be more efficient. These mappings are discussed in more detail in Section 6.3.1.

The next stage in the flow is partial mapping where parts in the netlist are first identified and then bound to available hard circuits on the FPGA. In the identification phase of partial mapping, the algorithm searches for parts of the design that could be

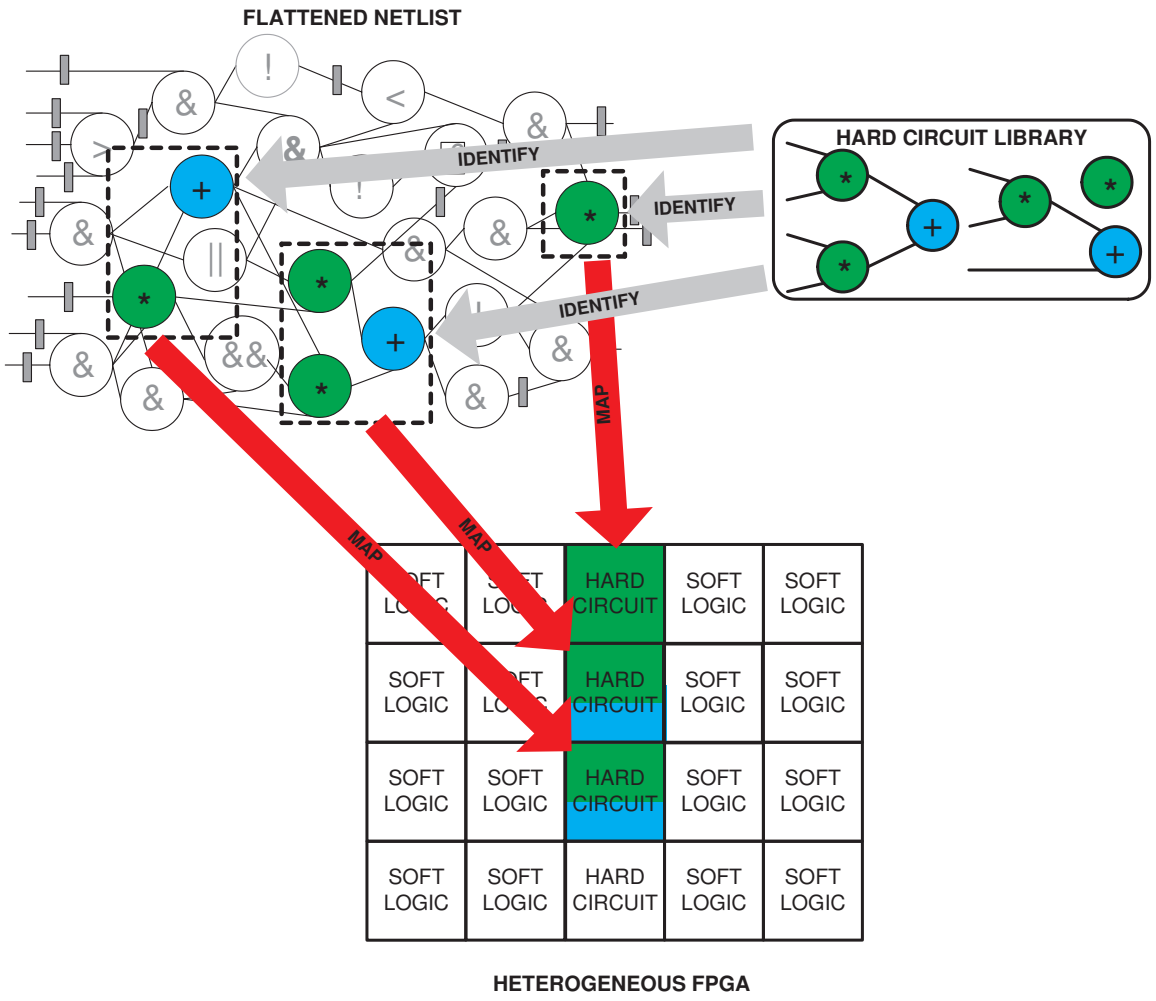


Figure 6.2: The partial mapping process

mapped to hard circuits on the target FPGA. These hard circuits include registers, adders, multipliers, and memories. In section 6.3.2, we discuss how important it is to identify and map registers and adders to an efficient implementation on the target FPGA.

Figure 6.2 shows an abstract illustration of the partial mapping process for identifying and mapping multipliers and more complex circuits that include multipliers. A hard circuit library, which is included with the input architectural description of the FPGA,

is used to identify parts of the netlist that could be mapped to the hard circuits on the FPGA. The binding stage then determines how to map these identified circuits to the heterogeneous FPGA. Complex hard circuits, such as the two multipliers joined by an addition seen in Figure 6.2, are searched for in the netlist using a matching algorithm, and this algorithm is discussed in Section 6.3.3.

The binding stage determines how each part of the netlist will be implemented. This is done by mapping parts of the netlist to hard circuits, soft programmable logic, or a mixture of both. In our experience, the binding stage only needs to make simple decisions; if a hard circuit exists that implements a piece of the netlist then bind it to that hard circuit. This choice always results in a faster and smaller mapped design for the designs we have experimented with. In Figure 6.2 the three identified multiplier circuits can be bound to hard circuits on the example FPGA since this FPGA includes four hard circuits, and therefore, the binding occurs. The specific location of these hard circuits is not determined at the binding stage and the decision is left to the downstream placement algorithm.

The output from our flow is a flat netlist consisting of connected complex logic structures and primitive gates.

## 6.3 Mapping Techniques

One of the key goals in the work in this chapter is to create a tool that achieves quality of results comparable to an industrial front-end synthesis tool. To achieve this we present several of the key optimizations discovered in the process of creating the high-quality tool. First, we discuss optimizations that are done immediately after flattening the original design. These include arithmetic optimizations, finite state machine re-encoding, and multiplexer collapsing optimizations. In subsequent sections, we discuss the partial mapper and how it efficiently maps registers and additions to a target FPGA and the steps that explicitly identify and bind tile-based hard circuits.

### 6.3.1 Mapping Soft Structures to an FPGA

We perform three optimizations that affect the final speed and area results of a design mapped to a heterogeneous FPGA. Specifically, these optimizations are arithmetic operations, detect and re-encode state machines to one-hot encoded, and collapse multiplexers.

#### Arithmetic Optimizations

For multiplications and additions, it is possible to shrink the size of the arithmetic operation by pushing constant values of one and zero through logic and computation if possible in a optimization called, constant propagation [Kil73]. These operations are best done at RTL synthesis instead of logic synthesis since the result of this optimization affects partial mapping decisions since a smaller arithmetic operation might fit into a smaller fraction of hard circuits on the FPGA.

Figure 6.3 (a) shows how additions can be improved by replacing "0" constant bits for low order bits with a wire. Similarly, Figure 6.3 (b) shows how "0" constants at most significant bits can shrink the size of an unsigned multiplier by simply eliminating these most significant constant bits. These optimizations can save significant numbers of hard multiplier units on the target FPGA. For example, if an 11x11 unsigned multiplier is to be implemented on a Stratix I, and the multiplier can be shrunk by 2 bits, then only one 9x9 multiplier is needed as opposed to two 9x9 multipliers.

We also look for arithmetic operations with constant inputs. Some of these arithmetic operations can be implemented more efficiently in soft logic as opposed to simply mapping them to hard circuits. For example, a multiplier with a constant input can be more efficiently implemented using a Read Only Memory (ROM) or shift operations depending on the bit-width and value of the constant. We also implement $A + B + 1$ or $A - B + 1$ in one adder using the carry in signal that commonly exists in the adder capabilities of a BLE.

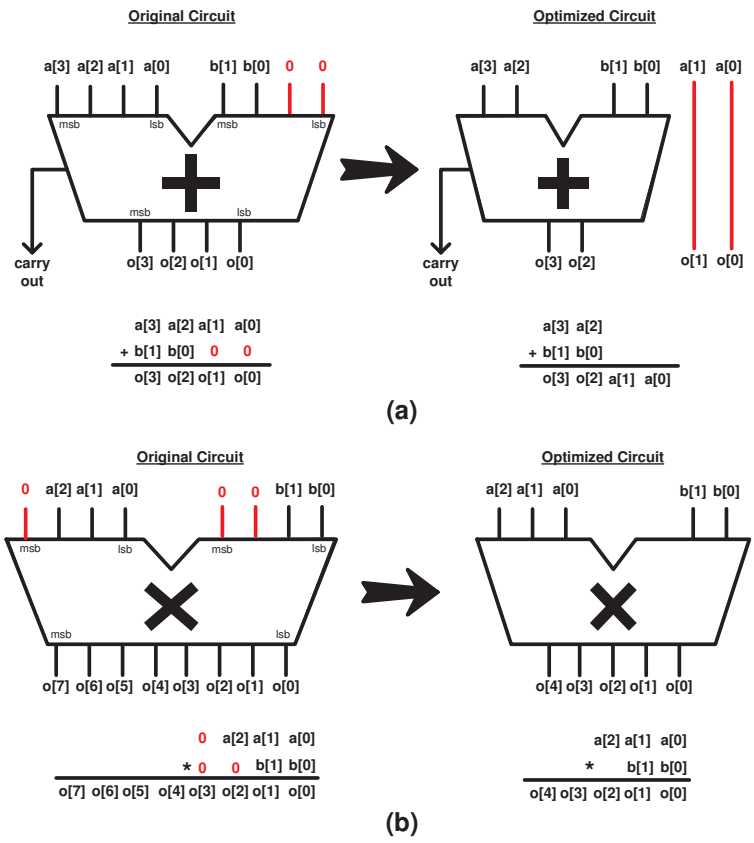


Figure 6.3: Two examples of simple arithmetic optimizations

### One-hot Re-Encoding of Finite State Machines

The implementation of finite state machines can benefit from being one-hot encoded when implemented on an FPGA [Gol93]. This results in one flip-flop per state, which reduces the number of bits needed to be written during each calculation of the next state. Since less bits are written per cycle, the amount of multiplexing and routing is also reduced, which often makes the finite state machine implementation both faster and smaller.

In the tool flow, finite state machines are first detected, and then states are re-encoded to a one hot-encoding. We identify state machines by checking if 3 characteristics of a potential state machine are satisfied. These characteristics are:

1. There is a case statement in a Verilog combinational *always* block.
2. The signal being compared in the combinational case statement comes from a register. This register is the state register.
3. There is a feedback loop to the state register through a multiplexer, and the only inputs to this multiplexer come from either a feedback loop from the state register or constant inputs that represent state encoded values.

These rules are sufficient, but not necessary. Therefore, not all finite state machines will be detected using these characteristics.

### Multiplexer Collapsing

**Verilog HDL**

```
module control (clock, a, b, c, d, reg1, reg2);
        input [1:0]a, b, c, d, clock;
        output reg1, reg2;
        reg reg2;
        assign reg1 = (b == 0) ? d : c;

        always @(posedge clock)
                case(a)
                        2'b00: reg2<=1'b0;
                        2'b01: reg2<=1'b1;
                        2'b10: reg2<=reg1;
                endcase
endmodule
```

Figure 6.4: Control statements in Verilog, which become multiplexers

Multiplexers are frequently used in circuit designs. Figure 6.4 shows Verilog code, which has both a *case* and an *if* structure and both of these structures are commonly implemented as decoded multiplexers.

We can collapse *case* and *if* structures together and group common signals together to reduce the number of logic levels in the data path potentially speeding up these paths.

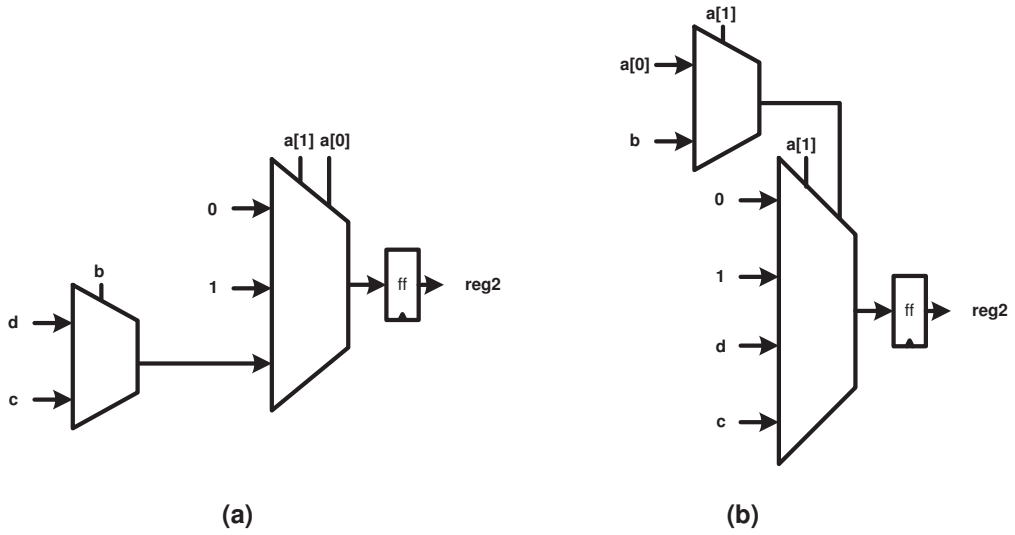


Figure 6.5: Example in Figure 6.4 collapses into a lower number of multiplexer levels

Figure 6.5(a) shows the initial implementation of the Verilog code in Figure 6.4, and Figure 6.5(b) shows how the two multiplexers can be collapsed at the cost of moving the logic to the control signals. This optimization increases the depth of logic for the control signals, with the benefit of amortizing logic and decreasing the number of logic levels on the data path.

We collapse multiplexers assuming that the additional logic complexity added to the control signals will not affect the overall speed of the design. This is not always the case, and to improve multiplexer collapsing, the algorithm would need to estimate path delays of the circuit and determine if the additional delay on the control paths will affect the overall speed of the design. These optimizations are similar to Metzgen *et. al.*'s work [MN05] in which they use a technique called compression to collapse 2:1 multiplexers together into 4:1 multiplexers so that they can be efficiently implemented on a 4-LUT FPGA. We, however, do not just collapse 2:1 multiplexers and compress all multiplexers.

### 6.3.2 Mapping to Soft Fabric Heterogeneity on FPGAs

To achieve industrial-quality results on modern FPGAs it is essential to correctly target the special-purpose adder structures that exist on all modern FPGAs and the features of the flip-flops that are provided within the soft fabric of the FPGA. We assume, for this discussion, that the target BLEs have one register and can implement either a logic function or one or more bits of addition or subtraction. This is similar to the BLEs on modern FPGAs available from both Altera [Alt03] and Xilinx [Xil05]. Mapping to soft fabric heterogeneity is done at the partial mapping stage in the tool flow.

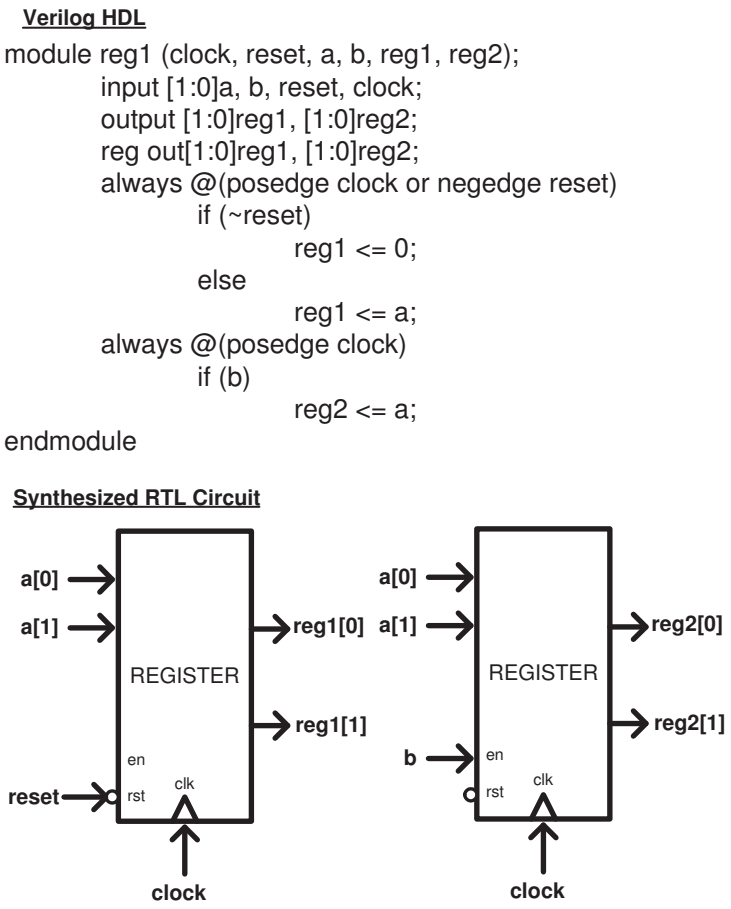


Figure 6.6: Verilog design with registers and a possible implementation

For example, when mapping parts of a design to flip-flops it is important to detect

logic that can be used for a flip-flop's enable and reset signals, which, if undetected, will result in an FPGA implementation that is both larger and slower. Figure 6.6 shows Verilog code that implements one 2-bit register with a reset signal, which if not detected correctly will cause the synthesis of extra multiplexer logic. Figure 6.6 also shows a Verilog design along with a logic schematic of a design in which the signal $b$ is a clock enable signal for the flip-flop. If $b$ is not properly detected and connected to the enable signal pin on the flip-flop, then a feedback loop through a multiplexer is needed to implement equivalent logic functionality.

### 6.3.3 Mapping to Tile-based Heterogeneity on FPGAs

Proper use of hard circuits is important since they can considerably decrease the effective area and improve speed of a design implemented on an FPGA. The most common tile-based hard circuits included on modern FPGAs are multipliers and block memories.

We call the process of mapping parts of a design to the hard circuits on an FPGA, partial mapping, because portions of the circuit are mapped to circuits available on the FPGA. Some logic is in a form that still needs to be technology mapped into BLEs and clustered into soft logic cluster tiles using downstream CAD algorithms. Partial mapping consists of an identification stage and a binding stage where the identification stage identifies pieces of the netlist that can be mapped to hard circuits and the binding stage makes the decision of how to map all pieces in the design to the downstream CAD flow. The following is a discussion of both of these stages for identifying and binding parts of the netlist that will map to tile-based hard circuits.

The initial inputs to the identification stage are a hard circuit library (which is included as part of the input architectural description to the entire front-end synthesis flow) and the flattened netlist. The hard circuit library describes what each hard circuit can implement. The output of the identification stage is the same flattened netlist with labels that describe what hard circuits each part of the netlist could be mapped to.

Identifying memories and multipliers in the input netlist is simple since these exist as primitives in the flattened netlist. Multipliers are instantiated in Verilog with the "*" symbol, which is treated as a primitive within the design netlist by the parser and elaborator. Similarly, memories in Verilog are array accesses where the array accessing

signal represents the address signal, and most parsers and elaborators treat memories as primitives. Since both of these structures are simple, no additional processing is needed to identify them.

