Peer Reviewed

Title:
Building Efficient, Reconfigurable Hardware using Hierarchical Interconnects

Author:
Wang, Chengcheng

Acceptance Date:
2013

Series:
UCLA Electronic Theses and Dissertations

Degree:
Ph.D., Electrical Engineering 0303UCLA

Advisor(s):
Markovic, Dejan

Committee:
Srivastava, Mani B., Kaiser, William J., Gerla, Mario

Permalink:
http://escholarship.org/uc/item/2vt0b5cb

Abstract:

UNIVERSITY OF CALIFORNIA

Los Angeles

Building Efficient, Reconfigurable Hardware using Hierarchical Interconnects

A thesis submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

Chengcheng Wang

2013

ABSTRACT OF THE DISSERTATION

Building Efficient, Reconfigurable Hardware using Hierarchical Interconnects

by

Chengcheng Wang

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2013

Professor Dejan Marković, Chair

In the semiconductor industry today, ASICs are able to offer 10x-1000x higher energy and area efficiencies than non-dedicated chips, such as programmable DSP processers, field-programmable gate arrays (FPGAs), and microprocessors. Not surprisingly, SoCs today have become an integration of many ASIC blocks, each performing a few dedicated tasks. The growing size of modern SoC chips, accelerated by the increasing demands for functionalities, has exposed the major drawback of ASIC: design cost. These large SoCs are re-designed a few times a year to rectify hardware-bugs and to support new features. Because ASICs are not reconfigurable, even the smallest hardware change would require a re-design. Additionally, design cost is rising exponentially with every technology generation.

The rising design cost of ASICs has exposed a huge need today: efficiency and flexibility must co-exist. But among flexible hardware candidates, microprocessors and programmable DSP

processors are far too slow to meet the throughput requirements of ASICs. FPGAs do come close in terms of performance, but are extremely inefficient due to its high energy and large area overhead. We must bridge the huge gap in efficiency for FPGA to become a viable contender to ASICs.

The primary culprit for FPGA inefficiency is interconnect, which accounts for over 75% of area and delay. For over 20 years, 2D-mesh network has been the back-bone of FPGA interconnects, but full connectivity in a 2D-mesh require O(N2) switches, requiring interconnects to grow much faster than Moore's Law. As a result, various heuristics are used to simplify switch-box arrays at the cost of resource utilization, but interconnect area of modern FPGA is still around 80%. This work builds FPGA using hierarchical interconnects based on Beneš networks, requiring O(N·log·N) switches. Although Beneš is commonly used in telecommunication, this work is its first silicon realization of a FPGA. To realize a highly efficient interconnect architecture, significant pruning of the network is required. Novel techniques such as fast-path U-turns and unbalanced branching are also implemented. A custom place-and-route software is developed to map benchmark designs on a variety of interconnect candidates. From mapping results, the architecture is updated based on network utilization until an optimized design is converged. The large area of FPGA chip requires aggressive power gating (PG), but interconnect signals often lack spatial locality, make it block-level PG difficult. A novel PG circuit technique is developed to power-gate individual interconnect switches with very small overhead in area and performance. Such technique requires fundamental circuit changes, even modifying the CMOS inverter.

With innovations in chip architecture, circuit design, and extensive software development, this work has demonstrated 5 user-mappable FPGAs (from 1K–16K LUTs) all

with around 50% interconnect area: a 3–4x reduction from commercial FPGAs while preserving connectivity. An energy efficiency of 1.1 GOPS/mW is the highest among reported FPGAs, and is 22x more efficient than the most efficient commercial FPGA today, significantly bridging the efficiency gap between FPGA and ASIC.

The dissertation of Chengcheng Wang is approved.

Mani B. Srivastava

William J. Kaiser

Mario Gerla

Dejan Marković, Committee Chair

University of California, Los Angeles

2013

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It has been six years since I started my graduate life at UCLA, and it has certainly been the best six years of my life so far. First of all, I am wholeheartedly thankful to my advisor, Dejan Marković, for his patience, knowledge, and sheer passion for this work. Additionally, I've also learned a great lot from him about presentation and communication skills.

I wish to thank Professor Mani Srivastava, William Kaiser, and Mario Gerla for being on my dissertation committee. Their helpful and thoughtful comments are definitely appreciated.

I am also grateful for having the best group members, especially Fang-Li Yuan and Tsung-Han Yu, who have endurance endless nights with me during tape-out madness and chip testing. They are the hardest working colleagues I've ever had, and yet also exert such positive energy. I would also like to acknowledge other lab members, especially Vaibhav Karkare and Yuta Toriyama for incessant discussions in the cubicles, technical or not. It is really difficult to find a group so diverse, and yet so unified; so technically strong, and yet so pleasant and interesting to be around.

I sincerely thank my parents for their never-ending care, and for always being my closest teacher and counselor. They shaped me the way I am today, and I am forever indebted to them. I also wish to thank my (soon-to-be-wife) Helen for her daily support and for being the biggest blessing in my life. Above all, I thank my God and Savior. His patience, grace, and love have been my greatest strength.

CURRICULUM VITAE

EDUCATION

M.S., Electrical Engineering, University of California, Los Angeles, (GPA 3.76)
2007-2009

B.S., Electrical Engineering and Computer Sciences, University of California, Berkeley, (GPA 3.8)
2003-2005

EMPLOYMENT HISTORY

Graduate Student Researcher – UCLA Electrical Engineering: Fall 2007 – Spring 2013

Design and Optimization of Low-Power ASIC and FPGA

- Developing FPGA with a novel interconnect architecture that significantly reduces interconnect area and power by 3-4x compared to existing FPGA architectures. Chips fabricated using IBM90, ST65, TSMC65, IBM45SOI, and TSMC40 processes. Single-handedly performing all aspects of the project, from chip architecture, circuit design, to software tool design. The most recent test chip is by far the largest VLSI chip made in UCLA, and is one of the most complex chips made by any academic institution.
- Extensive experience in high-performance, low-power digital circuit design. Developed novel circuits for low-leakage power gating and high-speed interconnect performance. Also developed a more accurate delay model to compensate for the lack of accuracy in logical effort models under low-power optimizations.

Nano-electro-mechanical Relays

- Designing circuits using nano-electro-mechanical relays, which have infinite off-impedance, low on-impedance, and low threshold voltage, making them attractive for digital-circuit, power-gating, and especially FPGA applications.

Word-length Optimization

- Developed and maintained a word-length optimization too to automatically determines the optimal word-lengths of every logic block given a quantization-error requirement. Very effective for power-performance optimization in the system level, especially when combined with architectural optimizations.

VLSI Design Engineer - Zoran Corporation, Sunnyvale, CA: Fall 2005 - Fall 2007

Designed numerous blocks for HDTV applications, including H*264 decoding, HD video capture (component and HDMI), histogram computation, MPEG post-processing, and others.

Involved with the entire design flow, including RTL design, verification, synthesis, timing-closure, place & route, ECO, FIB, driver design, SIMD microcode design, and chip-testing (using ATPG and FPGA).

HDTV Intern - Zoran Corporation, Sunnyvale, CA: Summer 2005 - Fall 2005

Developed and maintained a co-simulation environment that runs the RTL testbench in parallel with a software model

Developed Specmen code to randomly generate SIMD instructions for the co-simulation testbench.

Co-developed a complete set of microcode for H*264 that runs on the SIMD processor, including all modes of intra-/inter- prediction and reconstruction, for Luma and Chroma modes.

HONORS AND AWARDS

Outstand Dissertation Award, UCLA Electrical Engineering,	2013
Broadcom Fellowship (co-recipient),	2012
Jack Raper Award for Outstanding Technology Directions (co-recipient), ISSCC	2010
Department Fellowship, UCLA,	2007-2009
High Honors, UC Berkeley,	2003-2005

PUBLICATIONS:

Journals:

C. C. Wang, C. Shi, R. W. Brodersen, and D. Markovic, "An Automated Fixed-Point Optimization Tool in MATLAB XSG/SynDSP Enviornment," ISRN Signal Processing, Volume 2011

M. Spencer, F. Chen, C. C. Wang, R. Nathanael, H. Fariborzi, A. Gupta, H. Kam, V. Pott, J. Jeon, T-J. K. Liu, D. Markovic, E. Alon, V. Stojanovic, "Demonstration of Integrated Micro-Electro-Mechanical Relay Circuits for VLSI Applications," IEEE Journal of Solid State Circuits, Jan. 2011

C. C. Wang and D. Markovic, "Delay Estimation and Sizing of CMOS Logic Using Logical Effort with Slope Correction," IEEE Trans. of Circuits and Systems-II, vol. 56, issue 8, pp. 634-638, August 2009

Conferences:

C. C. Wang, F.-L. Yuan, H. Chen, D. Marković, "A 1.1 GOPS/mW FPGA Chip with Hierarchical Interconnect Fabric," in Proc. Int. Symposium on VLSI Circuits (VLSI'11), pp. 136-137, June 2011

F. Chen, M. Spencer, R. Nathanael, C. C. Wang, H. Fariborzi, A. Gupta, H. Kam, V. Pott, J. Jeon, T-J. K. Liu, D. Markovic, V. Stojanovic, E. Alon, "Demonstration of Integrated Micro-Electro-Mechanical Switch Circuits for VLSI Applications," in Proc. IEEE Int. Solid-State Conference (ISSCC'10), pp. 26-27, Feb. 2010

Magazine Articles:

D. Markovic, C. C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey, "Ultralow-Power Design in Near-Threshold Region," Proceedings of the IEEE, vol. 98, no. 2, pp. 237-252, Feb. 2010

Book Chapters:

D. Marković and R. W. Brodersen, DSP Architecture Design Essentials (Book Chapter 10 on Word-length Optimization), Springer, July 2012

Patents:

C. C. Wang, D. Markovic, "A Radix-3 Network Architecture For Boundary-Less Hierarchical Interconnects", March 2013, Application No. 61/786,676

C. C. Wang, D. Markovic, "Fine-Grained Power Gating in FPGA Interconnects", March 2013, Application No. 61/791,243

CHAPTER I

Introduction

1.1 The Drive Towards Efficiency

For 50 years, Moore's law has driven the rapid scaling in transistor count and feature size. Transistor performance also increased at this pace, essentially doubling its operation frequency with every generation. Few seemed to care that doubling the performance also doubles the power consumption, and by the early 2000s, consumer CPUs have reached over 3 GHz, consuming around 100 watts of power. It then became clear that frequency scaling is reaching the end of the road: power, thermal, and physical constraints became just as important as circuit performance.

"I don't want a kilowatt in my laptop," said Gordon Moore at the International Solid-Sates Circuits Conference (ISSCC) Keynote in 2003 [Moore03]. The industry was recognizing a turning point towards efficiency: design tradeoffs that balance performance, power, and area requirements. Often times, obtaining efficiency requires fundamental hardware changes. "General-purpose hardware is generally not power-efficient," said Shekhar Borkar of Intel at the same conference. Over the past 10 years, the industry has shifted from high-frequency, single-core CPUS, to a heterogeneous integration of multi-core CPUs and dedicated accelerators.

In 2003, many were concerned to *maintain* the 100W power budget. But in just a few years, the industry has commercialized sub-10W processors that fit in thin ultra-books, and even sub-1W processors for smartphones. Dictated by the changes in the scaling trend, these products are designed with *efficiency* in mind.

1.2 What is Efficiency?

Efficiency, unlike many traditional criteria, requires a combination of metrics. Energy efficiency (or power efficiency) is arguably the most common efficiency metric. It quantifies work per unit energy, and is generally measured in billions of operations per second (GOPS) per milliwatt (GOPS/mW). In VLSI circuits, this translates directly to battery life, thermal limit, and reliability.

One may wonder, for example, how energy efficiency differs from just low power. The difference is in *operations*. In an extreme case, any chip can consume 0 watts if it's off! But that is trivial because it is not performing not performing any operations. A similar analogy applies for performance: many smartphone processors today include 4 or 8 cores, but delivering peak performance in all cores will drain the battery very quickly, and can even exceed the phone's thermal budget. From these examples, it should be clear that low power, or high performance alone are not sufficient quantifications for real-life suitability. Efficiency balances these tradeoffs, to perform the *most* operations performed using the *least* amount of resources.

Area efficiency is also a common criterion, quantifying work per unit of area, generally measured in billions of operations per second per mm^2 ($GOPS/mm^2$). This translates directly to die size, cost, and yield. Naturally, we would like the smallest die size for the same functionality just to save cost. Although Moore's law is providing ever-increasing transistor density, designing a large chip is still expensive, and die yield remains an issue for large designs. Many complex VLSI designs are even divided into multi-chip modules (MCM) to avoid large die sizes, improving yield and easing debugging.

1.3 The Efficiency Tradeoff

Depending on the application, the designer may emphasize one efficiency metric over another (e.g. smartphone processors place huge requirements for energy efficiency, but a low-cost USB microcontroller would need high area efficiency). Energy and area efficiencies also contain a mutual tradeoff: a low-power chip may employ low-frequency cores with very low supply voltage (V_{DD}), but utilize massive parallelism to achieve its required throughput. Such a chip will have high energy efficiency, but lower area efficiency due to parallelism. In contrast, a single high-performance core running at nominal V_{DD} is generally less energy efficient, but achieves high area efficiency.

To have a fair comparison, it is necessary to evaluate a chip based on both criteria. This leads us to an interesting question: how efficient are today's chips? We have gathered VLSI chip data from ISSCC and VLSI Symposia (VLSI) conferences in the past decade, along with published data from field-programmable-gate-arrays (FPGAs). Normalizing and averaging all chips to 65nm technologies, we observe a clear trend in Figure 1.1.

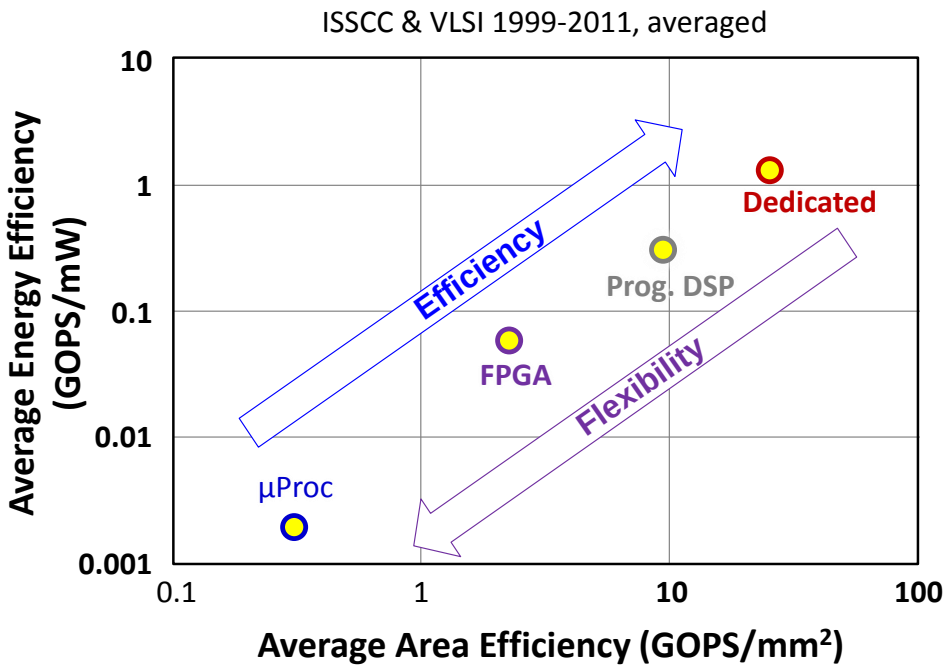


Figure 1.1: Energy and area efficiency of the ISSCC/VLSI chips from the past decade, averaged.

From Figure 1.1, we observe more than 1000× difference in efficiency between microprocessors and dedicated hardware (ASIC), with FPGAs and programmable digital-signal-processors (DSPs) in between. This exposes a key tradeoff between efficiency and flexibility, Intuitively, whenever we take a dedicated portion of a chip and make it programmable, we need to implement additional hardware. In the extreme case of a microprocessor, the actual arithmetic-logic-unit (ALU) becomes only a small portion of the entire chip, leading to its low energy and area efficiencies.

Today's semiconductor industry has a strong push for efficiency, so not surprisingly, more and more designs require specialized hardware. However, what if we need programmability? In the example of a smartphone, a microprocessor must remain, or else it cannot run any software. In addition to running an operating system, the smartphone must perform different multimedia tasks on-demand, while maintaining communications through the

digital-front-end. At the same time, the smartphone must have good battery life and maintain a reasonable cost. These conflicting requirements call for *both* efficiency and programmability, but how can we have both?

1.4 Efficiency and Flexibility – Current Solutions

Since no single hardware today can match the efficiency of dedicated chips, modern VLSI designs have integrated many dedicated blocks onto a single chip, centrally controlled by one or more microprocessors. This creates a system-on-a-chip (SoC) design where the flexible, inefficient microprocessors run the operating system, and then invokes dedicated, highly-efficient accelerators when necessary. Originally, SoCs were designed to integrate on-chip memory, I/O peripheries, and components to assist the central microprocessor. Modern SoCs have outsourced many more tasks to the accelerators, such that the microprocessor has become more like an arbiter.

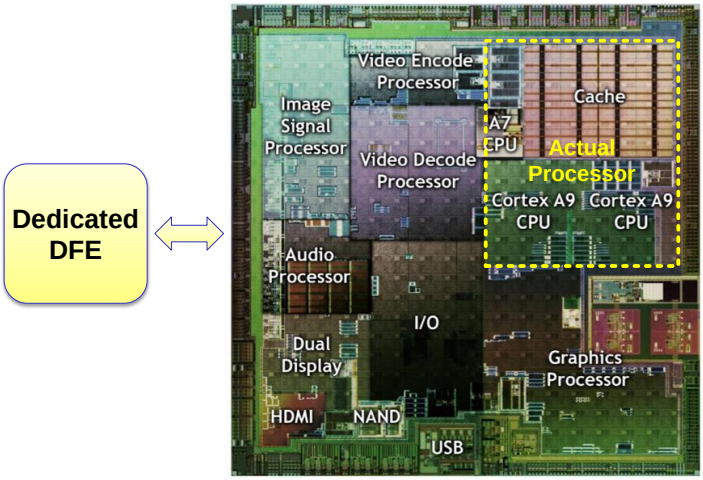


Figure 1.2: Block diagram of an NVIDIA Tegra 2 SoC for smartphones.

As shown in Figure 1.2, the NVIDIA Tegra 2 is marketed as a 1 GHz dual-core processor for Android smartphones. However, we see that the actual processor only occupies a small fraction of the total chip area. The majority of the chip is occupied by dedicated accelerators, and the digital-front-end (DFE) uses another ASIC chip. Recent research [Goulding11] has even proposed to use dedicated accelerator cores to replace portions of the operating system software to further improve chip efficiency. This accelerator-dominated SoC design exists in every smartphone today, and has enabled us to integrate unprecedented functionalities into our phones while still maintaining decent battery life. However, this is not a fully scalable solution.

First, this approach leads to large portions of "dark silicon" [Taylor12]. As smartphones are required to pack more functionality, the number of dedicated blocks will increase. Given the strict power budget in today's SoCs, generally only one or few blocks can be active at a time, while the remaining sections of the chip are idle or power gated. Mmodern SoCs are designed for this scenario, and enabling all the blocks will actually melt the chip. As technology scales, some predicted that only 25% of the chip is exploitable at 22nm, and only 10% is usable at 11nm [Donovan10]. Although energy efficiency of the idling blocks is remedied by power-gating, area efficiency is still reduced.

Second, and more importantly, the accelerators are *dedicated* hardware. Although the operating system can be updated via software, the accelerators cannot. As a result, a single change in one of the blocks would require a re-design. By integrating more blocks, we are more prone towards having this issue. Even when the hardware changes are not caused by design error, there are many other factors what would require a re-design.

1.5 Keeping Up with the Standards

Modern SoCs, especially for smartphones, integrate many multimedia and communication features, and most of these features are based on common standards. Since the accelerators are designed for these standards, whenever a communication or multimedia standard needs to be changed or updated, a re-design of the chip is required.

Unfortunately, standards for communication and multimedia are constantly evolving, and are being introduced at an accelerated pace (Figure 1.3). This figure is already not showing many additional features such as Bluetooth and near-field communications (NFC), or revisions within a current standard.

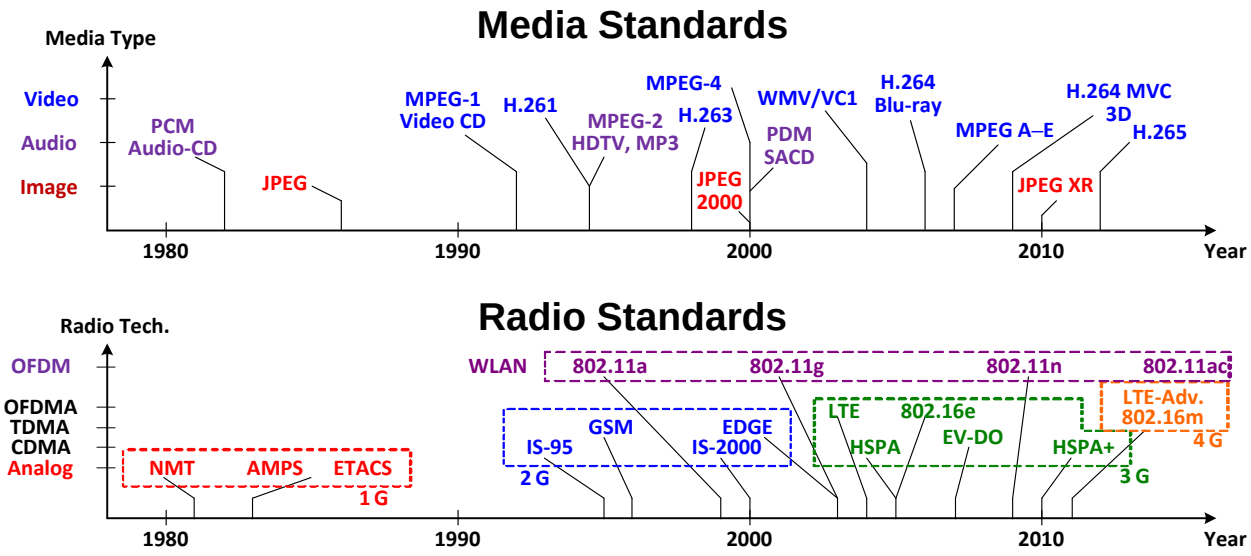


Figure 1.3: Evolution of common multimedia and radio standards.

As a result of frequent updates, the chips often need to "keep up" with the standards. It is common for ASIC chips to be re-designed at least once a year, even though most of the blocks on the chip are legacy designs. One or more additional revisions are often done between new designs to correct for hardware bugs. Therefore, it is common to see design teams taping out around the year. This may not be a big issue if chip designs are cheap. At almost 50 years after

Moore's Law, we would expect chip costs to have dropped down significantly, but what is really the cost of chip design?

1.6 The Cost of Chip Design

Even though Moore's Law provides lower transistor cost per generation, ASICs are still expensive to design, and are becoming increasingly so with every technology generation. The majority of the chip-design cost is not in the transistors. As shown in Figure 1.4, the increasing design complexity, increasing man-hours per design, expensive CAD licenses, and higher fabrication costs are driving up total cost at an accelerated pace. At 28nm, the non-reoccurring engineering (NRE) cost of a custom-made IC is more than $50 million [Sperling12].

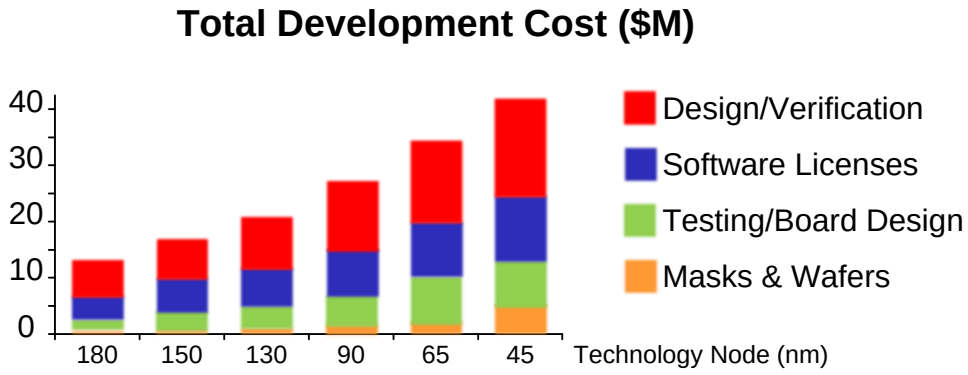


Figure 1.4: Cost of chip design with every technology node [Merritt13].

In the very near future, we will no longer be able to afford re-designing chips on an annual basis. Actually, most ASIC designs are now cost-limited to 65nm and older technologies, about 3 generations behind state-of-the-art processes. Only a few high-volume, high-profit chipmakers are able to maintain cutting-edge processes in their design. The newest processes are also far costlier for foundries to develop. Without many customers able to afford the newest products, it will take foundries far longer to recuperate the costs, thus creating a vicious cycle

that slows down the progress for new technologies.

How can more chip-makers afford the newest technology? They need to amortize the design cost. They need volume. Chips can no longer be designed, used for 1 year, and then thrown away. We need to enable hardware re-use. The designed chips need to remain competitive and up-to-date for more than one product cycle. To achieve such flexibility, we need reconfigurable hardware. But to maintain the efficiencies of current SoCs, we need *efficient*, reconfigurable hardware.

1.7 Candidates for Reconfigurable Hardware

Among the candidates for reconfigurable hardware, microprocessors are least efficient. They will not replace ASICs. In fact, the heterogeneous SoCs today is a result of processor inefficiency. Apart from microprocessors, FPGAs and programmable DSPs are possible candidates for reconfiguration, but they are very different. Programmable DSPs are software controlled, and it is difficult for them to execute the bit-true, cycle-true behaviors of a dedicated hardware. In addition, they generally have lower throughput unless massively-parallel architectures are used, such as single-instruction-multiple-data (SIMD). But in the case of a SIMD, all cores must perform the identical function, which is often not the behavior of a dedicated hardware. SIMD does find its use, for example, in image processors where all cores are required to perform the same operations on a massive array of data [Nakajima06, Noda07, Kurafuji11].

Among these candidates, FPGAs come closest in mimicking ASIC functionalities. They are designed to emulate dedicated hardware in a bit-true, cycle-true manner. They have high throughput due to the implicitly parallel, independent logic blocks, just like ASICs. In terms of

functionalities for the end-user, it is indistinguishable whether the chip is an ASIC or an FPGA. In terms of the design process, the difference is "night-and-day".

Comparing ASIC design versus FPGA design (Table I), FPGA is really appealing from a design perspective, especially for design verifications and prototypes. With supporting software, the user designs can be mapped to run on the FPGA in a matter of hours. Comparing to the expensive process of metallization rework or chip fabrication, the hardware reconfigurability of FPGAs places it in another league. For many VLSI designers, the ability to rapidly verify design changes in silicon without any fabrication rework is an indispensable tool. In many designs that require constant changes, or for small companies that cannot afford to support an entire physical design team (followed by a million-dollar fabrication process), FPGA is becoming used in end-products as well. In recent years, even analog FPGAs are being proposed in research [Scholo12].

Efficiency (ASIC)	**vs.**	**Flexibility (FPGA)**
Logic Design, Physical Design		Logic Design only
Licenses: Synthesis, P&R, etc.		Fewer Licenses
90nm or older		32nm or newer
2 – 4 months fabrication time		None
Expensive to Design		**Inexpensive to Design**
Efficient Operation		**Inefficient Operation**

Table 1-I: ASIC vs. FPGA – efficiency vs. flexibility.

But if FPGAs can behave just like ASICs, and are so much easier to design with, why are they not taking over? FPGA companies often use cost comparisons (as in Figure 1.4) for marketing purposes to steer designers away from building ASICs, but the fact is, modern SoCs rarely employ FPGA hardware. Why are chip designers still opting for the difficult, and expensive, ASIC design process? Because FPGAs pose a huge efficiency gap; it pays significant

penalties in area (17–54x), speed (2.5–6.7x), and power (5.7–62x) compared to ASIC designs [Kuon07, Kuon207, Ahmed10]. Such overheads are still prohibitively high for many ASIC designers to adopt an FPGA design.

For FPGA to be implemented in a mass-consumer market, especially in power-constrained environments such as smartphones, the inefficiency of FPGA operations must be corrected, or else our cellphone battery will last just a few minutes!

1.8 Thesis Outline

The focus of this work is to build efficient, reconfigurable hardware. In Chapter II, we first identify the source of inefficiency from today's FPGAs, and propose to use a hierarchical interconnect architecture to reduce the interconnect area. Chapter III highlights the interconnect architecture design and optimization techniques for hierarchical FPGA realization. Chapter IV illustrates the circuit-level techniques used in designing an energy-efficient large-scale hierarchical network. Chapter V illustrates the design process and hardware features of our FPGAs that achieve 3-4x interconnect area reduction over commercial FPGAs. Chapter VI discusses the algorithms and features of the software tool used for mapping designs onto our FPGA. Chapter VII highlights our testing platform and provides key measurement results. Chapter VIII concludes the thesis and provides an outlook on nano-electro-mechanical (NEM) relays as FPGA interconnects.