If a hard circuit has additional (typically programmable) functionality, it is a more difficult problem to identify these structures in the netlist. For example, the Altera DSP block [Alt03] can be configured to implement multiplication, multiply accumulation, and multiply summation. Multiply summation is the addition of two or more multiplications.

```
Input: Flattened netlist and hard circuit library
Output: Flattened netlist with identified possible hard circuit mappings
foreach f = function available on a hard circuit do
    seed = a unique part of f;
    foreach e = element in netlist of type seed do
        if subgraph_match(f, e) then
            label subgraph(e) with f;
        end
    end
end
```

**Algorithm 4**: The matching algorithm

A matching algorithm searches for instances of these complex hard circuits in the netlist. The initial input to the matching algorithm is a flattened netlist consisting of logic and computation and the hard circuit library. The goal of the algorithm is to find all parts of the input design that could make use of a hard circuit available on the FPGA. Algorithm 4 shows the steps in the matching algorithm. In this algorithm, a unique part of each complex hard circuit is used as a seed primitive in the netlist that allows us to quickly search candidate matches in the design netlist. The output of this algorithm is the netlist labelled with possible hard circuit implementations for each piece of the netlist.

Figure 6.7 shows a sample netlist and a hard circuit library. In the figure, the dotted lines surrounding multiplier and additions in the netlist represent matchings of hard circuits included in the library.

This matching problem is a form of sub-graph isomorphism, which has been used in

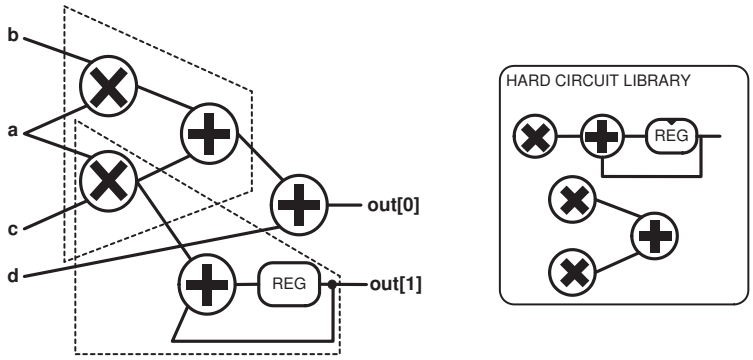


Figure 6.7: Library describing hard circuits on an FPGA and matching them in a netlist

instruction generation for reconfigurable and processor systems [KOMBS01, CFHZ04] and sub-circuit extraction for technology mapping [OEGS93].

We have found matching to be simple and successful for our purposes since the size of the sub-graphs representing complex hard circuits is very small consisting of at most five primitives - which makes the matching very fast. Secondly, while matching complex hard circuits, which all include multipliers, we use a multiply primitive as a seed to start each search, and multiplies appear infrequently in designs meaning that the matching is only run a small number of times.

Once the identification stage is complete, the next stage is binding, which decides how to map all pieces in the netlist to either soft logic or hard circuits.

**Input**: Flattened netlist with labelled possible hard circuit mappings
**Output**: Flattened netlist with each primitive marked either soft or hard
**foreach** *n = node of the netlist* **do**
  **if** *n not yet bound* **then**
    **foreach** *p = possible hard circuit mapping type for n* **do**
      count how many un-bound nodes in p there are;
      bind n and all nodes of p with the greatest count to p;
    **end**
  **end**
**end**

**Algorithm 5**: The binding algorithm

The input to the binding algorithm is the netlist labelled with the possible hard

circuits each of the pieces of the netlist can be mapped to. The output is the same netlist, but each piece of the netlist has, at most, one label indicating what that piece will be mapped to - either soft logic (which means no label), hard circuits, or a mixture of both. Note that a mixture of both means the computation is larger than what can fit in a hard circuit and its implementation on the FPGA will need to use possibly several hard circuits and soft logic. For multipliers, memories, and complex structures that can be implemented in a tile-based heterogeneous hard circuit we will choose the hard circuit that covers the largest portion of the netlist. Algorithm 5 shows the steps taken to bind all the primitives in the netlist to hard circuits.

When a tool implementing this algorithm outputs a netlist, the pieces of the netlist that are bound to complex hard circuits, multipliers, or memories are mapped to a macro circuit structure that later stages of an industrial CAD flow will implement on the FPGA. An example of a macro circuit structure supported by an industrial flow is LPMs [LPM93]; LPMs are used in the Quartus [Alt04c] flow so that designers can describe specific circuits they want to use. For example, there exists an LPM describing a multiply accumulator available on a Stratix I [Alt03].

## 6.4 CAD Flow and Verification

We have built a front-end HDL synthesis tool that implements the algorithms presented here and is designed to interface with Altera's Quartus CAD flow [Alt04c]. To attach to a CAD flow, first, our tool converts Verilog designs into structural Verilog netlists consisting of gate primitives and LPMs (or the equivalent of LPMs) targeted for a particular flow. These outputs are passed to the industrial flow that follows with downstream synthesis, placement, and routing to generate area and timing results. One of the major benefits of targeting industrial CAD flows, such as Quartus, is that it allows us to obtain real speed and area results on an industrial FPGA.

To verify that we generate functionally correct designs, we have built test benches for three of the benchmarks. These test benches include both the original Verilog design and a Verilog gate/LPM-level netlist mapped by our tool. The test bench generates inputs to both designs from one common source, and the outputs are joined through

XOR gates so that during simulation if any of the outputs generate a "1" it means that the two designs are not generating the same values, and there is an error in the design mapped by our tool. Our tool has been verified by simulating benchmarks cordicB, moleculeDynamics, and firB in ModelSim with random vectors. Aside from testing these three benchmarks, we have verified many Verilog structures including logic gates, control statements, arithmetic operations, and storage units.

## 6.5 Results

One of the key goals of this work is to build a front-end synthesis tool that generates results comparable to industrial front-end synthesis tools that target FPGAs. In this section, we describe our benchmarking methodology, and then we present and discuss the head-to-head comparison. Finally, we show the specific value of each of the different mapping optimizations described in Section 6.5.3.

The name of our front-end synthesis tool is *Odin*, and for the remainder of this chapter we will refer to it when distinguishing between the results generated by our tool and other tools.

### 6.5.1 Benchmarking Methodology

For this comparison, we use Quartus' CAD flow and map a set of benchmarks to Stratix I FPGAs [Alt03]. We use version 4.1 of Quartus and run our benchmarks through two CAD flows consisting of Quartus alone and Odin interfaced with Quartus. Figure 6.8 shows the two CAD flows.

Our benchmarks are Verilog HDL designs described in Chapter 3, Table 3.3 and Appendix B.

### 6.5.2 Comparison between Odin and Quartus' Front-End Synthesis Tool

Table 6.1 and Table 6.2 show the comparison of speed and area of both CAD flows for each benchmark. To reduce the experimental "noise" associated with placement

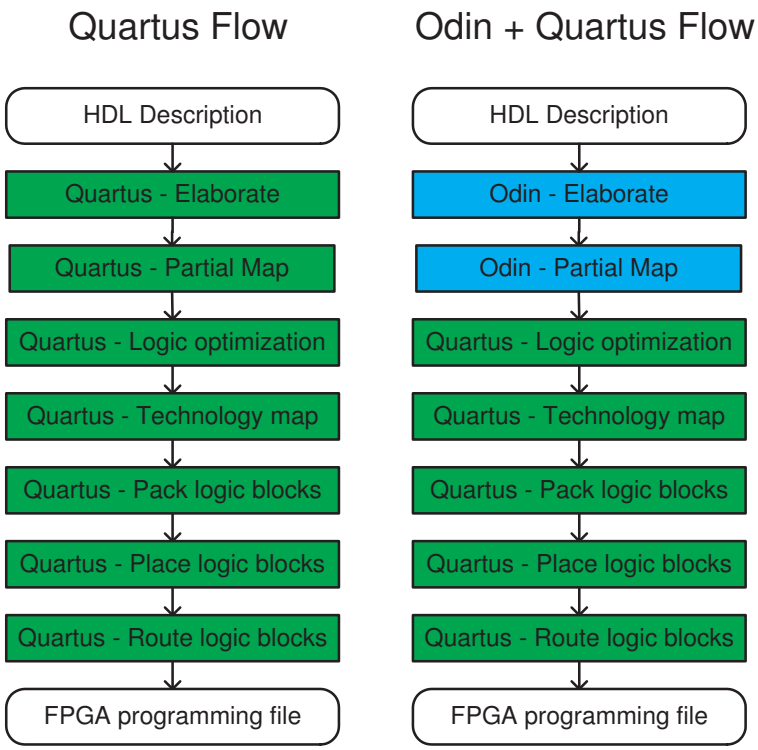


Figure 6.8: An industrial CAD flow and Odin joined into an industrial CAD flow

and routing, the results are averaged over 5 random seeds (where using random seeds attempts to average out the noise in the results found when using CAD optimization algorithms). In Table 6.1, columns 2 and 3 show a comparison between the number of LEs (which is Altera's BLE) used on a Stratix I FPGA mapped by Quartus and mapped by Odin interfaced with Quartus. Column 4 is a ratio that indicates how Odin performs compared to Quartus. For any comparison ratio, when it is less than one means that Odin is performing better than Quartus. Similarly, in Table 6.1 column 5, 6, and 7 show the number of 9-bit by 9-bit DSP blocks and a comparison ratio. Table 6.2 shows a speed comparison where column 2 and 3 show the maximum operating frequency of each circuit mapped by a pure Quartus and the Odin+Quartus flow. Column 4 shows the comparison ratio.

Overall, these results show that Odin generates comparable results to Quartus' front-end as it is only slightly worse in all cases. For the Stratix I FPGA the geometrically

Table 6.1: Area comparison between designs mapped by Odin and Quartus

| | Number of Logic Elements | | | Number of 9x9 DSP blocks | | |
|---|---|---|---|---|---|---|
| Designs | A - Quartus | B - Odin with Quartus | Ratio (B/A) | C - Quartus | D - Odin with Quartus | Ratio (D/C) |
| fft | 2374 | 3190 | 1.34 | 28 | 32 | 1.14 |
| iirA | 289 | 501 | 1.73 | 7 | 7 | 1.00 |
| iirB | 297 | 338 | 1.14 | 12 | 10 | 0.83 |
| firA | 84 | 84 | 1.00 | 4 | 4 | 1.00 |
| firB | 1598 | 1591 | 1.00 | 48 | 48 | 1.00 |
| firC | 998 | 548 | 0.55 | 0 | 17 | 0.00 |
| diffeqA | 221 | 271 | 1.23 | 24 | 40 | 1.67 |
| diffeqB | 512 | 369 | 0.72 | 24 | 40 | 1.67 |
| stereoVisionA | 17765 | 17145 | 0.97 | 80 | 96 | 1.20 |
| stereoVisionB | 35554 | 36194 | 1.02 | 176 | 144 | 0.82 |
| stereoVisionB_no_mem | 34279 | 33803 | 0.98 | 176 | 144 | 0.82 |
| rayTraceA | 2622 | 2679 | 1.02 | 27 | 27 | 1.00 |
| rayTraceA_no_mem | 2118 | 2815 | 1.33 | 27 | 27 | 1.00 |
| rayTraceB | 25056 | 28653 | 1.14 | 112 | 112 | 1.00 |
| rayTraceB_no_mem | 21557 | 29507 | 1.37 | 112 | 112 | 1.00 |
| oc45_cpu | 2191 | 3101 | 1.42 | 2 | 2 | 1.00 |
| reedSolDecoderA | 1151 | 1183 | 1.03 | 13 | 13 | 1.00 |
| reedSolDecoderB | 1799 | 1957 | 1.09 | 9 | 9 | 1.00 |
| molecularDynamics | 10542 | 14867 | 1.41 | 112 | 112 | 1.00 |
| cordicA | 591 | 838 | 1.42 | 0 | 0 | |
| cordicB | 2830 | 4104 | 1.45 | 0 | 0 | |
| MACA | 2864 | 2812 | 0.98 | 0 | 0 | |
| MACB | 9828 | 9720 | 0.99 | 0 | 0 | |
| crc33_d264 | 102 | 102 | 1.00 | 0 | 0 | |
| desArea | 1481 | 1305 | 0.88 | 0 | 0 | |
| desPerf | 4592 | 3838 | 0.84 | 0 | 0 | |
| stereoVisionC | 12433 | 12729 | 1.02 | 0 | 0 | |
| stereoVisionC_no_mem | 7281 | 7122 | 0.98 | 0 | 0 | |
| stereoVisionD | 170 | 134 | 0.79 | 0 | 0 | |
| rayTraceC | 766 | 909 | 1.19 | 0 | 0 | |
| rayTraceC_no_mem | 546 | 784 | 1.44 | 0 | 0 | |
| rayTraceD | 1519 | 2266 | 1.49 | 0 | 0 | |
| | | Average | 1.10 | | Average | 1.01 |

averaged area comparison ratio is 1.10 and the geometrically averaged speed ratio is 1.04. These ratios indicate that Odin is generating only slightly poorer mapped designs compared to Quartus, but for many of the benchmarks, we have quite comparable

Table 6.2: Speed comparison between designs mapped by Odin and Quartus

| | Speed in MHz | | |
|---|---|---|---|
| Designs | E - Quartus | F - Odin with Quartus | Ratio (E/F) |
| fft_258_6 | 101 | 146 | 0.70 |
| iirA | 85 | 83 | 1.03 |
| iirB | 116 | 109 | 1.06 |
| firA | 251 | 252 | 1.00 |
| firB | 84 | 75 | 1.11 |
| firC | 150 | 110 | 1.37 |
| diffeqA | 45 | 41 | 1.10 |
| diffeqB | 38 | 30 | 1.26 |
| stereoVisionA | 116 | 122 | 0.95 |
| stereoVisionB | 48 | 50 | 0.97 |
| stereoVisionB_no_mem | 53 | 57 | 0.97 |
| rayTraceA | 135 | 127 | 1.06 |
| rayTraceA_no_mem | 134 | 137 | 0.98 |
| ratTraceB | 45 | 52 | 0.87 |
| rayTraceB_no_mem | 48 | 54 | 0.89 |
| oc45_cpu | 86 | 61 | 1.40 |
| reedSolDecoderA | 86 | 83 | 1.04 |
| reedSolDecoderB | 68 | 54 | 1.27 |
| molecularDynamics | 42 | 35 | 1.20 |
| cordicA | 212 | 256 | 0.83 |
| cordicB | 167 | 223 | 0.75 |
| MACA | 107 | 99 | 1.09 |
| MACB | 83 | 75 | 1.11 |
| crc33_d264 | 0 | 0 | 0.00 |
| desArea | 235 | 194 | 1.21 |
| desPerf | 200 | 199 | 1.00 |
| stereoVisionC | Won't Fit | 120.16 | NA |
| stereoVisionC_no_mem | 163 | 146 | 1.11 |
| stereoVisionD_no_mem | 321 | 329 | 0.98 |
| rayTraceC | 120 | 128 | 0.94 |
| rayTraceC_no_mem | 125 | 139 | 0.90 |
| rayTraceD | 187 | 134 | 1.29 |
| | | Average | 1.04 |

results. One of the main reasons our tool generates results that are close to Quartus' front-end tool is because we have built Odin to deal with mapping functionality to both soft fabric and tile-based heterogeneity, and we have added the optimizations described

above.

### 6.5.3 Value of Specific Mapping Techniques in Odin

To see the effect the optimizations have on the quality of results generated by Odin we ran an experiment in which we turn on and off different mapping techniques. The mapping techniques for this experiment are:

1. Partial Mapping to the DSP block on the Stratix I
2. Arithmetic Optimizations
3. State Machine Identification and one-hot recoding
4. Multiplexer collapsing

We ran Odin on all the benchmarks for five different configurations and pass the mapped designs into the Quartus CAD flow to get speed and area results. Each configuration consists of different mapping techniques turned on or off. We use five configurations for this experiment where each subsequent configuration includes the mapping techniques of the previous configuration. The first configuration has all mapping techniques turned off. The second configuration only has mapping technique 1 turned on. The third technique has both mapping technique 1 and 2 turned on, and so on for the remaining three configurations where the fifth configuration has all techniques turned on.

In Figure 6.9(a), the bar graph shows how each mapping technique contributes to decreasing the number of LEs used to map the benchmarks to a Stratix I FPGA. All the mappings together provide a 4% decrease in the number of LEs used compared with the first configuration. We can see that *Multiplexer collapsing* contributes most of the LE savings at 81% of the 4% improvement.

Figure 6.9(b) shows the improvement of the number of mapped DSP blocks used. This metric is only affected by *Arithmetic optimizations*, where shrinking the size of multiplication decreases the number of DSP blocks by a total of 27%. This emphasizes

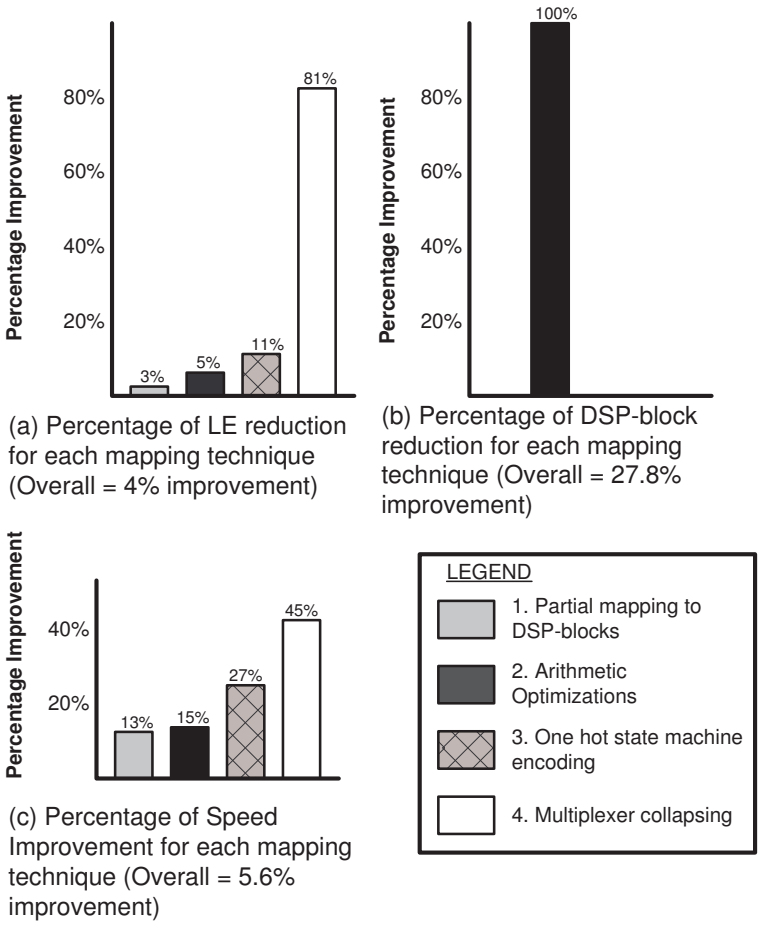


Figure 6.9: Impact on results for each of 5 optimization configurations

how important it is to propagate constants at front-end synthesis to ensure multipliers are mapped to the smallest implementation possible.