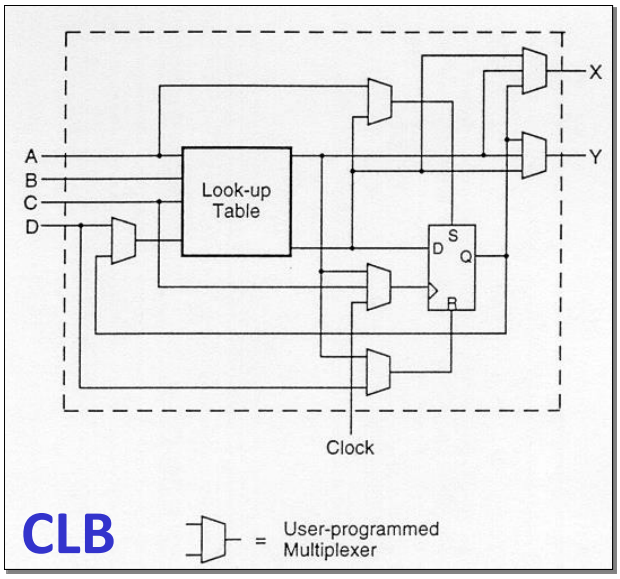
CHAPTER II

FPGA Interconnects: the Source of its Inefficiency

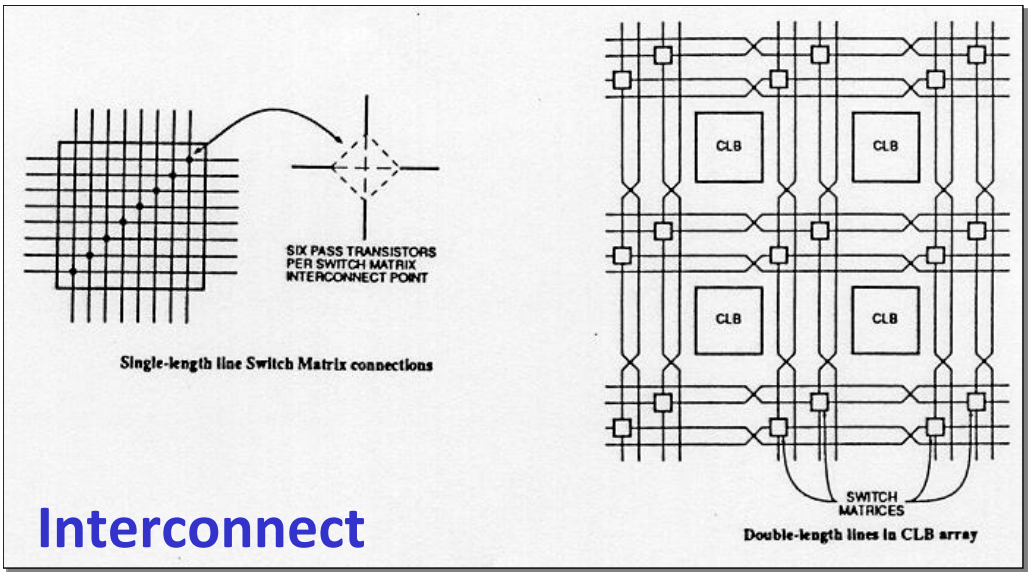
2.1 Brief History of FPGAs

The concept of a reconfigurable hardware started over 30 years ago, but it was regarded as prohibitively expensive because of its large overhead in area over ASICs. Transistors were expensive, and no one wanted to pay the huge area penalty for reconfigurability. Fortunately, the semiconductor industry rapidly expanded at the pace of Moore's law, and such large area overhead became more tolerable, finally leading to a first FPGA by Xilinx Corporation in 1985. The original FPGA, XC2000 series, had 64 or 128 look-up-tables (LUTs). As shown in Figure 2.1 a), each configurable-logic block (CLB) contains just one LUT and one selectable flip-flop. With so few CLBs, the interconnect network is also simple. The interconnects run in x- and y-direction around the CLBs, twisting with every segment, and some of the intersections have switch matrices placed diagonally, consisting of 6 pass-transistors per switch (Figure 2.1 b) [Brown92].

The initial perceptions of the XC2000 were "small, slow, expensive, and 'different'" [Alfke07], but the XC3000 introduced in 1987 became very successful even with very rudimentary software support. Fast-forward to today, FPGAs can support up to 500,000 LUTs per die, and the largest Xilinx Virtex-7 even supports 2 million LUTs using Stacked-Silicon Technology (Figure 2.2) [Saban12].



a)



b)

Figure 2.1: Schematic diagram from a Xilinx XC2000 of a) CLB and b) interconnects.

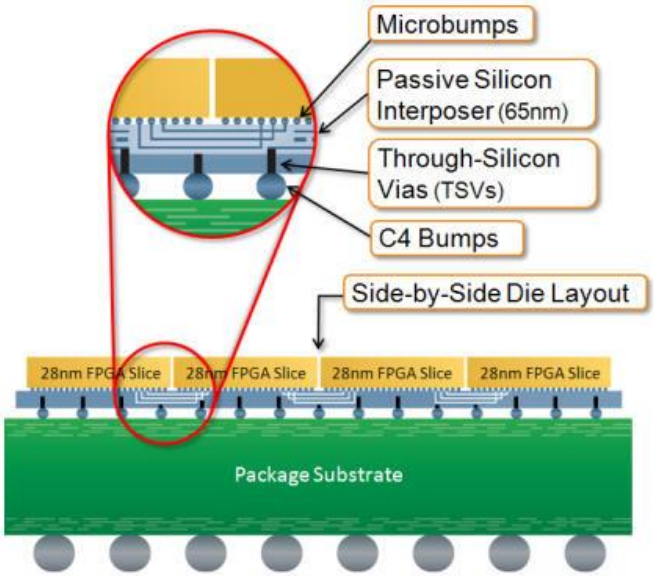
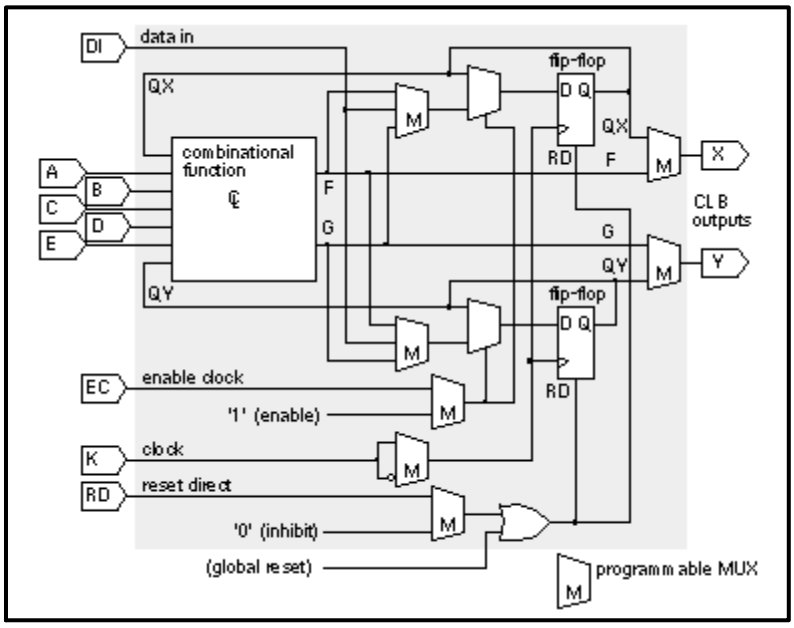


Figure 2.2: Illustration of Stacked-Silicon Technology in Xilinx Virtex-7.

Due to yield and fabrication constraints, each die is limited to around 500,000 LUTs, occupying 529 mm^2 in 28nm. "Stitching" the 4 chips together requires a very large interconnect bandwidth, far greater than that offered by standard packaging solutions. Therefore, a 65-nm passive silicon interposer is mounted onto the 4 FPGA dies to create a high-bandwidth interconnect, providing more than 10,000 connections between each adjacent die. For communication with external I/Os, the interposer uses through-silicon vias (TSVs) to connect the FPGA die to the C4 bumps on the package. Although the stacked silicon technology is not monolithic, many of the performance and cost benefits of a 3-D monolithic FPGA from [Lin07] still apply.

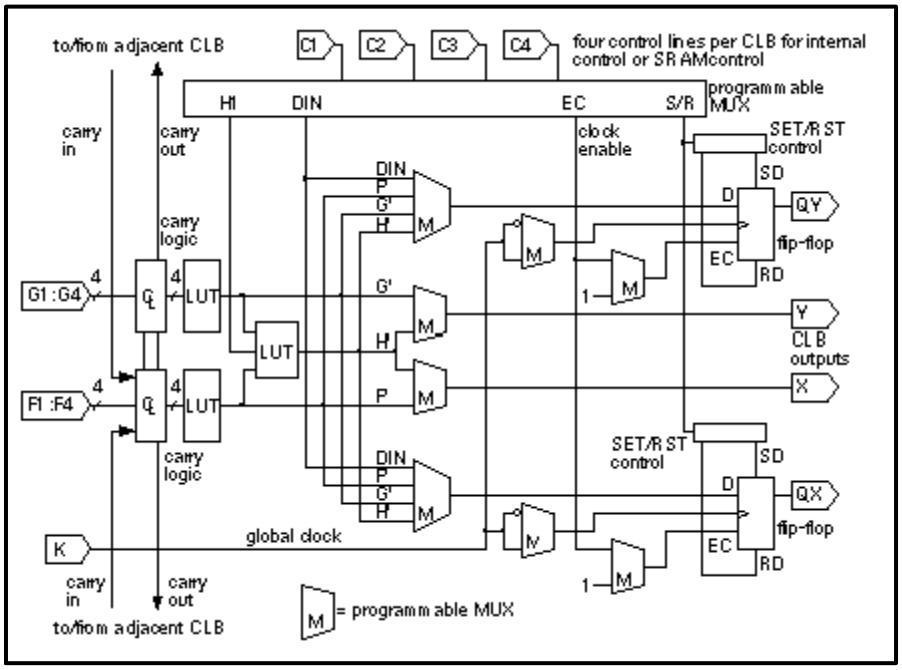
Of course, FPGA progressions are more than just area expansion, the CLB core of the FPGAs has also evolved over the years (Figure 2.3) [Rose93]. Many features are added to implement commonly-used ASIC features very effectively, such as multiple flip-flops with clock-enables (XC3000), a dedicated ripple-carry chain (XC4000), and LUT-combining multiplexers (XC5200).

XC3000

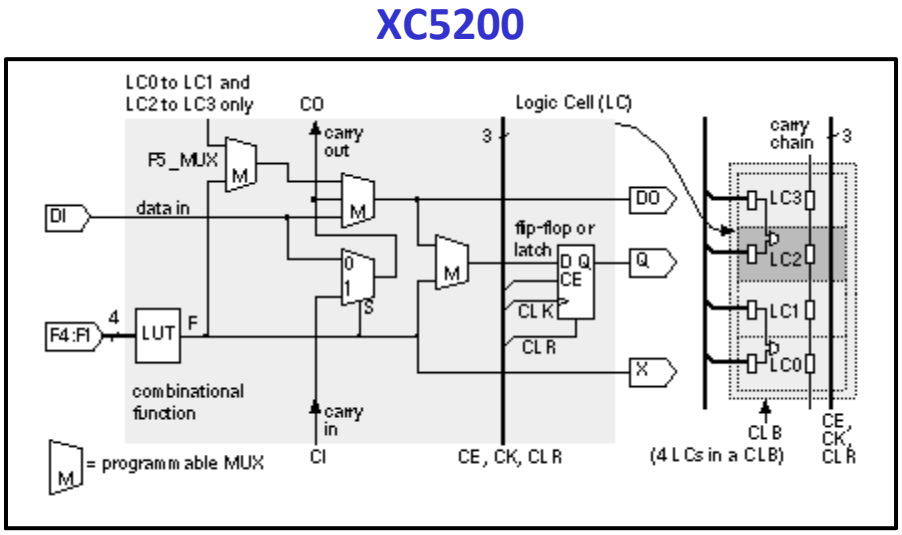


a)

XC4000



b)



c)

Figure 2.3: CLB diagram of Xilinx a) XC3000, b) XC4000, and c) XC5200.

Over the past ten years, CLB sizes grew even more. Xilinx has transitioned to four 4-input LUTs per CLB in its Virtex-4 [XlinxV408], then to four 6-input LUTs per CLB in Virtex-5 [XlinxV512]. The newer Virtex-6 and 7 even have dual flip-flops mated to each of the 6-input LUTs (Figure 2.4) [XilinxV6CLB12].

The newer CLBs place an even greater emphasis on software design. The performance of the FPGA depends heavily on the mapping algorithm – packing critical-path gates within a CLB would provide much faster performance than spreading the critical path across multiple CLBs. Since the interconnect network cannot provide full connectivity across all CLBs (Chapter III), packing LUTs locally into CLBs can reduce the number of I/Os required by the CLB [Betz98], and the software tool also needs to provide quality place-and-route results to ensure feasible design mapping.

Virtex 6/7

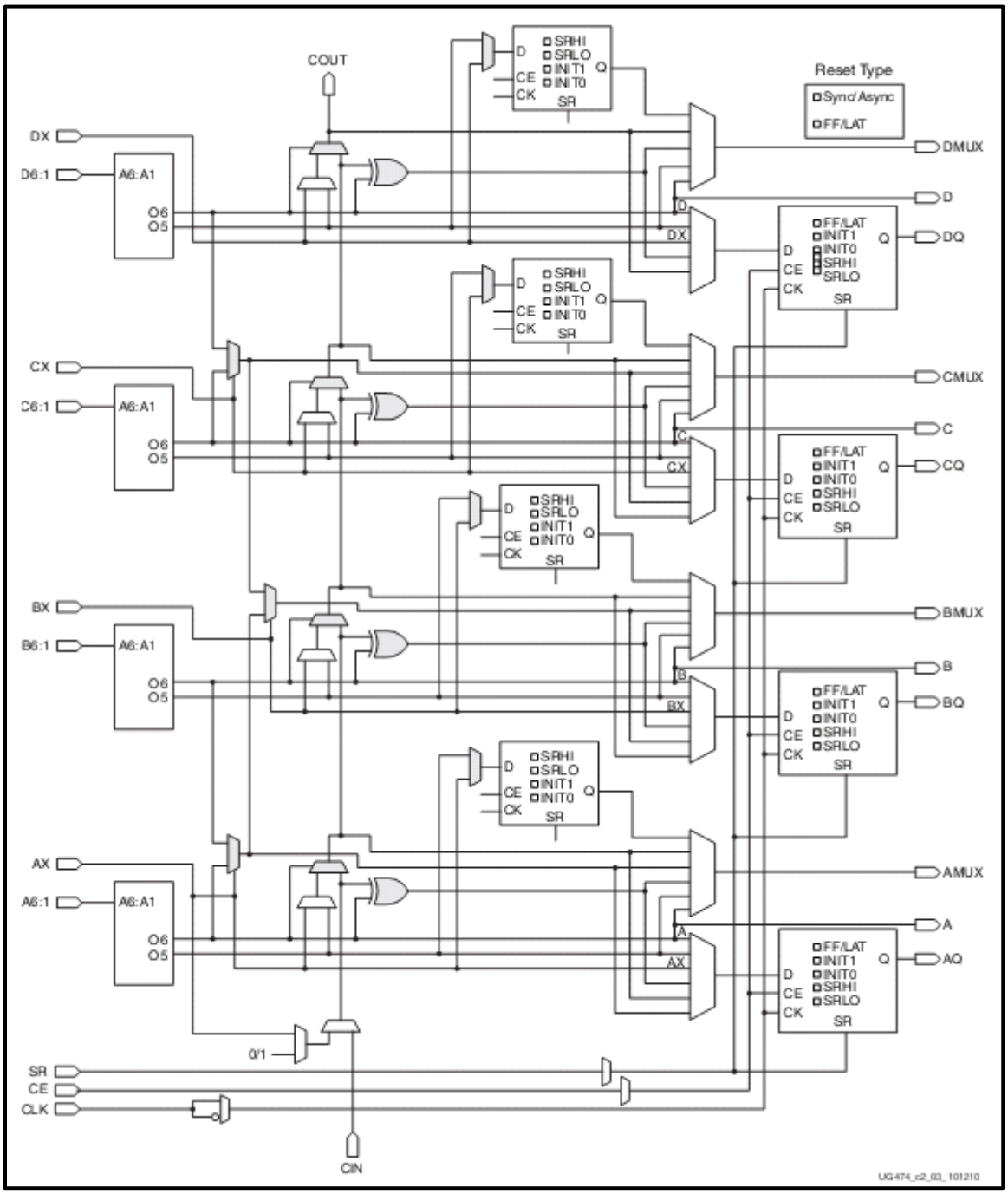


Figure 2.4: CLB diagram of Xilinx a Virtex-6 and 7 series FPGA.

Over the years, the FPGA software support has developed into a complete design suite. With extensive support for automated design mapping from HDL into bitstream, very little effort is required by the end-user. High-level synthesis tools even support mapping software programs (such as C or Matlab models) directly onto the FPGA. This many layers of abstraction provide a simple user experience, but it also shields us from seeing the intricate details of a FPGA design, especially interconnect routing.

2.2 Interconnects: the Backbone of an FPGA

In FPGA design, great emphasis is placed on the CLBs and other programmable blocks, and documentations are widely available. On the other hand, interconnects have mostly remained in the dark. Although FPGAs have grown enormously in size since the XC2000, the fundamental interconnect architecture still remains (Figure 2.5). In 2D-mesh interconnects, LUTs are placed in configurable logic blocks (CLBs), and interconnects run in the x- and y- direction surrounding the CLBs. I/O connection switches tie the CLB I/O to the interconnect network. Arrays of switch boxes are placed at interconnect crossings to select and buffer the programmed path. Each switch-box contains pass-transistors programmable by the configuration memory. Since a full switch-box array at every interconnect crossing requires too much area, various heuristics are used to simplify the arrays at the cost of interconnect connectivity [DeHon99, Tessier00, Lin09]. In Figure 2.5, the example network only implements switch boxes along one main diagonal and two sub-diagonals of the switch-box array. In this simplistic case, each interconnect trace enters a switch-box at every interconnect crossing, the selected path is then buffered to drive the next trace.

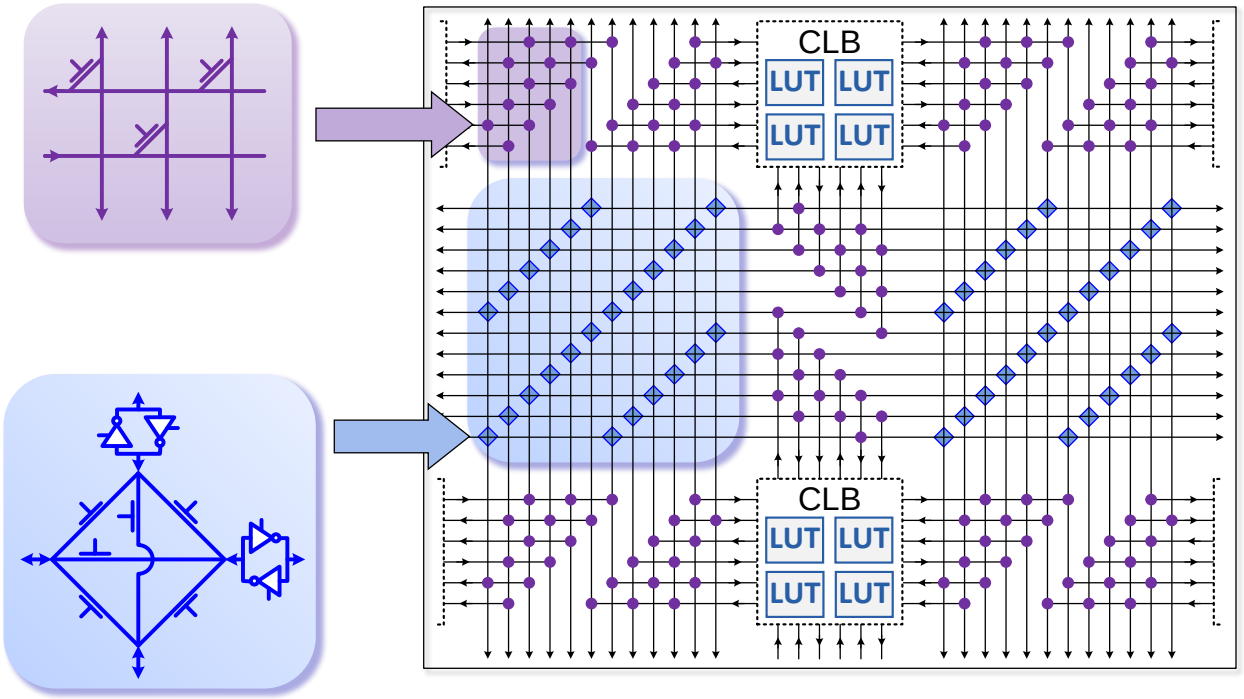


Figure 2.5: A sample 2D-mesh architecture with I/O connections and switch boxes.

To improve routing performance and add path diversity, each interconnect trace can be heuristically designed to travel for 1, 2, 4, 6, or even more CLBs before reaching the next switch. A path from one switch to the next is called a “hop”. From an illustration in Xilinx XC4000 interconnects (Figure 2.6) [XilinxXC99], we see different interconnects labeled as “single”, “double”, “quad”, “long”, or even “global” based on the distance of each hop. Coming out of a CLB, a signal can be connected to a selection of hop lengths, giving the router freedom to choose a longer or shorter hop based on its routing requirements. Modern FPGAs have also migrated towards uni-directional routing, thus removing bi-directional buffers and significantly reducing interconnect loading [Lemieux04, Lee06].

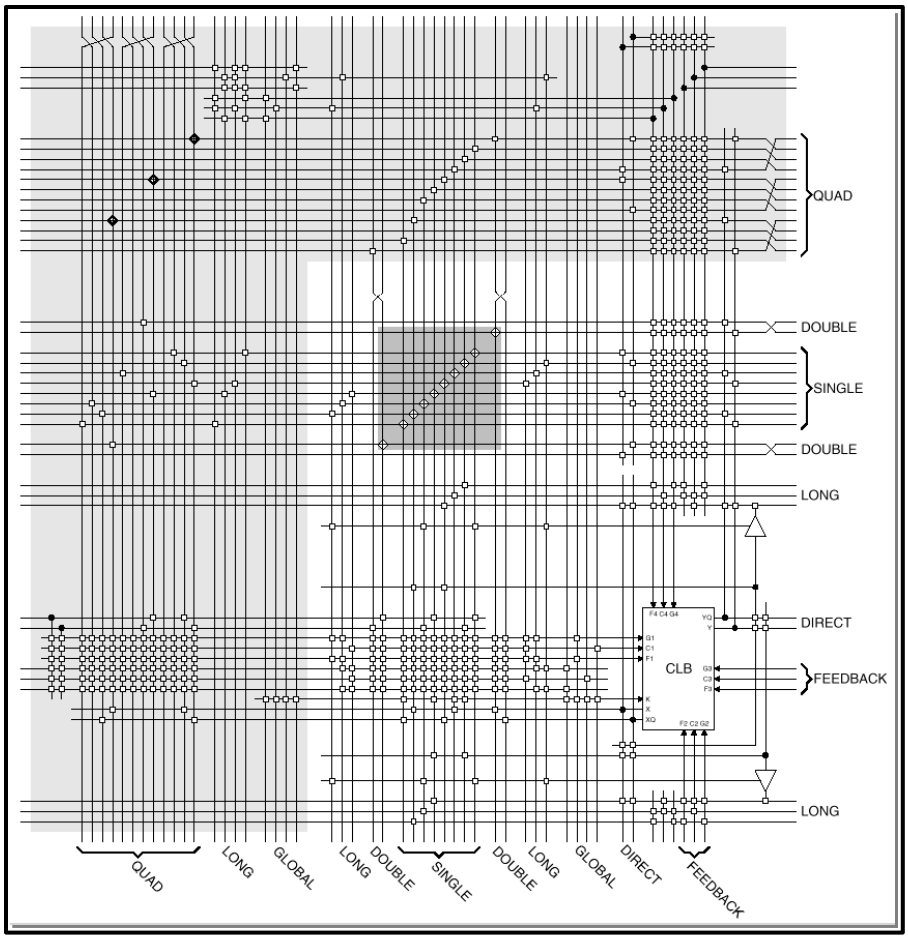


Figure 2.6: Interconnect architecture of a Xilinx XC4000 FPGA [XilinxXC99].

With extensive techniques in interconnect pruning, along with ever more complex CLBs, one may expect the FPGA area to be dominated by CLBs. It is called a "gate-array" after all. Surprisingly, even with such heuristics, 80% or more of the area on modern FPGAs are occupied by interconnects [Bolsens06]. The interconnect area is actually 4 times the logic area! In addition, interconnect also accounts for the majority of the delay and power in today's FPGAs (Figure 2.7). The reality could be even worse: if we were to remove the larger IP blocks and accelerators from the FPGA, and compare the area of interconnect versus the area of CLBs, the ratio could be closer to 10:1.

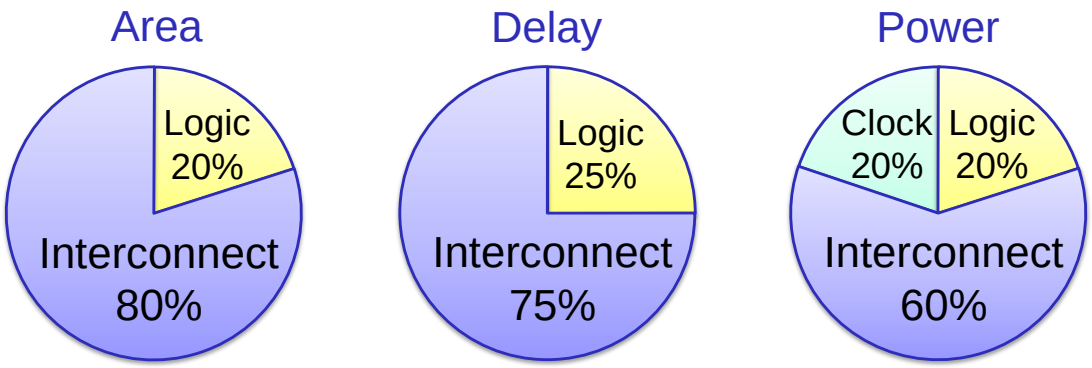


Figure 2.7: Area, delay and power breakdown of a modern 2D-mesh FPGA.

2.3 Scaling a 2D-mesh Network

The key cause for interconnect overhead is the scalability of 2D-mesh interconnects. In the worst case, the number of switch boxes grows as $O(N^2)$ with the number of LUTs. Although heuristics are able to reduce the number of switches, there is a limit. Rent's rule ($T = t \cdot g^p$) can be used to model interconnects, where g is the number of gates, exponent p is the Rent's coefficient for modeling the number of I/Os, and t is a constant of proportionality. In typical cases, the interconnect complexity per logic block is $O(N^{0.75})$ for random logic, which is still $O(N^{1.75})$ for a chip of N logic blocks [Landman71].

For very regular designs, such as a memory banks, the complexity per logic is $O(N^{0.5})$. Since FPGA mapping software employs intelligent gate placements, the logic is not completely random, but it is certainly not as regular as memory banks. We therefore expect the actual Rent's exponent p to be between 0.5 and 0.75 [Tessier00]. But for very large designs (large N), $O(N^{0.5})$ to $O(N^{0.75})$ provides too large of a range for this model to be useful. Nevertheless, it provides us theoretical lower and upper bounds on interconnect complexity.

Even using an optimistic exponent of $p = 0.5$, the total complexity of $O(N^{1.5})$ still

requires FPGA sizes to scale much faster than Moore's Law. Figure 2.8 shows the interconnect expansion from Xilinx Virtex-4 to Virtex-5 [XlinxV506, Minev09]. Adding 50% of interconnect logic per CLB poses a significant area increase even for just 1 product generation. Scaling N from 64 in XC2000 to 500,000 in modern FPGAs, it becomes clear why interconnect area is a key concern today.

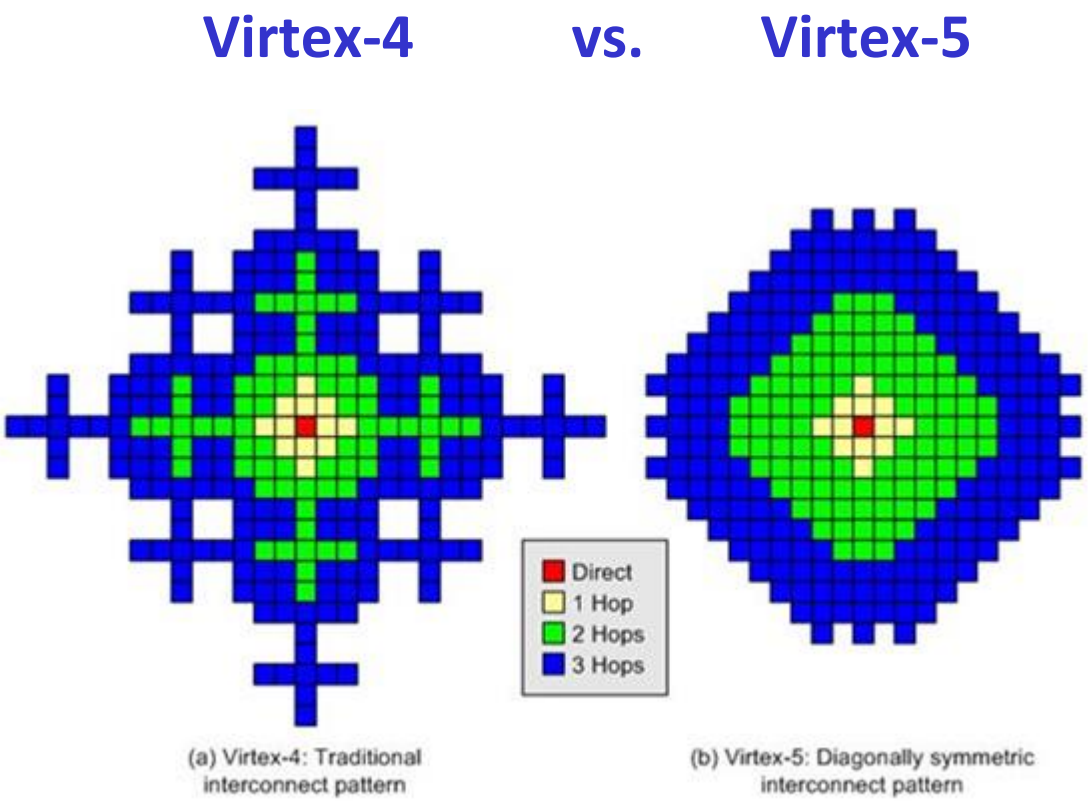


Figure 2.8: Interconnect resources per CLB for Xilinx Virtex-4 vs. Virtex-5 [XlinxV506].

In more recent years, many have proposed asynchronous architecture for FPGAs, aiming to improve its performance [Teifel04, Teifel204, Manohar06]. Such techniques have claimed to achieve > 1 GHz performance from FPGAs by using asynchronous hand-shake and token-based heavy pipelining. However, such technique failed to recognize the root cause of FPGA overhead, which is the scalability of the interconnect area. In contrast, asynchronous FPGAs require a 3x

overhead in interconnect area: replacing 1 signal with 3 asynchronous hand-shake signals, further exacerbating the effect of interconnect overhead. Whenever signal fan-outs are required, complex acknowledgement circuitry is required to wait for the slowest path to return the token before passing it on. More recent work by [LaFrieda10] acknowledged the large area and power overhead required by asynchronous FPGAs, and proposed a two-phase logic and voltage-scaling in the acknowledge signals to reduce the power consumption, but the large overhead in area remains. Although asynchronous FPGAs claims to run up to 3x faster than their synchronous counterparts, the 3x penalty in interconnect area will quickly nullify any performance advantages on large designs. Recent work in [Devlin11] uses dual pipeline (separate pipelines for precharge and evaluation phases) to further improve asynchronous performance, but requires 5 physical wires for 1 interconnect signal. Clearly, these approaches are not scalable to larger designs. For efficient, high-performance FPGAs, what we need is an interconnect architecture that is scalable in area and performance, and not brute-force circuit implementations.