Figure 6.9(c) shows how the mapping techniques in Odin affect the speed of the benchmarks. The total speed improvement due to these techniques is 5.6%. Multiplexer collapsing results in the greatest improvement in speed at 2.5%.

## 6.6 Summary

In this chapter, we have presented a front-end Verilog RTL synthesis tool called Odin with publicly available source code. Our flow uses various mapping techniques to generate designs that are comparable in area and speed to Quartus' front-end tool. We have also shown how certain mapping techniques improve our results. These techniques, though simple, are important in mapping designs automatically and efficiently to modern FPGAs. Finally, we provide this software to the academic community hoping that the availability of this tool will allow researchers to pursue other avenues in front-end synthesis for FPGAs (*www.eecg.toronto.edu/∼jayar/odin/*).

# 7 Conclusions

> In the second scroll of Wen the Eternally Surprised a story is written concerning one day when the apprentice Clodpool, in a rebellious mood, approached Wen and spake thusly: "Master, what is the difference between a humanistic, monastic system of belief in which wisdom is sought by means of an apparently nonsensical system of questions and answers, and a lot of mystic gibberish made up on the spur of the moment?" Wen considered this for some time, and at last said: "A fish!" And Clodpool went away, satisfied.
>
> Terry Pratchett

## 7.1 Summary and Contributions

The modern FPGA consists of both soft logic fabric and hard circuits where the latter have been included on FPGAs to improve the speed, area, and power consumption of FPGAs for designs that map functionality to these circuits. The central question in FPGA architecture is which hard circuits to include. Each potential hard circuit needs to provide some sort of benefit (area, speed, or power), and have sufficient demand in the designs created by the intended users of the FPGAs.

In this dissertation, we have investigated ideas on how to increase the utilization of hard circuits to improve the quality of FPGAs in terms of area efficiency. Our approach is to architect better hard circuits or create efficient algorithms to map to these hard circuits.

This dissertation includes the following contributions:

In Chapter 3, we introduced heterogeneous FPGA definitions and terminology. Besides classifying the different types of hard circuits added to an FPGA, Chapter 3 introduces key architectural parameters: supply ratio, which describes an experimental FPGA in terms of contained hard circuits and soft logic, and demand ratio, which describes the same ratio but in terms of a circuit characteristic. These definitions were first published in [Ros04] with additional definitions in my thesis proposal as well as [JR05a]. In this chapter, we introduce a scientific measurement methodology to measure the benefit of hard circuits and we perform some basic experiments on the benefit of including hard multipliers on FPGAs.

In Chapter 4, we introduce a new architectural concept called shadow clusters. In this chapter, we describe how to create an FPGA with a hard circuit combined with a shadow cluster, we provided a measurement methodology to measure the area efficiency of this new concept, and we described the created synthetic benchmarks to statistically analyze our architectural concept under different market conditions. Some of the relevant results show that a modern commercial architecture (with a fixed ratio of multipliers to soft logic) would gain 4.7% in area efficiency by employing shadow clusters. In addition, every architecture we studied under "reasonable" conditions never show a loss of area efficiency when shadow clusters are used. Furthermore, we show that our most area-efficient architecture that employs shadow clusters is 12.5% better than the most area-efficient architecture without shadow clusters. This work was published in [JR06].

In Chapter 5, we extended our shadow cluster idea, employing the concept with hard circuits that were previously impractical to add to FPGAs because there are insufficient designs in the FPGA target market demanding the hard circuit. In this chapter, we use crossbars as the hard circuit to combine with shadow clusters, we show how to architect a hard crossbar, and we modify our measurement methodology to map crossbars in designs to hard crossbars on an FPGA. Additionally, there is significant focus placed on how to synthetically generate benchmarks to test out our idea and model a range of possible target markets. Our results show that it is more practical to add hard crossbars to FPGAs only if they are

combined with shadow clusters. For example, we show that the need for hard 32 full-way crossbars in the FPGA's target benchmark suite for its inclusion to appear area neutral changes from 12% of benchmarks needing to have crossbars to 3% for FPGAs with hard crossbars combined with shadow clusters.

In Chapter 6, we describe the algorithms in a front-end synthesis tool targeting heterogeneous FPGAs. We show that designs mapped by a tool that employs these algorithms generates area and speed results that are almost in parity with an industrial strength tool. This work was published in [JR05a], and the software has been made available to the general public under an open source license. Presently, the software has been downloaded over 150 times by unique users. This tool was also used in our work to implement a technique that maps multiplexers in the design to unused multipliers. Though this technique does not provide meaningful area savings, it provides the means to maximize utilization of an FPGA's hard multipliers. This work was published in [JR05b].

## 7.2 Future Work

There are several potential research directions this work could take in the future. In this section, we suggest a few directions that could be considered.

### 7.2.1 Shadow Clusters with Multiple Hard Circuits

We have examined how shadow clusters affect the area efficiency with respect to only one type of hard circuit included on an architecture. In this dissertation, the hard circuit included on an FPGA was either a multiplier or crossbar. An extension of this is to consider an FPGA in which there are more than one type of hard circuit. For example, consider an FPGA with both multipliers and crossbars.

The challenge in such research is how to model the FPGAs and the targeting benchmarks. In both cases, we need to re-examine how to define supply ratio and demand ratio to deal with multiple hard circuits. This work becomes even more complex when we consider that a hard circuit might be designed to implement two different types of

functions. For example, a multiplier and crossbar might be combined together in one tile and could be output programmed using multiplexers similar to our shadow cluster concept. Here, the mapping algorithms need to determine how to allocate functionality of a design to the hard circuits such that the optimization goals are satisfied.

This extension would bring our study of shadow clusters closer to industrial FPGAs that contain multiple types of hard circuits. This work is also necessary since FPGAs will arguably contain more hard circuits in the future. For example, both Xilinx's and Altera's FPGAs include both multiplier and memory tiles. Extending our methodology will determine what type of benefit we can expect by employing shadow clusters on these architectures.

### 7.2.2 Heterogeneous Soft Logic

In Chapter 4, we performed two experiments in which we artificially changed the size of the BLE to study what effects this would have on architectures with and without shadow clusters. These experiments were motivated when FPGA architects, at Altera and Xilinx, noted that modern soft logic cluster tiles have about 40% of the tile area dedicated to logic and 60% of the area is for programmable routing.

In the first of these experiments in section 4.6.6, the size of a shadow cluster and a regular soft logic cluster were artificially increased together to determine how this affected our shadow cluster architectures, and in the second experiment we kept the shadow cluster size constant while increasing the size of the normal soft logic cluster.

Stepping away from the shadow cluster concept, one can imagine an FPGA in which soft logic cluster tiles include different soft-fabric heterogeneity and consume different amount of silicon area. A similar track of research examined the affect of different sizes of LUTs and their effect on area [HR93], but no research has looked at a soft fabric that has, for example, some soft logic cluster tiles that include adder capabilities including carry chains and other tiles that do not have this capability. This research has become more relevant as the ratio between logic and routing area consumption changes in modern FPGAs.

One of the key assumptions to study before proceeding with this work is the quality of our automatic sizing of transistors. The accuracy of our tool will have significant

impact on this research. From discussions with Ian Kuon [Kuo07], the LUT and flip-flop are the main contributors to the 40% tile area in industrial FPGAs, meaning the additional functionality in the logic (such as adders) takes up a small portion of the area. Since this is the case, a heterogeneous fabric would probably included differently sized tiles as opposed to different functional capabilities, but the concept is similar. A heterogeneous soft logic fabric will trade-off speed for area, but in this case using transistor sizing.

The question remains, would a heterogeneous soft logic fabric provide an improvement in FPGA area efficiency while maintaining performance?

### 7.2.3 Extension of Shadow Clusters Employed with Low-Demand Hard Circuits

In Chapter 5, we examined how shadow clusters make it possible to add new hard circuits to an FPGA where previously these hard circuits would not be added since there are insufficient designs targeting these structures. We chose to use crossbars as the example circuit to demonstrate this concept.

With respect to area efficiency, the two factors affecting to the benefit of an included hard circuit on an FPGA are:

1. Soft logic savings - how many BLEs or soft logic cluster tiles a hard circuit saves when implementing design functionality.

2. Hard circuit size - the silicon area of a hard circuit.

The interesting thing with hard crossbars is that they take up relatively little silicon area (as per point 2), especially when compared to a circuit like a multiplier. Hard crossbars, however, save about the same amount of FPGA soft logic compared to a hard multiplier and an equivalent soft implementation.

It would be interesting to study other types of hard circuits that save a different amount of soft logic resources to get a complete picture of how different hard circuits will benefit FPGAs. For example, imagine a hard circuit that only saved 3 to 4 times

the soft logic compared to the 6 to 20 times soft logic saving provided by hard crossbars and hard multipliers.

## 7.3 Concluding Remarks

We believe over time FPGAs will include more and more hard circuits in the fabric to narrow the area gap between FPGAs and ASICs. The ideas proposed in this dissertation provide both a methodology to study these inclusions on an FPGA, as well as some concepts to make hard circuits more area efficient. The most promising idea is the shadow cluster, and hopefully, using this concept will allow a new range of hard circuits to be introduced to FPGAs and will increase their usefulness to an even larger part of society.

# A Automatic Transistor Sizing of FPGAs

## A.1 Introduction

Due to the time and labour intensive work needed to create modern FPGAs, efforts have been made to automate the creation of such devices [KER05, Egi05, Kuo04]. In this appendix, we describe an automated transistor sizing methodology that plays a key role in the circuit lead design of FPGAs. This tool plays a key role in the work presented in chapters 3, 4, and 5, as one of the integral parts of the measurement methodology that requires an estimate of the area of FPGA-specific components. These components include programmable routing in the hard circuit tile and soft logic cluster tiles as well as the LUT-based logic. In this appendix, we provide details about our methodology.

### A.1.1 Method for Automatic Transistor Sizing

We use a simple flow to automatically size transistors in an FPGA to optimize the tile in terms of area and speed. Figure A.1 shows the flow.

The inputs to the automatic transistor sizer are a description of the FPGA architecture, the order in which to size the transistors, the range of transistors sizes to try, and a set of important paths (which we usually extract from benchmarks). The number of benchmark paths can be large and using them directly for transistor sizing might be excessively time consuming. Instead, we create representative paths from the set of input paths by first collecting statistics about these paths. These statistics are used to generate a specified number of paths that have characteristics similar to those of the input paths.

The final output of the automated transistor sizer is the widths for each type of transistor.

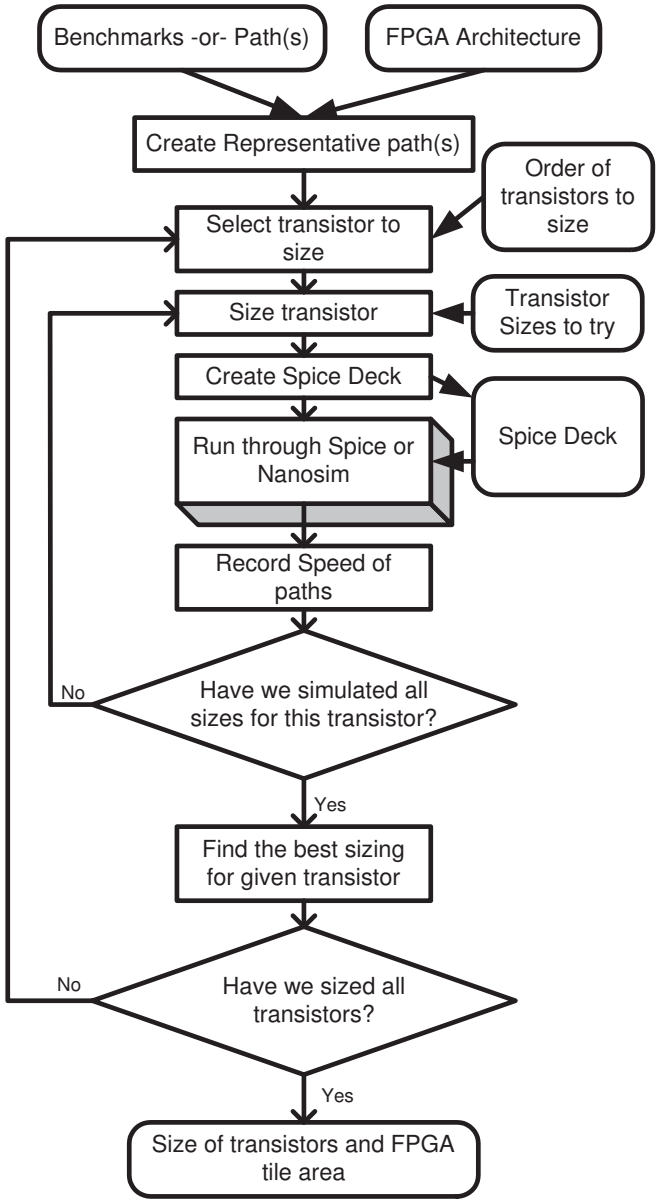


Figure A.1: This is the flow for the automatic transistor sizer.

Once there are a set of paths to size, we size the transistors based on the provided order of sizing. For each transistor type, we choose the size of the transistor, create a spice deck, simulate the spice deck using HSPICE [Syn99] or NanoSim [Syn01], and

record the resulting speed for each of the paths. With all the speeds for each transistor sizing, we determine which transistor size results in the best satisfaction of our criteria and set this transistor type size as the size for subsequent sizings. We proceed to the next transistor type and repeat this process.

The criteria we use to determine the best transistor sizing for a transistor type is area-delay product. To calculate the speed for a given transistor type size, we first normalize the speed of the path by dividing it by the speed of the path with a minimum width transistor for the current transistor type. We then geometrically average the speeds for all the paths in the set of representative paths. The area is calculated by summing up the minimum width transistor area of the entire FPGA tile [BRM99]. The area and speed results are multiplied together to give an area-delay product that is compared against other area-delay products for the same transistor type. The transistor sizing that yielded the smallest area-delay product is selected as the best transistor size for a given transistor type.

## A.2 Automatic Transistor Sizing Constraints

Transistor sizing of an FPGA involves determining appropriate sizes for all the transistors used to create the structures shown in Figure A.2. Various optimization goals such as speed, area, power consumption, or a combination thereof, can be used to guide this optimization. There has been significant amounts of prior work aimed at automating transistor sizing for custom (i.e. non-programmable) designs [NRSVT88, CEWWM+99, FD85]. These past approaches demonstrated the benefits of automation but purely for custom designs.

In Figure A.2, there are many repeated structures in the FPGA tile and it is typically desired that these groups of structures are sized together. When a transistor size grows, *all* transistors in that architectural class grow, which could include many transistors. For example, if one transistor in the first stage of the multiplexer that implements the LUT is sized to twice a minimum width transistor size, then all the transistors in the first stage of the multiplexer implementing every LUT in the FPGA is similarly sized to two times the size of a minimum width transistor.

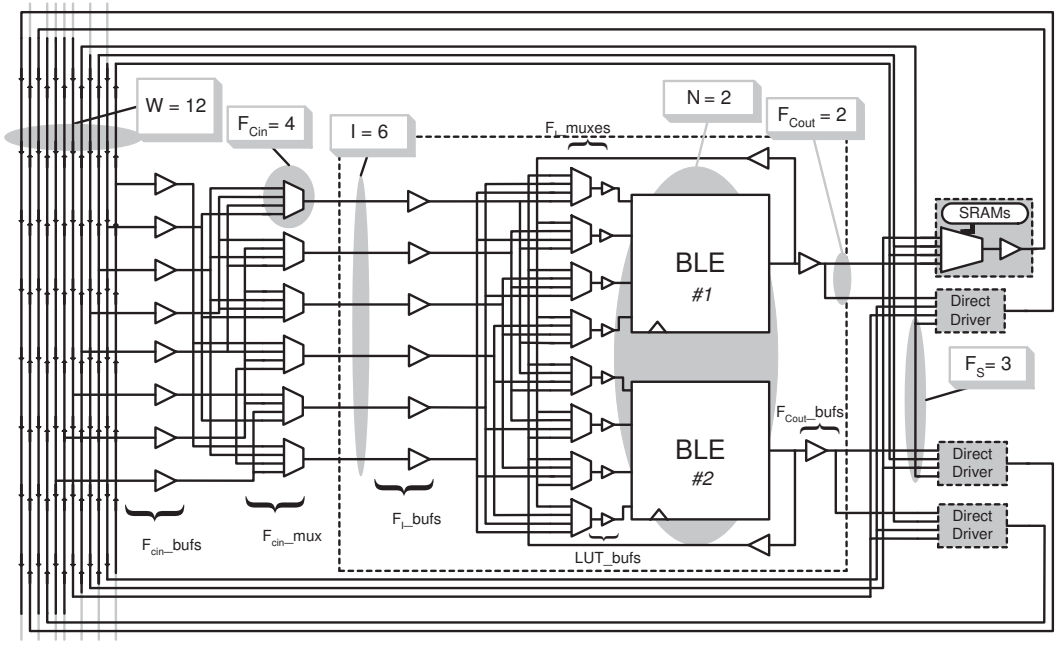


Figure A.2: Logic schematic of a soft logic cluster tile

With FPGAs it is worth noting that transistor sizing is different from regular transistor sizing because an FPGA may have many different near critical paths using different circuit structures when actually programmed with a design, so we are required to size multiple paths. It is difficult to select which path to size in an FPGA since it is not known which set of transistors are most important for a design's speed when it is actually mapped to an FPGA. Note that a path is a register to register series of components.

There have been no past automated approaches for the circuit design of FPGAs. Previous academic work that has focused on either layout [CSR+99, KER05] or accurate CAD estimation of speed and area [BRM99] use a manual approach to size transistors.

### A.2.1 Converting Transistor Sizes to Tile Area

After using our automatic transistor sizing tool to determine the widths of the transistors within a tile, we use these widths to estimate the area of a tile in square microns.

To do this conversion we adapted a method from [Egi05] to 90nm technology. This involves first converting each transistor into a measurement of the number of minimum width transistors occupying the same area as the sized transistor using the following equation [BRM99]:

$$area(minWidthTransistors) = 0.5 + \frac{driveStrength}{2} \tag{A.1}$$

Once each transistor is converted into a unit of minimum width transistors, we sum the minimum transistor widths for all the transistors in buffers, SRAMs, or multiplexers. To estimate the actual layout area, we model the area in terms of grid units where a grid unit is a square with height and width equal to the area required between metal vias adjacent to one another in the given technology. The number of layout grid units needed is estimated using the following equation [Egi05]:

$$\begin{aligned} cellArea(gridSquares) = complexity \times 3.3 \\ \times cellArea(minWidthTransistors) \end{aligned} \tag{A.2}$$

The complexity factor is 1.455 for SRAMs and multiplexers and 1.0 for all other structures, which reflects the difficulty of laying out SRAMs and multiplexers. Finally, we sum the number of grid units for all the structures in the tile and multiply this by the grid size for the 90nm technology. The final estimated layout area is in square microns.

### A.2.2 Quality of Sizing Tool

To verify that our automatic transistor sizing tool is generating an FPGA tile with area similar to modern industrial FPGA tiles, we compared the area of two of our tiles with the area of Altera's Stratix I [Alt03] and Stratix II [Alt04d] tiles.