2.4 Hierarchical Network – A Scalable Solution

To address the non-scalability of 2D-mesh, we adopted a hierarchical interconnect architecture based on a Beneš network. In telecommunication, Clos, Beneš, and similar hierarchical networks are well-known to be rearrangeably non-blocking network for point-to-point connections, and are commonly used in communications [Clos53, Benes62, Kleinrock77, Yang99, Dally04]. There has not been a silicon realization of a Beneš network for FPGAs until this work. To demonstrate its feasibility, the original Beneš network is first modified into a realizable FPGA architecture.

As a demonstration, we start with 2 LUTs, each with just 2 inputs and 2 outputs (Figure

2.9). This network requires 3 stages, and each stage uses 2x2 switch matrices (SMs) for signal routing. Each SM can support both uni-cast and multi-cast of incoming signals, as shown. This network is rearrangeably non-blocking for uni-cast, meaning the signal routing can be rearranged to support arbitrary LUT-to-LUT connections.

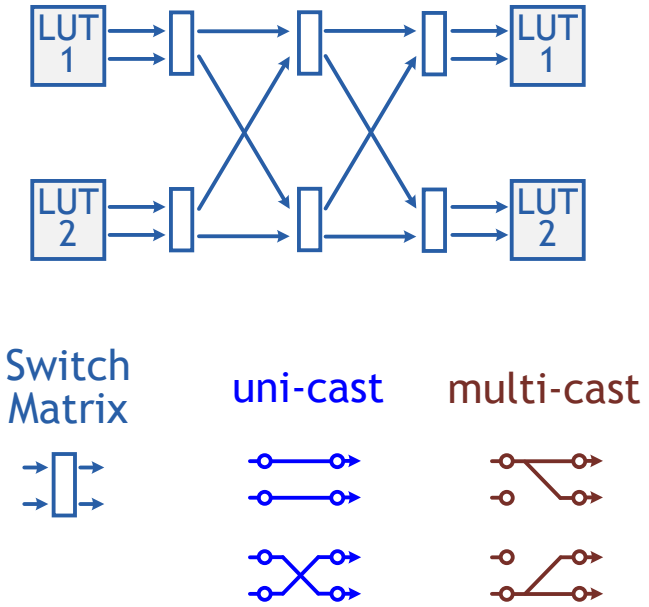


Figure 2.9: A simple 3-stage Beneš network connecting 2 LUTs.

In FPGA applications, it is common to use 4 to 6 input LUTs with 2 outputs. To illustrate a 4-input, 2-output LUT network, the 3 stage network is recursively extended to a 5-stage network (Figure 2.10), and can be further extended to larger networks. This network remains non-blocking for uni-cast, and because there are only half as many LUT outputs as inputs, it is virtually non-blocking even for multi-cast based on our simulations. Since each LUT only has 2 outputs, the red SMs can always multi-cast the signals, and can be removed. In addition, the 4 inputs to a LUT may arrive in any order, therefore the gray SMs can be removed as well. Note that for some CLBs, such as DSP accelerators or control signals, the inputs may not arrive in any order, and in those cases the grey SMs must remain. For simplicity, the center 3 stages are

abstracted as a single 4-input, 4-output SM, which is essentially a 2-bit 2x2 switch because it propagates two paths in each direction. The simplified diagram is shown on the bottom of Figure 2.10.

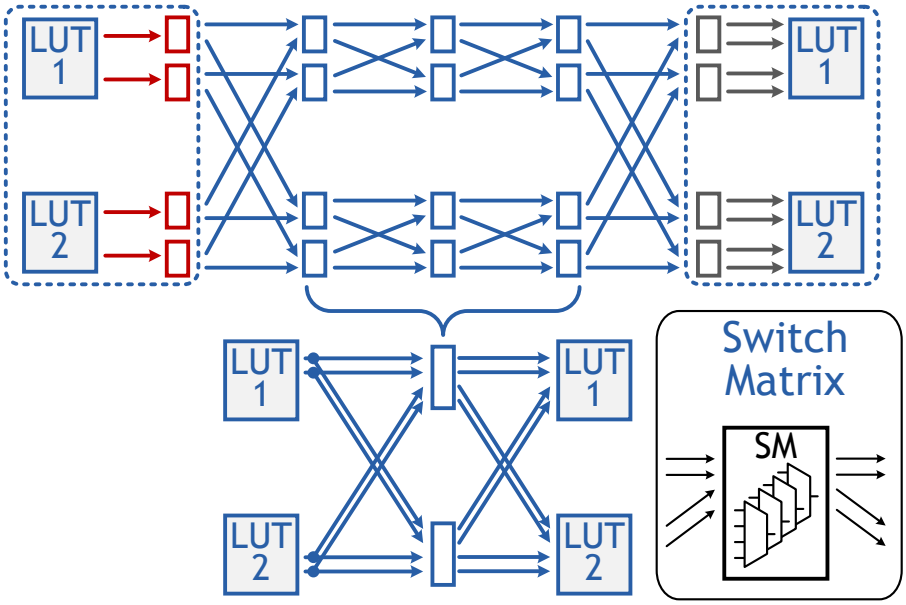


Figure 2.10: A 5-stage Beneš network merged into a 3-stage using 2-bit 2x2 switches.

Scaling to a larger network, we observe one key problem with the original Beneš network. Figure 2.11 shows an 8-LUT network using 5 SM stages. The downside is that *all* paths are required to traverse on all 5 stages regardless of the physical distance between the source and destination. As shown in Figure 2.11, LUT 7 and 8 are physically adjacent to each other, but the network requires the signal to traverse through all hierarchies while a simple switch in the first stage would suffice. Another issue with this network is input/output locality. In an FPGA, the input and output of a LUT is coming from one hardware block, but in this network, the inputs and outputs are split across two sides of the network. Since this diagram is not representing physical implementation, it can be misleading to the FPGA designer.

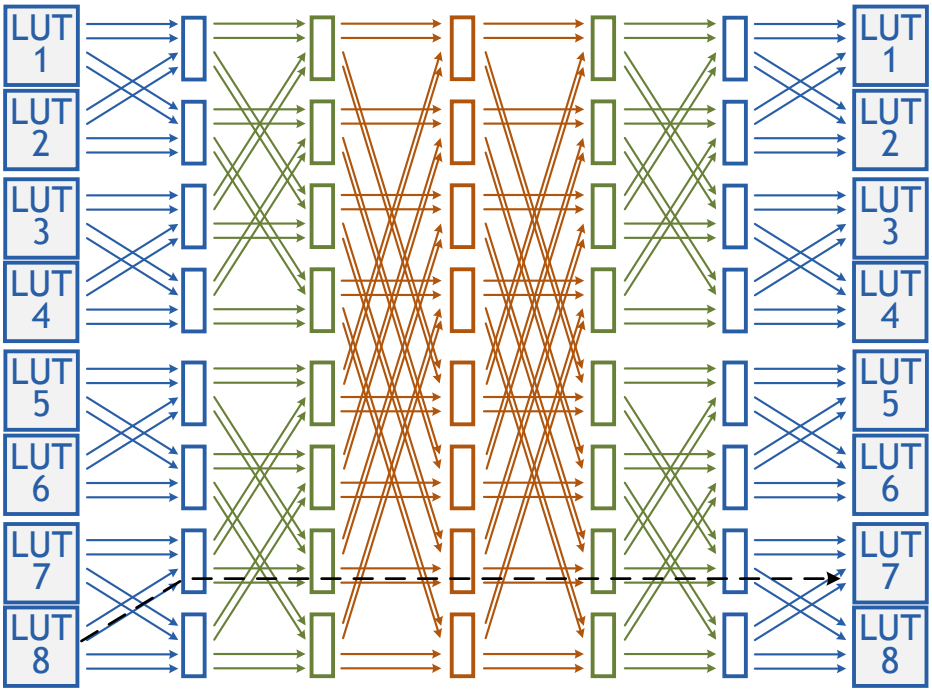


Figure 2.11: A 5-stage Beneš network connecting 8 LUTs.

To avoid traversing unnecessary hierarchies to speed up interconnect routing, and to provide an interconnect that closely resembles the physical implementation, we employ a folded Beneš network (Figure 2.12), also called a fat-tree network by [Leiserson85]. This similar architecture has been employed in supercomputing machines, such as the Connection Machine CM-5 with 256, 544, and even over 1000 processing nodes [Leiserson96].

As shown, 4 LUTs are connected via 2 stages of SM, and another 4 LUTs are to be connected with a 3rd SM stage. This effectively leads to an interconnect complexity of $O(N \cdot \log N)$, which scales much better than $O(N^2)$ in 2D-mesh interconnects.

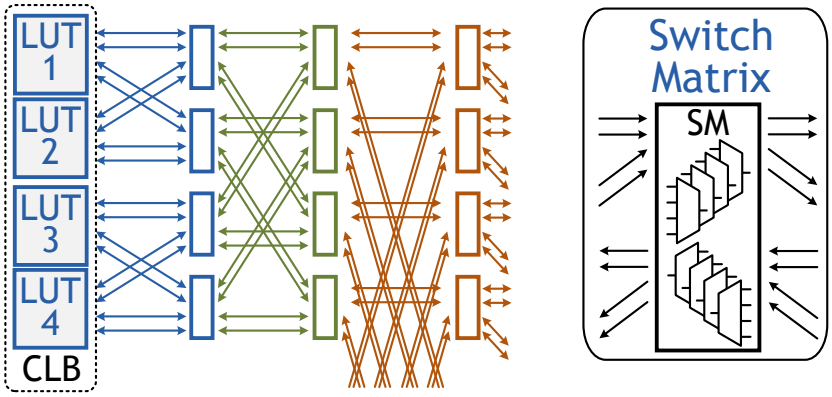


Figure 2.12: A 3-stage folded Beneš network connecting 8 LUTs (4 LUTs shown).

Although drawn with 2 arrows, each trace is actually 2 uni-directional signals. Each switch matrix then performs 4 uni-directional connections both upwards and downwards. Signals will come from the LUT output, traverse up to the required hierarchy, and traverse back down to the LUT input. Because the network is still rearrangeably non-blocking, full connectivity can be obtained.

Although this architecture reduces interconnect complexity by reducing the number of switches, routing congestion remains an issue. In Figure 2.12, the first SM stage has 2x2 wires crossing each other, but the second stage has 4x4 wires crossing, and the 3rd stages has 8x8. Each additional SM stage doubles the routing congestion. This O(N) congestion requires much larger area for higher level SMs, making physical design more difficult and less area-efficient.

Fortunately, implementing a Beneš network on silicon gives us freedom in both x- and y-directions. Although [Manuel 07] illustrated a manual layout method for a Beneš layout on a 1-dimensional array, most silicon implementations allow for a 2-dimensional layout. To alleviate congestion, routing is alternated between the x-y directions, doubling the routing congestion for every 2 stages. The routing congestion is reduced from O(N) to O($N^{0.5}$) (Figure 2.13), and the fully symmetrical implementation also eases physical design.

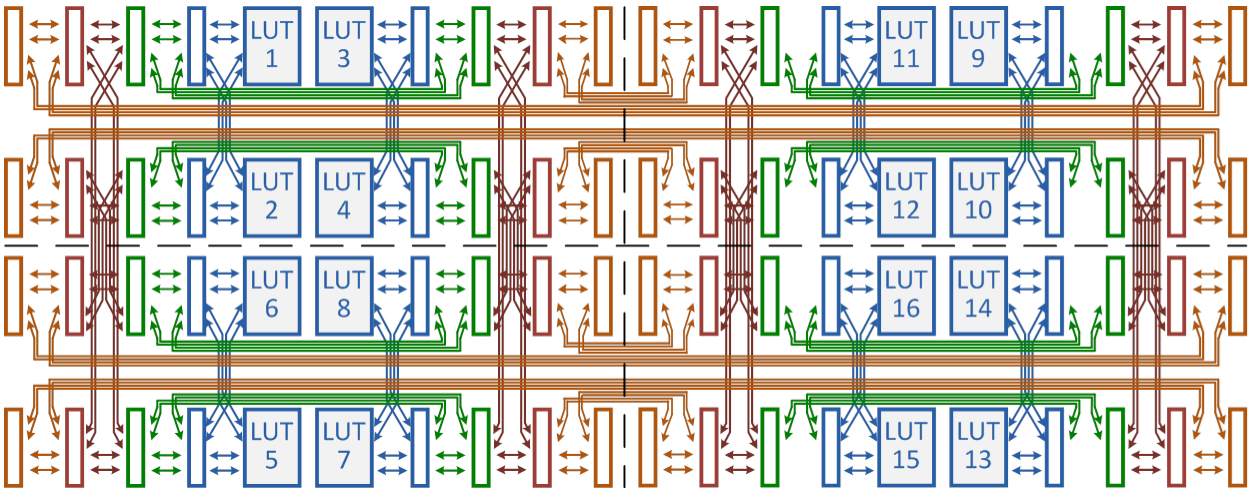


Figure 2.13: A hierarchical Beneš interconnect architecture using alternated x-y routing.

Another change from the original Beneš network is unequal wire lengths. At every hierarchy, the LUTs near the center are connected to create shorter routes, and the LUTs near the edges have longer routes. In terms of logic connectivity, this wiring difference is an isomorphic transformation from the original network, thus the interconnect connectivity remains unchanged [Wu80, Duato02, Konda08]. Yet this difference in wire lengths gives routing tools options for faster paths on timing-critical routes. In physical design, this also allows the center routes to remain at the lower metal layers without crossing over the longer routes on the upper metal layers, further avoiding congestion.

2.5 Prior Attempts at Hierarchical FPGAs

Numerous publications have discussed hierarchical FPGA implemented as tree-of-meshes (Figure 2.14) [Greenberg88, Lai97, Tsu99, Wong04, DeHon04]. It is a limited bisection network, where the mesh connectivity decreases for upper hierarchies. In some implementations [Tsu 99], even connectivity at local levels is limited. Additionally, a centralized routing network is required at every hierarchy, which increases routing congestion, and central switches are still

based on 2D-mesh. The layout in [Greenberg88] intelligently distributes the meshes across the layout into "cubies", but the complexity of every hierarchy remains that of a mesh-based switch.

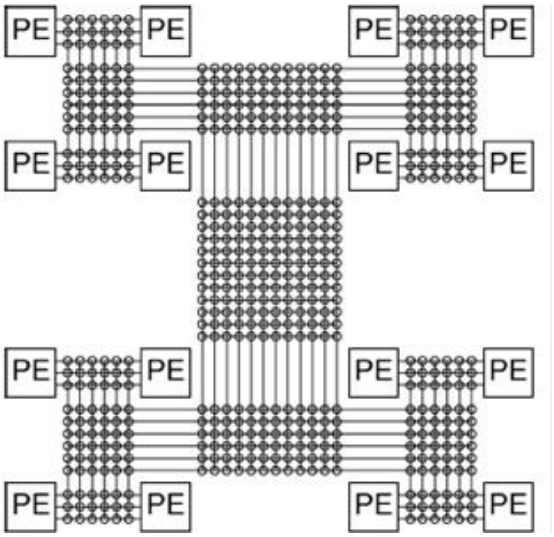


Figure 2.14: A hierarchical interconnect architecture using alternated x-y routing [DeHon04].

Unlike tree-of-mesh interconnects, our Beneš interconnect architecture evenly distributes routing across all LUTs instead of crowding them into centralized "hubs," easing routing congestion and shortening the wire length significantly. This is different from the butterfly layout in [DeHon00, Wong04] where centralized hubs are used, but hubs are distributed across different "cubies," thus requiring each signal to traverse across different hubs in different cubies just to switch hierarchy, significantly increasing interconnect delay.

There is one known silicon implementation of a tree-of-mesh FPGA, the hierarchical, synchronous reconfigurable array (HRSA) [Tsu99]. The architecture uses a Radix-4 topology with centralized switches and bi-directional routing. Rent's exponent of 0.5 is used, so every hierarchy prunes the interconnect connectivity by 50%. Due to the centralized hubs used in this architecture, processing elements (PEs, equivalent to LUTs) that are physically close to each other may be required to use a detour routing. A heuristic is then employed to add "shortcuts" to

connect these PEs using additional wiring (Figure 2.15).

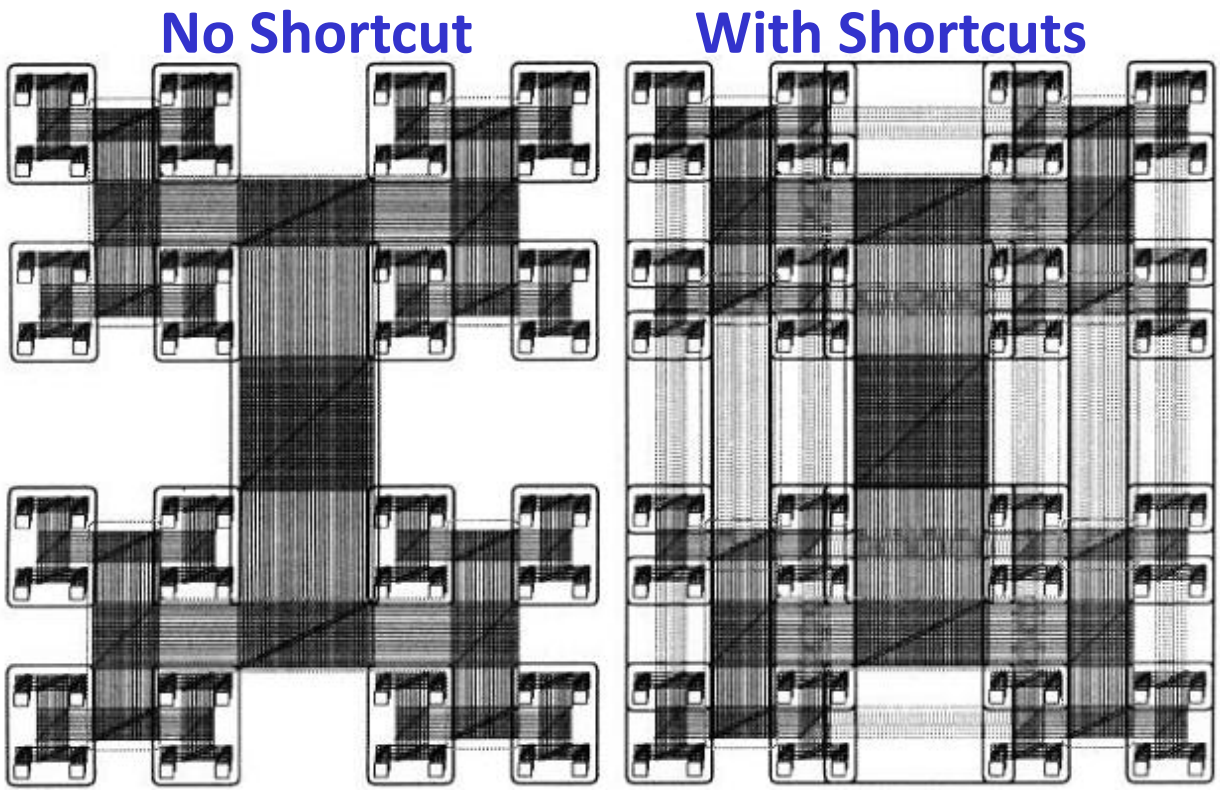


Figure 2.15: The HSRA architecture without (left) and with (right) wiring shortcuts.

The HSRA architecture was able to maintain good operation frequency due to its heavy pipelining, but the interconnect network with a Rent's exponent of 0.5 offered "very limited" connectivity. There has not been a follow-up chip after the original HSRA in 1999.

A multilevel hierarchical FPGA was published by [Mrabet06], although no silicon realization is attempted. The architectures use a Radix-4 topology with a Rent's exponent of 1, but only on the downward paths. The upward path, on the other hand, provides no path diversity (Figure 2.16). Therefore, the overall path diversity of this architecture is very limited, and the interconnect connectivity when mapping real-world designs is about 30-50%, often requiring a 2K-LUT FPGA to map 1K-LUT designs.

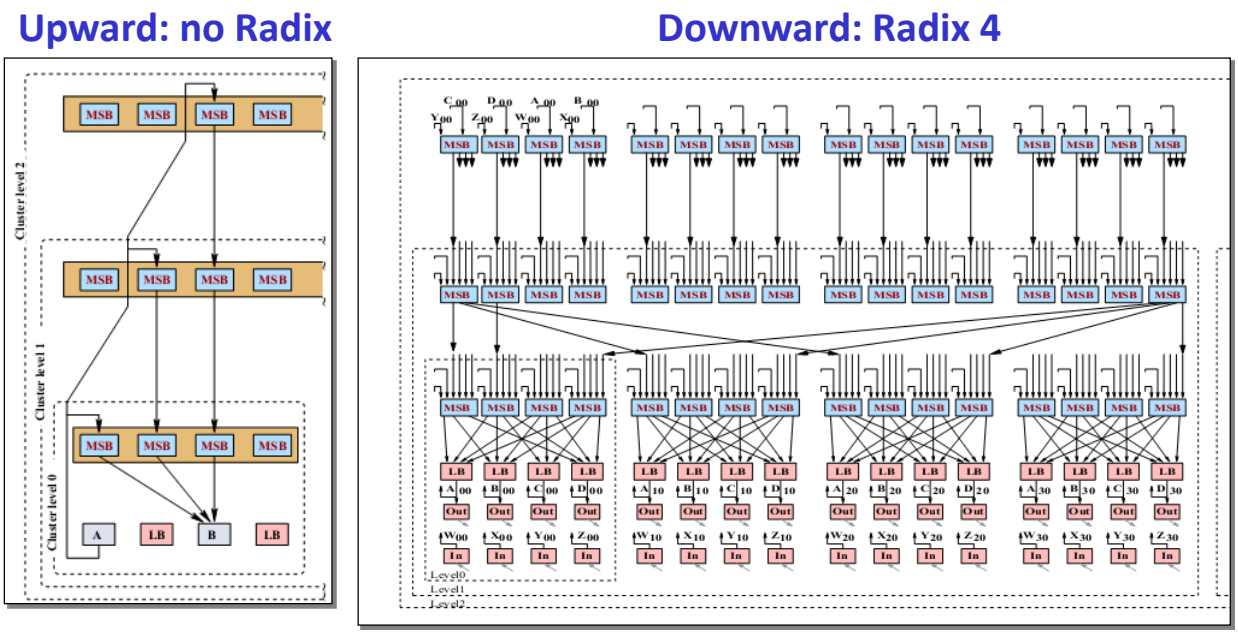


Figure 2.16: The multilevel hierarchical FPGA architecture.

2.6 Our Challenges

Although hierarchical FPGA has great appeal on paper, it has not received much attention in practice. The main reason is that it has yet to demonstrate *any* advantage over 2D-mesh: its 30-50% logic utilization is significantly lower than the 85% utilization achievable by commercial FPGAs, and it has yet to demonstrate any notable performance, power, or area advantage. The speed improvement in HRSA is due to heavy pipelining, not interconnect improvements.

On the other hand, commercial FPGAs today are already very mature products, often made as full-custom designs with state-of-the-art processes (and needing more than 10 layers of metal). The CAD tools are also capable of delivering very high quality-of-results (QoR) within a easy-to-use framework.

For our work to be considered worthwhile, we need to demonstrate and realize a

hierarchical FPGA with significant benefits in performance and efficiency. To demonstrate its practical values, software development is also needed to allow users to map their own designs. Overall, this project requires innovation and extensive work in creating an interconnect architecture, realizing it in silicon, and developing software tools to demonstrate its advantages. These details are covered in the following chapters of this thesis.

CHAPTER III

Architecture Design of Hierarchical FPGAs

3.1 Realizing Large-Scale Beneš Networks

To illustrate the silicon realization of the Beneš network, we start with the architecture design applied to our two FPGAs. The two chips shown in this dissertation have approximately 10× difference in logic capacity, and have different interconnect architecture as well. The first chip is a more straight-forward implementation, while the second chip utilized extensive architectural optimization techniques illustrated in Section 3.3 through 3.7.

The first test chip we published in [Wang11] contained 2048 look-up-tables (LUTs), each with 4 inputs and 2 outputs. Built on a Radix-2 architecture, it requires 11 levels of interconnects. Since every level translates to one SM stage, 11 levels of SMs are required. To ensure 100% connectivity in all cases, every LUT would need to have 11 levels of SM to preserve the full Beneš network. Using the 2D-layout method illustrated in Figure 2.13, expanding from 4 stages for 16 LUTs to 11 stages for 2048 LUTs would still be feasible to route, but it would occupy a significant amount of area. According Rent's rule, this brute-force implementation represents a Rent's exponent of $p = 1$. Realistically, there is no need to implement an interconnect network with more than $p = 0.75$ connectivity, as the area penalty associated with building larger interconnects far outweighs the benefits from chip utilization [Tessier00].

Mathematically speaking, implementing a network with $p < 1$ requires interconnect pruning at every stage. For example, when $p = 0.75$, every additional stage should implement 25% fewer wires than the previous stage. For FPGA realizations, there are three key reasons that

make this exact implementation impractical.

First, mapping FPGA design is a very non-deterministic process that depends heavily on the design to be mapped and the algorithms used by the place-and-route (P&R) software. The design to be mapped can have a Rent's exponent p anywhere between 0.5 and 0.75, which is a very wide range for interconnect routing. A very regular design, such as a feed-forward finite-impulse-response (FIR) filter, combined with a high-quality P&R tool, could be easily mapped onto an architecture with $p = 0.5$. On the other hand, a more complex design such as fast Fourier transform (FFT) will consume significantly more interconnect resources. There is no single exponent that can accurately represent all design complexities.

Second, the interconnect utilization is uneven across the SM stages. An effective P&R software would attempt to keep most of the signals local, thus shortening the critical path and reducing the active wire lengths. As a result, it is important to have sufficient routing resources for the lower levels to provide sufficient path diversity for the P&R tool. It can be worthwhile to use a Rent's exponent of $p = 1$ for the lower hierarchies, and use a more aggressive pruning (e.g. $p = 0.5$) for the upper hierarchies. From our architecture evaluations, pruning the lower hierarchies, even with $p = 0.75$, can lead to sever routing problems and performance degradation.

Lastly, and most importantly, the FPGA architecture needs to be realized in a 2-dimensional layout, and its large size can lead to a very complex physical design if not planned carefully. As shown in Figure 3.1, an efficient physical implementation can allow the FPGA chip designer to start with creating just one LUT macro and its SMs. Although the interconnect wire length between the macros can be different, the hardware logic and the I/O port for each macro are identical. The fully symmetrical architecture allows the LUT macro to be replicated throughout the entire chip, drastically improving design time. The designer can also add more

hierarchies to the physical design flow, such as creating a 4-LUT macro out of the 1-LUT macro, then creating a 16-LUT macro from the 4-LUT macro. However, if the interconnect is to be pruned at every stage, the regularity of the layout can no longer be preserved: assuming all LUTs have SMs at stage 1, using $p = 0.75$, only 75% of the LUTs will have stage-2 SM, and only 56% of the LUTs will have stage-3 SM, and so on. Without regularity in the layout, not only will the interconnect take much longer to design, the reduced SM does not necessarily lead to reduced area. In Figure 3.1, if SMs are reduced for LUT 4, 8, 12, and 16, it would leave a gap in the middle of the layout because the surround macros are larger. This results in a worst-case situation of lost interconnect connectivity *and* lower layout density due to wasted area. When pruning SMs, the designer needs to make sure the reduced SM actually leads to reduced area, and must not over-complicate the layout process. This requires very judicious SM pruning at very strategic locations.

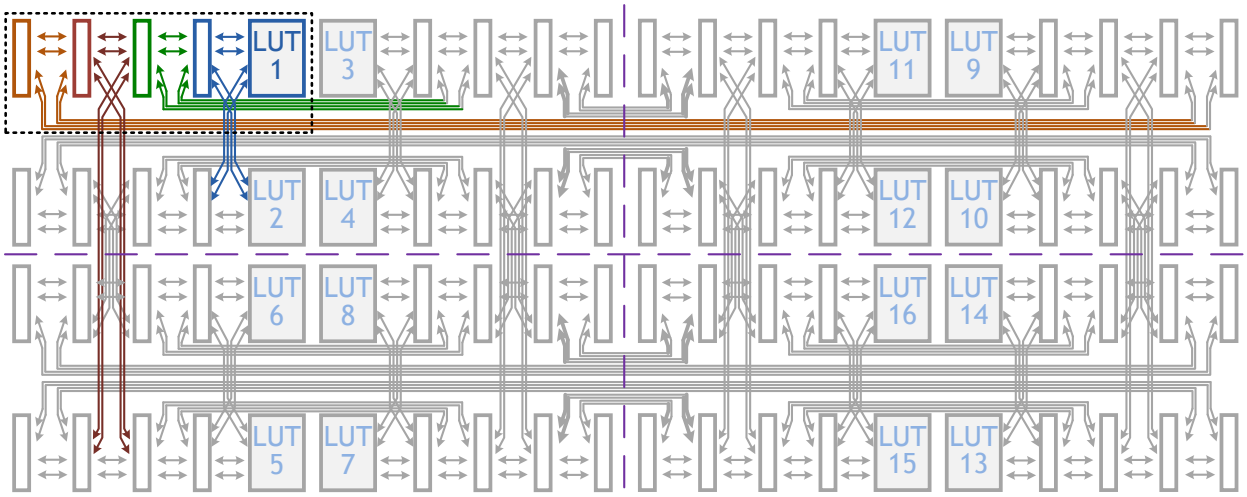
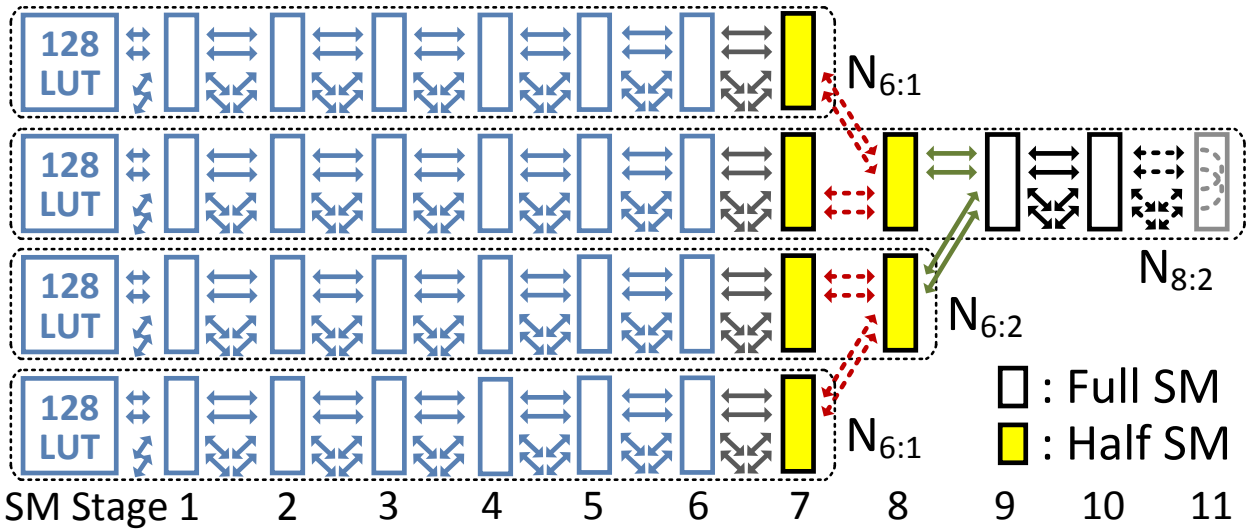


Figure 3.1: A hierarchical macro-based implementation of a 2D-Beneš network.