Table 3.4 in chapter 3 shows the parameters that we selected to make our comparison. Architecture 1 is similar to the Stratix I FPGA and Architecture 2 has parameters similar to the Stratix II FPGAs. To select our track width (W) we looked at the Stratix I and Stratix II architecture and took an average of the horizontal and vertical routing tracks. For the Stratix II we chose N to be 8 and K to be 6, which is an

estimation of Altera's ALM [Alt04d].

Due to non-disclosure agreements, the absolute area numbers cannot be reported. Comparing the measurements relatively, we found that both of our automatically sized tiles for the Stratix I and Stratix II are approximately one third the size of the actual industrial tile areas. We account for our smaller area for the following reasons:

1. Our modeled cluster does not include all the elements present in a Stratix Lab (also known as a cluster) including clock buffers, registers with control signals such as enable and reset, adder capabilities, and carry chains. Many of these additional units add significantly to the silicon area required to implement the structure. From discussions with industrial architects [Lew06] we know that the area in a modern tile is approximately 40% for the logic and 60% for the programmable routing. This is due to the additional structures within a modern soft logic cluster tile.

2. The clusters that we model do not completely capture the Stratix architectures. While the size and number of LUTs in a cluster, the channel width, and various other routing parameters are set to be similar to the Stratix architectures, these parameters do not fully describe the Stratix devices. For example, our model of the Stratix devices uses only length four wire segments while the actual devices have a range of wire lengths. Longer wire segments will significantly impact the size of buffers that drive these segments.

3. The methodology we use to convert minimum width transistor counts to square microns only provides an estimate of layout area, and this method was originally created for a different CMOS technology.

4. Sizing is determined by finding a minimum area-delay product iteratively for each transistor. This cost function is not necessarily the criteria that FPGA designers use to size transistors, and instead, designers may be mainly concerned about speed or yield on paths through the FPGA resulting in the use of larger transistors than would be selected by our transistor sizer. In our discussions with industrial FPGA design experience, we now know that the primary sizing criteria is speed, which will significantly impact the sizing of all transistors in the cluster.

5. From discussions [Kuo07] we have learned that initial conditions, in terms of the size of each transistor, is a major factor in getting good results from an automated transistor sizing algorithm. Our initial sizings for all transistors was minimum width, and this has been shown to be poor starting conditions.

## A.3 Summary

In chapters 3, 4, and 5 we measure the quality of architectural concepts that depend on the quality of our area estimation of FPGA tiles. In this appendix, we reviewed details on how we automatically size transistors and how we convert our transistor sizes into area measurements in square microns. We compared our final results to industrial tiles and provided reasons why our tiles are approximately three times smaller.

# B Benchmark Details

## B.1 Introduction

Throughout this dissertation we use a set of benchmarks that is first introduced in Chapter 2, section 3.4. In this appendix, we provide additional information about these benchmarks. This information includes the number and sizes of multipliers, the number of input and output pins, and the number of memory bits in each benchmark. In addition to these benchmarks, we have created a number of synthetic benchmarks, and we provide details of these benchmarks in this appendix.

## B.2 Benchmark Details

As described earlier, our benchmarks are a collection of Verilog designs from around the Internet from various sources including: The Opencores organization [ope07], SCU-RTL [scu98], Texas-97 [tex97], and the Benchmarks for Placement 2001 [Pla01].

We have also made a considerable effort to convert applications developed locally from VHDL to Verilog. These designs include, Raytrace [FR03], Stereo Vision [DRM03], and Molecular Dynamic system [AKE+04]. The conversion process used both an automated tool (called X-HDL which was produced by a now defunct company) to start the conversion from VHDL to Verilog, and the remainder of the design was converted by hand.

Table B.1 shows the number of BLEs and the number of input and output pins. These numbers were obtained by mapping these circuits to Stratix I FPGAs [Alt03] using Quartus version 4.1 [Alt04c].

Table B.2 gives a breakdown of each benchmark in terms of how many and what are the sizes of the multipliers. These statistics were gathered by going into each

Table B.1: Benchmarks Basic Details

| Benchmark | BLEs | I/O Pins |
|---|---|---|
| fft | 2374 | 69 |
| iirA | 289 | 59 |
| iirB | 297 | 38 |
| firA | 84 | 60 |
| firB | 1598 | 455 |
| firC | 998 | 20 |
| diffeqA | 221 | 162 |
| diffeqB | 512 | 258 |
| stereoVisionA | 17765 | 213 |
| stereoVisionB | 35554 | 331 |
| stereoVisionB_no_mem | 34279 | 331 |
| rayTraceA | 2622 | 560 |
| rayTraceA_no_mem | 2118 | 560 |
| rayTraceB | 25056 | 337 |
| rayTraceB_no_me | 21557 | 337 |
| oc45_cpu | 2191 | 140 |
| reedSolDecoderA | 1151 | 20 |
| reedSolDecoderB | 1799 | 32 |
| moleculeDynamics | 10542 | 362 |
| cordicA | 591 | 51 |
| cordicB | 2830 | 111 |
| MACA | 2864 | 211 |
| MACB | 9828 | 415 |
| crc33_d264 | 102 | 330 |
| desArea | 1481 | 190 |
| desPerf | 4592 | 190 |
| stereoVisionC | 12433 | 367 |
| stereoVisionC_no_me | 7281 | 367 |
| stereoVisionD | 170 | 53 |
| rayTraceC | 766 | 103 |
| rayTraceC_no_me | 546 | 103 |
| rayTraceD | 1807 | 468 |

Table B.2: Multiplier Details for our Benchmarks

| Benchmark | Number of Mults | Number of Multipliers and their Size | | | | |
|---|---|---|---|---|---|---|
| fft | 32 | 32 8x8 | | | | |
| iirA | 5 | 2 10x10 | 3 8x8 | | | |
| iirB | 5 | 2 16x16 | 3 8x16 | | | |
| firA | 4 | 4 8x8 | | | | |
| firB | 25 | 25 16x16 | | | | |
| firC | 17 | 17 8x8 | | | | |
| diffeqA | 5 | 5 32x32 | | | | |
| diffeqB | 5 | 5 32x32 | | | | |
| stereoVisionA | 152 | 152 8x8 | | | | |
| stereoVisionB | 528 | 24 7x3 | 12 7x4 | 72 7x5 | | |
| | | 36 7x6 | 36 7x7 | 72 7x8 | 24 7x9 | 252 16x9 |
| stereoVisionB_no_mem | 528 | 24 7x3 | 12 7x4 | 72 7x5 | | |
| | | 36 7x6 | 36 7x7 | 72 7x8 | 24 7x9 | 252 16x9 |
| rayTraceA | 18 | 9 16x16 | 9 8x7 | | | |
| rayTraceA_no_mem | 18 | 18 | 9 16x16 | 9 8x7 | | |
| rayTraceB | 31 | 6 16x16 | 6 29x16 | 3 16x33 | | |
| | | 3 29x33 | 6 14x46 | 3 16x16 | 3 24x16 | 1 1x16 |
| rayTraceB_no_me | 31 | 6 16x16 | 6 29x16 | 3 16x33 | | |
| | | 3 29x33 | 6 14x46 | 3 16x16 | 3 24x16 | 1 1x16 |
| oc45_cpu | 1 | 1 16x16 | | | | |
| reedSolDecoderA | 13 | 13 4x4 | | | | |
| reedSolDecoderB | 9 | 9 8x8 | | | | |
| moleculeDynamics | 19 | 1 36x51 | 3 47x38 | | | |
| | | 3 38x38 | 2 22x51 | 3 22x43 | 3 2x43 | 3 50x43 |

benchmark and extracting the size and number of multipliers.

Table B.3: Memory Details for our Benchmarks

| Benchmark | Number of Memory bits |
|---|---|
| fft | 73278 |
| firC | 96 |
| stereoVisionB | 143360 |
| rayTraceA | 1008 |
| rayTraceB | 1043 |
| oc45_cpu | 256 |
| stereoVisionC | 457280 |
| rayTraceC | 24192 |

Table B.3 shows the benchmarks that contain memory. In this table, column 2 shows how many bits are in each benchmark. These numbers were obtained by mapping these

benchmarks to Stratix I FPGAs using Quartus version 4.1.

## B.3 Synthetic Benchmarks with Multipliers

We have created a set of synthetic benchmarks that use multipliers to test our concepts in Chapter 3 and Chapter 4. In this section, we show each of these benchmarks including the number of multipliers in each benchmark suite and the demand ratio. These benchmarks are summarized in Chapter 4 in Table 4.2, and the benchmark suite details are provided in the following tables B.4, B.5, B.6, and B.7.

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_0 | 4066 | 28 | 1:14 |
| BM_1 | 2920 | 8 | 1:36 |
| BM_2 | 285 | 0 | 1:$\infty$ |
| BM_3 | 698 | 4 | 1:17 |
| BM_4 | 660 | 0 | 1:$\infty$ |
| BM_5 | 995 | 3 | 1:33 |
| BM_6 | 4656 | 0 | 1:$\infty$ |
| BM_7 | 4482 | 11 | 1:40 |
| BM_8 | 1364 | 1 | 1:136 |
| BM_9 | 4868 | 1 | 1:486 |
| BM_10 | 2203 | 9 | 1:24 |
| BM_11 | 3777 | 19 | 1:19 |
| BM_12 | 2968 | 14 | 1:21 |
| BM_13 | 3706 | 6 | 1:61 |
| BM_14 | 2813 | 0 | 1:$\infty$ |
| BM_15 | 755 | 0 | 1:$\infty$ |
| BM_16 | 1858 | 0 | 1:$\infty$ |
| BM_17 | 2021 | 14 | 1:14 |
| BM_18 | 4731 | 16 | 1:29 |
| BM_19 | 1346 | 0 | 1:$\infty$ |
| BM_20 | 4642 | 9 | 1:51 |
| BM_21 | 405 | 1 | 1:40 |
| BM_22 | 4946 | 17 | 1:29 |
| BM_23 | 1494 | 5 | 1:29 |
| BM_24 | 2774 | 20 | 1:13 |
| BM_25 | 2643 | 8 | 1:33 |
| BM_26 | 2450 | 7 | 1:35 |

Continued on next page

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_27 | 4358 | 15 | 1:29 |
| BM_28 | 3146 | 13 | 1:24 |
| BM_29 | 3748 | 31 | 1:12 |
| BM_30 | 1146 | 0 | 1:∞ |
| BM_31 | 215 | 0 | 1:∞ |
| BM_32 | 2583 | 6 | 1:43 |
| BM_33 | 2267 | 0 | 1:∞ |
| BM_34 | 2325 | 0 | 1:∞ |
| BM_35 | 1609 | 2 | 1:80 |
| BM_36 | 3004 | 5 | 1:60 |
| BM_37 | 1002 | 2 | 1:50 |
| BM_38 | 3089 | 0 | 1:∞ |
| BM_39 | 3091 | 0 | 1:∞ |
| BM_40 | 3189 | 9 | 1:35 |
| BM_41 | 1118 | 2 | 1:55 |
| BM_42 | 251 | 1 | 1:25 |
| BM_43 | 1312 | 0 | 1:∞ |
| BM_44 | 1503 | 2 | 1:75 |
| BM_45 | 1756 | 8 | 1:21 |
| BM_46 | 1674 | 3 | 1:55 |
| BM_47 | 2414 | 0 | 1:∞ |
| BM_48 | 3131 | 11 | 1:28 |
| BM_49 | 531 | 1 | 1:53 |
| BM_50 | 1393 | 0 | 1:∞ |
| BM_51 | 3660 | 1 | 1:366 |
| BM_52 | 3778 | 0 | 1:∞ |
| BM_53 | 2244 | 2 | 1:112 |
| BM_54 | 2070 | 0 | 1:∞ |
| BM_55 | 1070 | 0 | 1:∞ |
| BM_56 | 968 | 0 | 1:∞ |
| BM_57 | 2368 | 7 | 1:33 |
| BM_58 | 2735 | 13 | 1:21 |
| BM_59 | 398 | 1 | 1:39 |
| BM_60 | 4912 | 0 | 1:∞ |
| BM_61 | 659 | 2 | 1:32 |
| BM_62 | 1760 | 11 | 1:16 |
| BM_63 | 1608 | 6 | 1:26 |
| BM_64 | 4079 | 14 | 1:29 |
| BM_65 | 1836 | 5 | 1:36 |
| BM_66 | 220 | 1 | 1:22 |
| BM_67 | 1393 | 0 | 1:∞ |
| BM_68 | 1428 | 3 | 1:47 |
| BM_69 | 1080 | 0 | 1:∞ |

Continued on next page

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_70 | 2783 | 4 | 1:69 |
| BM_71 | 3102 | 2 | 1:155 |
| BM_72 | 4453 | 4 | 1:111 |
| BM_73 | 2511 | 10 | 1:25 |
| BM_74 | 1293 | 2 | 1:64 |
| BM_75 | 1798 | 8 | 1:22 |
| BM_76 | 3382 | 13 | 1:26 |
| BM_77 | 1183 | 1 | 1:118 |
| BM_78 | 578 | 2 | 1:28 |
| BM_79 | 1027 | 0 | 1:$\infty$ |
| BM_80 | 3087 | 0 | 1:$\infty$ |
| BM_81 | 750 | 0 | 1:$\infty$ |
| BM_82 | 4449 | 0 | 1:$\infty$ |
| BM_83 | 225 | 0 | 1:$\infty$ |
| BM_84 | 3856 | 13 | 1:29 |
| BM_85 | 2161 | 8 | 1:27 |
| BM_86 | 2230 | 6 | 1:37 |
| BM_87 | 4525 | 4 | 1:113 |
| BM_88 | 1373 | 0 | 1:$\infty$ |
| BM_89 | 1915 | 1 | 1:191 |
| BM_90 | 3374 | 0 | 1:$\infty$ |
| BM_91 | 712 | 1 | 1:71 |
| BM_92 | 1855 | 2 | 1:92 |
| BM_93 | 2449 | 8 | 1:30 |
| BM_94 | 2111 | 4 | 1:52 |
| BM_95 | 3874 | 6 | 1:64 |
| BM_96 | 2700 | 12 | 1:22 |
| BM_97 | 1189 | 2 | 1:59 |
| BM_98 | 1286 | 7 | 1:18 |
| BM_99 | 3294 | 11 | 1:29 |
| BM_100 | 1797 | 3 | 1:59 |
| BM_101 | 4033 | 15 | 1:26 |
| BM_102 | 1325 | 5 | 1:26 |
| BM_103 | 3354 | 14 | 1:23 |
| BM_104 | 240 | 0 | 1:$\infty$ |
| BM_105 | 994 | 3 | 1:33 |
| BM_106 | 1886 | 0 | 1:$\infty$ |
| BM_107 | 2367 | 11 | 1:21 |
| BM_108 | 4008 | 4 | 1:100 |
| BM_109 | 2577 | 11 | 1:23 |
| BM_110 | 3751 | 18 | 1:20 |
| BM_111 | 4321 | 4 | 1:108 |
| BM_112 | 3748 | 30 | 1:12 |

Continued on next page

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_113 | 2614 | 1 | 1:261 |
| BM_114 | 893 | 3 | 1:29 |
| BM_115 | 2461 | 4 | 1:61 |
| BM_116 | 619 | 2 | 1:30 |
| BM_117 | 4537 | 13 | 1:34 |
| BM_118 | 3088 | 15 | 1:20 |
| BM_119 | 872 | 1 | 1:87 |
| BM_120 | 4550 | 9 | 1:50 |
| BM_121 | 809 | 1 | 1:80 |
| BM_122 | 2197 | 8 | 1:27 |
| BM_123 | 3407 | 4 | 1:85 |
| BM_124 | 853 | 0 | 1:$\infty$ |
| BM_125 | 4083 | 4 | 1:102 |
| BM_126 | 4098 | 7 | 1:58 |
| BM_127 | 2514 | 0 | 1:$\infty$ |
| BM_128 | 3543 | 0 | 1:$\infty$ |
| BM_129 | 635 | 2 | 1:31 |
| BM_130 | 993 | 1 | 1:99 |
| BM_131 | 2907 | 6 | 1:48 |
| BM_132 | 4126 | 4 | 1:103 |
| BM_133 | 1414 | 1 | 1:141 |
| BM_134 | 1251 | 4 | 1:31 |
| BM_135 | 2236 | 5 | 1:44 |
| BM_136 | 1384 | 4 | 1:34 |
| BM_137 | 3666 | 5 | 1:73 |
| BM_138 | 1135 | 0 | 1:$\infty$ |
| BM_139 | 2102 | 1 | 1:210 |
| BM_140 | 2535 | 3 | 1:84 |
| BM_141 | 4881 | 0 | 1:$\infty$ |
| BM_142 | 1165 | 4 | 1:29 |
| BM_143 | 372 | 0 | 1:$\infty$ |
| BM_144 | 3685 | 19 | 1:19 |
| BM_145 | 2008 | 2 | 1:100 |
| BM_146 | 4269 | 15 | 1:28 |
| BM_147 | 4538 | 1 | 1:453 |
| BM_148 | 3155 | 5 | 1:63 |
| BM_149 | 2501 | 5 | 1:50 |
| BM_150 | 2455 | 1 | 1:245 |
| BM_151 | 2674 | 14 | 1:19 |
| BM_152 | 2807 | 10 | 1:28 |
| BM_153 | 4121 | 12 | 1:34 |
| BM_154 | 4386 | 3 | 1:146 |
| BM_155 | 3740 | 0 | 1:$\infty$ |

Continued on next page

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_156 | 2933 | 12 | 1:24 |
| BM_157 | 3679 | 0 | 1:∞ |
| BM_158 | 4972 | 9 | 1:55 |
| BM_159 | 2182 | 4 | 1:54 |
| BM_160 | 3455 | 1 | 1:345 |
| BM_161 | 1236 | 6 | 1:20 |
| BM_162 | 2160 | 5 | 1:43 |
| BM_163 | 2578 | 7 | 1:36 |
| BM_164 | 2580 | 7 | 1:36 |
| BM_165 | 2329 | 4 | 1:58 |
| BM_166 | 449 | 0 | 1:∞ |
| BM_167 | 2487 | 7 | 1:35 |
| BM_168 | 869 | 2 | 1:43 |
| BM_169 | 3975 | 18 | 1:22 |
| BM_170 | 695 | 0 | 1:∞ |
| BM_171 | 2035 | 8 | 1:25 |
| BM_172 | 1863 | 5 | 1:37 |
| BM_173 | 586 | 1 | 1:58 |
| BM_174 | 2147 | 0 | 1:∞ |
| BM_175 | 805 | 5 | 1:16 |
| BM_176 | 2820 | 0 | 1:∞ |
| BM_177 | 368 | 1 | 1:36 |
| BM_178 | 1868 | 10 | 1:18 |
| BM_179 | 2576 | 9 | 1:28 |
| BM_180 | 306 | 0 | 1:∞ |
| BM_181 | 1931 | 1 | 1:193 |
| BM_182 | 3094 | 3 | 1:103 |
| BM_183 | 2846 | 15 | 1:18 |
| BM_184 | 4203 | 3 | 1:140 |
| BM_185 | 3275 | 1 | 1:327 |
| BM_186 | 3270 | 0 | 1:∞ |
| BM_187 | 3719 | 9 | 1:41 |
| BM_188 | 3274 | 3 | 1:109 |
| BM_189 | 2505 | 14 | 1:17 |
| BM_190 | 2131 | 4 | 1:53 |
| BM_191 | 2276 | 14 | 1:16 |
| BM_192 | 631 | 3 | 1:21 |
| BM_193 | 4802 | 7 | 1:68 |
| BM_194 | 554 | 2 | 1:27 |
| BM_195 | 1801 | 2 | 1:90 |
| BM_196 | 4675 | 0 | 1:∞ |
| BM_197 | 4632 | 0 | 1:∞ |
| BM_198 | 1757 | 10 | 1:17 |