Overall, realizing a large Beneš network in FPGAs requires 3 things to keep in mind: interconnect connectivity, layout regularity, and layout density.

3.2 Implementing a 2048-LUT FPGA Interconnect

The 2048-LUT test chip requires 11 levels of interconnects. To preserve interconnect connectivity for lower levels, we maintained connectivity (Rent's p = 1) until SM stage 7, followed by 2 stages of p = 0.5, and full connectivity for the top 2 stages. One quadrant of the FPGA architecture is shown in Figure 3.2: the quadrant is divided into 4 macros, each containing 128 LUTs. Inside each 128-LUT macro, all the LUT macros are identical; they are implemented similarly to Figure 3.1, but with 7 stages of SM per LUT. The half-SMs shown in yellow allow 2 out of 4 inputs to propagate upwards, realizing Rent's p = 0.5. Two concatenated half-SMs leads to a top-level connectivity of 25%.



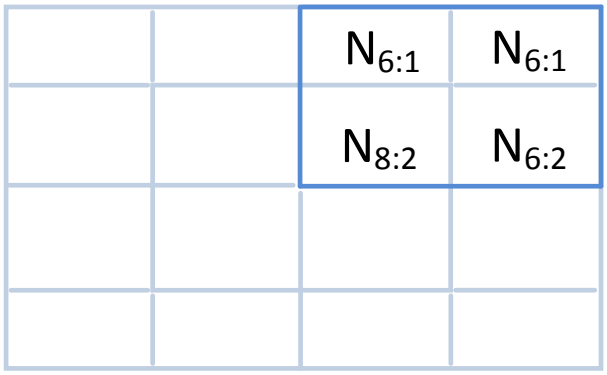


Figure 3.2: Interconnect architecture for our 2048-LUT FPGA, one quadrant shown.

The interconnect network is partitioned into three sub-networks: $N_{8:2}$, $N_{6:2}$, and $N_{6:1}$, where $N_{P:Q}$ represent a network of P full-SMs and Q half-SMs. Intelligent SM-pruning also requires the pruned SM to translate to an area reduction. From the architecture in Figure 3.2, it is clear that the 3 types of SM macros, $N_{8:2}$, $N_{6:2}$, and $N_{6:1}$, will each occupy a different area, because they each contain different number of SM stages. $N_{8:2}$ is the largest macro, followed by $N_{6:2}$, with $N_{6:1}$ being the smallest. To avoid gaps in the layout area, all SMs have the same width. Therefore $N_{6:1}$ macros are shorter. $N_{6:2}$ is also shorter than $N_{8:2}$, leading to some open space. In Chapter V, we will see that the opened space is used by Block RAMs. Because BRAM CLBs are larger than regular CLBs, the area pieces together very densely.

The top level of the chip is shown in Figure 3.3 with the 4 hierarchies of top-level wires shown in colors corresponding to those in Figure 3.2. The top-level layout is symmetrical in the x- and y- direction, allowing the single 512-LUT quadrant to be replicated to form the other 3 quadrants. The chip is divided into 16 macros of 128 LUTs each: macros with $N_{8:2}$ interconnects are placed near the center for shorter top-level routing, branching into $N_{6:2}$ on the left and right. $N_{8:2}$ and $N_{6:2}$ then both branch into $N_{6:1}$ on the top and bottom. This physical placement avoids long wires at the top level, and therefore minimizes interconnect buffers and further reduces area.

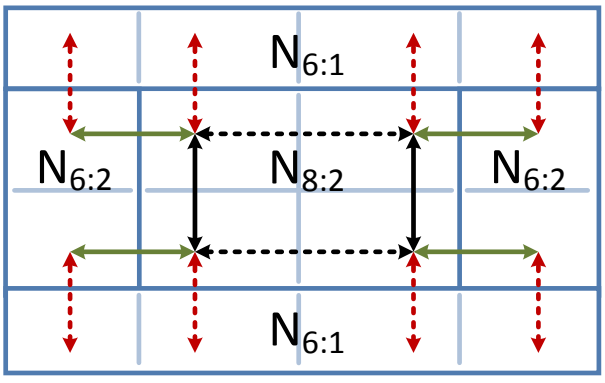


Figure 3.3: Interconnect architecture for our 2048-LUT FPGA, one quadrant shown.

This 2048-LUT architecture is relatively straightforward, using only 2 types of SMs to form 3 types of LUT macros. Scaling into larger designs with even more hierarchies, more advanced architectural techniques are used to further optimize the design. They are highlighted in the following sections (3.3 – 3.6).

3.3 Radix-3 Boundary-less Interconnect

Although hierarchical routing's O($N \cdot \log N$) complexity is much better than O(N^2) from 2D-mesh, it is sometimes inefficient for local routing if the leaves are crossing a high-radix boundary. For example, In Figure 3.4a), LUT 8 and 9 are neighbors, but signals have to traverse up 4 stages of network, and then zig-zag their way down the hierarchy to for LUTs to communicate with each other. Such lack of spatial locality is not desirable.

One method to shorten the nearest-neighbor routing lengths is an isomorphic transformation, as shown in Figure 3.4b). Connections from LUT 8 to LUT 9 can now traverse directly up to stage 4, make a U-turn, and traverse straight down. In terms of connectivity, it is well known that isomorphic butterfly structures maintain the same logic connectivity [Wu80]. Although the wire length travelled has reduced, the number of switches has not: the signal still needs to traverse up and down 4 hierarchies for communication between LUT 8 and 9.

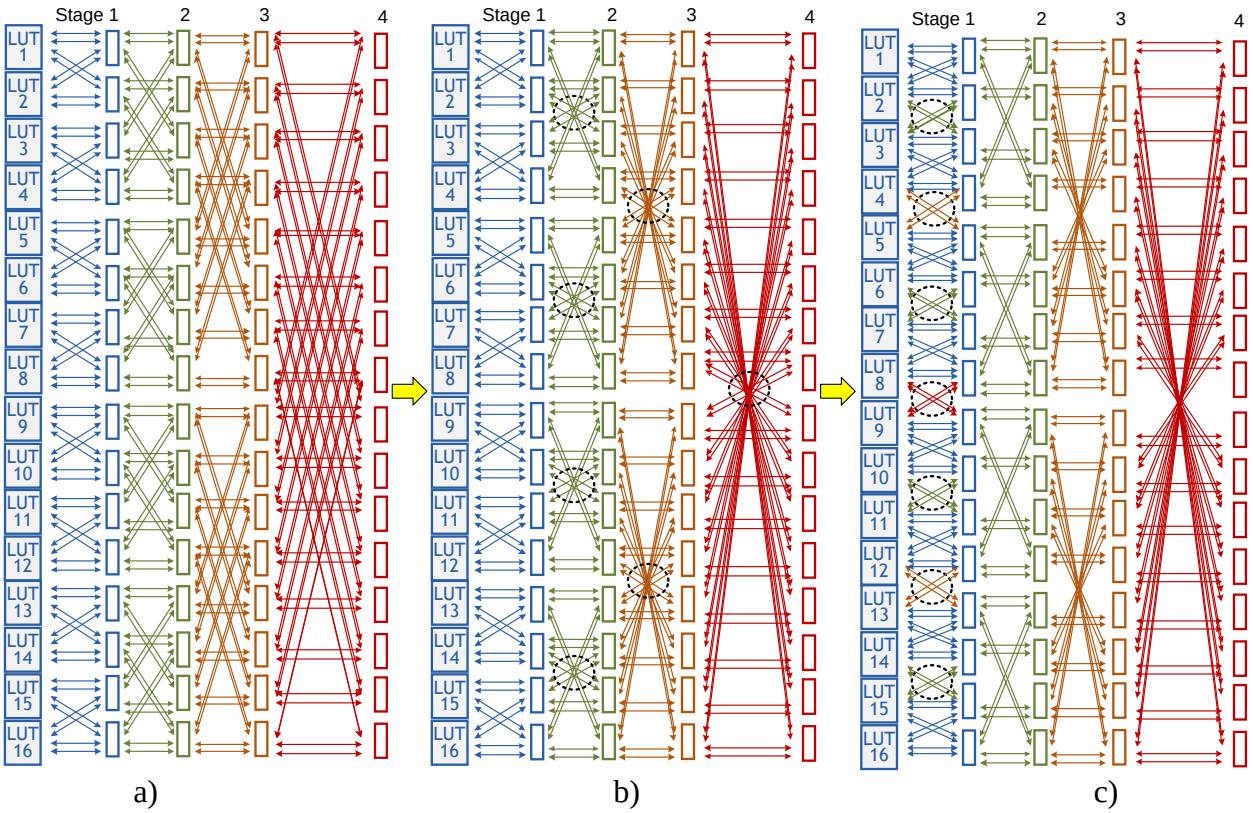


Figure 3.4: a) An original 16-LUT Beneš network, b) with isomorphic transformation to shorten nearest-neighbor lengths, and c) with boundary-less radix-3 switches in stage 1.

In this section, we propose a method of applying higher radix switches on the lower SM levels to utilize spatial locality in routing, allowing efficient interconnect routing for direct neighbors. We call such network a boundary-less radix-3 network [Wang13].

To convert a radix-2 network to a boundary-less radix-3 network, we first identify the center 2x2 routing of each stage, shown in the dashed circle in Figure 3.4b). It is noted that such center 2x2 routing only connects across an interconnect length of 1 (2^0). The first stage transformation into a radix-3 boundary-less interconnect is shown in Figure 3.4c). All center 2x2 routing in the dashed circles are moved to stage 1. This converts stage 1 into a radix-3 interconnect, and all stage-1 switches are capable of communicating with their immediate neighbors, both up and down the SM stages.

With stage 1 completes, we now convert stage 2 to a boundary-less radix-3 switch. We first identify the remaining center 2x2 routing above stage 2 (Figure 3.5a), shown in dashed circles. Note that these 2x2 routings only connect across an interconnect length of 3 (2^1+1). These 2x2 routings are then moved down to between stages 1 and 2 (Figure 3.5b), converting the second stage into a radix-3 boundary-less interconnect.

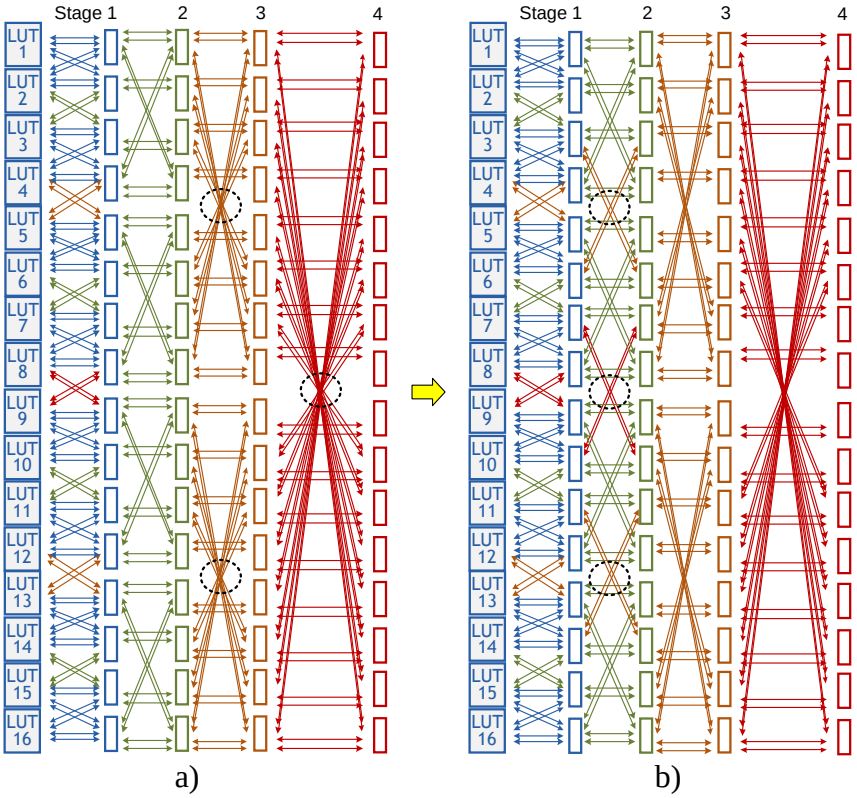


Figure 3.5: A 16-LUT Beneš network with a) boundary-less radix-3 switches in stage 1, and b) with boundary-less radix-3 switches in stages 1 and 2.

The same transformation continues for stage 3-4: we first identify the remaining center 2x2 switches above stage 3, shown in dashed circle (Figure 3.6a). For stage 2-3, we can note the remaining 2x2 switches are actually double pairs, one for LUTs 6−11, and one for LUTs 5−12. The inner 2x2 of the double pair connects across a distance of 5 (2^2+1), while the outer 2x2 connects across a distance of 7 (2^2+3). To maintain consistency, we then move the center double

pair from stage 3-4 (dashed circle) down to stage 2-3 (Figure 3.6b), transforming stages 2-3 into a boundary-less interconnect. It is clear that this stage-by-stage transformation can be continued to the top of the hierarchy. Alternatively, the designer may also choose to stop the transformation at any hierarchy, and preserve the remaining upper hierarchies as traditional radix-2 network.

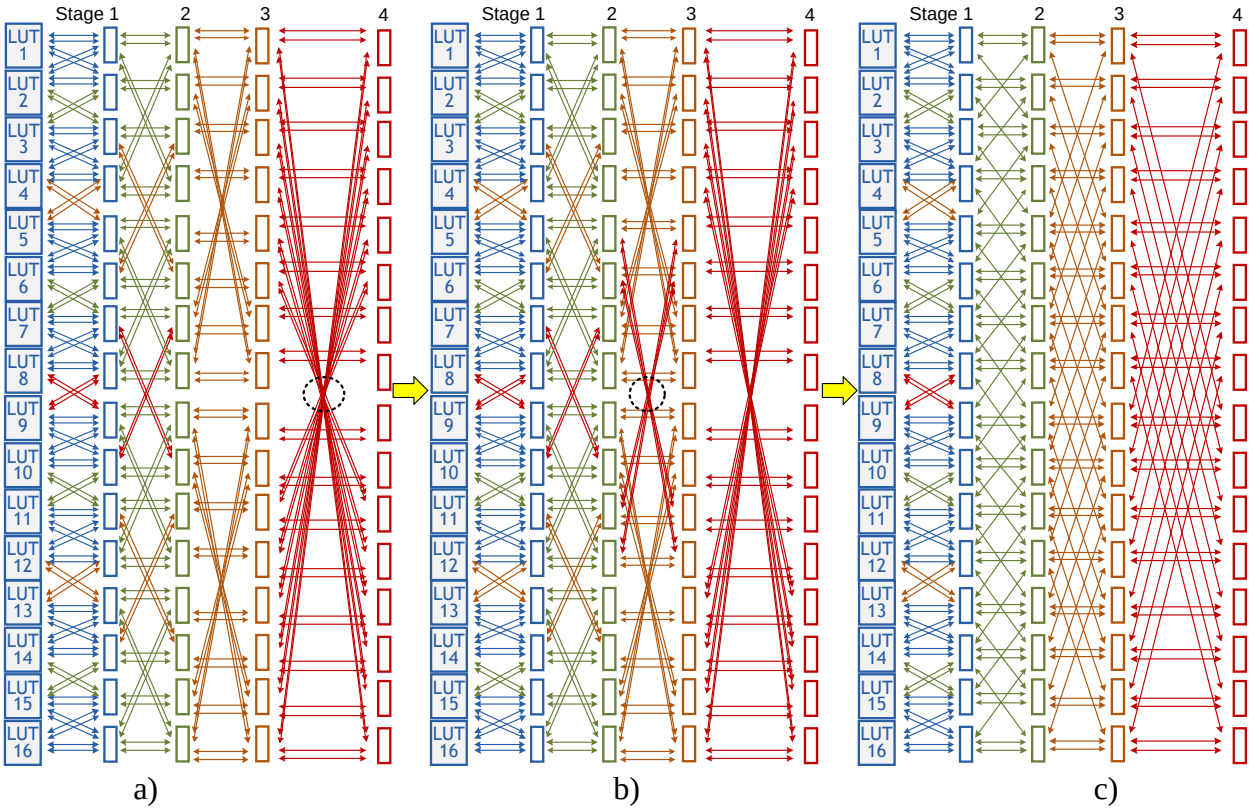


Figure 3.6: A 16-LUT Beneš network, a) with boundary-less radix-3 switches in stages 1 and 2, b) with boundary-less radix-3 switches in stage 1-3, and c) rearranged for distributed routing.

From the intermediate result in Figure 3.6b), we have shown that 50% of the wires branching out above stage 1 have been removed, and the wires on the bottom-most stage have doubled. Since the upper-stage wires are long, and the bottom-stage wires are very short, such tradeoff results in significant wire-length reduction for the architecture. Though shown for a 16-LUT example, this methodology can be extended to a network of arbitrary size.

From this illustration, we see that all stages above stage 1 have unevenly distributed

routing: some switches have to connect more routing than others. This scenario occurs because the wires above stage 1 have been reduced by 50%. To form a regular routing pattern, one method is to evenly re-distribute the interconnect routing: the dual routes branching out of stages 1-4 are re-distributed across all switches, resulting in the final routing architecture shown in Figure 3.6c). We see that the re-distributed routes for stages 1-4 use only single 2x2 butterflies, as opposed to the double 2x2 butterflies used below stage 1.

Given the 50% wiring reduction above stage 1, an alternative method of wire re-distribution is to prune the number of switches above a certain hierarchy. As shown in Figure 3.7a), one method is to prune the switches in stage 3 by moving some wires to a double wire, reducing the number of stage-3 switches by half. Since the remaining stage-3 switches are centered, this results in shorter interconnect length for stage 3-4, and reduces the number of switches in stage 4 by 50%.

Another method is to prune the switches in stage 4 by moving some wires to a double wire, reducing the number of stage-4 switches by half (Figure 3.7b). This can allow the stage-4 switches to reside on 1 half of the network, which can be useful in reducing the wire length of upper hierarchies. For example, for the 2048-LUT FPGA in Figure 3.2, SM stages 7 and 8 can benefit from this technique because the wires are merged toward the center, where the $N_{8:2}$ interconnects reside.

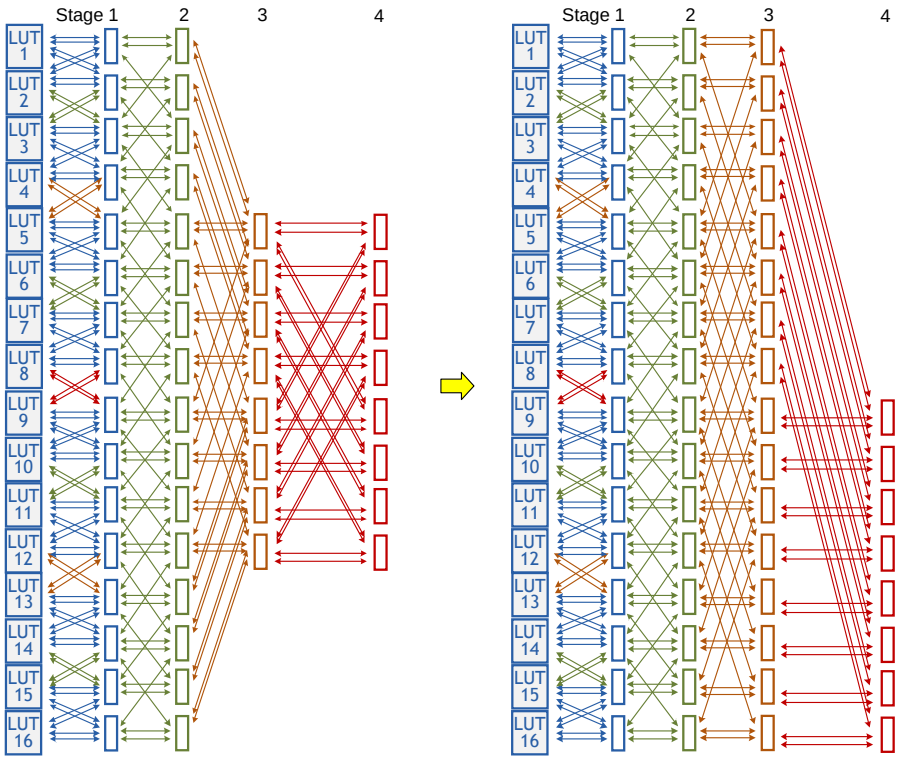


Figure 3.7: A boundary-less radix-3 network with switches pruned at a) stage 3 and b) stage 4.

Although the illustrations above use a radix-3 boundary-less architecture as an expansion to radix-2, it is not limited to this case. For example, a radix-6 architecture can be used as an expansion to radix-4; a radix-12 architecture can be used as an expansion to radix-8; and so on.

For the sake of completeness, Figure 3.8a) illustrates a radix-4 Fat Tree using 4x4 switches. Two stages of radix-4 switches are required to implement a 16-LUT network. To construct a boundary-less network, we first identify the wires in stage 1-2 that have a distance of 4 (4^0): these wires are bolded in Figure 3.8a). These selected wires are then moved down to below stage 1 (Figure 3.8b) to form a boundary-less network in the first stage. The center switches for LUTs 5-12 are radix-6, while LUTs 1-4 and LUTs 13-16 are only radix-5 in this illustration because they rest on the boundary of the network.

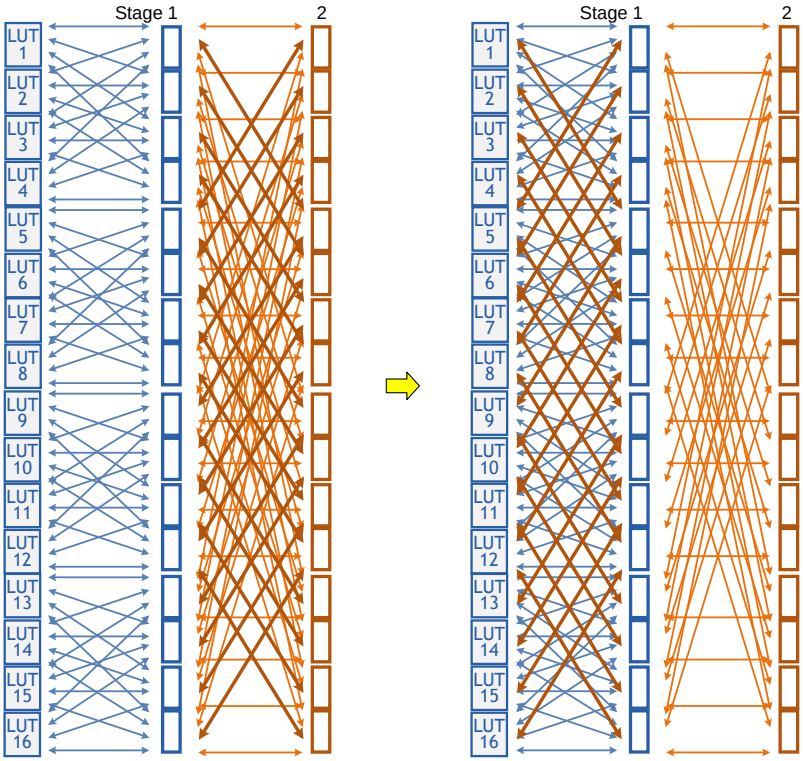


Figure 3.8: a) An original radix-4 16-LUT Beneš network and b) with boundary-less radix-6 switches in stage 1.

3.4 Fast-Path Interconnect

In VLSI designs, there usually exists a critical path, that is, a path that is more difficult to meet timing constraints. In most VLSI designs, the vast majority of the paths do not reside on the critical path, but those that are on the critical path usually determine the performance of the entire design. We therefore propose an addition to the interconnect SMs to allow faster performance for critical-path gate: fast path.

In the example in Figure 3.9a), we see an example routing from LUT 2 to LUT 16. One possible route is highlighted. Beneš network offers many path diversity (thus it is rearrangeably

non-blocking; without path diversity, the network offers very limited connectivity (such as [Mrabet06] from Section 2.5)), and we are simply choosing one path as an example. The signal needs to traverse up to stage 4 before U-turning back down. With the addition of fast-path, signals are allowed to travel from the LUT output directly to all SMs within its macro (Figure 3.9b). Therefore, the signal is able to travel directly from the output of LUT 2 to the SM on stage 4, and then U-turning back down. Following the macro-based design methodology highlighted in Figure 3.1, a LUT is placed with all its SMs in one macro during physical design, so adding fath-path routing within the macro does not add any interconnect routing outside the macro.

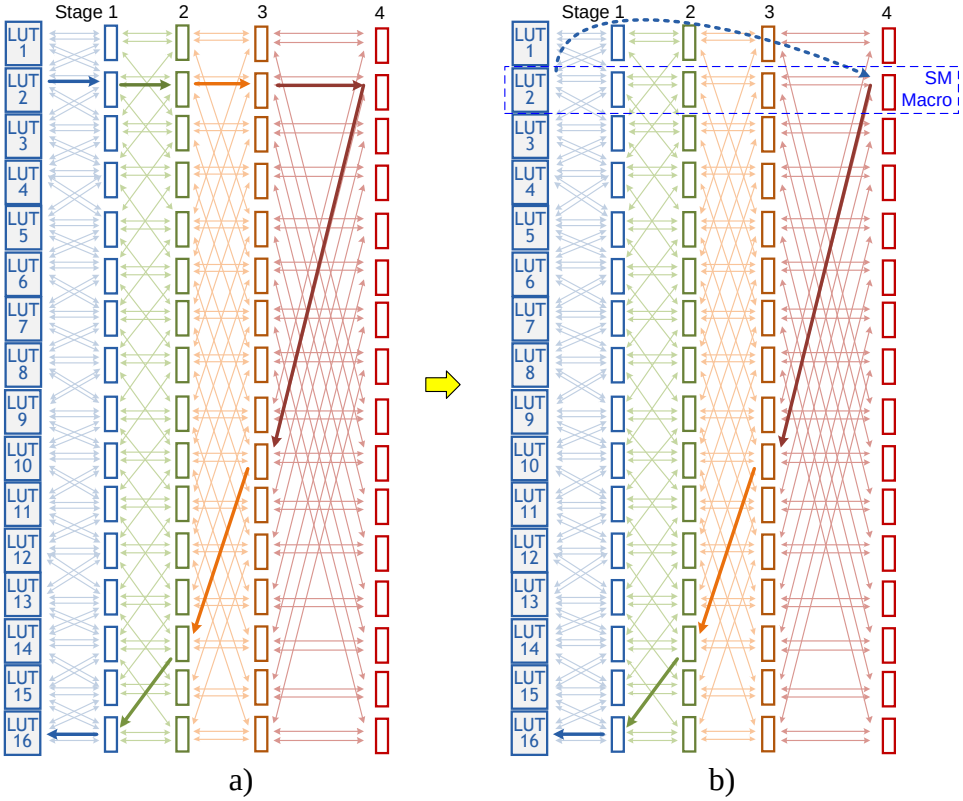


Figure 3.9: A routing example from LUT 2 to 16 a) without fast path and b) with fast path.

For each point-to-point connection, there is always at least one fast-path available, but other routes that conflict with the fast-path routes must take the slower route. In a timing-driven place-and-route flow, this gives the software tool freedom to choose a faster path for more

timing-critical routes.

In cases where routing obstructions occur, it is sometimes still possible to utilize portions of a fast-path, and use regular routing for the remainder of the routes. One such example is illustrated in Figure 3.10a), although it would be ideal to have fast-path directly connected to SM stage 4, the router can still connect fast-path to SM stage 3, and use regular routing to complete the route. In other cases, it is sometime impossible to use any fast-path, and regular routing must be used entirely (Figure 3.10b). Even under such cases, path-diversity allows for many routing choices, and the boundary-less radix-3 network sometimes even allows for fewer SM stages. In Figure 3.10b), one example route requires 4 SM stages, while another requires just 3 SM stages. It is up to the timing-driven P&R tool to select the faster path for timing-critical nets.

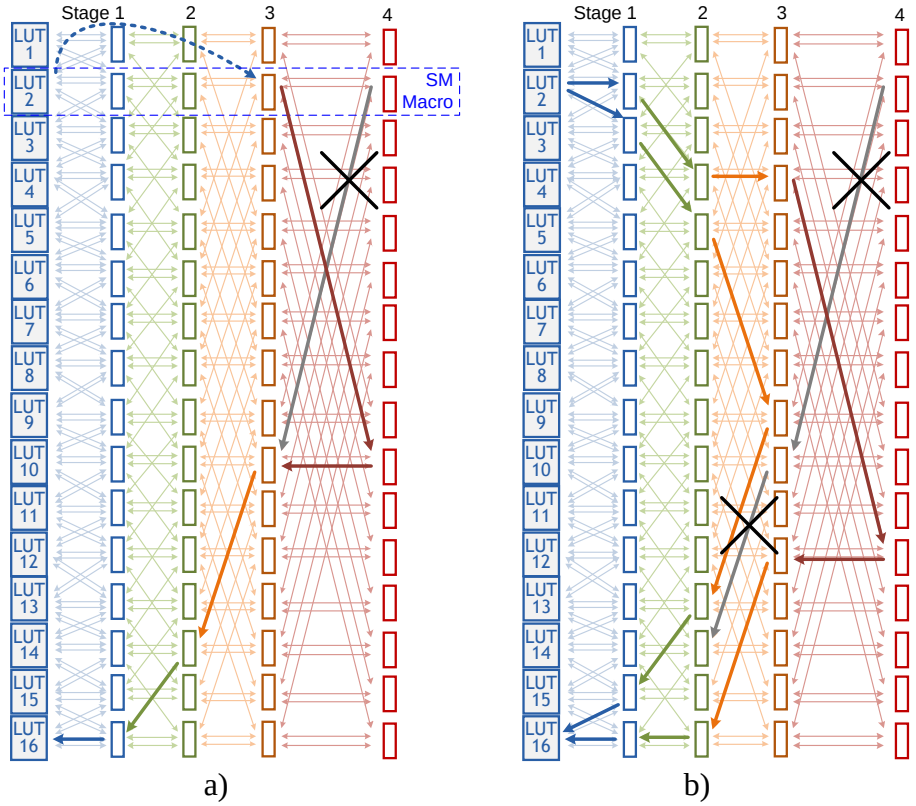


Figure 3.10: A routing example with routing obstruction that a) still allows a slower fast-path and b) allowing no fast-path.

3.5 Interconnect Cost vs. Gate Cost

In an FPGA, upper-level interconnects are often required to travel long distances, and it would be beneficial to reduce the number of these nets. On the other hand, interconnect switches are also a dominating factor for chip area, and it would be beneficial to reduce the number of these gates as well. Although it is ideal to reduce both, there also exists a trade-off between these two factors.

From the simple example in Figure 3.11, the two types of SMs have the same gate cost. Actually the 4-input muxes in Figure 3.11b) cost more when implemented as a traditional mux, as it takes three 2-input muxes to implement. As a static parallel mux (Chapter IV), a 4-input mux occupies as much area as two 2-input muxes. The muxes in Figure 3.11a) only allow for odd-to-odd and even-to-even switching, but the SM has double the number of muxes.

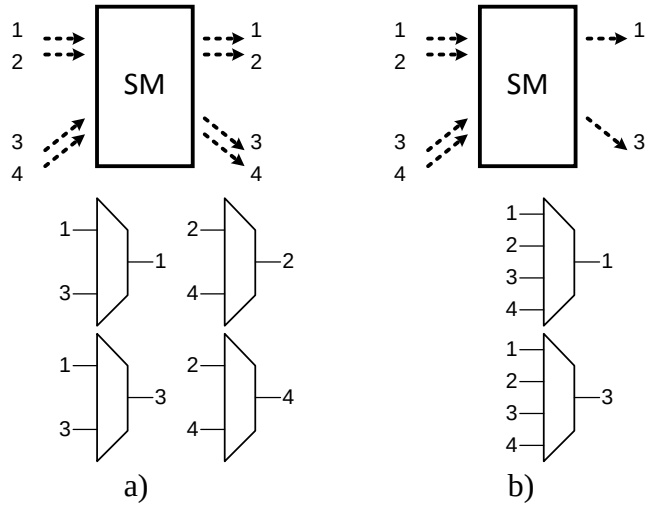


Figure 3.11: Two SM design with same gate cost, but a) with more wiring than b).

In terms of connectivity, the design in Figure 3.11a) is superior. For example, if input (1) travels to output (1), the design in Figure 3.11b) will not be able to send another signal in the horizontal direction. But the design in a) is still able to send another signal through output (2) as long as it does not need to route input (3). Overall, design in a) provides more path diversity for

routing.

A different scenario arises when the wire lengths are long, and signals (3) and (4) would need to be buffered (sometimes more than once). When the wires are long and the buffers are large, the signal buffering area can easily outweigh the mux area. In this case, the design in a) is clearly inferior: it requires double the number of buffers but does not provide double the connectivity of b).

For lower-level SMs, where the wiring is short and does not require additional buffers, it is beneficial to use limited-input muxes, but implement more of them to improve path diversity. For upper-level SMs with high wiring cost, it is beneficial to reduce to number of wires, in which case full-input muxes should be used, but fewer should be implemented to save wiring cost.

3.6 Local Interconnect vs. Branch Interconnect

In FPGA, interconnect wiring is expensive, because it contributes to routing congestion and buffer gate area. But local interconnects are much cheaper to implement. In traditional Beneš networks, each SM provides just as much local interconnects as branch interconnects (Figure 3.12), even though interconnects that branch to long wires cost significantly more hardware area. To reduce hardware, it is more effective to prune branch interconnects before pruning local interconnects. Local interconnects alone can also contribute to path diversity. In the example in Figure 3.12 (right), the fastest route from LUT 2 to LUT 14 is using the fast path, but let us assume two downward paths between SM stage 1 and 2 are blocked by other timing-critical signals. In this case, a design with traditional SM switches would be required to take a longer route, but a SM design with more local interconnects (4 in this example) can still provide a downward path for this route.

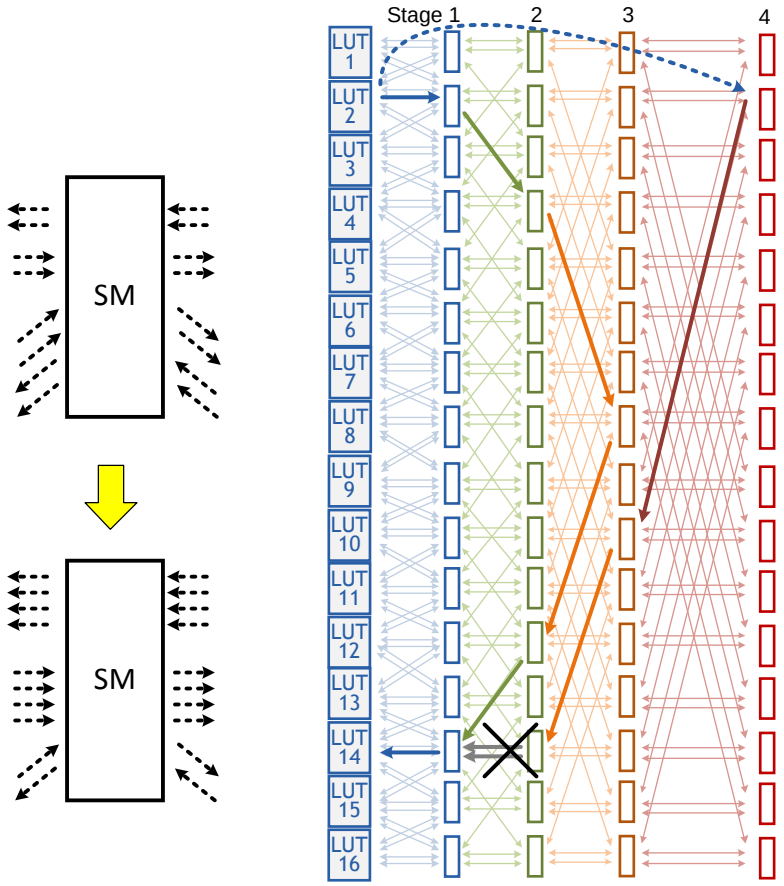


Figure 3.12: An example where traditional-Beneš based SM experiences local interconnect congestion, whereas a SM design with more local interconnects can utilize the fast path.

An example SM design with 4 local interconnect and 1 branch interconnect is shown in Figure 3.13. When implemented as a SM macro, the local interconnects are contained inside the macro. Compared to the traditional-Beneš based SM design, the new design reduces the interconnect wiring in and out of the macro by 50%, but doubles the local interconnects. Such SM design is very effective for upper-level SMs where the branch interconnects are expensive. This essentially follows the same optimization strategy from Section 3.5: it adds more wires and uses simpler muxes when the wire cost is low, but use larger muxes and fewer wires when the wire costs are high.

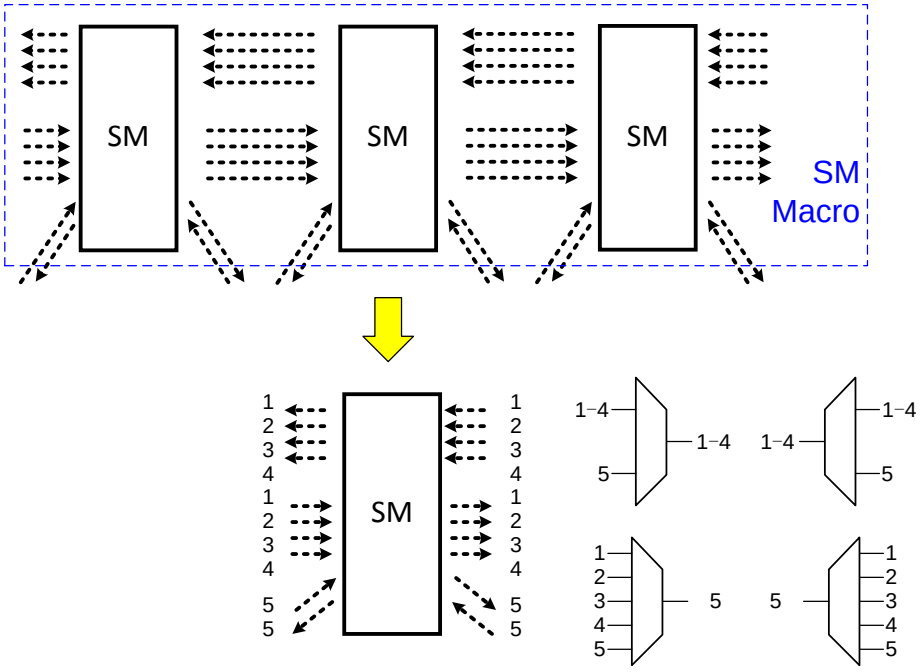


Figure 3.13: A switch-matrix example with more local interconnects than branch interconnects.

3.7 Micro-architecture of a Switch Matrix

A switch matrix (SM) is the most commonly used building block in the hierarchical FPGA – our FPGA has more than 10x as much SMs as LUTs. It is therefore important to have an SM design that is as small as possible, yet provides sufficient connectivity. Figure 3.14 shows an example SM micro-architecture of a radix-3 SM used in our most recent FPGA (details in Section 3.8). Not surprisingly, a SM consists of simply of a collection of muxes. The number of SM outputs determines the number of muxes it needs, but we need to carefully decide how much connectivity to build into each mux, for that has a large impact on the SM area, which has a significant impact on the final chip area.

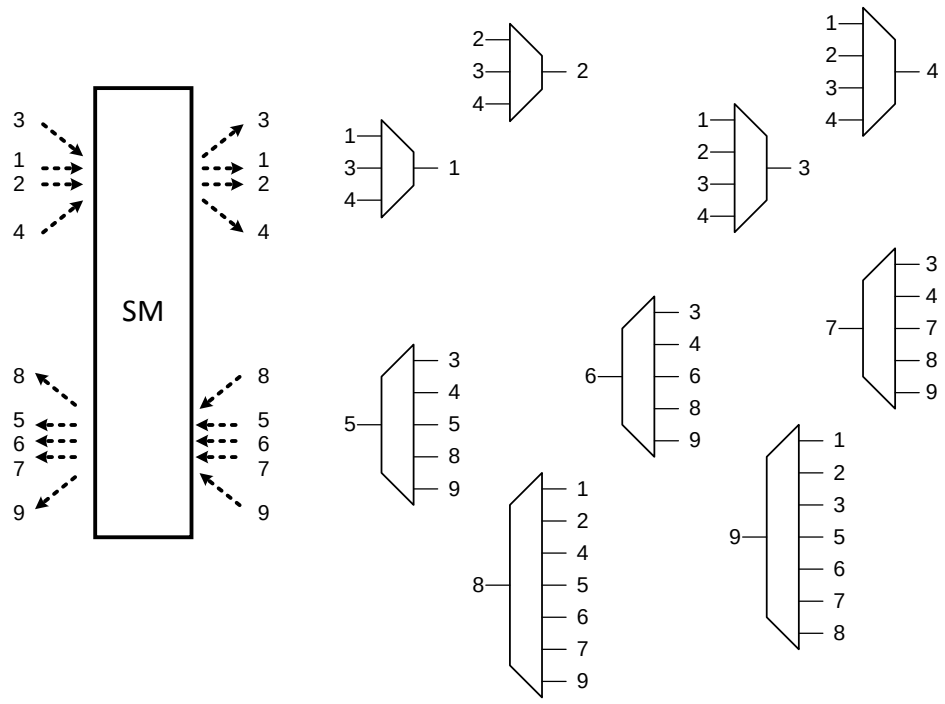


Figure 3.14: Internal mux interconnect of an example radix-3 switch matrix.

In Figure 3.14, signals 1–4 are upstream signals. Signals 1 and 2 travel internally inside the SM macro, and signals 3 and 4 are branches. From the mux design of 1 and 2, we see the first pruning heuristic: muxes 1 and 2 are allowed to propagate only signals 1 and 2 upwards, respectively, and both 3 and 4 are allowed. This is because outputs 1 and 2 travel in the same path. Not allowing switching between paths 1 and 2 has minimal impact on routing results, but reduces the mux complexity for 1 and 2. Using this simplification, the incoming signal from branch 3 and 4 will be assigned to path 1 or 2 (or both if decided by the router), and remain in the assigned path until it branches out again.

Similar approach applies for the downward paths: incoming signals can be assigned downward paths 5, 6, or 7, and are not allowed to switch between these paths until the signal branches out again. For U-turns, another simplification can be made. For example, there is no need to U-turn from input 1 and 2 back down to output 5, 6, or 7, because they come from the

same SM. There is never a need to ascend one hierarchy and U-turn back to the same SM. Similarly, output 8 travels back to the same SM that input 3 is coming from, so there is no need to performance that U-turn either; the same case applies to output 9 and input 4.

These micro-architectural techniques are effective in reducing SM complexity, thus reducing area and improving mux performance. But even with these techniques, the muxes still poses a large overhead on area and performance. Many circuit-level techniques are applied to implement these SMs efficiently, as discussed in Chapter IV.

3.8 Implementing a 16K-LUT FPGA Interconnect

In the previous 2048-LUT FPGA chip (Section 3.2), the architecture was optimized manually, and two types of SMs are utilized. To fully demonstrate the scalability of hierarchical interconnects, the new FPGA has expanded the interconnect architecture by 10x. Since there is no theoretical method to calculate the optimal connectivity at every level of the hierarchy, we have also developed a software tool to map designs onto our architecture (Chapter VI), which allows us to explore the optimal interconnect architecture using an iterative, closed-loop design process: we explore different architectures by mapping benchmarks and commonly-used designs, then examine the interconnect usage across different SM stages and locations, then refine the architecture accordingly and perform the mapping process again.

This FPGA consists of 16K "LUTs" arranged on a 64×320 array. Because it is a heterogeneous FPGA (Chapter V), each "LUT" is limited not to a look-up table, but is more like a SM macro that provides I/O capabilities: in this case, each SM macro provides 5 inputs and 2 outputs to any CLBs, logic, memory, DSP, or others. For example, a SLICE L CLB contains 30 inputs and 12 outputs, it therefore requires 6 SM macros to implement its interconnect; on the

other hand a DSP CLB requires 165 inputs and 66 outputs, requiring 33 SM macros in a 3×11 array.

The SM architecture of the 16K-LUT FPGA is shown in Figure 3.15 and 3.16. Figure 3.15 illustrates the lower 10 SM levels on a 1-dimensional drawing, although physical implementation is 2-D. Figure 3.16 illustrates the top-level physical architecture, highlighting wiring for the top 5 SM stages. The SM architecture is symmetrical across horizontal bisection, and is composed of 7 types SM macros, ranging from 10 to 14 stages of SM. The bottom 10 stages of SM are common across all SM macros, and are illustrated in Figure 3.15.

The CLB-input requirements in this chip ranges from 30, 35, 165, or 180 inputs, therefore the switch matrix in this architecture is chosen to contain 5 inputs and 2 outputs as a common denominator. From Figure 3.15, it shows each LUT to provide 5 inputs and 4 outputs, that is because each output is multi-casted to both local and branch interconnects, similar to the multi-cast concept from Section 2.4. The bottom 5 stages of the SM utilizes boundary-less radix-3 interconnect, providing short routing distance to neighbors and providing extra path diversity for the network. Above stage 5, we transition back to a radix-2 network to save interconnect area. Additionally, having all radix-3 network in all hierarchies would make the entire architecture boundary-less, which drastically increases place-and-route time. The current timing-driving routing algorithm is based on breadth-first search, and by having radix-2 in the upper hierarchies, the P&R tool is able to converge more quickly due to its reduced search radius. From our P&R evaluations, a radix-3 to radix-2 transition at SM stage 5 provides sufficient path diversity and routing performance.

This SM architecture uses 2 local interconnects per SM on the upward path, but 3 local interconnects per SM on the downward path. This is due to the assistance of fast-path, which

allows many signals to travel directly from the LUT output to the upper-level SM without occupying local interconnects along the way. This alleviates the routing congestion upwards, but does not alleviate the congestion downwards (the fast-path signals still need to traverse downwards on regular interconnects).

Another key distinction between the lower 5 SM stages and upper stages are the upward branch interconnects. From Figure 3.15, we the lower 5 SM stages to have branching on the upward path, but above stage 5, upward branching has been pruned, and only local upward interconnects remain. The exception is for SM stages 10, 11, and 12, for those stages allow the SM to branch upwards upon the termination of the SM macro. As shown in Figure 3.15, the SMs on the bottom half only have 9 stages, and therefore must branch into the SMs on the top half at stage 10 to continue signal propagation, else the signal would reach a "dead-end". This pruning methodology trades off local vs. branch interconnects: it allows branching when the wire costs are low, therefore providing more path diversity, but for the upper hierarchies, path diversity is sacrificed to reduce interconnect congestion and gate area. However, local interconnects are not pruned even for upper hierarchies, because those wire costs remain low, and having 3 local interconnects on the downward paths provides additional path diversity without increasing the area significantly.

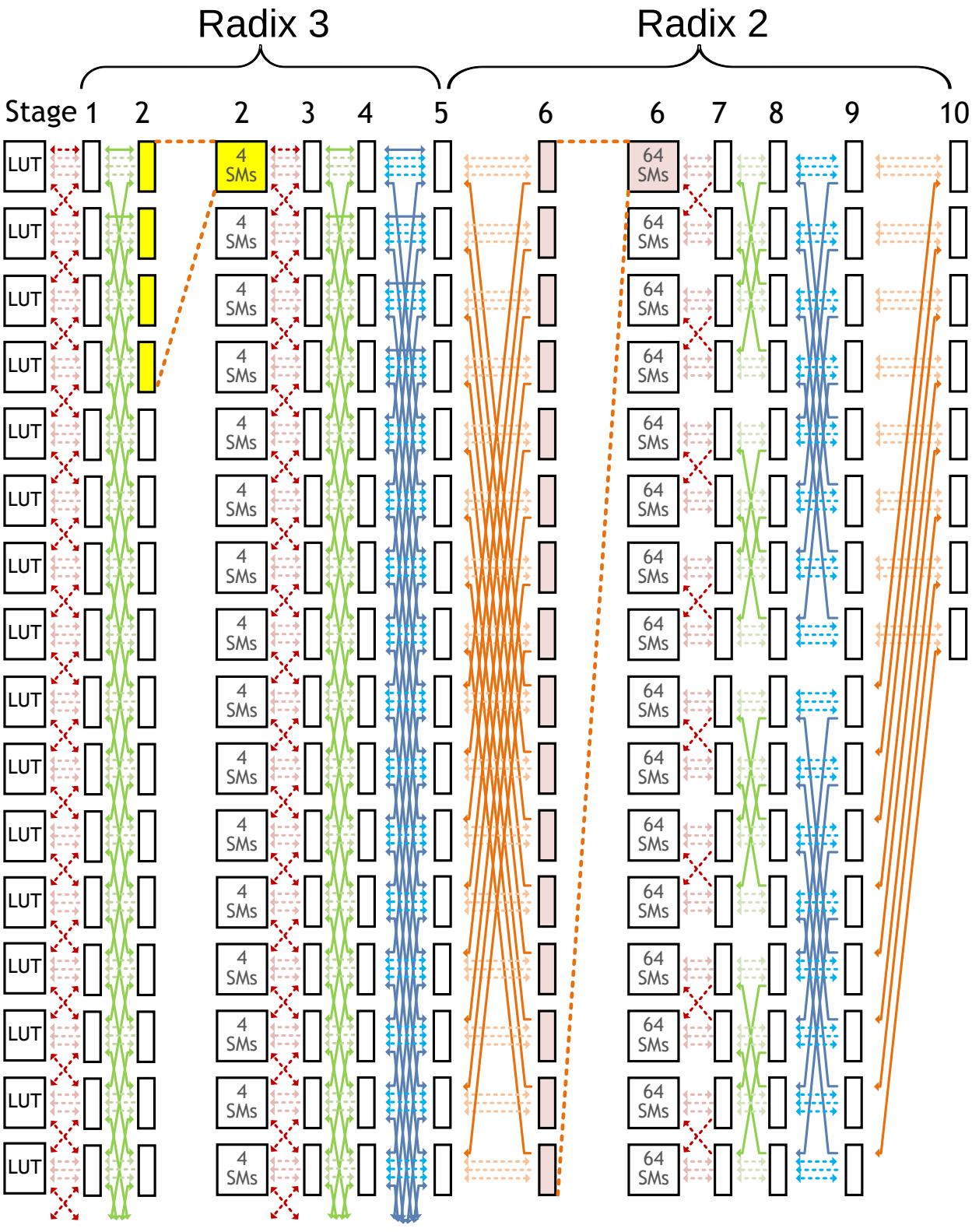


Figure 3.15: 1-D SM architecture of the 16K-LUT FPGA, showing the lower 10 SM stages.

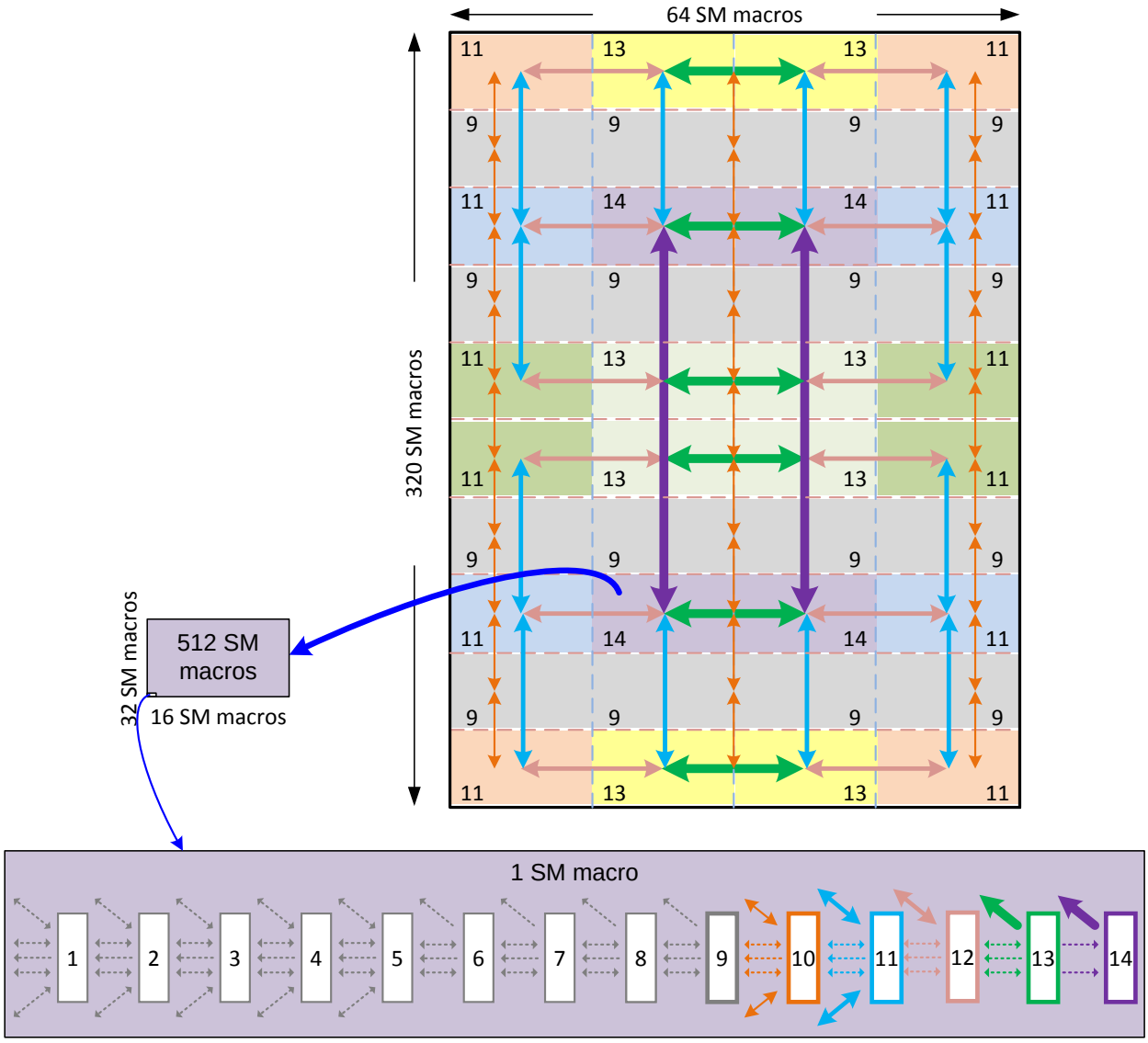


Figure 3.16: 2-D SM architecture of the 16K-LUT FPGA, showing the top 5 stages of wiring.

In the top level, the SM architecture is divided into 40 macros, each containing 512 SM macros. From the iterative interconnect optimization process, we've converged to an architecture shown in Figure 3.16. There are 7 types of SM macros, shown in 7 different colors. The most commonly-used SM macro has 9 stages, spanning across rows 2, 4, 7, and 9. The remaining SM macros have 11, 13, or 14 stages of SM (labeled in Figure 3.16). The largest SM has 14 stages, shown in the inset of Figure 3.16. These SMs reside in the center of the top and bottom halves of

the network.

Figure 3.16 also illustrates the mixed-radix implementation in the top level. SM stages 10 and 11 are actually radix 3, but are not boundary-less (with the exception of some stage-10 routing that crosses the horizontal bi-section). This is partially because the number of rows (320) is not a radix-2 number. Without utilizing radix-3 SM, another stage of SM would be required. However, since 320 is not much larger than 256, adding a SM stage appears wasteful. The other reason is due to the wiring cost of stage-14 routing, which needs to span half the height of the FPGA. This results in very long wires, and are very expensive to buffer. To reduce the requirements on the number of stage-14 routing, boundary-less stage-10 routes are implemented along the horizontal bisection. This addition allows gates that are placed near the horizontal bisection to use the shorter, and faster, stage-10 routing. Only the gates that are required to communicate across the entire chip need to occupy stage-14 routing.

The final architecture in Figure 3.16 is arrived through extensive iterative improvements to the architecture. The automated P&R flow (Chapter VI) greatly expedited the evaluation process, and gives us confidence in the routability and performance of the optimized design. The architecture techniques discussed in 3.3−3.6 have greatly improved the routing quality of the interconnect network, and reduced interconnect area. Although we have expanded from 3 types of SM macros to 7, it remains a feasible implementation. The circuit-level implementation of the interconnect are detailed in Chapter IV, and the physical integration details are discussed in Chapter V.

CHAPTER IV

Interconnect Circuit Design

4.1 Key Building Blocks in Interconnect Circuits

FPGA interconnect is a complex network that spans the entire area of the chip, but it consists of just three families of circuits: multiplexers, buffers, and configuration bit-cells (BC). In Figure 4.1, an example switch matrix (SM) is illustrated with its internal circuitry.

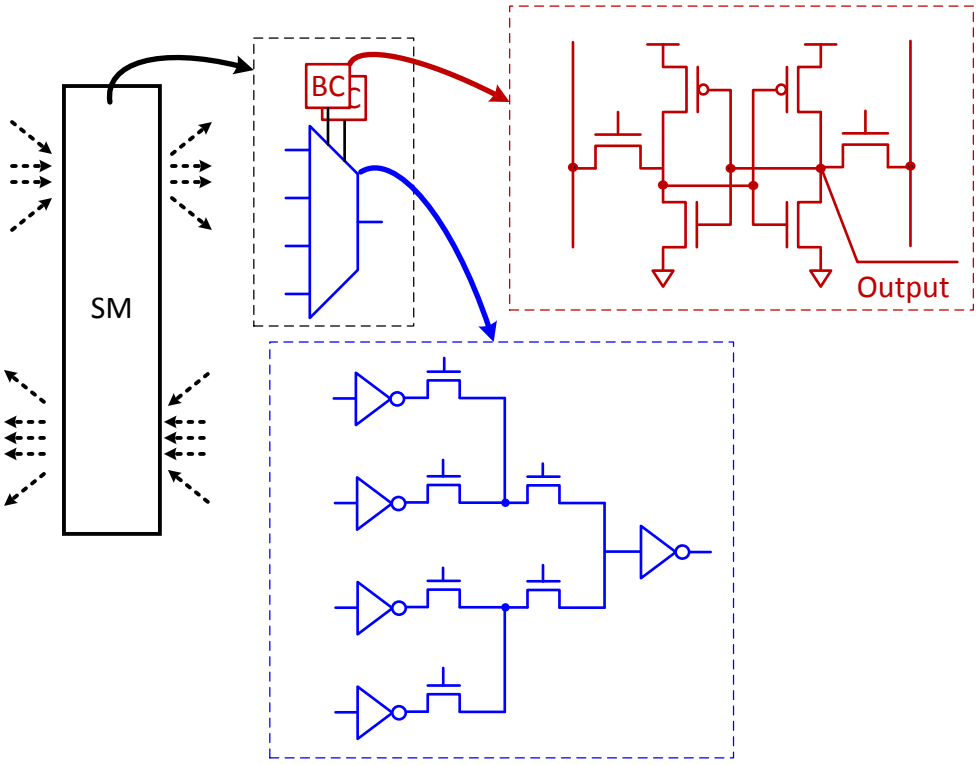


Figure 4.1: An example switch matrix with its internal circuitry.

As previously shown in Section 3.7, each mux propagates interconnect signal from one of its inputs to its output. The input to be propagated is pre-configured and stored in the bit-cells. The bit-cells are generally SRAM-based, though non-volatile configurations generally use flash memories, and are used to drive pass transistors to perform signal selection.

Buffers can be inverting or non-inverting, generally composed of one of more CMOS inverters. The buffering requirements are generally determined by the number of pass-transistors used and the interconnect wire-length.

Although the individual components of the interconnect network can appear simple, each component is replicated thousands or even millions of times on the entire chip. Therefore even a small reduction in area, delay, or power of an individual block can have a significant impact on the chip level. The following sections highlight the circuit-level techniques used to optimize these individual blocks.

4.2 Static Multiplexers and Area-Performance Tradeoff

In traditional CMOS circuits, muxes are generally implemented in a tree structure, where N inputs are controlled by log_2N select lines, such as the mux in Figure 4.1. This structure has its benefits for CMOS circuits, because it uses the fewest number of select lines, and the output of the mux is always deterministic (that is, there is always one input driving the output, and there will never be more than 1 input "fighting" for the same output). Although Figure 4.1 has only NMOS pass-transistors, they are generally not used as stand-alone pass-gates, each pass transistor is usually constructed using a NMOS-PMOS transistor pair, whose gates are driven by true and complement select-line signals.