Continued on next page

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_199 | 1595 | 6 | 1:26 |
| BM_200 | 2629 | 18 | 1:14 |
| BM_201 | 918 | 3 | 1:30 |
| BM_202 | 4774 | 26 | 1:18 |
| BM_203 | 348 | 0 | 1:$\infty$ |
| BM_204 | 3848 | 0 | 1:$\infty$ |
| BM_205 | 4535 | 9 | 1:50 |
| BM_206 | 1771 | 2 | 1:88 |
| BM_207 | 2399 | 10 | 1:23 |
| BM_208 | 2424 | 6 | 1:40 |
| BM_209 | 4680 | 1 | 1:468 |
| BM_210 | 2736 | 1 | 1:273 |
| BM_211 | 4245 | 13 | 1:32 |
| BM_212 | 3861 | 12 | 1:32 |
| BM_213 | 2257 | 11 | 1:20 |
| BM_214 | 3117 | 7 | 1:44 |
| BM_215 | 3450 | 12 | 1:28 |
| BM_216 | 3748 | 0 | 1:$\infty$ |
| BM_217 | 4844 | 0 | 1:$\infty$ |
| BM_218 | 4332 | 16 | 1:27 |
| BM_219 | 2500 | 4 | 1:62 |
| BM_220 | 802 | 2 | 1:40 |
| BM_221 | 4975 | 15 | 1:33 |
| BM_222 | 2725 | 0 | 1:$\infty$ |
| BM_223 | 1749 | 0 | 1:$\infty$ |
| BM_224 | 4219 | 11 | 1:38 |
| BM_225 | 4292 | 0 | 1:$\infty$ |
| BM_226 | 4093 | 17 | 1:24 |
| BM_227 | 2020 | 7 | 1:28 |
| BM_228 | 2475 | 13 | 1:19 |
| BM_229 | 4526 | 0 | 1:$\infty$ |
| BM_230 | 3360 | 13 | 1:25 |
| BM_231 | 1399 | 0 | 1:$\infty$ |
| BM_232 | 2347 | 10 | 1:23 |
| BM_233 | 3844 | 0 | 1:$\infty$ |
| BM_234 | 927 | 2 | 1:46 |
| BM_235 | 1974 | 2 | 1:98 |
| BM_236 | 4490 | 24 | 1:18 |
| BM_237 | 1119 | 0 | 1:$\infty$ |
| BM_238 | 4861 | 3 | 1:162 |
| BM_239 | 1477 | 4 | 1:36 |
| BM_240 | 320 | 0 | 1:$\infty$ |
| BM_241 | 3478 | 19 | 1:18 |

Continued on next page

Table B.4: Benchmark suite SB45 containing multipliers and with a Demand Ratio of 1:45

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_242 | 4065 | 9 | 1:45 |
| BM_243 | 4764 | 8 | 1:59 |
| BM_244 | 561 | 1 | 1:56 |
| BM_245 | 2008 | 2 | 1:100 |
| BM_246 | 4269 | 15 | 1:28 |
| BM_247 | 4538 | 1 | 1:453 |
| BM_248 | 3155 | 5 | 1:63 |
| BM_249 | 2501 | 5 | 1:50 |

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_0 | 18347 | 93 | 1:19 |
| BM_1 | 15343 | 0 | 1:$\infty$ |
| BM_2 | 17224 | 174 | 1:9 |
| BM_3 | 12603 | 64 | 1:19 |
| BM_4 | 13350 | 67 | 1:19 |
| BM_5 | 18719 | 95 | 1:19 |
| BM_6 | 12435 | 63 | 1:19 |
| BM_7 | 12895 | 130 | 1:9 |
| BM_8 | 12345 | 0 | 1:$\infty$ |
| BM_9 | 18486 | 93 | 1:19 |
| BM_10 | 12863 | 130 | 1:9 |
| BM_11 | 14578 | 74 | 1:19 |
| BM_12 | 10971 | 111 | 1:9 |
| BM_13 | 19892 | 202 | 1:9 |
| BM_14 | 15224 | 0 | 1:$\infty$ |
| BM_15 | 10892 | 55 | 1:19 |
| BM_16 | 13547 | 68 | 1:19 |
| BM_17 | 13912 | 70 | 1:19 |
| BM_18 | 14054 | 142 | 1:9 |
| BM_19 | 16573 | 168 | 1:9 |
| BM_20 | 19675 | 99 | 1:19 |
| BM_21 | 14831 | 75 | 1:19 |
| BM_22 | 10636 | 53 | 1:20 |
| BM_23 | 17462 | 88 | 1:19 |
| BM_24 | 14745 | 149 | 1:9 |
| BM_25 | 14806 | 150 | 1:9 |
| BM_26 | 19567 | 99 | 1:19 |

Continued on next page

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_27 | 10776 | 109 | 1:9 |
| BM_28 | 12726 | 129 | 1:9 |
| BM_29 | 15940 | 161 | 1:9 |
| BM_30 | 10160 | 51 | 1:19 |
| BM_31 | 14226 | 72 | 1:19 |
| BM_32 | 15772 | 160 | 1:9 |
| BM_33 | 10830 | 55 | 1:19 |
| BM_34 | 17359 | 176 | 1:9 |
| BM_35 | 13582 | 68 | 1:19 |
| BM_36 | 19049 | 0 | 1:$\infty$ |
| BM_37 | 19144 | 194 | 1:9 |
| BM_38 | 18844 | 191 | 1:9 |
| BM_39 | 19019 | 96 | 1:19 |
| BM_40 | 18907 | 96 | 1:19 |
| BM_41 | 17549 | 89 | 1:19 |
| BM_42 | 15914 | 80 | 1:19 |
| BM_43 | 15366 | 78 | 1:19 |
| BM_44 | 15346 | 155 | 1:9 |
| BM_45 | 15646 | 158 | 1:9 |
| BM_46 | 17194 | 87 | 1:19 |
| BM_47 | 15566 | 158 | 1:9 |
| BM_48 | 14364 | 145 | 1:9 |
| BM_49 | 12793 | 64 | 1:19 |
| BM_50 | 10152 | 103 | 1:9 |
| BM_51 | 18757 | 190 | 1:9 |
| BM_52 | 17494 | 88 | 1:19 |
| BM_53 | 15239 | 77 | 1:19 |
| BM_54 | 17211 | 87 | 1:19 |
| BM_55 | 16280 | 82 | 1:19 |
| BM_56 | 10787 | 54 | 1:19 |
| BM_57 | 14594 | 74 | 1:19 |
| BM_58 | 18507 | 0 | 1:$\infty$ |
| BM_59 | 14397 | 73 | 1:19 |
| BM_60 | 10322 | 52 | 1:19 |
| BM_61 | 15711 | 0 | 1:$\infty$ |
| BM_62 | 19917 | 101 | 1:19 |
| BM_63 | 10483 | 106 | 1:9 |
| BM_64 | 14026 | 71 | 1:19 |
| BM_65 | 12356 | 125 | 1:9 |
| BM_66 | 12357 | 62 | 1:19 |
| BM_67 | 19732 | 100 | 1:19 |
| BM_68 | 11718 | 118 | 1:9 |
| BM_69 | 12541 | 63 | 1:19 |

Continued on next page

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_70 | 11933 | 121 | 1:9 |
| BM_71 | 15048 | 152 | 1:9 |
| BM_72 | 19003 | 96 | 1:19 |
| BM_73 | 12126 | 123 | 1:9 |
| BM_74 | 18658 | 189 | 1:9 |
| BM_75 | 10419 | 52 | 1:20 |
| BM_76 | 12919 | 131 | 1:9 |
| BM_77 | 12849 | 65 | 1:19 |
| BM_78 | 10112 | 102 | 1:9 |
| BM_79 | 13469 | 68 | 1:19 |
| BM_80 | 19313 | 98 | 1:19 |
| BM_81 | 11020 | 111 | 1:9 |
| BM_82 | 13290 | 0 | 1:$\infty$ |
| BM_83 | 17873 | 90 | 1:19 |
| BM_84 | 18427 | 0 | 1:$\infty$ |
| BM_85 | 13938 | 141 | 1:9 |
| BM_86 | 11940 | 121 | 1:9 |
| BM_87 | 12137 | 123 | 1:9 |
| BM_88 | 11333 | 115 | 1:9 |
| BM_89 | 11157 | 56 | 1:19 |
| BM_90 | 17138 | 174 | 1:9 |
| BM_91 | 18972 | 96 | 1:19 |
| BM_92 | 14059 | 0 | 1:$\infty$ |
| BM_93 | 19246 | 195 | 1:9 |
| BM_94 | 11505 | 58 | 1:19 |
| BM_95 | 11013 | 111 | 1:9 |
| BM_96 | 18541 | 94 | 1:19 |
| BM_97 | 15193 | 0 | 1:$\infty$ |
| BM_98 | 19138 | 97 | 1:19 |
| BM_99 | 13192 | 66 | 1:19 |
| BM_100 | 16186 | 164 | 1:9 |
| BM_101 | 16982 | 86 | 1:19 |
| BM_102 | 12469 | 63 | 1:19 |
| BM_103 | 15422 | 78 | 1:19 |
| BM_104 | 19734 | 200 | 1:9 |
| BM_105 | 10145 | 51 | 1:19 |
| BM_106 | 14508 | 147 | 1:9 |
| BM_107 | 13282 | 67 | 1:19 |
| BM_108 | 17290 | 175 | 1:9 |
| BM_109 | 15442 | 78 | 1:19 |
| BM_110 | 11597 | 58 | 1:19 |
| BM_111 | 12319 | 62 | 1:19 |
| BM_112 | 18488 | 93 | 1:19 |

Continued on next page

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_113 | 14674 | 74 | 1:19 |
| BM_114 | 16931 | 171 | 1:9 |
| BM_115 | 17896 | 181 | 1:9 |
| BM_116 | 11349 | 57 | 1:19 |
| BM_117 | 17596 | 89 | 1:19 |
| BM_118 | 10349 | 105 | 1:9 |
| BM_119 | 13972 | 0 | 1:$\infty$ |
| BM_120 | 15949 | 80 | 1:19 |
| BM_121 | 13307 | 67 | 1:19 |
| BM_122 | 19587 | 198 | 1:9 |
| BM_123 | 11728 | 59 | 1:19 |
| BM_124 | 15471 | 157 | 1:9 |
| BM_125 | 18378 | 93 | 1:19 |
| BM_126 | 19977 | 101 | 1:19 |
| BM_127 | 13558 | 137 | 1:9 |
| BM_128 | 14561 | 147 | 1:9 |
| BM_129 | 18796 | 95 | 1:19 |
| BM_130 | 12452 | 126 | 1:9 |
| BM_131 | 17314 | 0 | 1:$\infty$ |
| BM_132 | 14274 | 144 | 1:9 |
| BM_133 | 13972 | 141 | 1:9 |
| BM_134 | 17928 | 182 | 1:9 |
| BM_135 | 17410 | 88 | 1:19 |
| BM_136 | 18414 | 93 | 1:19 |
| BM_137 | 17039 | 173 | 1:9 |
| BM_138 | 13660 | 69 | 1:19 |
| BM_139 | 17584 | 178 | 1:9 |
| BM_140 | 12290 | 62 | 1:19 |
| BM_141 | 12551 | 127 | 1:9 |
| BM_142 | 13011 | 66 | 1:19 |
| BM_143 | 17086 | 86 | 1:19 |
| BM_144 | 17046 | 86 | 1:19 |
| BM_145 | 11769 | 59 | 1:19 |
| BM_146 | 15279 | 77 | 1:19 |
| BM_147 | 13300 | 67 | 1:19 |
| BM_148 | 10346 | 105 | 1:9 |
| BM_149 | 16881 | 85 | 1:19 |
| BM_150 | 17920 | 182 | 1:9 |
| BM_151 | 14857 | 75 | 1:19 |
| BM_152 | 11361 | 115 | 1:9 |
| BM_153 | 17041 | 173 | 1:9 |
| BM_154 | 17016 | 86 | 1:19 |
| BM_155 | 16856 | 85 | 1:19 |

Continued on next page

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_156 | 10587 | 53 | 1:19 |
| BM_157 | 17927 | 182 | 1:9 |
| BM_158 | 18221 | 92 | 1:19 |
| BM_159 | 17522 | 88 | 1:19 |
| BM_160 | 19951 | 101 | 1:19 |
| BM_161 | 11118 | 56 | 1:19 |
| BM_162 | 14728 | 74 | 1:19 |
| BM_163 | 17488 | 177 | 1:9 |
| BM_164 | 14394 | 73 | 1:19 |
| BM_165 | 19701 | 100 | 1:19 |
| BM_166 | 15012 | 76 | 1:19 |
| BM_167 | 15001 | 76 | 1:19 |
| BM_168 | 12807 | 65 | 1:19 |
| BM_169 | 17747 | 0 | 1:$\infty$ |
| BM_170 | 13019 | 132 | 1:9 |
| BM_171 | 15482 | 78 | 1:19 |
| BM_172 | 10363 | 52 | 1:19 |
| BM_173 | 13556 | 137 | 1:9 |
| BM_174 | 17478 | 177 | 1:9 |
| BM_175 | 16556 | 84 | 1:19 |
| BM_176 | 13075 | 66 | 1:19 |
| BM_177 | 19012 | 193 | 1:9 |
| BM_178 | 18444 | 187 | 1:9 |
| BM_179 | 12877 | 65 | 1:19 |
| BM_180 | 17221 | 87 | 1:19 |
| BM_181 | 11504 | 116 | 1:9 |
| BM_182 | 19841 | 100 | 1:19 |
| BM_183 | 14960 | 151 | 1:9 |
| BM_184 | 11252 | 0 | 1:$\infty$ |
| BM_185 | 18824 | 0 | 1:$\infty$ |
| BM_186 | 13663 | 69 | 1:19 |
| BM_187 | 14230 | 144 | 1:9 |
| BM_188 | 16772 | 85 | 1:19 |
| BM_189 | 11667 | 59 | 1:19 |
| BM_190 | 11570 | 58 | 1:19 |
| BM_191 | 18032 | 91 | 1:19 |
| BM_192 | 15897 | 161 | 1:9 |
| BM_193 | 18800 | 95 | 1:19 |
| BM_194 | 16249 | 164 | 1:9 |
| BM_195 | 19883 | 0 | 1:$\infty$ |
| BM_196 | 16960 | 86 | 1:19 |
| BM_197 | 12057 | 122 | 1:9 |
| BM_198 | 16950 | 86 | 1:19 |

Continued on next page

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_199 | 12154 | 61 | 1:19 |
| BM_200 | 12428 | 63 | 1:19 |
| BM_201 | 15178 | 77 | 1:19 |
| BM_202 | 14341 | 72 | 1:19 |
| BM_203 | 12823 | 65 | 1:19 |
| BM_204 | 19900 | 101 | 1:19 |
| BM_205 | 11715 | 118 | 1:9 |
| BM_206 | 19066 | 193 | 1:9 |
| BM_207 | 18819 | 191 | 1:9 |
| BM_208 | 13145 | 66 | 1:19 |
| BM_209 | 12009 | 60 | 1:20 |
| BM_210 | 12000 | 60 | 1:20 |
| BM_211 | 19217 | 97 | 1:19 |
| BM_212 | 12336 | 62 | 1:19 |
| BM_213 | 18467 | 187 | 1:9 |
| BM_214 | 10995 | 55 | 1:19 |
| BM_215 | 13841 | 70 | 1:19 |
| BM_216 | 14304 | 72 | 1:19 |
| BM_217 | 15868 | 80 | 1:19 |
| BM_218 | 11132 | 113 | 1:9 |
| BM_219 | 19758 | 0 | 1:$\infty$ |
| BM_220 | 15885 | 0 | 1:$\infty$ |
| BM_221 | 15206 | 154 | 1:9 |
| BM_222 | 17573 | 178 | 1:9 |
| BM_223 | 12619 | 0 | 1:$\infty$ |
| BM_224 | 17996 | 91 | 1:19 |
| BM_225 | 12817 | 130 | 1:9 |
| BM_226 | 19224 | 195 | 1:9 |
| BM_227 | 11662 | 59 | 1:19 |
| BM_228 | 16013 | 162 | 1:9 |
| BM_229 | 17199 | 87 | 1:19 |
| BM_230 | 17102 | 0 | 1:$\infty$ |
| BM_231 | 16784 | 85 | 1:19 |
| BM_232 | 12764 | 64 | 1:19 |
| BM_233 | 16403 | 166 | 1:9 |
| BM_234 | 17695 | 89 | 1:19 |
| BM_235 | 11281 | 57 | 1:19 |
| BM_236 | 13987 | 142 | 1:9 |
| BM_237 | 14884 | 75 | 1:19 |
| BM_238 | 16865 | 171 | 1:9 |
| BM_239 | 11335 | 57 | 1:19 |
| BM_240 | 14835 | 150 | 1:9 |
| BM_241 | 11210 | 56 | 1:20 |

Continued on next page

Table B.5: Benchmark suite SB15_V1 containing multipliers and with a Demand Ratio of 1:15 and Variance of 1:1111

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_242 | 18890 | 0 | 1:∞ |
| BM_243 | 18325 | 186 | 1:9 |
| BM_244 | 19353 | 98 | 1:19 |
| BM_245 | 14616 | 148 | 1:9 |
| BM_246 | 19269 | 97 | 1:19 |
| BM_247 | 14214 | 72 | 1:19 |
| BM_248 | 10405 | 52 | 1:20 |
| BM_249 | 18633 | 94 | 1:19 |

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_0 | 4066 | 85 | 1:4 |
| BM_1 | 2920 | 26 | 1:11 |
| BM_2 | 285 | 0 | 1:∞ |
| BM_3 | 698 | 13 | 1:5 |
| BM_4 | 660 | 0 | 1:∞ |
| BM_5 | 995 | 10 | 1:9 |
| BM_6 | 4656 | 0 | 1:∞ |
| BM_7 | 4482 | 33 | 1:13 |
| BM_8 | 1364 | 4 | 1:34 |
| BM_9 | 4868 | 3 | 1:162 |
| BM_10 | 2203 | 30 | 1:7 |
| BM_11 | 3777 | 57 | 1:6 |
| BM_12 | 2968 | 44 | 1:6 |
| BM_13 | 3706 | 18 | 1:20 |
| BM_14 | 2813 | 0 | 1:∞ |
| BM_15 | 755 | 0 | 1:∞ |
| BM_16 | 1858 | 1 | 1:185 |
| BM_17 | 2021 | 43 | 1:4 |
| BM_18 | 4731 | 48 | 1:9 |
| BM_19 | 1346 | 0 | 1:∞ |
| BM_20 | 4642 | 29 | 1:16 |
| BM_21 | 405 | 3 | 1:13 |
| BM_22 | 4946 | 51 | 1:9 |
| BM_23 | 1494 | 15 | 1:9 |
| BM_24 | 2774 | 60 | 1:4 |
| BM_25 | 2643 | 24 | 1:11 |
| BM_26 | 2450 | 22 | 1:11 |