FPGA interconnect pass-transistors are used differently: the select lines of the mux come from bit-cells, which are static: the user (or the software tool) controls the data to write into the BCs, which are programmed before the chip starts its operation, and the BCs remain static during the operation of the FPGA. We can utilize this scenario to enhance the performance of the

muxes.

The BCs drive the gate of the pass-transistors, which turns the pass transistor on and off by effectively changing the resistance across the source and drain of the pass-transistor. We can improve the performance of the pass-transistors by operating the BCs on a higher VDD. This effectively over-drives the gate of the pass-transistors to reduce its on-resistance. Although increasing the VDD generally increases the active power quadratically, it does not affect the active power of the FPGA at all: the switching signals travel across the source and drain of the pass transistors are still toggling at the lower VDD (Figure 4-2). This is especially useful when VDD_L, is scaled down. In this scenario, total energy will decrease as $C \cdot VDD_L^2$, just like normal circuits, but system performance will not degrade by as much, because the on-resistance of the pass-transistors remains constant. The only power penalty of this implementation is increased leakage through the bit-cells, and increased gate-leakage through the gate of the pass-transistor. This implementation also has a potential area penalty with implementing two voltage domains, one for BCs and one for the remaining circuits. But having two voltage domains enables us to further optimize the multiplexer circuits, as shown in the next section. In the chip design chapter (Chapter V), we see will that the area overhead from dual-VDD is kept to a minimum.

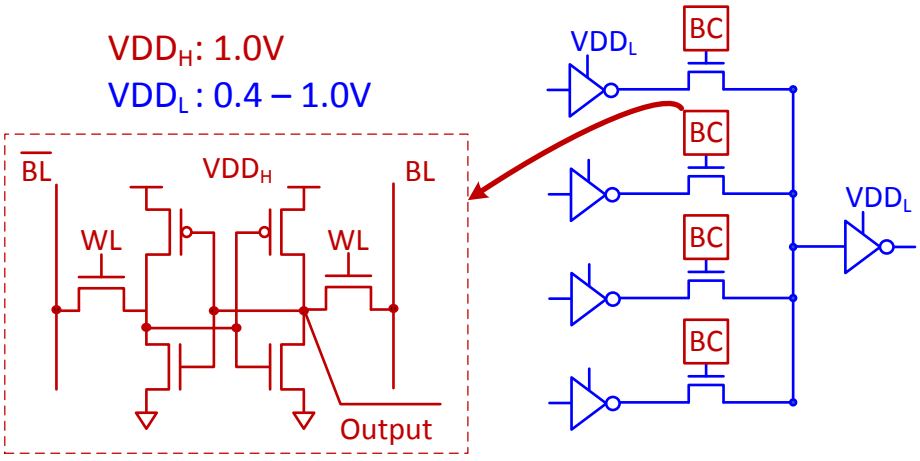


Figure 4.2: A static pass-transistor mux with high VDD for the bit-cells.

Since the BCs are controlled by the user or the software tool, and are static during the operation of the FPGA, we can convert to a static, one-hot-bit mux design (Figure 4.2) [Lewis05, Lee06]. In this case design, *N* independent BCs are used to control an *N*-input mux. To avoid driving conflicts, the user or the software tool must ensure that one and only one of the BCs are enabled at all times. The BCs must not be all off either, as that would cause the output buffer of the mux to be floating.

The static muxes have a significant performance advantage over traditional muxes, for it reduces the number of pass-transistor stages from log_2N to 1. It also results in smaller mux area, but pays a much larger penalty in BC area, because it now requires *N* BCs instead of log_2N BCs. It also requires the BCs to be configured properly before the rest of the circuitry (VDD_L) can be powered-on [Calhoun10, Ryan10].

The performance benefit of static pass-transistor mux is clear, but the area penalty of *N* bit-cells per *N*-input mux can become significant. For example, an 8-input mux implemented as a parallel mux requires 8 bit-cells, doubling the area compared to a traditional 8-input mux with 3 bit-cells. We mitigate the area overhead by adopting a partially-parallel approach, as illustrated in Figure 4.3. For the 10-input mux, a traditional mux would require 4 BCs, with a critical path of 3 or 4 pass-transistors. Using a fully-parallel mux would require 10 BCs, but a critical path of 1 pass-transistor. However, for very wide parallel muxes, the leakage through the 9 "off" pass-transistors can also be significant. The design in Figure 4.3 provides a viable compromise between performance, area, and leakage. Assuming input 1 and 2 are critical-path inputs (generally branch inputs into the SM), and the remaining inputs are less timing-critical (such as local inputs or U-turns), the micro-architecture in Figure 4.3 allows the critical-path inputs to propagate with just 1 pass-transistor. Inputs 3-10 requires 2 pass-transistors, but it is still much

faster than the 4 pass-transistors otherwise required by a traditional mux. The total BC count of this design compromise is 8 bits. For large-input muxes, the inputs can often be categorized into timing-critical and non-timing-critical, and this design approach has proven to be very effectively in providing faster performance for timing-critical nets, yet with a smaller area and leakage penalty.

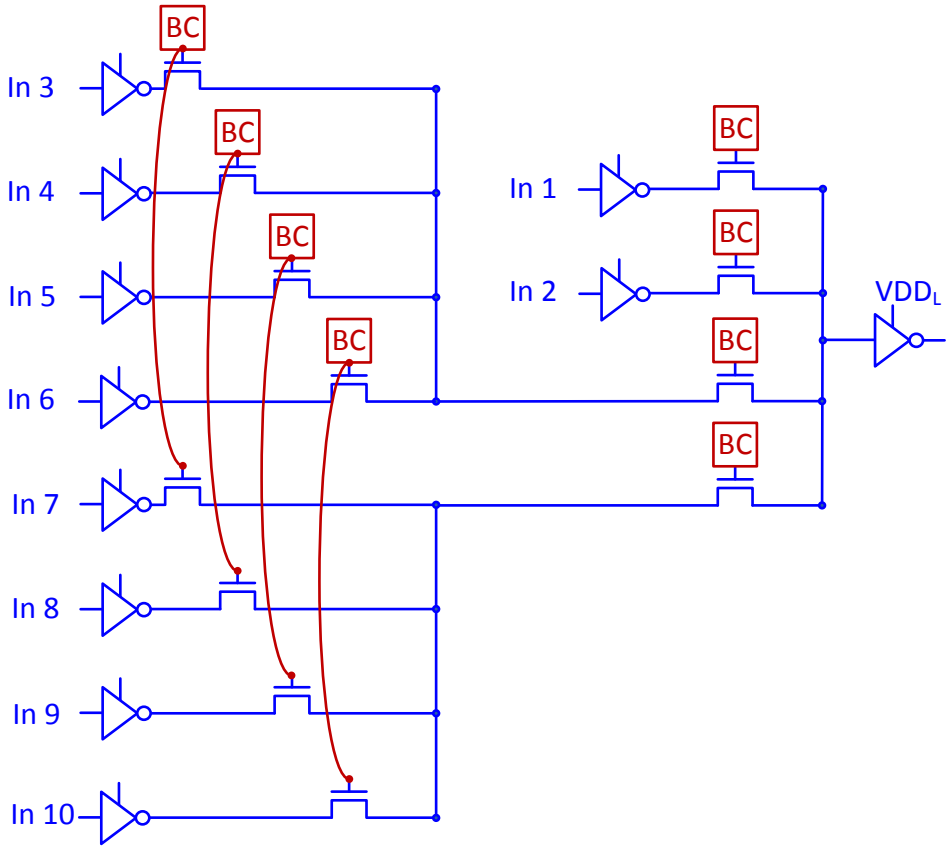


Figure 4.3: A 10-input static pass-transistor mux with 2 critical-path inputs and 8 non-critical-path inputs, requiring 8 bit-cells.

In addition to parallel static muxes and dual-VDD, additional techniques such as using multi-threshold-voltage transistors, multi-oxide-thickness transistors [Curd07], and even body biasing [Rahman04] can be applied to interconnect pass-transistors to further exploit the static-nature of the gate signals. They can be implemented when available. In our process, the lack of

triple-oxide process disallows us from applying body biasing, and unlike Virtex FPGAs [Curd07], we do not have a separate medium-thickness oxide transistor for interconnects.

4.3 Strategies for Interconnect Buffering

The mux designs illustrated in this chapter have been non-inverting – CMOS inverters always exist at both the input and output of each multiplexer, so the polarity of the signal is preserved. Having the input of the cell tied to the gate of the transistor makes the cell more immune to latch-up issues, and such robust design is a standard practice in almost all standard-cells. If the input is tied to source or drain of the transistor, the designer needs to make sure the input voltage never exceeds the body voltage of the PMOS transistors, else latch-up will occur.

In designing our FPGA chips, we separated the input buffer from the remaining of the mux, essentially creating inverting muxes (Figure 4.4). Inverters are placed at the input to buffer the incoming signals. The inverted signals are then used to drive the inverting muxes. By sharing the input buffers among all the muxes, we can reduce the buffer area and buffer power. For example, input 3 in Figure 4.4 is tied to 4 muxes. Previously, it would be required to drive all 4 input inverters inside the 4 muxes, even if it only needs to drive one of the 4 muxes. With the new design, it drives just one input inverter, and the inverter will only drive the pass-transistors that are turned on.

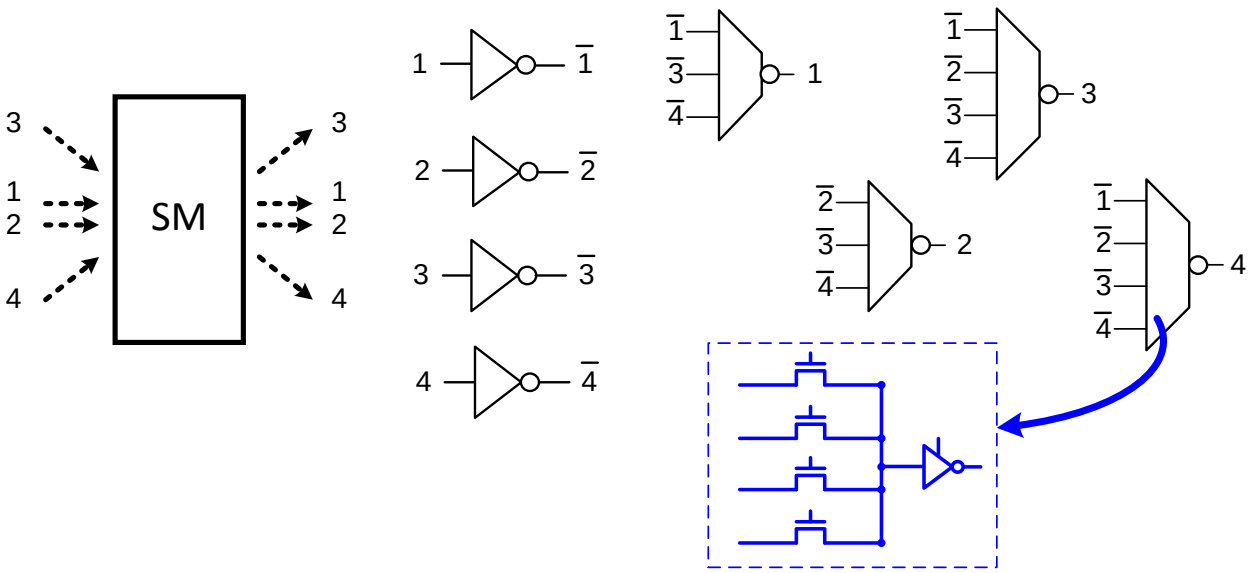


Figure 4.4: Illustration of input-buffer sharing inside a switch matrix.

Generally, not all muxes are propagating from the same input, therefore moving to a single, larger input inverter actually improves performance in most cases (e.g. we can use a 4x input inverter instead of 4 1x inverters, so if the incoming signal only needs to propagate to 1 mux, the mux can switch faster because the 4x inverter offers more drive strength).

In pass-transistor design, the general rule is to not exceed 2 pass-transistors per inverter or buffer [Sutherland99, Rabaey03]. In cases where larger capacitances are involved, such as wires or large gate capacitances, anything more than one pass-transistor per buffer results in performance loss. Based on our simulation results, we do not implement 2 pass-transistor designs unless the 2 pass-transistors are placed in close proximity, within the same macro, such as the mux design from Figure 4.3.

In the non-inverting mux designs, the output inverter drives the input inverter directly, albeit with some wiring parasitic in between (Figure 4.5a). Such scenario of "over-buffering" actually results in performance loss. There are two alternative approaches, illustrated in Figure 4.5b) and c), that both result in one inverter per pass-transistor. Although the two approaches

appear similar, they result in very different functionalities. Generally, the output inverter of the mux is larger than the input inverter. If design b) is applied, its input inverters would all need to be upsized due to its lack of output inverter. However, b) still results in a performance loss: the wire parasitic is generally dominated by capacitance, while the pass-transistor parasitic is generally dominated by resistance. Using simple Elmore delay model, design b) has a large capacitance driven by the pass-transistor, while design c) has the buffer directly driving the large wire capacitance, followed by the pass-transistor. Design c) is clearly the superior design, and matches our simulations.

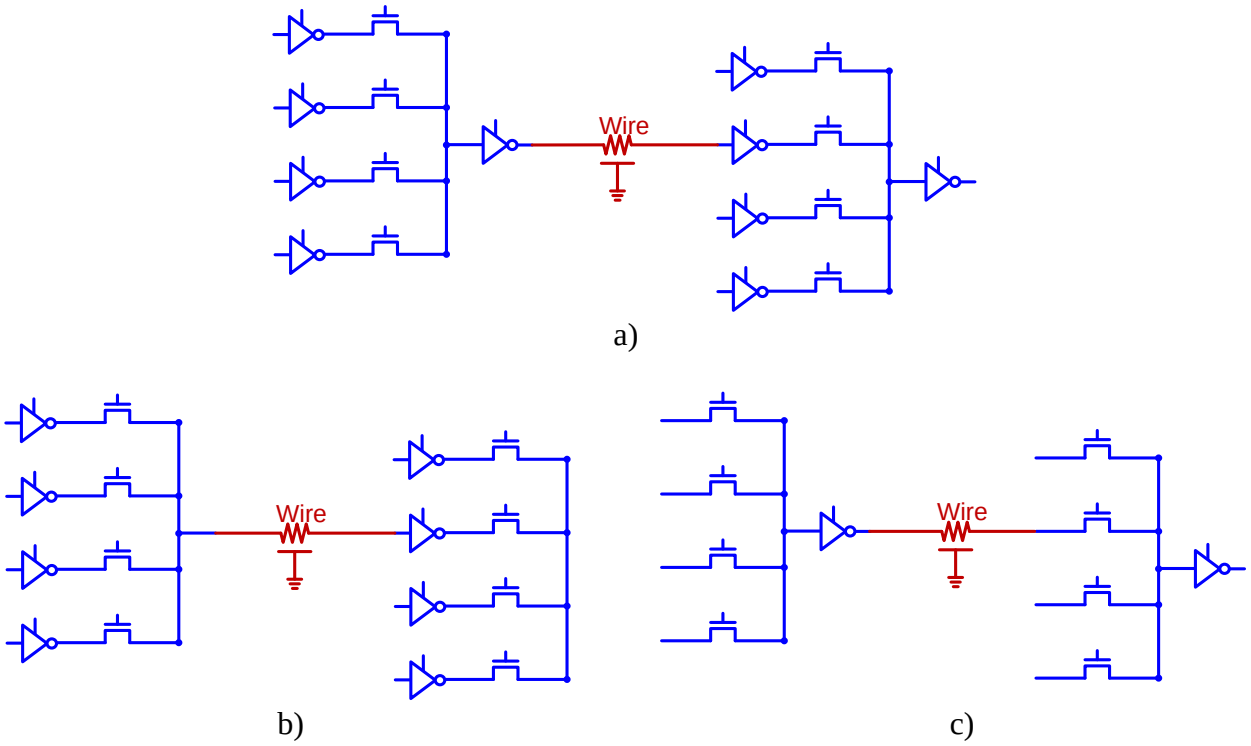


Figure 4.5: Illustration of signal buffer across interconnects of a) a non-inverting mux, b) an inverting mux with input inverters, and c) an inverting mux with output inverters.

Based on our simulations, design c) consistently resulted in faster performance than a) and b) even when the wire capacitance is large. Given its smaller area and faster performance,

we have utilized this inverting-mux design extensively in our design.

The disadvantage of inverting muxes is the alteration of signal polarity. Based on the route of the signal, the routing tool needs to determine the final polarity of the signal when it reaches the input of the CLBs. A selectable inverter is placed at the input of the CLB to invert the signal back if the polarity is flipped. Although the selectable inverter adds an inverter and a mux to the critical path, the overall performance benefits from inverting muxes far outweigh this overhead.

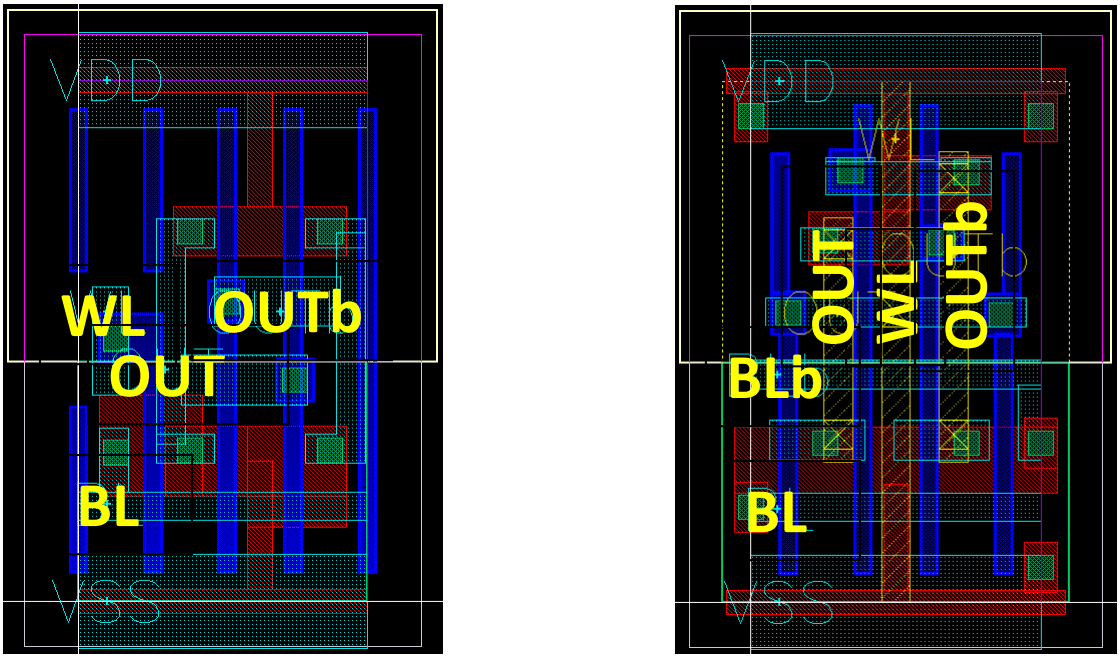
4.4 Designing Configuration Bit-Cells

The interconnect muxes and the CLBs are all configured by bit-cells. In our FPGA, the bit-cells are created as distributed arrays of SRAM cells. Unlike traditional SRAM arrays, the values stored inside the SRAM cells needs not be read out by an address-decoder. Instead, the configuration values stored in the SRAM cells are routed directly to the gate of the muxes. Traditional SRAM cells are not suitable because all bits cannot be accessed simultaneously, and an address decoder is required for reading. A scan-chain approach is another alternative, as adopted in Intel's 32nm DSP CLB [Agarwal10]; this was feasible for 6 CLBs, but is not scalable to larger designs. Because bit-cells are heavily used (millions of bits per FPGA), any area savings from the individual bit-cells can result in a large saving in chip area.

The low-powered FPGA designed in [Ryan10] uses a 5-transistor (5T) SRAM cell to save chip area. We have evaluated this approach against a more traditional 6T SRAM. Although SRAM arrays can benefit from the area savings by switching to 5T [Carlson04, Nalam11], our bit-cell designs are different. Our design not only requires the storage node of the bit-cell to be directly routable during physical design, the bit-cell also needs to conform to the design height

and width requirements of a standard-cell-based design. Therefore, the height of the SRAM cells must be same as that of regular standard-cells, its N-well must be at the same location, and the cell width and all pin locations must be a multiple of the routing pitch.

After careful physical design, the corresponding 5T and 6T SRAM designs are shown in Figure 4.6. We see they occupy the same area, although the 5T SRAM is routable with only the first metal layer (M1). Since the WL needs to travel vertically, a vertical WL on M2 is still required when routing the 5T SRAM, but the 6T design has the vertical WL embedded into its design. Since the stored value (and often times its complement) need to be routed out, the 5T SRAM would need M2 routing to access the *OUT* and *OUTb* pins. The 6T SRAM, on the other hand, has *OUT* and *OUTb* already on the M2 layer. Even though at a first glance, the 5T design occupies less metal layer, the actual metallization usage of the two designs are very similar once all the pins are routed. Under a closer look, we see that the 6T SRAM has only 50% of its *BL* and *BLb* geometries inside its area boundary. That is because its *BL* and *BLb* pins can be shared with its neighbors when the neighboring cells are mirrored across the y-axis. The 5T SRAM cells do not have such symmetry because it lacks a *BLb* signal, therefore it cannot share geometries with its neighbors.



a) b)

Figure 4.6: Physical design of the configuration bit-cells in a) 5T SRAM and b) 6T SRAM.

Overall, the 5T SRAM has not demonstrated an advantage in area or metal usage for our application. Additionally, it requires write-assist to be able to successfully write in a '1' due to its asymmetric design. Designing the write-assist circuitry would require another voltage domain. Since the 6T SRAM poses no area or metal overhead for our application, it was chosen for our bit-cell implementation. It also provides a more stable write, and avoids adding a third voltage domain to our design.

4.5 Power-gating Switch Matrices

The interconnect network in an FPGA can involve long wires, which requires large output buffers and inverters to drive them. These buffers contribute significantly to leakage. To alleviate leakage, power gating is employed in many modern designs to power off the output driver when unused. Power gating is generally achieved by adding a footer transistor that turns

off during power-gating mode. Footer transistors work well to power-gate an entire block, but interconnect signals are long, and long signals lack spatial locality. It is therefore difficult to power-gate an entire block of interconnects, because even one active interconnect buffer would prevent an entire region from being power-gated. Therefore, power-gating circuitry needs to be implemented at a fine-grained level, such as adding footer transistors and power-gating (PG) control to individual buffers (Figure 4.7). The footer transistor can be NMOS or PMOS, as in [Anderson04].

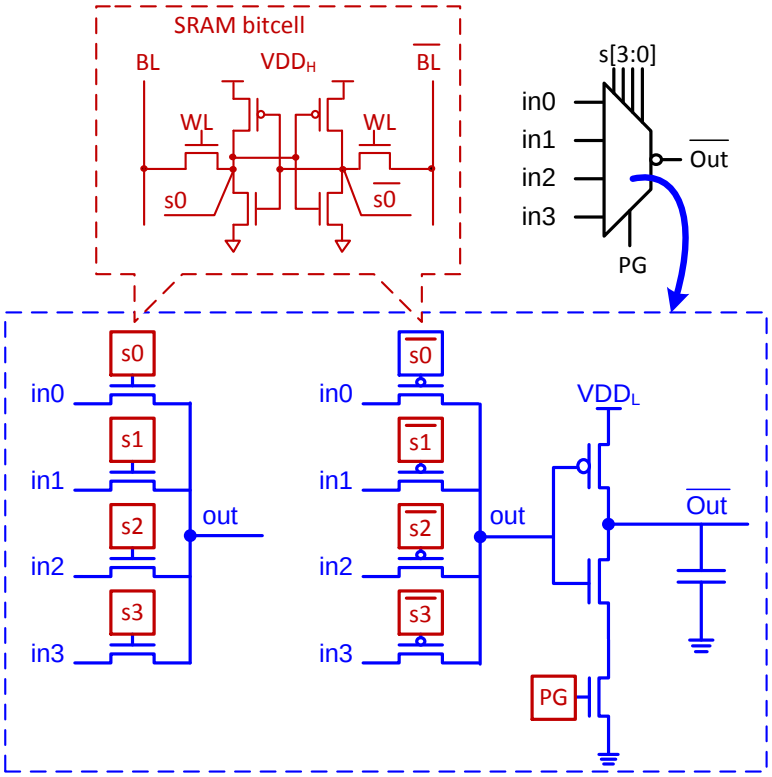


Figure 4.7: A 4-input static mux with output inverter and traditional power gating.

In FPGA multiplexers, the output buffers are already large (often 8x or more). Because the power-gating footer transistors are high-threshold-voltage (HVT) devices, they need to be made larger just to conduct the same amount of current, and because these transistors are in series with the NMOS transistor of the inverter, the footer transistors need to be even larger than

the output inverter to not significantly impact the circuit performance. Such large area overhead from the footer makes this fine-grained implementation impractical.

We suggest a novel power-gating method for interconnects that requires minimal area overhead *and* has no significant performance impact [Wang13b]. The PG signal is used to drive an additional PMOS pass-gate (Figure 4.8). During power gating, the select bits *s*[*N*-1:0] are programmed to '0', and *PG* is programmed to '1' to enable power gating. Since the *PG* signal is from VDD_H domain, this drives the gate input to the inverter to VDD_H, which is above the supply voltage of the inverter, VDD_L. Such voltage drastically reduces the leakage current of the PMOS transistor, thus reducing the leakage of the inverter. The performance penalty of this circuit is minimal, for adding a minimum-sized PMOS pass-gate in parallel to the other *N* pass gates produces very little performance impact on the static mux. However, the drawback of such implementation is that the NMOS transistor of the inverter remains fully on, driving the output of the inverter to '0'. Since this output node is connected to other pass gates, even when those gates are off, leakage current can occur through those gates. In addition, having an output wire driven to '0' does not reduce the coupling capacitance experienced by neighboring wires (Figure 4.8, bottom right).

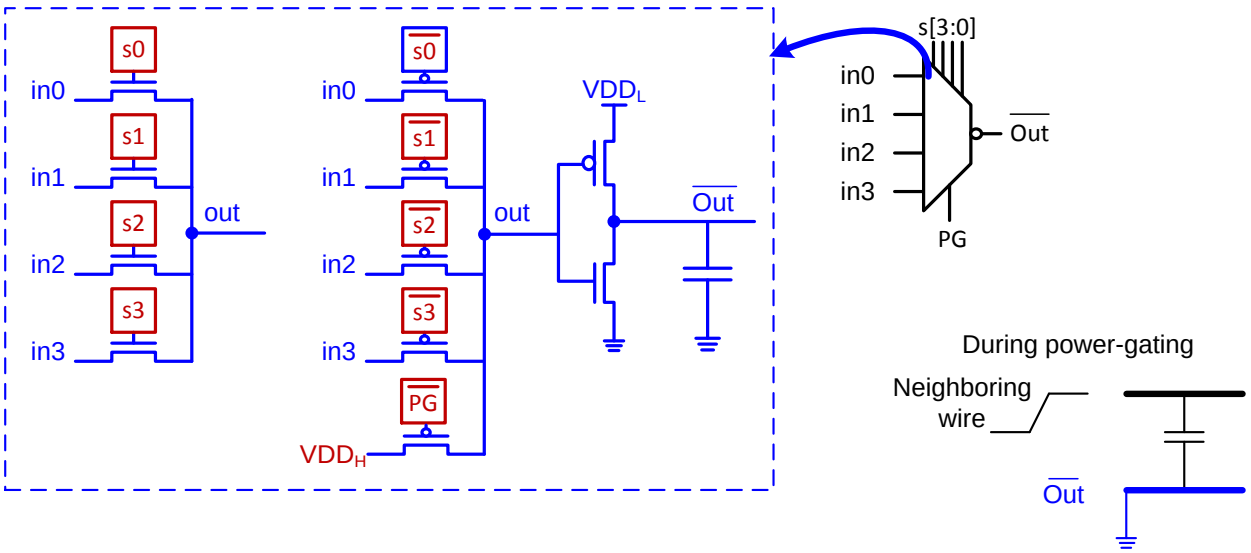


Figure 4.8: A 4-input static mux with output inverter and our proposed power gating.

For further leakage reduction, we propose another static multiplexer that achieves tri-state output during power-gating (Figure 4.9). The inputs to the output inverter is separated into *pmos* and *nmos* inputs, joined by a minimum-size, high-threshold-voltage (HVT) transmission gate as keeper. During power gating, *PG* is '1', the keeper is off, and the *pmos* and *nmos* signals are driven to opposite polarities, '1' and '0', respectively. This not only drastically reduces the leakage current of the PMOS transistor, but also turns off the NMOS transistor. The output therefore enters tri-state mode during power gating. This tri-state buffer will not cause leakage current through other pass gates, and also drastically reduces the coupling capacitance experienced by neighboring wires by forming a capacitive divider.

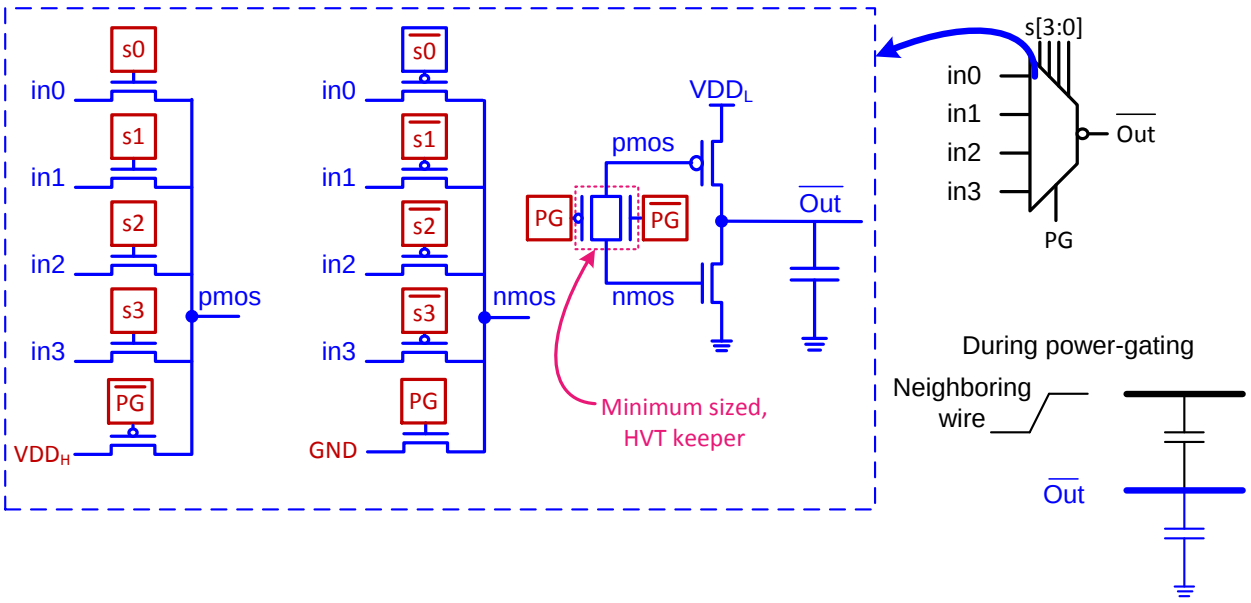


Figure 4.9: A 4-input static mux with output inverter and our proposed, tri-state PG.