Continued on next page

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_27 | 4358 | 48 | 1:9 |
| BM_28 | 3146 | 40 | 1:7 |
| BM_29 | 3748 | 94 | 1:3 |
| BM_30 | 1146 | 0 | 1:∞ |
| BM_31 | 215 | 2 | 1:10 |
| BM_32 | 2583 | 18 | 1:14 |
| BM_33 | 2267 | 0 | 1:∞ |
| BM_34 | 2325 | 0 | 1:∞ |
| BM_35 | 1609 | 6 | 1:26 |
| BM_36 | 3004 | 17 | 1:17 |
| BM_37 | 1002 | 6 | 1:16 |
| BM_38 | 3089 | 0 | 1:∞ |
| BM_39 | 3091 | 0 | 1:∞ |
| BM_40 | 3189 | 29 | 1:10 |
| BM_41 | 1118 | 7 | 1:15 |
| BM_42 | 251 | 3 | 1:8 |
| BM_43 | 1312 | 1 | 1:131 |
| BM_44 | 1503 | 6 | 1:25 |
| BM_45 | 1756 | 24 | 1:7 |
| BM_46 | 1674 | 12 | 1:13 |
| BM_47 | 2414 | 0 | 1:∞ |
| BM_48 | 3131 | 34 | 1:9 |
| BM_49 | 531 | 4 | 1:13 |
| BM_50 | 1393 | 0 | 1:∞ |
| BM_51 | 3660 | 4 | 1:91 |
| BM_52 | 3778 | 0 | 1:∞ |
| BM_53 | 2244 | 6 | 1:37 |
| BM_54 | 2070 | 0 | 1:∞ |
| BM_55 | 1070 | 0 | 1:∞ |
| BM_56 | 968 | 2 | 1:48 |
| BM_57 | 2368 | 23 | 1:10 |
| BM_58 | 2735 | 41 | 1:6 |
| BM_59 | 398 | 3 | 1:13 |
| BM_60 | 4912 | 0 | 1:∞ |
| BM_61 | 659 | 7 | 1:9 |
| BM_62 | 1760 | 33 | 1:5 |
| BM_63 | 1608 | 19 | 1:8 |
| BM_64 | 4079 | 43 | 1:9 |
| BM_65 | 1836 | 15 | 1:12 |
| BM_66 | 220 | 3 | 1:7 |
| BM_67 | 1393 | 0 | 1:∞ |
| BM_68 | 1428 | 11 | 1:12 |
| BM_69 | 1080 | 0 | 1:∞ |

Continued on next page

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_70 | 2783 | 13 | 1:21 |
| BM_71 | 3102 | 8 | 1:38 |
| BM_72 | 4453 | 13 | 1:34 |
| BM_73 | 2511 | 30 | 1:8 |
| BM_74 | 1293 | 6 | 1:21 |
| BM_75 | 1798 | 24 | 1:7 |
| BM_76 | 3382 | 39 | 1:8 |
| BM_77 | 1183 | 5 | 1:23 |
| BM_78 | 578 | 9 | 1:6 |
| BM_79 | 1027 | 0 | 1:$\infty$ |
| BM_80 | 3087 | 0 | 1:$\infty$ |
| BM_81 | 750 | 1 | 1:75 |
| BM_82 | 4449 | 0 | 1:$\infty$ |
| BM_83 | 225 | 2 | 1:11 |
| BM_84 | 3856 | 39 | 1:9 |
| BM_85 | 2161 | 25 | 1:8 |
| BM_86 | 2230 | 18 | 1:12 |
| BM_87 | 4525 | 14 | 1:32 |
| BM_88 | 1373 | 0 | 1:$\infty$ |
| BM_89 | 1915 | 5 | 1:38 |
| BM_90 | 3374 | 0 | 1:$\infty$ |
| BM_91 | 712 | 4 | 1:17 |
| BM_92 | 1855 | 8 | 1:23 |
| BM_93 | 2449 | 25 | 1:9 |
| BM_94 | 2111 | 12 | 1:17 |
| BM_95 | 3874 | 18 | 1:21 |
| BM_96 | 2700 | 38 | 1:7 |
| BM_97 | 1189 | 7 | 1:16 |
| BM_98 | 1286 | 21 | 1:6 |
| BM_99 | 3294 | 34 | 1:9 |
| BM_100 | 1797 | 11 | 1:16 |
| BM_101 | 4033 | 47 | 1:8 |
| BM_102 | 1325 | 15 | 1:8 |
| BM_103 | 3354 | 44 | 1:7 |
| BM_104 | 240 | 0 | 1:$\infty$ |
| BM_105 | 994 | 9 | 1:11 |
| BM_106 | 1886 | 0 | 1:$\infty$ |
| BM_107 | 2367 | 34 | 1:6 |
| BM_108 | 4008 | 12 | 1:33 |
| BM_109 | 2577 | 33 | 1:7 |
| BM_110 | 3751 | 56 | 1:6 |
| BM_111 | 4321 | 13 | 1:33 |
| BM_112 | 3748 | 90 | 1:4 |

Continued on next page

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_113 | 2614 | 5 | 1:52 |
| BM_114 | 893 | 9 | 1:9 |
| BM_115 | 2461 | 13 | 1:18 |
| BM_116 | 619 | 8 | 1:7 |
| BM_117 | 4537 | 39 | 1:11 |
| BM_118 | 3088 | 47 | 1:6 |
| BM_119 | 872 | 4 | 1:21 |
| BM_120 | 4550 | 29 | 1:15 |
| BM_121 | 809 | 4 | 1:20 |
| BM_122 | 2197 | 25 | 1:8 |
| BM_123 | 3407 | 13 | 1:26 |
| BM_124 | 853 | 0 | 1:$\infty$ |
| BM_125 | 4083 | 12 | 1:34 |
| BM_126 | 4098 | 23 | 1:17 |
| BM_127 | 2514 | 0 | 1:$\infty$ |
| BM_128 | 3543 | 0 | 1:$\infty$ |
| BM_129 | 635 | 6 | 1:10 |
| BM_130 | 993 | 4 | 1:24 |
| BM_131 | 2907 | 18 | 1:16 |
| BM_132 | 4126 | 14 | 1:29 |
| BM_133 | 1414 | 5 | 1:28 |
| BM_134 | 1251 | 13 | 1:9 |
| BM_135 | 2236 | 15 | 1:14 |
| BM_136 | 1384 | 14 | 1:9 |
| BM_137 | 3666 | 15 | 1:24 |
| BM_138 | 1135 | 1 | 1:113 |
| BM_139 | 2102 | 4 | 1:52 |
| BM_140 | 2535 | 9 | 1:28 |
| BM_141 | 4881 | 1 | 1:488 |
| BM_142 | 1165 | 12 | 1:9 |
| BM_143 | 372 | 0 | 1:$\infty$ |
| BM_144 | 3685 | 58 | 1:6 |
| BM_145 | 2008 | 6 | 1:33 |
| BM_146 | 4269 | 46 | 1:9 |
| BM_147 | 4538 | 4 | 1:113 |
| BM_148 | 3155 | 17 | 1:18 |
| BM_149 | 2501 | 17 | 1:14 |
| BM_150 | 2455 | 3 | 1:81 |
| BM_151 | 2674 | 44 | 1:6 |
| BM_152 | 2807 | 30 | 1:9 |
| BM_153 | 4121 | 37 | 1:11 |
| BM_154 | 4386 | 10 | 1:43 |
| BM_155 | 3740 | 0 | 1:$\infty$ |

Continued on next page

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_156 | 2933 | 38 | 1:7 |
| BM_157 | 3679 | 0 | 1:∞ |
| BM_158 | 4972 | 28 | 1:17 |
| BM_159 | 2182 | 13 | 1:16 |
| BM_160 | 3455 | 3 | 1:115 |
| BM_161 | 1236 | 19 | 1:6 |
| BM_162 | 2160 | 16 | 1:13 |
| BM_163 | 2578 | 24 | 1:10 |
| BM_164 | 2580 | 21 | 1:12 |
| BM_165 | 2329 | 13 | 1:17 |
| BM_166 | 449 | 1 | 1:44 |
| BM_167 | 2487 | 21 | 1:11 |
| BM_168 | 869 | 7 | 1:12 |
| BM_169 | 3975 | 55 | 1:7 |
| BM_170 | 695 | 2 | 1:34 |
| BM_171 | 2035 | 24 | 1:8 |
| BM_172 | 1863 | 15 | 1:12 |
| BM_173 | 586 | 5 | 1:11 |
| BM_174 | 2147 | 0 | 1:∞ |
| BM_175 | 805 | 15 | 1:5 |
| BM_176 | 2820 | 0 | 1:∞ |
| BM_177 | 368 | 5 | 1:7 |
| BM_178 | 1868 | 31 | 1:6 |
| BM_179 | 2576 | 28 | 1:9 |
| BM_180 | 306 | 2 | 1:15 |
| BM_181 | 1931 | 5 | 1:38 |
| BM_182 | 3094 | 11 | 1:28 |
| BM_183 | 2846 | 46 | 1:6 |
| BM_184 | 4203 | 9 | 1:46 |
| BM_185 | 3275 | 5 | 1:65 |
| BM_186 | 3270 | 0 | 1:∞ |
| BM_187 | 3719 | 29 | 1:12 |
| BM_188 | 3274 | 11 | 1:29 |
| BM_189 | 2505 | 44 | 1:5 |
| BM_190 | 2131 | 13 | 1:16 |
| BM_191 | 2276 | 44 | 1:5 |
| BM_192 | 631 | 9 | 1:7 |
| BM_193 | 4802 | 23 | 1:20 |
| BM_194 | 554 | 8 | 1:6 |
| BM_195 | 1801 | 8 | 1:22 |
| BM_196 | 4675 | 0 | 1:∞ |
| BM_197 | 4632 | 0 | 1:∞ |
| BM_198 | 1757 | 32 | 1:5 |

Continued on next page

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_199 | 1595 | 19 | 1:8 |
| BM_200 | 2629 | 54 | 1:4 |
| BM_201 | 918 | 9 | 1:10 |
| BM_202 | 4774 | 80 | 1:5 |
| BM_203 | 348 | 0 | 1:∞ |
| BM_204 | 3848 | 0 | 1:∞ |
| BM_205 | 4535 | 27 | 1:16 |
| BM_206 | 1771 | 7 | 1:25 |
| BM_207 | 2399 | 31 | 1:7 |
| BM_208 | 2424 | 20 | 1:12 |
| BM_209 | 4680 | 4 | 1:117 |
| BM_210 | 2736 | 4 | 1:68 |
| BM_211 | 4245 | 39 | 1:10 |
| BM_212 | 3861 | 37 | 1:10 |
| BM_213 | 2257 | 34 | 1:6 |
| BM_214 | 3117 | 22 | 1:14 |
| BM_215 | 3450 | 37 | 1:9 |
| BM_216 | 3748 | 0 | 1:∞ |
| BM_217 | 4844 | 0 | 1:∞ |
| BM_218 | 4332 | 50 | 1:8 |
| BM_219 | 2500 | 14 | 1:17 |
| BM_220 | 802 | 8 | 1:10 |
| BM_221 | 4975 | 45 | 1:11 |
| BM_222 | 2725 | 0 | 1:∞ |
| BM_223 | 1749 | 0 | 1:∞ |
| BM_224 | 4219 | 36 | 1:11 |
| BM_225 | 4292 | 0 | 1:∞ |
| BM_226 | 4093 | 52 | 1:7 |
| BM_227 | 2020 | 24 | 1:8 |
| BM_228 | 2475 | 39 | 1:6 |
| BM_229 | 4526 | 0 | 1:∞ |
| BM_230 | 3360 | 41 | 1:8 |
| BM_231 | 1399 | 0 | 1:∞ |
| BM_232 | 2347 | 31 | 1:7 |
| BM_233 | 3844 | 1 | 1:384 |
| BM_234 | 927 | 6 | 1:15 |
| BM_235 | 1974 | 8 | 1:24 |
| BM_236 | 4490 | 73 | 1:6 |
| BM_237 | 1119 | 1 | 1:111 |
| BM_238 | 4861 | 11 | 1:44 |
| BM_239 | 1477 | 14 | 1:10 |
| BM_240 | 320 | 0 | 1:∞ |
| BM_241 | 3478 | 60 | 1:5 |

Continued on next page

Table B.6: Benchmark suite SB15_V2 containing multipliers and with a Demand Ratio of 1:15 and variance of 1:333

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_242 | 4065 | 29 | 1:14 |
| BM_243 | 4764 | 26 | 1:18 |
| BM_244 | 561 | 3 | 1:18 |
| BM_244 | 3685 | 58 | 1:6 |
| BM_245 | 2008 | 6 | 1:33 |
| BM_246 | 4269 | 46 | 1:9 |
| BM_247 | 4538 | 4 | 1:113 |
| BM_248 | 3155 | 17 | 1:18 |
| BM_249 | 2501 | 17 | 1:14 |

Table B.7: Benchmark suite SB15_V3 containing multipliers and with a Demand Ratio of 1:15 and a variance of 1:128

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_0 | 10750 | 0 | 1:∞ |
| BM_1 | 15864 | 0 | 1:∞ |
| BM_2 | 18054 | 416 | 1:4 |
| BM_3 | 14428 | 0 | 1:∞ |
| BM_4 | 18211 | 0 | 1:∞ |
| BM_5 | 19846 | 458 | 1:4 |
| BM_6 | 19576 | 0 | 1:∞ |
| BM_7 | 17607 | 135 | 1:13 |
| BM_8 | 17232 | 0 | 1:∞ |
| BM_9 | 12679 | 0 | 1:∞ |
| BM_10 | 18074 | 278 | 1:6 |
| BM_11 | 18918 | 0 | 1:∞ |
| BM_12 | 17269 | 398 | 1:4 |
| BM_13 | 19768 | 0 | 1:∞ |
| BM_14 | 15055 | 0 | 1:∞ |
| BM_15 | 10766 | 0 | 1:∞ |
| BM_16 | 16062 | 0 | 1:∞ |
| BM_17 | 13105 | 100 | 1:13 |
| BM_18 | 10465 | 0 | 1:∞ |
| BM_19 | 18714 | 288 | 1:6 |
| BM_20 | 15839 | 0 | 1:∞ |
| BM_21 | 16320 | 125 | 1:13 |
| BM_22 | 13174 | 101 | 1:13 |
| BM_23 | 14165 | 327 | 1:4 |
| BM_24 | 11607 | 0 | 1:∞ |
| BM_25 | 15582 | 239 | 1:6 |
| BM_26 | 12204 | 93 | 1:13 |
| BM_27 | 16950 | 0 | 1:∞ |
| BM_28 | 17904 | 275 | 1:6 |
| BM_29 | 10197 | 78 | 1:13 |
| BM_30 | 16280 | 375 | 1:4 |
| BM_31 | 11468 | 352 | 1:3 |
| BM_32 | 13945 | 0 | 1:∞ |
| BM_33 | 12810 | 98 | 1:13 |
| BM_34 | 18458 | 142 | 1:12 |
| BM_35 | 15790 | 243 | 1:6 |
| BM_36 | 16436 | 126 | 1:13 |
| BM_37 | 12417 | 0 | 1:∞ |
| BM_38 | 12390 | 95 | 1:13 |
| BM_39 | 16369 | 125 | 1:13 |
| BM_40 | 19778 | 760 | 1:2 |
| BM_41 | 13678 | 315 | 1:4 |

Continued on next page

Table B.7: Benchmark suite SB15_V3 containing multipliers and with a Demand Ratio of 1:15 and a variance of 1:128

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_42 | 12803 | 394 | 1:3 |
| BM_43 | 14160 | 436 | 1:3 |
| BM_44 | 13699 | 105 | 1:13 |
| BM_45 | 10018 | 0 | 1:∞ |
| BM_46 | 11672 | 359 | 1:3 |
| BM_47 | 18673 | 0 | 1:∞ |
| BM_48 | 12411 | 382 | 1:3 |
| BM_49 | 12731 | 196 | 1:6 |
| BM_50 | 15711 | 241 | 1:6 |
| BM_51 | 11518 | 88 | 1:13 |
| BM_52 | 16983 | 0 | 1:∞ |
| BM_53 | 18110 | 418 | 1:4 |
| BM_54 | 16869 | 129 | 1:13 |
| BM_55 | 16074 | 247 | 1:6 |
| BM_56 | 19134 | 147 | 1:13 |
| BM_57 | 14982 | 0 | 1:∞ |
| BM_58 | 13085 | 302 | 1:4 |
| BM_59 | 18996 | 0 | 1:∞ |
| BM_60 | 16203 | 249 | 1:6 |
| BM_61 | 13115 | 0 | 1:∞ |
| BM_62 | 16852 | 259 | 1:6 |
| BM_63 | 16196 | 0 | 1:∞ |
| BM_64 | 15975 | 0 | 1:∞ |
| BM_65 | 16484 | 380 | 1:4 |
| BM_66 | 13800 | 0 | 1:∞ |
| BM_67 | 10186 | 0 | 1:∞ |
| BM_68 | 14642 | 225 | 1:6 |
| BM_69 | 12861 | 99 | 1:12 |
| BM_70 | 11831 | 0 | 1:∞ |
| BM_71 | 14511 | 335 | 1:4 |
| BM_72 | 15041 | 463 | 1:3 |
| BM_73 | 19254 | 296 | 1:6 |
| BM_74 | 11340 | 0 | 1:∞ |
| BM_75 | 11263 | 0 | 1:∞ |
| BM_76 | 13616 | 104 | 1:13 |
| BM_77 | 11771 | 0 | 1:∞ |
| BM_78 | 18871 | 0 | 1:∞ |
| BM_79 | 16083 | 495 | 1:3 |
| BM_80 | 13247 | 203 | 1:6 |
| BM_81 | 19237 | 148 | 1:12 |
| BM_82 | 16921 | 0 | 1:∞ |
| BM_83 | 14184 | 218 | 1:6 |
| BM_84 | 10423 | 240 | 1:4 |

Continued on next page

Table B.7: Benchmark suite SB15_V3 containing multipliers and with a Demand Ratio of 1:15 and a variance of 1:128

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_85 | 19427 | 0 | 1:∞ |
| BM_86 | 12333 | 94 | 1:13 |
| BM_87 | 19571 | 0 | 1:∞ |
| BM_88 | 16136 | 0 | 1:∞ |
| BM_89 | 11853 | 182 | 1:6 |
| BM_90 | 17075 | 262 | 1:6 |
| BM_91 | 17725 | 0 | 1:∞ |
| BM_92 | 13846 | 106 | 1:13 |
| BM_93 | 17511 | 0 | 1:∞ |
| BM_94 | 13816 | 106 | 1:13 |
| BM_95 | 13916 | 321 | 1:4 |
| BM_96 | 18865 | 0 | 1:∞ |
| BM_97 | 10464 | 0 | 1:∞ |
| BM_98 | 15280 | 0 | 1:∞ |
| BM_99 | 10054 | 0 | 1:∞ |
| BM_100 | 18045 | 138 | 1:13 |
| BM_101 | 14309 | 0 | 1:∞ |
| BM_102 | 16123 | 124 | 1:13 |
| BM_103 | 14105 | 0 | 1:∞ |
| BM_104 | 19057 | 0 | 1:∞ |
| BM_105 | 12441 | 287 | 1:4 |
| BM_106 | 15426 | 0 | 1:∞ |
| BM_107 | 19527 | 0 | 1:∞ |
| BM_108 | 19662 | 151 | 1:13 |
| BM_109 | 14694 | 0 | 1:∞ |
| BM_110 | 13336 | 205 | 1:6 |
| BM_111 | 13247 | 101 | 1:13 |
| BM_112 | 18115 | 0 | 1:∞ |
| BM_113 | 14339 | 0 | 1:∞ |
| BM_114 | 11063 | 0 | 1:∞ |
| BM_115 | 15450 | 0 | 1:∞ |
| BM_116 | 10226 | 0 | 1:∞ |
| BM_117 | 12191 | 0 | 1:∞ |
| BM_118 | 12695 | 0 | 1:∞ |
| BM_119 | 14962 | 115 | 1:13 |
| BM_120 | 18607 | 0 | 1:∞ |
| BM_121 | 10588 | 81 | 1:13 |
| BM_122 | 13921 | 0 | 1:∞ |
| BM_123 | 16011 | 0 | 1:∞ |
| BM_124 | 19857 | 458 | 1:4 |
| BM_125 | 19766 | 152 | 1:13 |
| BM_126 | 15668 | 120 | 1:13 |
| BM_127 | 18098 | 0 | 1:∞ |

Continued on next page

Table B.7: Benchmark suite SB15_V3 containing multipliers and with a Demand Ratio of 1:15 and a variance of 1:128

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_128 | 19428 | 0 | 1:∞ |
| BM_129 | 19883 | 306 | 1:6 |
| BM_130 | 19535 | 0 | 1:∞ |
| BM_131 | 12304 | 94 | 1:13 |
| BM_132 | 17797 | 136 | 1:13 |
| BM_133 | 12464 | 0 | 1:∞ |
| BM_134 | 17144 | 0 | 1:∞ |
| BM_135 | 14949 | 230 | 1:6 |
| BM_136 | 17533 | 0 | 1:∞ |
| BM_137 | 18842 | 145 | 1:12 |
| BM_138 | 18357 | 282 | 1:6 |
| BM_139 | 17741 | 273 | 1:6 |
| BM_140 | 11910 | 91 | 1:13 |
| BM_141 | 14112 | 217 | 1:6 |
| BM_142 | 13225 | 305 | 1:4 |
| BM_143 | 14066 | 108 | 1:13 |
| BM_144 | 17352 | 0 | 1:∞ |
| BM_145 | 15849 | 0 | 1:∞ |
| BM_146 | 11986 | 184 | 1:6 |
| BM_147 | 13002 | 0 | 1:∞ |
| BM_148 | 19001 | 0 | 1:∞ |
| BM_149 | 12980 | 199 | 1:6 |
| BM_150 | 11948 | 0 | 1:∞ |
| BM_151 | 14601 | 337 | 1:4 |
| BM_152 | 19818 | 305 | 1:6 |
| BM_153 | 13639 | 0 | 1:∞ |
| BM_154 | 14648 | 0 | 1:∞ |
| BM_155 | 12742 | 98 | 1:13 |
| BM_156 | 17944 | 414 | 1:4 |
| BM_157 | 16841 | 0 | 1:∞ |
| BM_158 | 11768 | 90 | 1:13 |
| BM_159 | 16528 | 0 | 1:∞ |
| BM_160 | 17560 | 0 | 1:∞ |
| BM_161 | 12361 | 0 | 1:∞ |
| BM_162 | 16883 | 129 | 1:13 |
| BM_163 | 12034 | 92 | 1:13 |
| BM_164 | 13533 | 208 | 1:6 |
| BM_165 | 18381 | 0 | 1:∞ |
| BM_166 | 12943 | 199 | 1:6 |
| BM_167 | 17666 | 543 | 1:3 |
| BM_168 | 12778 | 98 | 1:13 |
| BM_169 | 10937 | 84 | 1:13 |
| BM_170 | 15389 | 0 | 1:∞ |

Continued on next page

Table B.7: Benchmark suite SB15_V3 containing multipliers and with a Demand Ratio of 1:15 and a variance of 1:128

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_171 | 16690 | 0 | 1:∞ |
| BM_172 | 13237 | 0 | 1:∞ |
| BM_173 | 10856 | 83 | 1:13 |
| BM_174 | 10791 | 0 | 1:∞ |
| BM_175 | 10479 | 80 | 1:13 |
| BM_176 | 16113 | 124 | 1:12 |
| BM_177 | 14250 | 0 | 1:∞ |
| BM_178 | 10907 | 0 | 1:∞ |
| BM_179 | 15232 | 0 | 1:∞ |
| BM_180 | 17651 | 271 | 1:6 |
| BM_181 | 18977 | 146 | 1:12 |
| BM_182 | 16597 | 127 | 1:13 |
| BM_183 | 19709 | 151 | 1:13 |
| BM_184 | 10652 | 81 | 1:13 |
| BM_185 | 19506 | 750 | 1:2 |
| BM_186 | 17030 | 0 | 1:∞ |
| BM_187 | 15727 | 0 | 1:∞ |
| BM_188 | 19728 | 0 | 1:∞ |
| BM_189 | 11625 | 268 | 1:4 |
| BM_190 | 16671 | 256 | 1:6 |
| BM_191 | 11574 | 0 | 1:∞ |
| BM_192 | 14000 | 0 | 1:∞ |
| BM_193 | 18485 | 0 | 1:∞ |
| BM_194 | 14814 | 342 | 1:4 |
| BM_195 | 12051 | 0 | 1:∞ |
| BM_196 | 13684 | 0 | 1:∞ |
| BM_197 | 19412 | 0 | 1:∞ |
| BM_198 | 14023 | 107 | 1:13 |
| BM_199 | 12360 | 0 | 1:∞ |
| BM_200 | 16120 | 124 | 1:13 |
| BM_201 | 17069 | 0 | 1:∞ |
| BM_202 | 14415 | 0 | 1:∞ |
| BM_203 | 16121 | 0 | 1:∞ |
| BM_204 | 18436 | 0 | 1:∞ |
| BM_205 | 17366 | 133 | 1:13 |
| BM_206 | 12555 | 0 | 1:∞ |
| BM_207 | 10596 | 244 | 1:4 |
| BM_208 | 17470 | 134 | 1:13 |
| BM_209 | 17815 | 0 | 1:∞ |
| BM_210 | 15504 | 0 | 1:∞ |
| BM_211 | 14749 | 0 | 1:∞ |
| BM_212 | 13599 | 104 | 1:13 |
| BM_213 | 18525 | 0 | 1:∞ |

Continued on next page

Table B.7: Benchmark suite SB15_V3 containing multipliers and with a Demand Ratio of 1:15 and a variance of 1:128

| Benchmark Name | Num BLEs | Number of Multipliers | Demand Ratio |
|---|---|---|---|
| BM_214 | 19635 | 604 | 1:3 |
| BM_215 | 15002 | 0 | 1:∞ |
| BM_216 | 15456 | 118 | 1:13 |
| BM_217 | 15257 | 0 | 1:∞ |
| BM_218 | 17572 | 270 | 1:6 |
| BM_219 | 10367 | 0 | 1:∞ |
| BM_220 | 15478 | 0 | 1:∞ |
| BM_221 | 17467 | 134 | 1:13 |
| BM_222 | 13027 | 0 | 1:∞ |
| BM_223 | 17293 | 0 | 1:∞ |
| BM_224 | 19850 | 305 | 1:6 |
| BM_225 | 19133 | 0 | 1:∞ |
| BM_226 | 17680 | 136 | 1:13 |
| BM_227 | 13598 | 0 | 1:∞ |
| BM_228 | 14396 | 110 | 1:13 |
| BM_229 | 11606 | 0 | 1:∞ |
| BM_230 | 19360 | 0 | 1:∞ |
| BM_231 | 16139 | 124 | 1:13 |
| BM_232 | 15032 | 231 | 1:6 |
| BM_233 | 11041 | 84 | 1:13 |
| BM_234 | 19128 | 147 | 1:13 |
| BM_235 | 14964 | 0 | 1:∞ |
| BM_236 | 15494 | 357 | 1:4 |
| BM_237 | 15343 | 0 | 1:∞ |
| BM_238 | 16848 | 259 | 1:6 |
| BM_239 | 11722 | 360 | 1:3 |
| BM_240 | 10450 | 0 | 1:∞ |
| BM_241 | 11108 | 0 | 1:∞ |
| BM_242 | 17442 | 0 | 1:∞ |
| BM_243 | 15761 | 121 | 1:13 |
| BM_244 | 13181 | 202 | 1:6 |
| BM_245 | 12222 | 0 | 1:∞ |
| BM_246 | 10329 | 158 | 1:6 |
| BM_247 | 10062 | 77 | 1:13 |
| BM_248 | 12463 | 0 | 1:∞ |
| BM_249 | 14318 | 0 | 1:∞ |

## B.4 Synthetic Benchmarks with Crossbars

We have created a set of synthetic benchmarks that use crossbars to test our concepts in Chapter 5. In this section, we show each of these benchmark suites including the number of crossbars and the demand ratio in each benchmark. Note that for each of these 100 benchmark suites we are showing the number of 16-16 crossbars, and for 32-32 or 64-64 you simply divide this number by four or sixteen respectively.

All the data for the other benchmarks, such as SB_1, can be extracted from Table B.8. For example, Benchmark suite SB_1 has the same data as Table B.8 except that benchmarks BM_1 through BM_99 have 0 crossbars and 1:$\infty$ demand ratio.

Table B.8: Benchmark suite SB_100 containing crossbars and with a Demand Ratio of 1:15

| Benchmark Name | Num BLEs | Number of Crossbars | Demand Ratio |
|---|---|---|---|
| BM_0 | 1811784 | 11323 | 1:15 |
| BM_1 | 1156971 | 96163 | 1:19 |
| BM_2 | 1038038 | 143812 | 1:11 |
| BM_3 | 1546097 | 64916 | 1:38 |
| BM_4 | 1123256 | 170881 | 1:10 |
| BM_5 | 1129751 | 115938 | 1:15 |
| BM_6 | 1966727 | 164269 | 1:19 |
| BM_7 | 1980449 | 76249 | 1:41 |
| BM_8 | 1908872 | 152542 | 1:20 |
| BM_9 | 1431429 | 107700 | 1:21 |
| BM_10 | 1804627 | 100636 | 1:28 |
| BM_11 | 1723559 | 96867 | 1:28 |
| BM_12 | 1336816 | 105814 | 1:20 |
| BM_13 | 1091996 | 62706 | 1:27 |
| BM_14 | 1240360 | 34262 | 1:57 |
| BM_15 | 1041397 | 81651 | 1:20 |
| BM_16 | 1737804 | 16144 | 1:172 |
| BM_17 | 1252973 | 70188 | 1:28 |
| BM_18 | 1087754 | 1956 | 1:891 |
| BM_19 | 1466427 | 10218 | 1:229 |
| BM_20 | 1476901 | 81801 | 1:28 |
| BM_21 | 1821685 | 74414 | 1:39 |
| BM_22 | 1970330 | 183079 | 1:17 |
| BM_23 | 1335776 | 74107 | 1:28 |
| BM_24 | 1059936 | 8121 | 1:209 |
| BM_25 | 1831120 | 33094 | 1:88 |

Continued on next page

Table B.8: Benchmark suite SB_100 containing crossbars and with a Demand Ratio of 1:15

| Benchmark Name | Num BLEs | Number of Crossbars | Demand Ratio |
|---|---|---|---|
| BM_26 | 1285459 | 114373 | 1:17 |
| BM_27 | 1880349 | 59442 | 1:50 |
| BM_28 | 1487814 | 73011 | 1:32 |
| BM_29 | 1379676 | 84929 | 1:25 |
| BM_30 | 1101072 | 161461 | 1:10 |
| BM_31 | 1296672 | 147986 | 1:14 |
| BM_32 | 1135793 | 16748 | 1:108 |
| BM_33 | 1131071 | 88130 | 1:20 |
| BM_34 | 1381420 | 141160 | 1:15 |
| BM_35 | 1467263 | 34751 | 1:67 |
| BM_36 | 1710028 | 29746 | 1:91 |
| BM_37 | 1334654 | 116206 | 1:18 |
| BM_38 | 1607435 | 145668 | 1:17 |
| BM_39 | 1592264 | 97691 | 1:26 |
| BM_40 | 1953988 | 131447 | 1:23 |
| BM_41 | 1564638 | 138222 | 1:18 |
| BM_42 | 1010769 | 84606 | 1:19 |
| BM_43 | 1894427 | 13352 | 1:227 |
| BM_44 | 1933684 | 80501 | 1:38 |
| BM_45 | 1743110 | 120347 | 1:23 |
| BM_46 | 1727880 | 43493 | 1:63 |
| BM_47 | 1906336 | 23398 | 1:130 |
| BM_48 | 1622132 | 106063 | 1:24 |
| BM_49 | 1603573 | 71596 | 1:35 |
| BM_50 | 1151257 | 8114 | 1:227 |
| BM_51 | 1894577 | 96730 | 1:31 |
| BM_52 | 1876137 | 109890 | 1:27 |
| BM_53 | 1057996 | 169493 | 1:9 |
| BM_54 | 1416015 | 197449 | 1:11 |
| BM_55 | 1183647 | 155684 | 1:12 |
| BM_56 | 1589998 | 39729 | 1:64 |
| BM_57 | 1642155 | 76322 | 1:34 |
| BM_58 | 1709624 | 125048 | 1:21 |
| BM_59 | 1615702 | 92416 | 1:27 |
| BM_60 | 1843389 | 68828 | 1:42 |
| BM_61 | 1710768 | 79706 | 1:34 |
| BM_62 | 1431299 | 44490 | 1:51 |
| BM_63 | 1434942 | 134489 | 1:17 |
| BM_64 | 1170864 | 142760 | 1:13 |
| BM_65 | 1089687 | 96905 | 1:17 |
| BM_66 | 1142056 | 101270 | 1:18 |
| BM_67 | 1699759 | 51588 | 1:52 |
| BM_68 | 1729849 | 13732 | 1:201 |

Continued on next page

Table B.8: Benchmark suite SB_100 containing crossbars and with a Demand Ratio of 1:15

| Benchmark Name | Num BLEs | Number of Crossbars | Demand Ratio |
|---|---|---|---|
| BM_69 | 1534469 | 23626 | 1:103 |
| BM_70 | 1924369 | 104513 | 1:29 |
| BM_71 | 1548571 | 97476 | 1:25 |
| BM_72 | 1499105 | 73330 | 1:32 |
| BM_73 | 1147838 | 87700 | 1:20 |
| BM_74 | 1760012 | 30601 | 1:92 |
| BM_75 | 1368389 | 187033 | 1:11 |
| BM_76 | 1638705 | 63141 | 1:41 |
| BM_77 | 1174670 | 28796 | 1:65 |
| BM_78 | 1728966 | 120899 | 1:22 |
| BM_79 | 1639788 | 32973 | 1:79 |
| BM_80 | 1680833 | 121292 | 1:22 |
| BM_81 | 1738303 | 124351 | 1:22 |
| BM_82 | 1230093 | 10348 | 1:190 |
| BM_83 | 1873787 | 176561 | 1:16 |
| BM_84 | 1228523 | 76079 | 1:25 |
| BM_85 | 1138510 | 42139 | 1:43 |
| BM_86 | 1159077 | 99118 | 1:18 |
| BM_87 | 1699963 | 76877 | 1:35 |
| BM_88 | 1910512 | 108541 | 1:28 |
| BM_89 | 1866912 | 67851 | 1:44 |
| BM_90 | 1644045 | 119467 | 1:22 |
| BM_91 | 1273926 | 36654 | 1:55 |
| BM_92 | 1347163 | 88407 | 1:24 |
| BM_93 | 1869108 | 143389 | 1:20 |
| BM_94 | 1090913 | 98651 | 1:17 |
| BM_95 | 1492311 | 10518 | 1:227 |
| BM_96 | 1754724 | 75435 | 1:37 |
| BM_97 | 1627340 | 89692 | 1:29 |
| BM_98 | 1185029 | 151092 | 1:12 |
| BM_99 | 1977463 | 65088 | 1:48 |

## B.5 Summary

In this appendix, we have provided more of the details of the benchmarks used throughout this dissertation. This includes not only the size of these benchmarks in terms of BLEs, but also details about the number of multipliers or crossbars, the number of memory bits, and the number of input and output pins in the benchmarks.

# References


[ACH+97] Charles J. Alpert, Tony F. Chan, Dennis Huang, Andrew Kahnh, Igor Markov, Pep Mulet, and Kenneth Yan. Faster Minimization of Linear Wirelength for Global Placement. In *International Symposium on Physical Design*, pages 4–11, Napa Valley, CA, 1997.

[Act96] Actel. *ACT 1 Series FPGAs*, 1996.

[Act02] Actel. *ProASIC 500K Family*, 2002.

[AKE+04] Navid Azizi, Ian Kuon, Aaron Egier, Ahmad Darabiha, and Paul Chow. Reconfigurable Molecular Dynamics Simulator. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 197–206, April 2004.

[Alt02] Altera. *Using Stratix GX in Switch Fabric Systems*, 2002. Altera White Paper.

[Alt03] Altera. *Stratix Device Handbook*, Jul 2003.

[Alt04a] Altera. *APEX 20K, Programmable Logic Device Family*, March 2004.

[Alt04b] Altera. *Crosspoint Switch Matrices in MAX II & MAX 2000A Devices*, 2004. Altera Application Note 294.

[Alt04c] Altera. *Quartus II Handbook, Volumes 1, 2, and 3*, 2004.

[Alt04d] Altera. *Stratix II Device Handbook*, 2004.

[Alt04e] Altera. Stratix II Performance and Logic Efficiency Analysis. Feb 2004.

[Alt04f] Altera. www.altera.com/products/devices/arm/overview/arm-overview.html, 2004.

[Alt06] Altera. *Stratix III Device Handbook*, 2006.

[Alt07a] Altera. *Cyclone III Device Handbook*, 2007.

[Alt07b] Altera. *Cyclone III Device Handbook*, 2007.

[AR00] E. Ahmed and J. Rose. The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density. In *ACM/SIGDA International Symposium on FPGAs*, pages 3–12, Feb 2000.