When one of the select bits, *s*[*N*-1:0] is on, power-gating is disabled. In this case, the static multiplexer must transmit the selected input to the output inverter rapidly. The NMOS and PMOS transistors of the transmission gate in the static mux are drawn separately for clarity. The NMOS pass gate is fast when conducting a '0', but is unable to rapidly conduct a '1' (it only pulls up to $VDD_H - V_T$). In contrast, the PMOS pass gate is fast when conducting a '1', but is slow when conducting a '0' (it only pulls down to V_T). This invention exploits such driving property of pass gates: the NMOS pass gate is connected to the wire *pmos* to rapidly turn on the PMOS inverter, and the PMOS pass gate is connected to the wire *nmos* to rapidly turn on the NMOS inverter. When one transistor in the output inverter is turning on, the other is not yet fully off (its gate voltage hovers around V_T until the keeper finishes the transition), but the current *difference* between the two transistors is large enough to not impact the performance by more than 5%. Eventually, the keeper connects the *pmos* and *nmos* nets to the same voltage, and the static leakage behavior is no different from a traditional CMOS inverter. This novel power gating circuitry allows the usage of low-threshold-voltage (LVT) inverters at the multiplexer output,

thus compensating for the 5% performance penalty to reach comparable performance as traditional CMOS inverters.

The downside of the design in Figure 4.9 is that the outputs cannot directly drive CMOS gates, because a floating net can lead to large short-circuit currents. Therefore, all nets driven by tri-state buffers must be set as *dont_touch* during the synthesis and physical-design flow. For very long nets that would require buffering, the design from Figure 4.8 is applied. Commercial place-and-route tools can then be used for buffer insertion during physical design.

4.6 Power-On Sequence of the Interconnect Network

This section describes the power-on sequence associated with the dual-VDD interconnect circuit design. It will then discuss the potential power-on issues with the designs proposed in Section 4.5, and suggest 2 viable solutions.

When the bit-cells are not programmed, they can contain arbitrary value. For example, Figure 4.10 shows a pass-transistor mux when both *s0* and *s3* are '1', while *s1*, *s2*, and *PG* are '0'.

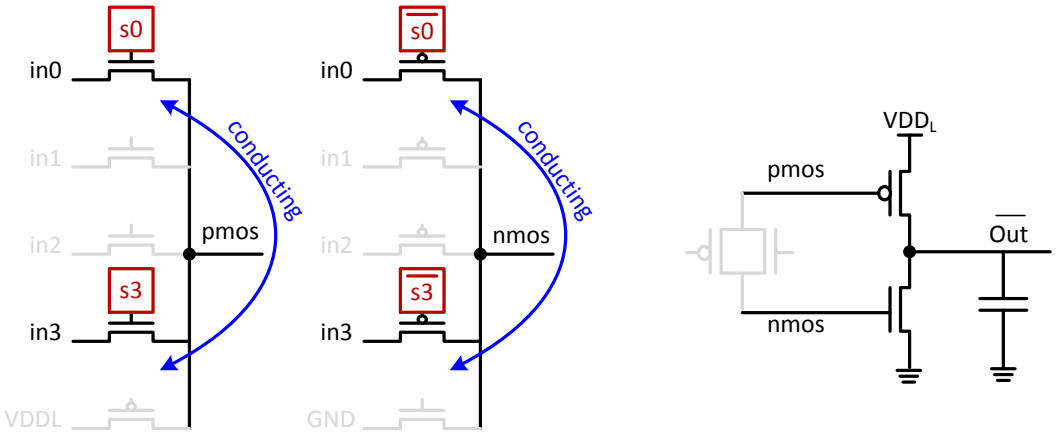


Figure 4.10: An example of an unconfigured mux where *s0* and *s3* are both conducting.

From this example, we see that *in0* and *in3* are now conducting. Since the chip is not yet programmed, we have no control over the voltages at *in0* and *in3*. This can be catastrophic if *in0* and *in3* are at different voltages, causing large current to flow and potentially damaging the chip.

One way to ensure safe operation is to set VDD_L to '0' before the bit-cells are programmed. When VDD_L is '0', all the output inverters and buffers will output '0'. Since the inputs of a mux are driven by the output of other muxes, the inputs are always at '0' when VDD_L is '0'. This way, even when multiple pass-transistors are turned on, there is no current flowing (Figure 4.11).

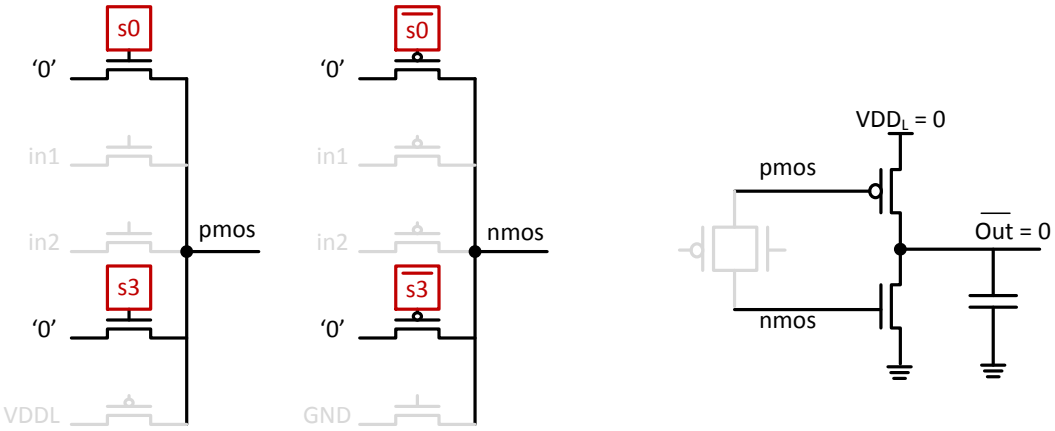


Figure 4.11: An example of an unconfigured mux where VDD_L is '0', no current flows.

A different issue arises when power-gating signals are involved. When *PG* happens to be '1' in the unconfigured state, the proposed designs from Section 4.5 may still cause current to flow. Figure 4.12 shows the potential issue with the proposed design from Figure 4.8. Because *PG* is driven by VDD_H, it can be producing a VDD_H signal while the inputs *in0* through *in3* are producing a '0'. This causes current to flow through the pass-transistors, again causing large current and potential chip failure.

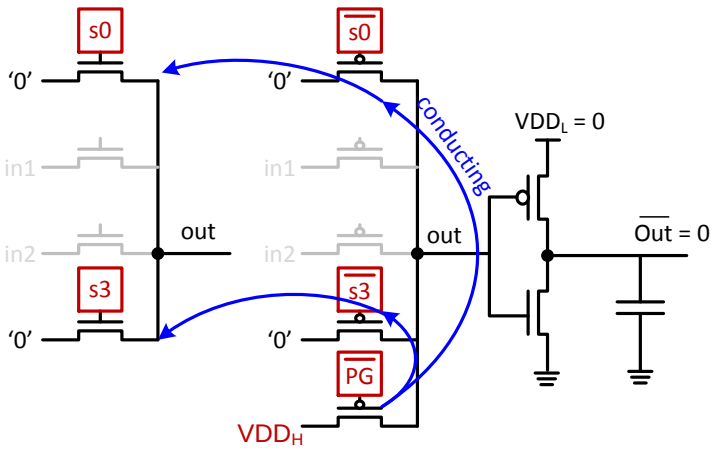


Figure 4.12: An example of an unconfigured mux from Figure 4.8, where *VDD*$_L$ is ‘0’ but *PG* is ‘1’, causing current flow.

Similar issue exists for the proposed design from Figure 4.9. As shown in Figure 4.13, the *PG* signal in *VDD*$_H$ domain can conduct current with the ‘0’ at the inputs of the mux. The *pmos* and *nmos* wires are also shorted when *PG* is on, causing additional conduction paths from *pmos* to *nmos* to ground.

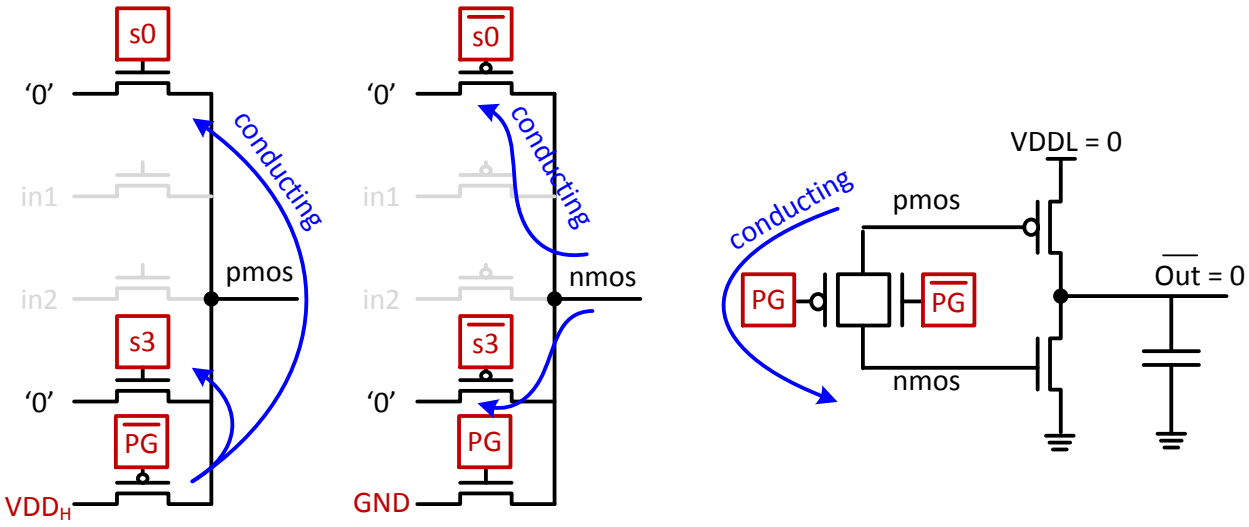


Figure 4.13: An example of an unconfigured mux from Figure 4.9, where *VDD*$_L$ is ‘0’ but *PG* is ‘1’, causing current flow.

Figure 4.14 present a simple fix by tying the source input of the *PG* pass-transistor to VDD_L instead of VDD_H, which ensures all mux inputs to be '0' when VDD_L is '0', causing no current to flow.

The downside of the design in Figure 4.14 from Figure 4.8 and 4.9 is a less effective power-gating during normal operation: since the VDD of the output inverter is VDD_L, power-gating with a signal of VDD_L only presents a V_{GS} of 0 for the PMOS (green, Figure 4.14), instead of a negative V_{GS} of VDD_L–VDD_H, which is more effective at leakage reduction.

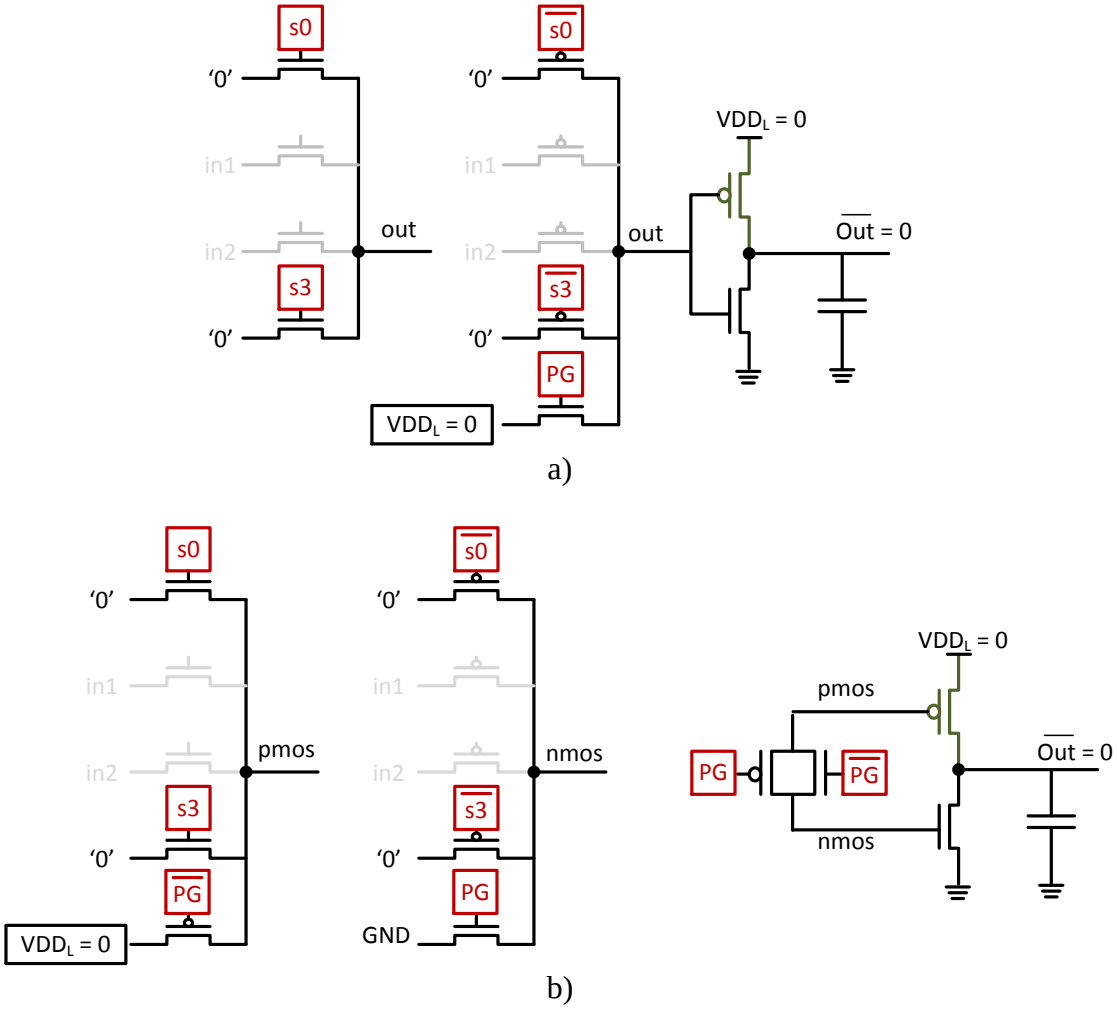


Figure 4.14: Example illustration with an updated design that uses VDD_L signals, applied on a) the design from Figure 4.8 and b) the design from Figure 4.9.

Figure 4.15 presents an alternative fix by tying the source input of the *PG* signal to a 3rd *VDD* domain, $VDD_{H,LATE}$, which remains '0' until after the SRAMs are programmed. This design maintains the benefits of power-gating with a high *VDD*, VDD_H, but without the potential large currents, because $VDD_{H,LATE}$ remains '0' when the chip is unconfigured. Once $VDD_{H,LATE}$ is powered-on, the power-gating behavior is the same as the original proposal from Figures 4.8 and 4.9.

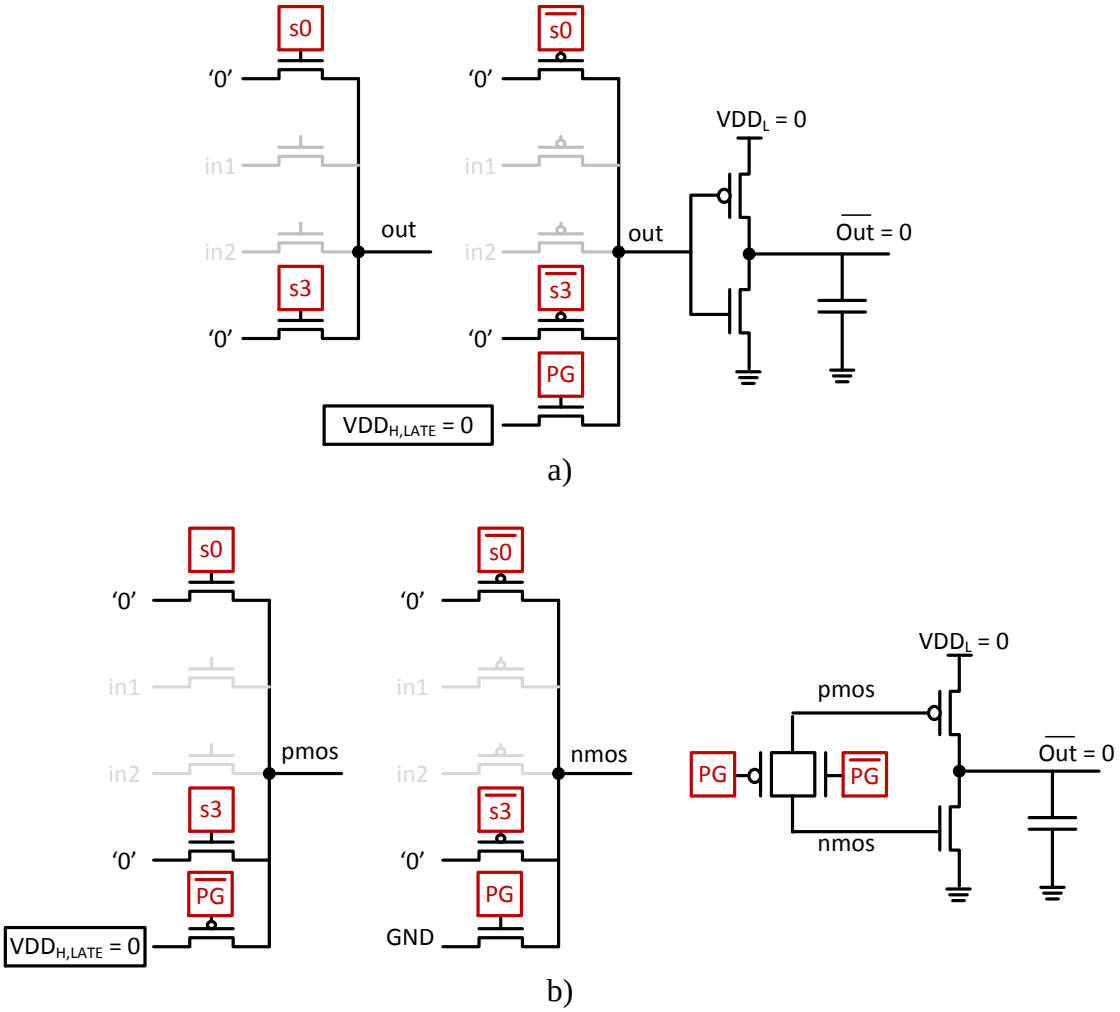


Figure 4.15: Example illustration with an updated design that uses $VDD_{H,LATE}$ signals, applied on a) the design from Figure 4.8 and b) the design from Figure 4.9.

The downside with the above implementation is that a 3rd *VDD* domain is required, which poses a penalty in chip area. In our chip design, we decided to maintain only 2 *VDD* domains due to area constraint. We therefore employed the design from Figure 4.14 for power gating. The chip design details and physical implementation flow are discussed in Chapter V.

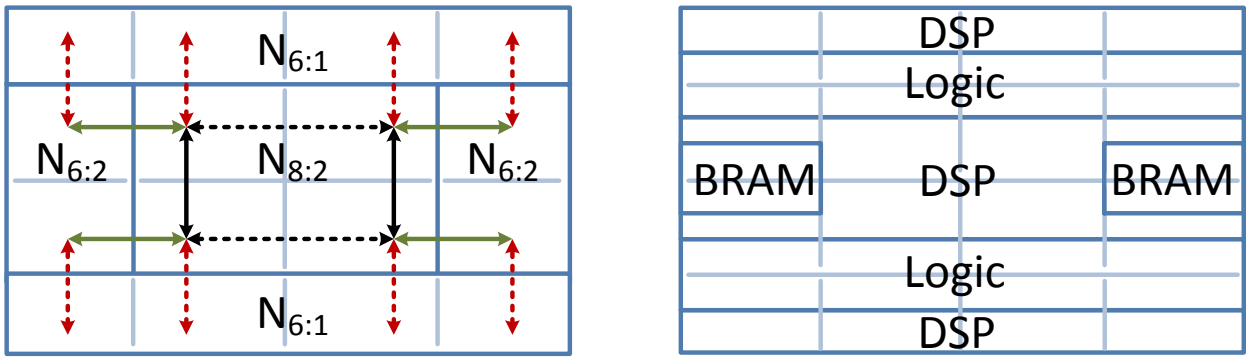
CHAPTER V

Configurable Logic Block Design and Chip Integration

The previous two sections have described the architecture design and circuit-level techniques of the hierarchical interconnect network. To realize a FPGA hardware, the interconnect switch matrices (SMs) are implemented with the configurable logic blocks (CLBs) make a complete macro. Different CLB-SM macros are then integrated to form the complete chip. This section illustrates the design process of the CLBs, and the chip integration process.

5.1 Configurable Logic Blocks for the 2048-LUT FPGA

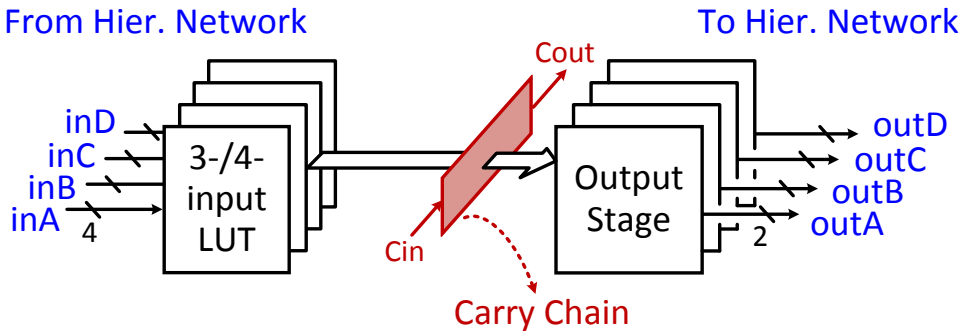
From the architecture discussions in Chapter III, the interconnect resources of the 2048-LUT FPGA are allocated into 16 macros of 128 SM-macros each, shown in Figure 5.1a). Analogously, the resource allocations of the CLBs are arranged in a similar way, shown in Figure 5.1b). The chip has 2048 LUTs divided into 16 CLB macros, and each macro contains a heterogeneous integration of different types of CLBs. To map a variety of designs, the resource allocation is as follows: 1024 LUTs are logic-only, 896 LUTs are configurable for logic or arithmetic functions, and 128 LUTs are used for block RAMs (BRAMs). These LUTs are grouped into configurable logic blocks (CLBs): 4 logic-only LUTs form a Logic CLB, 4 logic/arithmetic LUTs form a DSP CLB, and 8 BRAM LUTs form a 1 Kb, dual-port BRAM CLB. Since some DSP and BRAM operations require many input bits, this grouping allows input-sharing between the LUTs within a CLB.



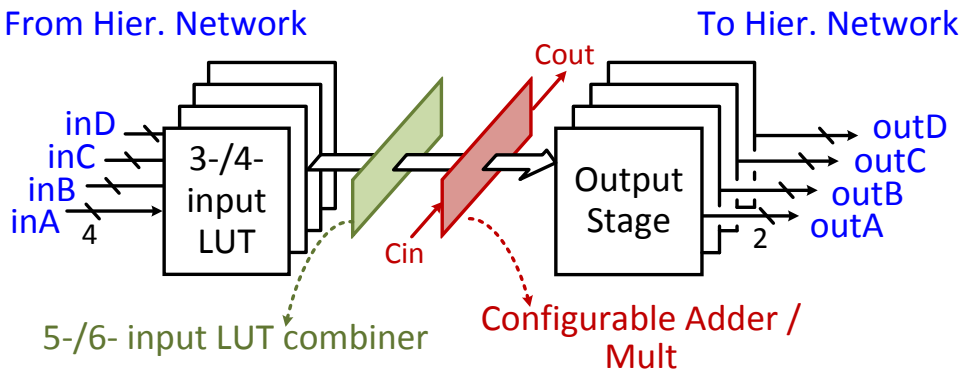
a) b)

Figure 5.1: Resource allocation for a) interconnects and b) CLBs.

Four LUTs are combined with intermediate arithmetic blocks to form a Logic or DSP CLB. Each Logic CLB is composed of four 4-input LUTs, a carry chain, and 4 output stages with selectable flip-flops (Figure 5.2a). Each LUT is configurable as one 4-input LUT, or two 3-input LUTs with up to 4 unique inputs. The carry chain supports 4b additions where Propagate and Generate are driven from LUTs. Since each output stage support two outputs, the Logic CLB is especially useful when two outputs per bit are required, such as in 3:2 compressors.



a)



b)

Figure 5.2: Block diagram of a) a Logic CLB and b) a DSP CLB.

The DSP CLBs are a more flexible for intensive arithmetic computing demands. Each DSP CLB is composed of four 4-input LUTs, a LUT combiner, a partial product generator, a configurable adder tree, and 4 parallel output stages (Figure 5.2b). The detailed block diagram is shown in Figure 5.3. Each 4-LUT is able to perform one 4-input logic, two 3-input logic by sharing two common inputs (same as Logic CLB), or functions with 5 or 6 inputs by combining LUTs. Two 4b ripple-carry adders can perform two separate 4b, one 8b, or one 3-operand 4b addition with the support of 3:2 compressors (built from LUTs). The Wallace-tree multiplier reuses the adder cells, and uses dedicated partial-product generators. Overall, the CLB has the flexibility to support 10 operating modes, which includes (i) random logic with 3 to 6 inputs; (ii) 4 to 8b addition/subtraction for 2 to 3 operands; and (iii) 4b×4b signed/unsigned multiplication. To achieve this degree of configurability, the synthesized CLB has 50 gates on its critical path (shaded in Figure 5.3), amounting to a 1.1ns delay.

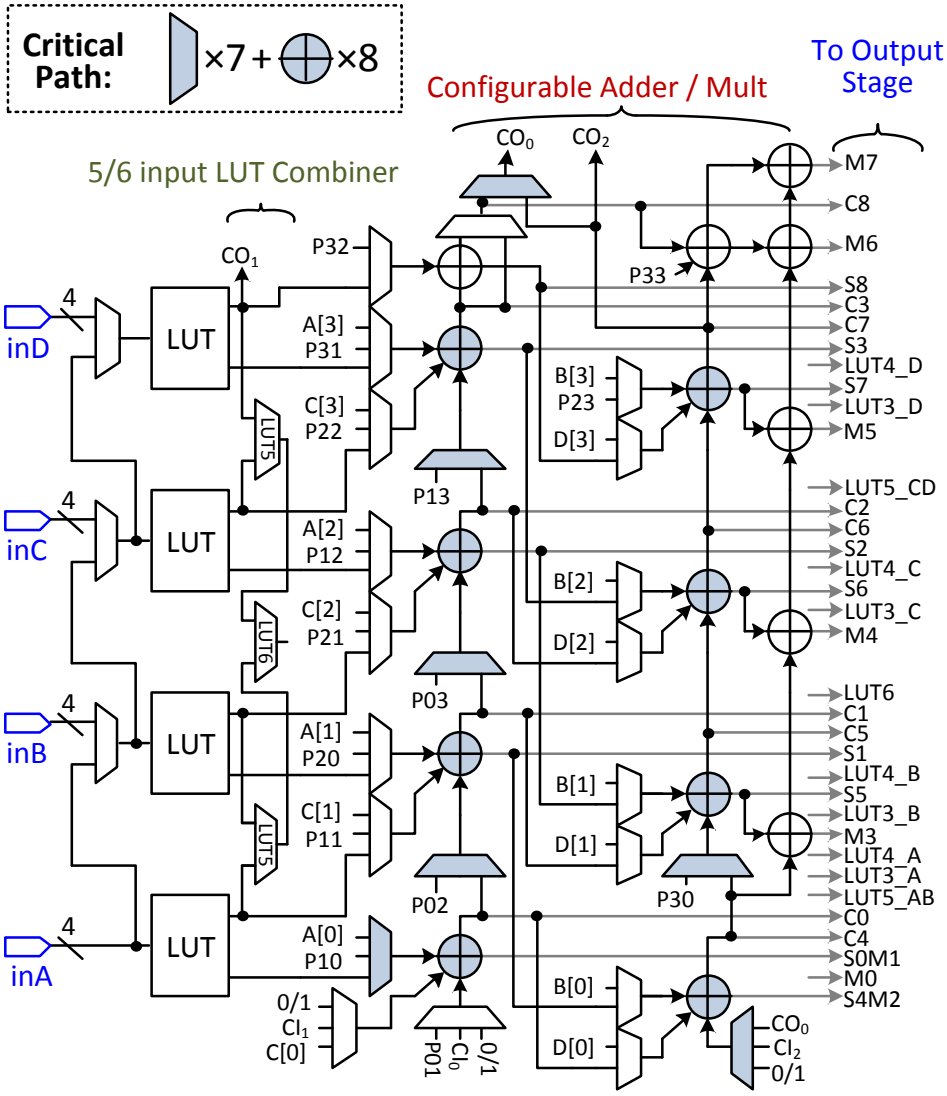


Figure 5.3: Block diagram of a) a Logic CLB and b) a DSP CLB.

The 1Kb block-RAM (BRAM) is a custom dual-port memory with two 7-bit addresses (*addr0* and *addr1*), an 8-bit data (*din*) and 8-bit write mask (*wr_mask*), and two 8-bit outputs (*dout0* and *dout1*). The memory modes are selectable by a 2-bit control signal (*wr_mode*), and in certain cases, *wr_mask[7]* bit is also used for configuration. The BRAM configurations are shown in Figure 5.4.