[ARM01] ARM. *ARM922T System-on-Chip Platform OS Processor*, 2001.

[BCC+91] J. Birkner, A. Chan, H.T Chua, A. Chao, K. Gordon, B. Kleinman, P. Kolze, and R. Wong. A Very High-Speed Field Programmable Gate Array Using Metal-to-Metal Anti-Fuse Programmable Elements. In *New Hardware Product Introduction at CICC '91*, 1991.

[BFRV92] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, Boston, 1992.

[BHSV90] R. K. Brayton, G. D. Hachtel, and A. L. Sangionvanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, 78(2):264–300, February 1990.

[BHUH06] M. J. Beauchamp, S. Hauck, K.D. Underwood, and K.S. Hemmert. Embedded Floating Point Units in FPGAs. In *ACM/SIGDA International Symposium on FPGAs*, pages 12–20, Feb 2006.

[BL90] J. Bhasker and Huan-Chih Lee. An Optimizer for Hardware Synthesis. *IEEE Design & Test*, 7(5):20–36, October 1990.

[BL03] G. Brebner and D. Levi. Networking on Chip with Platform FPGAs. In *IEEE International Conference on Field-Programmable Technology*, pages 13–20, Dec 2003.

[Bor99] S. Borkar. Design challenges of technology scaling. *IEEE MICRO*, 19(4):23–29, July–August 1999.

[BRM99] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[BRV90] S. Brown, J. Rose, and Z. G. Vranesic. A Detailed Router for Field-Programmable Gate Arrays. In *Proceedings of the International Conference on Computer Aided Design*, pages 382–385, 1990.

[BRV92] S. Brown, J. S. Rose, and Z. Vranesic. A Detailed Router for Field Programmable Gate Arrays. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 11(5):620–628, May 1992.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[BVOP03] Murat Becer, Radi Vaidyanathan, Chanchee Oh, and Rajendra Panda. Signal Integrity Management in an SoC Physical design Flow. In *International Symposium on Physical Design*, pages 110–117, Monterey, CA, 2003.

[CC04] D. Chen and J. Cong. DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs. In *IEEE International Conference on Computer Aided Design*, pages 752–759, Nov 2004.

[CCD+92] K. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar. DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization. In *IEEE Design and Test of Computers*, pages 7–20, September 1992.

[CD94] J. Cong and Y. Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimizationin Lookup-Table Based FPGA Designs. *IEEE Trans. Computer-aided Design*, 13(1):1–12, 1994.

[CEWWM+99] A. R. Conn, I. M. Elfadel, Jr. W. W. Molzen, P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan. Gradient-based optimization of custom circuits using a static-timing formulation. In *Proceedings of Design Automation Conference*, pages 452–459, New York, NY, USA, 1999.

[CFHZ04] Jason Cong, Yiping Fan, Guoling Han, and Zhiru Zhang. Application-specific instruction generation for configurable processor architectures. In *ACM/SIGDA International Symposium on FPGAs*, pages 183–189, 2004.

[CH00] Jason Cong and Yean-Yow Hwang. Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA designs. *ACM Transactions on Design Automation of Elctronic Systems*, 5(2):193–225, 2000.

[CPD96] Jason Cong, John Peck, and Yuzheng Ding. RASP: A General Logic Synthesis System for SRAM-Based FPGAs. In *ACM/SIGDA International Symposium on FPGAs*, pages 137–143, 1996.

[CSR+99] Paul Chow, Soon Ong Seo, Jonathan Rose, Kevin Chung, Gerard Paez Monzon, and Immanuel Rahadja. The Design of a SRAM-Based Field-Programmable Gate Array - Part II: Circuit Design and Layout. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(3):321–331, September 1999.

[CTZW94] Y.-W. Chang, S. Thakur, K. Zhu, and D. F. Wong. A New Global Routing Algorithm for FPGAs. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 356–361, 1994.

[CX00] J. Cong and S. Xu. Performance-Driven Technology Mapping for Heterogeneous FPGAs. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 19(11):1268–1280, November 2000.

[DK85] Alfred E. Dunlop and Brian W Kernighan. A Procedure for Placement of Standard-Cell VLSI Circuits. *IEEE Transactions on Computer-Aided Design*, 4(1):92–98, January 1985.

[DMNSV87] Srinivas Devada, Hi-Keung Ma, A. Newton, and A. Sangiovanni-Vincentelli. MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations. *IEEE Transactions on Computer-Aided Design*, (12):1290–1300, December 1987.

[DRM03] A. Dharabiha, J. Rose, and W.J. MacLean. Video-Rate Stereo Depth Measurement on Programmable Hardware. In *IEEE Computer Society Conference on Computer Vision & Pattern Recognition*, pages 203–210, 2003.

[Egi05] Aaron C. Egier. Enhancing and Using an Automatic Design System for Creating FPGAs. Master's thesis, University of Toronto, 2005.

[FD85] J. P. Fishburn and A.E. Dunlop. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *International Conference on Computer Aided Design*, pages 326–328, November 1985.

[FR03] J. Fender and J. Rose. A High-Speed Ray TRacing Engine Built on a Field-Programmable System. In *IEEE International Conf. On Field-Programmable Technology*, pages 188–195, 2003.

[FRV91] R. J. Francis, J. Rose, and Z. Vranesic. Chortle-crf: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of Design Automation Conference*, pages 613–619, 1991.

[Gam81] A. El Gamal. Two-dimensional stochastic model for interconnections in master slice integrated circuits. *IEEE Transactions on Circuits and Systems*, 28(2):127–138, 1981.

[Gig04] Paul Gigliotti. Implementing Barrel Shifters Using Multipliers. *XAPP - Application Note: Virtex-II Family*, pages 1–4, August 2004.

[Gol93] Steve Golson. One-hot state machine design for FPGAs. In *3rd PLD Design Conference*, pages 1–6, Santa Clara, CA, March 1993.

[HBO+93] D. Hill, B. Britton, B. Oswald, N-S Woo, S. Singh, C-T Chen, and B Krambeck. Optimized Reconfigurable Cell Array Architecture for High-performance Field Programmable Gate Arrays. In *Proceeding of IEEE Custom Integrated Circuits Conference 1993*, pages 7.2.1–7.2.5, 1993.

[HHF98] Scott Hauck, Matthew M. Hosler, and Thomas W. Fry. High-Performance Carry Chains for FPGAs. In *ACM/SIGDA International Symposium on FPGAs*, pages 223–233, 1998.

[HK97] Dennis Huang and Andrew Khang. Partitioning-Based Standard-cell global placement with an Exact Objective. In *International Symposium on Physical Design*, pages 18–25, Napa Valley, CA, 1997.

[HLL+06] C.H Ho, P.H.W. Leong, W. Luk, S.J.E. Wilton, and S. Lopex-Buedo. Virtual Embedded Blocks: A methodology for evaluating embedded elements in FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 35–44, 2006.

[HR93] Jianshe He and Jonathan Rose. Advantages of Heterogeneous Logic Block Architectures for FPGAs. In *IEEE Custom Integrated Circuits Conf.*, pages 7.4.1–7.4.5, San Diego, CA, 1993.

[IEE87] IEEE. *IEEE Standard VHDL Language Reference Manual*, 1987.

[JR05a] Peter Jamieson and Jonathan Rose. A Verilog RTL Synthesis Tool for Heterogeneous FPGAs. In *Field-Programmable Logic and Applications*, pages 305–310, 2005.

[JR05b] Peter Jamieson and Jonathan Rose. Mapping Multiplexers onto Hard Multipliers in FPGAs. In *3rd International IEEE Northeast Workshop on Circuits & Systems*, pages 215–226, 2005.

[JR06] Peter Jamieson and Jonathan Rose. Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters. In *IEEE International Conference on Field-Programmable Technology*, pages 1–8, 2006.

[JW03] C. Jones and S. Wilton. Cascadable bus based crossbar switch in a Programmable Logic Device. *U.S. Patent 6,590,417. Issued July 8th*, 2003.

[JW04] C. Jones and S. Wilton. Cascadable bus based crossbar switching in a Programmable Logic Device. *U.S. Patent 6,710,623. Issued March 23rd*, 2004.

[KER05] Ian Kuon, Aaron Egier, and Jonathan Rose. Design, Layout, and Verification of an FPGA using Automated Tools. In *ACM/SIGDA International Symposium on FPGAs*, pages 215–226, Feb 2005.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[Kil73] G. A. Kildall. A unified approach to global program optimization. In *Annual Symposium on Principles of Programming Laguages*, pages 194–206, 1973.

[KN02] H. Kariniemi and J. Numi. A Crossbar-Based ATM Switch on FPGA for 2.488 Gbits/s CATV Network with Scaleable Header Remapping Function. In *Communication Systems, Networks, and Digital Signal Processing Symposium*, pages 82–85, July 2002.

[Kna99] Steve Knapp. Constant-coeficient multipliers save FPGA space, time. *Personal Engineering*, pages 45–48, July 1999.

[KOMBS01] Ryan Kastner, Seda Ogrenci-Memik, Elaheh Bozorgzadeh, and Majid Sarrafzadeh. Instruction Generation for Hybrid Reconfigurable Systems. In *IEEE International Conference on Computer Aided Design*, pages 127–131, San Jose, CA, 2001.

[KR06] Ian Kuon and Jonathan Rose. Measuring the Gap Between FPGAs and ASICs. In *ACM/SIGDA International Symposium on FPGAs*, pages 21–30, Feb 2006.

[KSJA91] Jurgen Kleinhans, George Sigl, Frank M. Johannes, and Kurt J. Antreich. GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization. *IEEE Transactions on Computer-Aided Design*, 10(3):356–365, March 1991.

[Kuo04] Ian Kuon. Automated FPGA Designs. Verification and Layout. Master's thesis, University of Toronto, 2004.

[Kuo07] Ian Kuon. Private Communications with Ian Kuon. 2007.

[LAB+05] David Lewis, Elias Ahmed, Gregg Baeckler, Vaughn Betz, Mark Bourgeault, David Cashman, David Galloway, Mike Hutton, Chris Lane, Andy Lee, Paul Leventis, Sandy Marquardt, Cameron McClintock, Ketan Padalia, Bruce Pedersen, Giles Powell, Boris Ratchev, Srinivas Reddy, Jay Schleicher, Kevin Stevens, Richard Yuan, Richard Cliff, and Jonathan Rose. The Stratix II Logic and Routing Architecture. In *ACM/SIGDA International Symposium on FPGAs*, pages 14–20, Feb 2005.

[Lat04] Lattice. *ispXPGA Family*, Jan 2004.

[Lat07a] Lattice. *LatticeECP*, Feb 2007.

[Lat07b] Lattice. *LatticeECP2*, Apr 2007.

[LB93] G. Lemieux and S. Brown. A Detailed Routing Algorithm for Allocating Wire. In *ACM/SIGDA Physical Design Workshop*, pages 215–226, 1993.

[Lew06] David Lewis. Private Communications with David Lewis. 2006.

[Lie03] Lars Liebmann. Layout Impact of Resolution Enhancement techniques: Impediment or Opportunity? In *International Symposium on Physical Design*, pages 110–117, Monterey, CA, 2003.

[LL01] G. Lemieux and D. Lewis. Using Sparse Crossbars within LUT clusters. In *ACM/SIGDA International Symposium on FPGAs*, pages 59–68, Feb 2001.

[LL04] G. Lemieux and D. Lewis. Directional and Single-Driver Wires in FPGA Interconnect. In *IEEE International Conference on Field-Programmable Technology*, pages 41–48, Dec 2004.

[LPM93] Electronic Industries Association standard for Library Parametrized Modules. www.edif.org/lpmweb/intro/what_is_lpm.htm, 1993.

[LW95] Y.-S. Lee and A. Wu. A Performance and Routability Driven Router for FPGAs Considering Path Delay. In *Proceedings of Design Automation Conference*, pages 557–561, 1995.

[Mag05] Magma Design Automation Inc. Blast FPGA. 2005.

[MBR99] Alexander Marquardt, Vaughn Betz, and Jonathan Rose. Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density. In *ACM/SIGDA International Symposium on FPGAs*, pages 37–46, Monterey, CA, 1999.

[MC92] D. Marple and L. Cooke. An MPGA Compatible FPGA Architecture. In *ACM/SIGDA Workshop on FPGAs*, pages 39–44, 1992.

[MCC05] G. Morris, G. Constantinides, and P. Cheung. Using DSP Blocks for ROM Replacement: A Novel Synthesis Flow. In *Field-Programmable Logic and Applications*, pages 77–82, Aug 2005.

[MDM+95] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *IEEE Journal of Solid-State Circuits*, 30(8):847–854, 1995.

[Men01] Mentor Graphics. LeanardoSpectrum. 2001.

[Mic94] Giovanni Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.

[Mic07] CMC Microsystems, 2007. http://www.cmc.ca.

[MN05] Paul Metzgen and Dominic Nancekievill. Multiplexer restructuring for FPGA implementation cost reduction. In *Proceedings of Design Automation Conference*, pages 421–426, 2005.

[Nga94] T. Ngai. *An SRAM-programmable Field-Reconfigurable Memory*. PhD thesis, University of Toronto, 1994.

[NRSVT88] William Nye, David C. Riley, Alberto Sangiovanni-Vincentelli, and Andre L. Tits. DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits. *IEEE Transactions on CAD*, 7(4):501–519, 1988.

[OEGS93] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm. In *Proceedings of Design Automation Conference*, pages 31–37, 1993.

[Ope93] Open Verilog International. *Verilog Hardware Description Reference*, March 1993.

[ope07] www.opencores.org. 2007.

[Pla01] www.cs.nthu.edu.tw/∼ylin/, 2001.

[Ple90] Plessey. *Plessey Semiconductor ERA60100 Advance Information, Data Sheet*, 1990.

[Qui03] QuickLogic. *Eclipse Family Data Sheet*, 2003.

[RFCL89] J. S. Rose, R. J. Francis, P. Chow, and D. Lewis. The Effect of Logic Block Complexity on Area of Programmable Gate Arrays. In *IEEE Custom Integrated Conference*, pages 5.3.1 – 5.3.5, May. 1989.

[RFLC90] J.S. Rose, R. J. 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*, 25(5):1217–1225, October 1990.

[Roo06] Ajay Roopchansingh. Private Communications with Ajay Roopchansingh. 2006.

[Ros04] Jonathan Rose. Hard vs. Soft: The Central Question of Pre-Fabricated Silicon. In *34th International Symposium on Multiple-Valued Logic (ISMVL'04)*, pages 2–5, Toronto, ON, May 2004.

[SBR98] Jordan S. Swartz, Vaughn Betz, and Jonathan Rose. A fast routability-driven router for FPGAs. In *Proceedings of the ACM/SIGDA International symposium on Field programmable gate arrays*, pages 140–149, 1998.

[scu98] www.engr.scu.edu/mourad/benchmark/RTL-Bench.html, 1998.

[SDJ91] Georg Sigl, Konrad Doll, and Frank M. Johannes. Analytical Placement: A Linear or a Quadratic Objective Function? In *Proceedings of Design Automation Conference*, pages 427–432, 1991.

[Smi06] A. M. Smith. *Heterogeneous Reconfigurable Architecture Design: An Optimisation Approach*. PhD thesis, Imperial College, 2006.

[SMS02] Amit Singh and Malgoratza Marek-Sadowska. Efficient Circuit Clustering for Area and Power Reduction in FPGAs. In *ACM/SIGDA International Symposium on FPGAs*, pages 59–66, 2002.

[SRRJ00] Guenter Stenz, Bernhard M. Reiss, Bernhard Rohfleisch, and Frank M. Johannes. Performance Optimization by Interacting Netlist Transformations and Placement. *IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems*, 19(3):350–358, March 2000.

[STM05] STMicroelectronics. 90nm CMOS090 Design Platform, 2005. http://www.st.com/stonline/prodpres/dedicate/soc/asic/90plat.htm.

[Syn99] Synopsys. *Star-Hspice Manual*, 1999.

[Syn01] Synopsys. *Nanosim Reference Guide*, 2001.

[Syn03] Synplicity. Synplify Pro. 2003.

[Syn04] Synopsys. Design Compiler FPGA. 2004.

[tex97] www-cad.eecs.berkeley.edu/Respep/Research/vis/texas-97/, 1997.

[Wil97] S. Wilton. *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories*. PhD thesis, Toronto, ON, 1997.

[Wil99] S.J.E. Wilton. FPGA Embedded Memory Architectures: Recent Research Results. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1999.

[Wil02] Steven J.E. Wilton. Implementing Logic in FPGA Memory Arrays: Heterogeneous Memory Architectures. In *IEEE Custom Integrated Circuits Conference*, pages 142–149, May 2002.

[Wil07] Stephen Williams. ICARUS Verilog at www.icarus.com/eda/verilog/. 2007.

[WKH99] Steven J.E. Wilton William K.C. Ho. Logical-to-Physical Memory Mapping for FPGAs with Dual-Port Embedded Arrays. In *International Workshop on Field Programmable Logic and Applications*, 1999.

[WRV96] S.J.E. Wilton, J. Rose, and Z.G. Vranesic. Memory/Logic Interconnect Flexibility in FPGAs with Large Embedded Memory Arrays. In *IEEE Custom Integrated Circuits Conference*, 1996.

[WRV97] S.J.E. Wilton, J. Rose, and Z.G. Vranesic. Memory-to-Memory Connection Structures in FPGAs with Embedded Memory Arrays. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 10–16, 1997.

[WRV99] S.J.E. Wilton, J. Rose, and Z.G. Vranesic. The Memory/Logic Interface in FPGA's with Large Embedded Memory Arrays. *IEEE Transactions on Very-Large Scale Integration Systems*, 7(1), March 1999.

[Xil89] Xilinx. *The Programmable Gate Array Data Book*, 1989.

[Xil00] Xilinx. *High-Speed Buffered Crossbar Switch Design Using Virtex-EM Devices*, 2000. Xilinx Application Note 240.

[Xil03] Xilinx. *Virtex-II Pro Platform FPGAs: Functional Description*, Oct 2003.

[Xil04] Xilinx. *Xilinx ISE 6 Software Manuals and Help*, 2004.

[Xil05] Xilinx. *Virtex-4 Family Overview*, March 2005.

[Xil06] Xilinx. *Virtex-5 Family Overview*, June 2006.

[Xil07] Xilinx. *Spartan-3A DSP FPGA FAmily: Complete Data Sheet*, Apr 2007.

[YAF+03] S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget, and D. Levi. A high I/O reconfigurable crossbar switch. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 3–10, April 2003.

[Ye06] Andy Ye. Private Communications with Andy Ye. 2006.

[YW94] Honghua Yang and D. F. Wong. Edge-Map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs. In *Proceedings of the International Conference on Computer Aided Design*, pages 150–155, 1994.

[ZJC+06] Y. Zhang, T. Jeong, F. Chen, H. Wu, R. Nitzsche, and G. Gao. A study of the on-chip interconnection network for the IBM Cyclops64 multi-core architecture. In *IEEE International Parallel & Distributed Processing Symposium*, pages 1–10, April 2006.