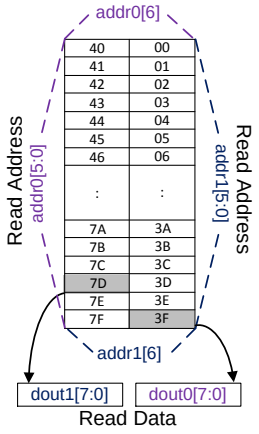
Mode 00-0

Dual 8-bit read

Configuration: wr_mode[1:0] = 00, wr_mask[7] = 0

Write Address: Not Supported
Write Data: Not Supported
Write Mask: Not Supported

Read Address0: addr0[6:0]
Read Data0: dout0 = Read(addr0[6:0])
Read Address1: addr1[6:0]
Read Data1: dout1 = Read(addr1[6:0])



a)

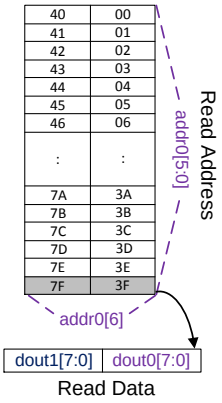
Mode 00-1

16-bit read

Configuration: wr_mode[1:0] = 00, wr_mask[7] = 1

Write Address: Not Supported
Write Data: Not Supported
Write Mask: Not Supported

Read Address0: addr0[6:0]
Read Data0: dout0 = Read(addr0[6:0])
Read Data1: dout1 = Read({ 1, addr0[5:0] })



b)

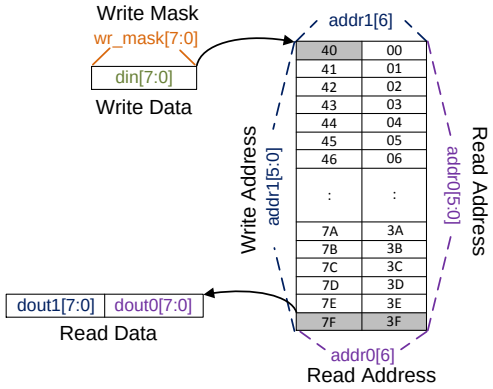
Mode 01

8-bit write with 1-bit precision write mask, 16-bit read

Configuration: wr_mode[1:0] = 01

Write Address: addr1[6:0]
Write Data: din[7:0]
Write Mask: wr_mask[7:0]

Read Address0: addr0[6:0]
Read Data0: dout0 = Read(addr0[6:0])
Read Data1: dout1 = Read({ 1, addr0[5:0] })



c)

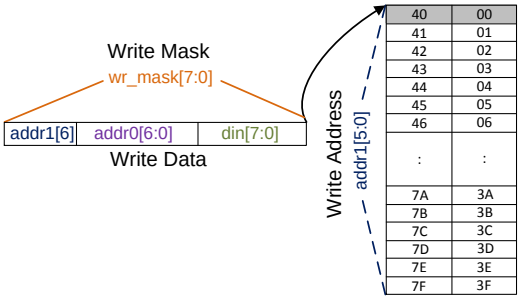
Mode 11

16-bit write with 2-bit-precision write-mask

Configuration: wr_mode[1:0] = 11

Write Address: addr1[5:0]
Write Data: { addr1[6], addr0[6:0], din[7:0] }
Write Mask: { wr_mask[7:0] }, each mask-bit controls 2 data bits

Read Address: Not Supported
Read Data: Not Supported



d)

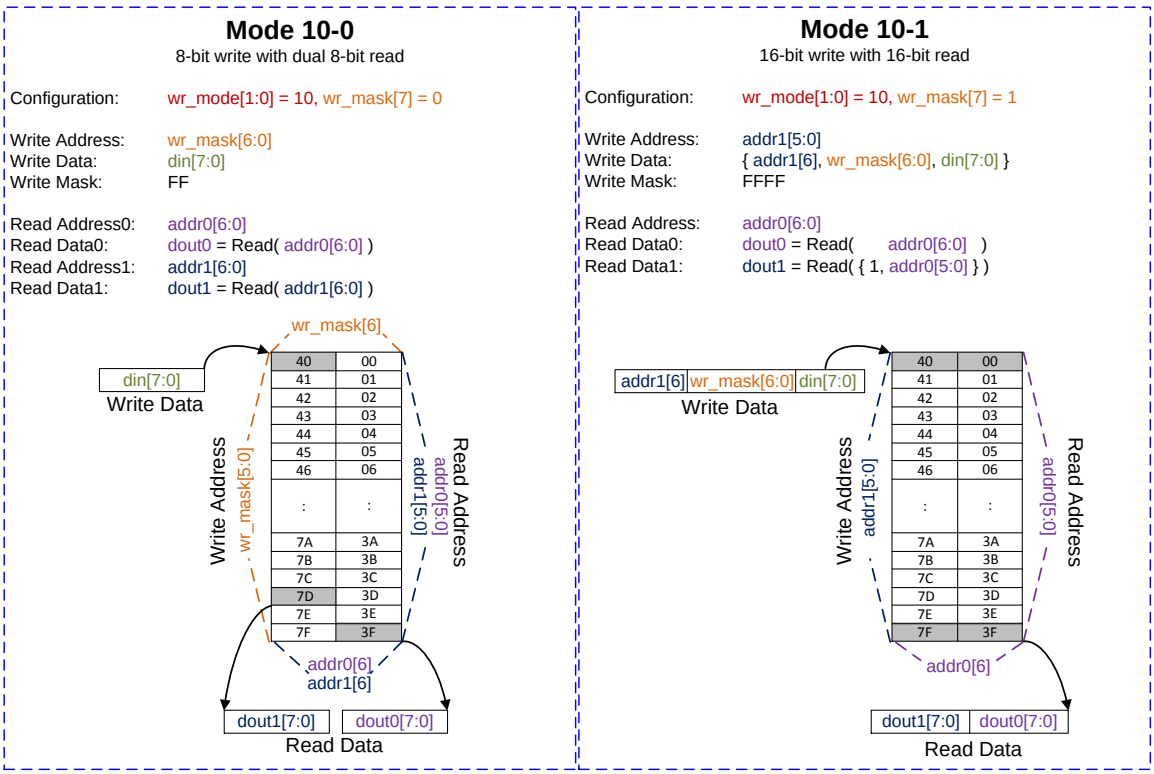


Figure 5.4: The 6 BRAM modes: a) dual 8-bit read, b) 16-bit read, c) 8-bit masked write, 16-bit read, d) 16-bit masked write, e) 8-bit write, dual 8-bit read, and f) 16-bit write, 16-bit read.

In certain configurations, this BRAM is even configurable as a tri-port memory, with one 8-bit write and two independent 8-bit reads (Figure 5.4e). The mode selections are propagated to the interconnect network as regular signals, and can be changed during chip operation. The detailed diagrams in Figure 5.4 also illustrates the actual memory locations that each mode accesses. No explicit write-enable signals exist; write is disabled by toggling *wr_mode* back to '00'.

Although the BRAM supports many user-programmable modes, the actual implementation is done without too much hardware overhead by utilizing hardware sharing and optimized control design. The physical implementation of the BRAM is realized in two 64x8b arrays for reconfigurability into 128x8b or 64x16b modes. The memory array is implemented

using regular flip-flops, because 1 Kb is not large enough for compiled register-file or SRAM to achieve sufficient area savings. The write-logic architecture is highlighted in Figure 5.5, and the read-logic architecture is highlighted in Figure 5.6.

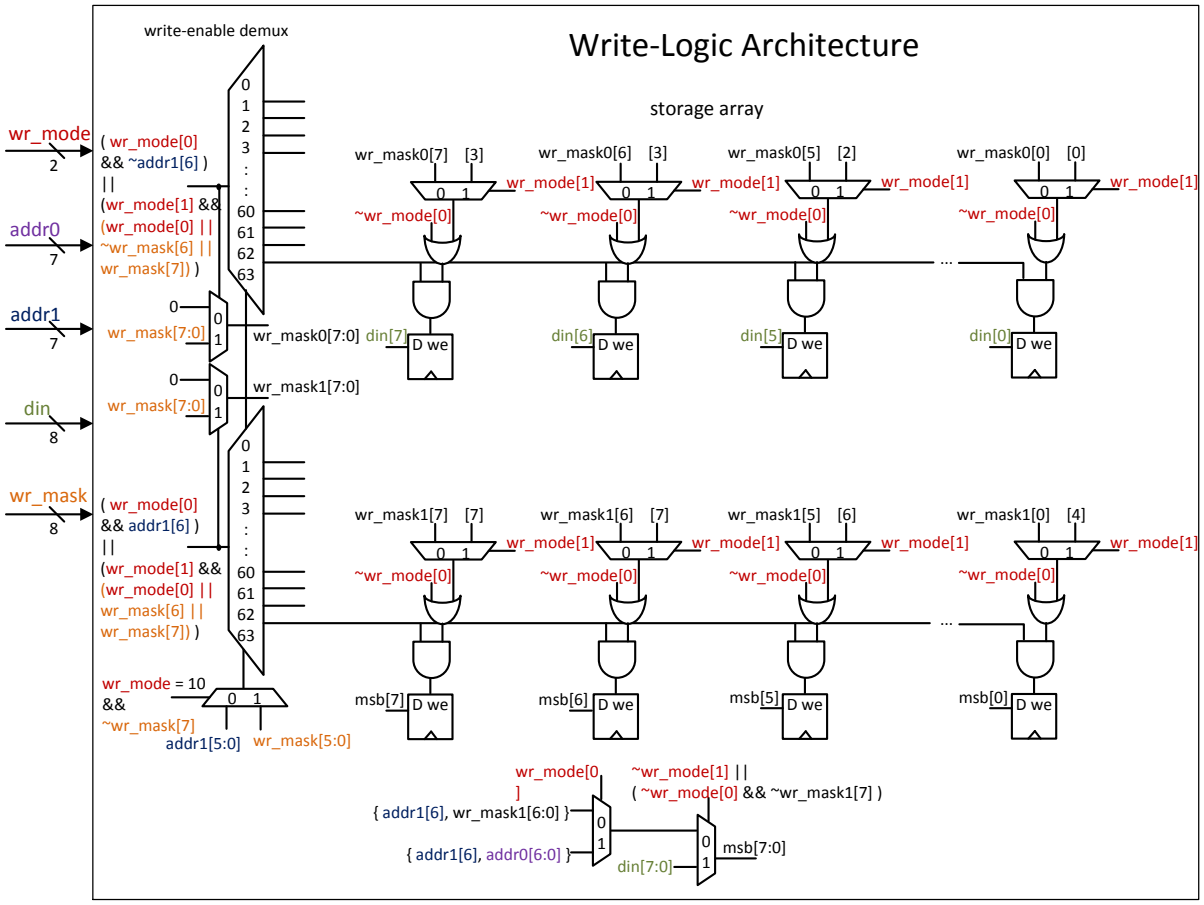


Figure 5.5: Write-logic architecture of the 1Kb reconfigurable dual-port BRAM.

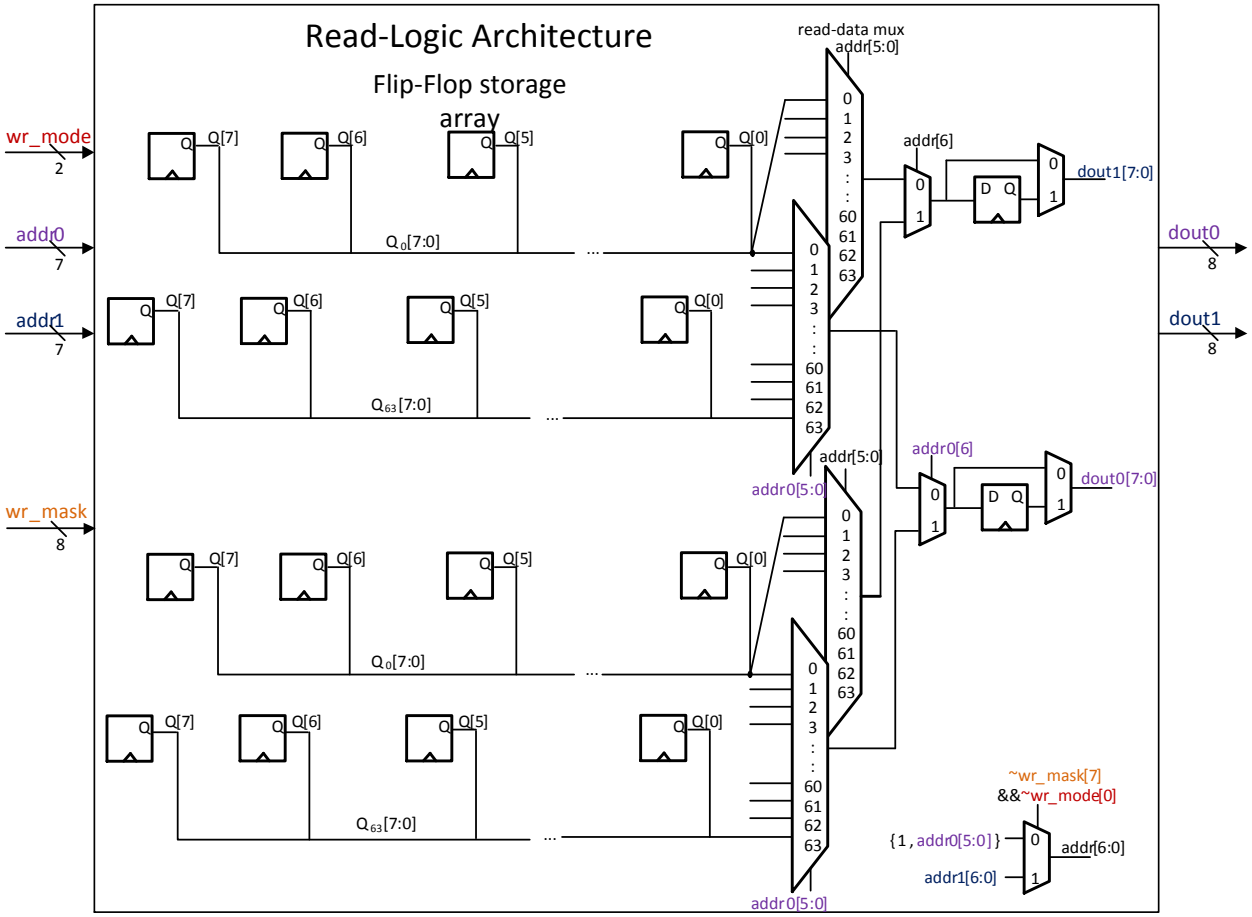


Figure 5.6: Read-logic architecture of the 1Kb reconfigurable dual-port BRAM.

5.2 Macro-based Chip Integration for the 2048-LUT FPGA

A macro-based design approach is used for efficient integration. Each CLB is integrated with its switch matrices into a complete macro. From Figure 5.1, we see that the chip is divided into 3 types of interconnects ($N_{6:1}$, $N_{6:2}$, and $N_{8:2}$) and 3 types of CLBs (Logic, DSP, and BRAM). Based on the location of each CLB, it will be integrated with the corresponding interconnect SM, making a total of 7 combinations: $N_{8:2}$ Logic, $N_{8:2}$ DSP, $N_{6:2}$ Logic, $N_{6:2}$ DSP, $N_{6:2}$ BRAM, $N_{6:1}$ Logic, and $N_{6:1}$ DSP.

Both the CLBs and the interconnect SMs require configuration at power-up. Scan-chains

are the easiest approach, but the area overhead from scan is very high because we need a very large number of configuration bits (320,000 in this "small" FPGA). We therefore implemented a custom 6T SRAM bit-cell (BC), and an array of BCs is placed inside each CLB, alongside the CLB and SM blocks, and the output of each BC is routed directly to the SM and CLB for configuration. The SRAM-based BCs achieve a 5x area reduction over scan flip-flops. The schematic diagram is shown in Figure 5.7.

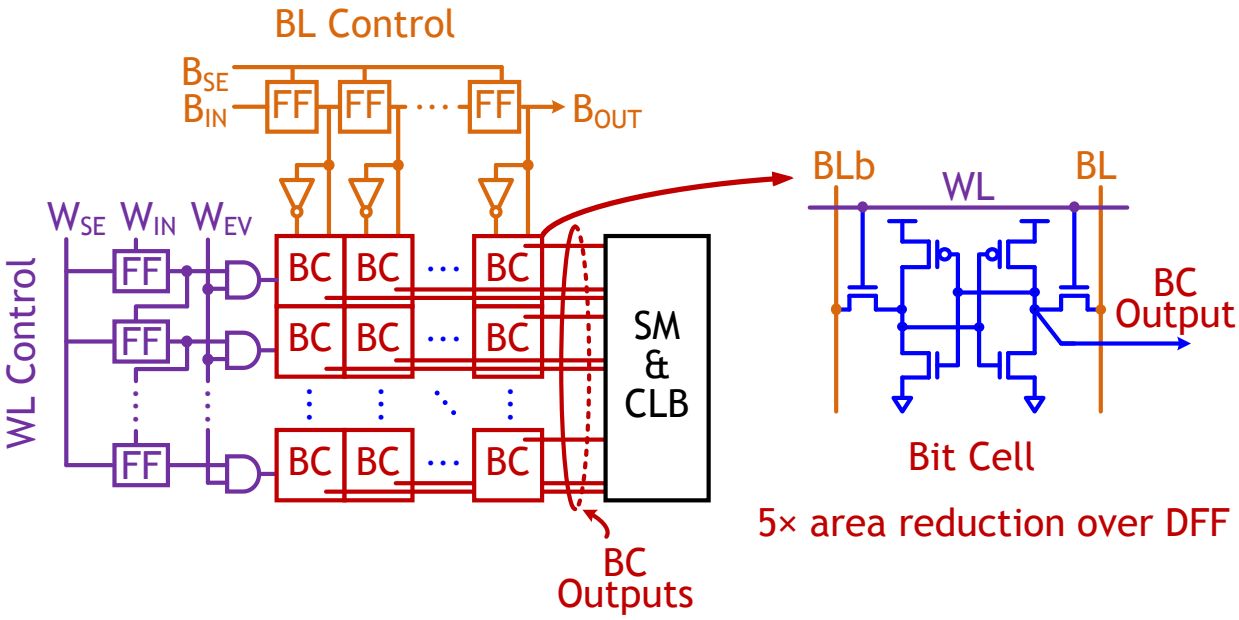


Figure 5.7: Design of a bit-cell (BC) array with its bit-line (BL) and word-line (WL) controls.

Scan-chains are used to implement the word-line (WL) and bit-line (BL) controls. The bit-line control signals (BL and BLb) of each CLB are generated from a BL scan chain, which is 14b per CLB. Once all configuration bits are shifted into the scan chain, B_{SE} stops the shifting and activates the word-line (WL) control circuit. The WL scan chain shifts a single '1' down the scan chain, and its writing operation has two phases: (i) write-scan-enable (W_{SE}) shifts the WL scan chain forward for one clock cycle, propagating the '1' to the next row; (ii) write-evaluate (W_{EV}) then changes to '1', which is AND'ed with the WL scan chain values to enable just one

row of BCs for programming. The selected row of BCs is then programmed with the value from the BL scan chain. W_{EV} is then de-asserted to stop the writing process. We then shift in the next row of configuration bits through the BL scan chain, and the WL scan chain advances by another row and repeats the writing process. This scan-chain based writing process avoids the use of an area-inefficient address decoder, utilizing that fact that no random-access read and writes are required for configuration bits.

The physical integration of a complete CLB-SM macro is shown in Figure 5.8. A CLB is integrated with 4 SM-macros. Each SM macro contains 8 SMs in this case ($N_{6:2}$). The WL controls run down the center, and BL control runs down the bottom of the BC arrays.

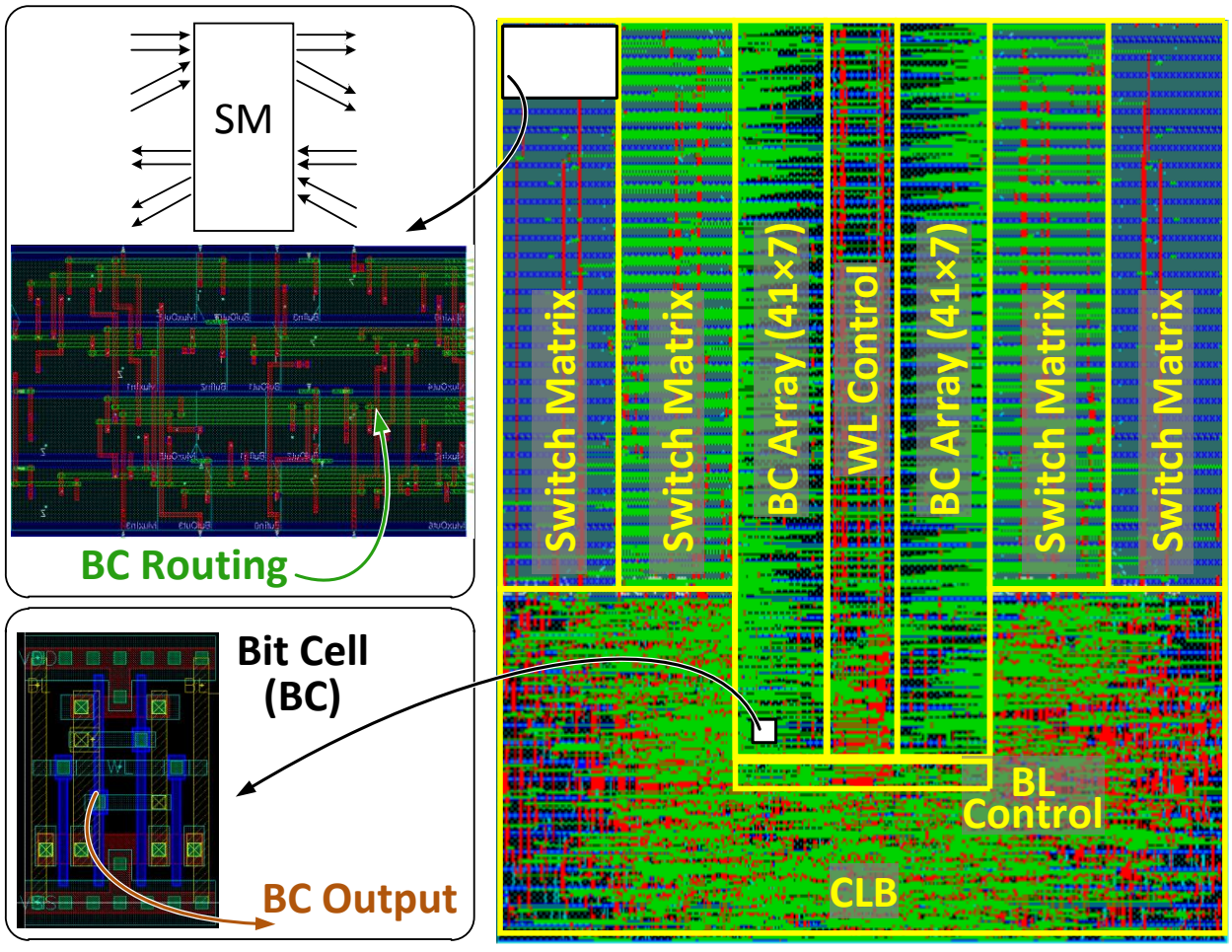


Figure 5.8: Layout of a CLB-SM macro with 4 SMs, a BC array, and BL and WL controls.

From the physical design in Figure 5.8, we see that the 2 SM macros reside on each side of the BC array. Therefore the configuration bits must route from the BC output, through the SM macro on the inside, to reach the SM macro on the outside. To not use additional metal layer, each SM is designed to use no more than half of the resources on metal M3, so the BC routing for the other SM can route in the other half (Figure 5.8, upper left, green).

Although commercial place-and-route tools are used for physical design, the SM macros, the BL and WL controls, and the BC arrays are placed manually, using scripts and hand-placement. The manually-placed design is then routed using the automated router. In the case of the SM macros, the routes are then altered manually to occupy no more than half of the M3 resources. The physical designs for the CLBs are synthesized and placed-and-routed using only automated, commercial flows. Overall, the CLB-SM macros are constructed with 98-99% layout density.

Intermediate macros are constructed by integrating 32 CLBs (128 LUTs), and 16 intermediate macros are integrated in the top level. The top-level of the chip floorplan is shown in Figure 5.9, labeled into 7 different types of SM-CLB combinations. Because of the x- and y-symmetry of the top-level design, only 4 of the 16 macros need to be designed. Additionally, 2 out of the 4 macros are identical (both are Slice L $N_{6:1}$), so only 3 unique macros are designed, and are then replicated in top-level. The intermediate macros are not shown, but we can directly see the CLBs placed on the floorplan. The red areas inside the macros are the regions occupied by switch matrices.

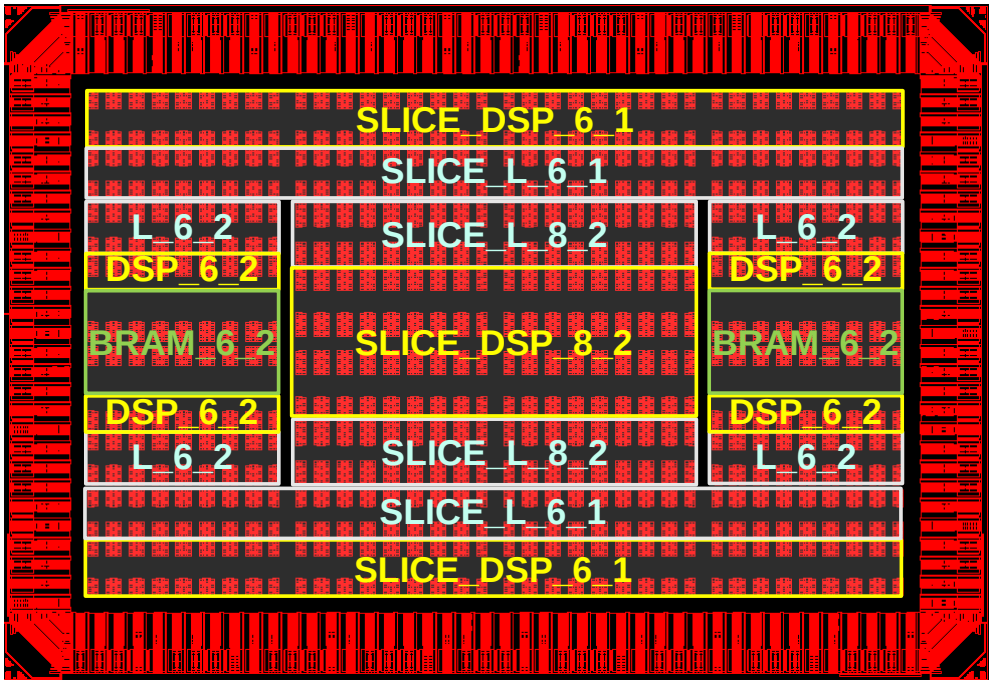


Figure 5.9: Top-level layout floorplan of the 2048-LUT FPGA with 512 CLBs.

Inside the FPGA core, the layout area occupied interconnects is 51%. Compared to commercial 2D-mesh FPGAs, where interconnects occupy 80% of the total chip area or more [Lin07], we have achieved a 3-4x reduction in interconnects area for a fixed logic area. The interconnect-to-logic ratio is reduced from 4:1 to 1:1 (Figure 5.10).

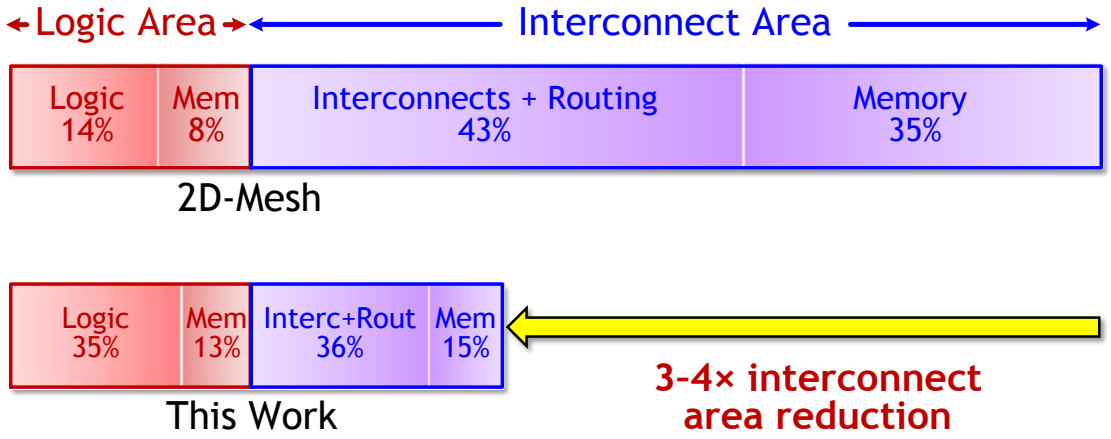


Figure 5.10: Area impact of our work: a 1:1 logic-to-interconnect ratio.

Unlike heuristics in previous attempts that resulted in limited interconnect connectivity, this area reduction is achieved without sacrificing connectivity, thus usable logic density is preserved. Therefore, this 2.5x reduction in chip area actually translates to 2.5x higher logic density.

5.3 Fine-Grained CLBs for the 16K-LUT FPGA

Our initial 2048-LUT FPGA has demonstrated significant area reduction from commercial FPGAs. Building from its initial design, the new 16K-LUT FPGA expands the logic capability by 10x, and it is designed to be a heterogeneous integration of CLBs with different granularities. The chip incorporates 3 granularities of CLBs: fine-grained reconfigurable blocks such as LUTs, simple arithmetics, and distributed memories, medium-grain blocks such as DSP accelerators and block RAMs (Section 5.4), and coarse-grain accelerators for targeted applications, in this case a 64-8192 point Fast-Fourier-Transform (FFT) processor and a 16-core communications signal processor (Section 5.5).

The fine-grained CLBs are mainly composed of LUTs and its surrounding logic. In our prior design, we utilized 4-input LUTs with 4 LUTs per CLB, similar to the structure in Xilinx Virtex-4. We then add our own improvements to the Logic CLB, and even create a DSP CLB that allows 8-bit additions and multiplications. The Logic and DSP CLB designs where efficient, but they have two shortcomings in mapping real-life designs. First, the CLBs that we designed were not compatible with the CLBs from commercial FPGAs, so when the users perform logic synthesis to map their designs, they cannot use commercial FPGA synthesis tools. In Chapter VI, we see that we first created a custom standard-cell library for our LUTs, enabling the user to synthesize designs for our FPGA using commercial ASIC synthesis tools. Although this

approach works, FPGA synthesis tools in general create much better quality-of-results for FPGA mapping. Second, the critical-path of our CLBs is longer than that of commercial FPGAs, which is caused by having too many configuration modes, causing too much logic to reside on the critical path. The CLBs need to be modified so each CLB's most commonly-used features need to be executed as fast as possible.

The new fine-grained CLBs are designed to target these issues. The CLBs are made to be logically-compatible with the newest CLB designs from Xilinx Virtex-6 and Virtex-7, and is also backward-compatible with Virtex-5. We have added our own improvements to the CLBs, but we were careful to maintain the compatibility with commercial CLBs. This not only allows our FPGA to be synthesized with commercial synthesis tools (e.g. Synplify Pro or Xilinx ISE), it also allows a true apple-to-apple comparison of performance and power between our FPGAs and commercial FPGAs, while mapping and executing the same user-design.

To avoid having excessive configuration modes that slow down the CLB, we have separated the logic CLBs from DSP features such as multipliers and 3-input adders, which are now built into dedicated DSP (non-LUT) CLBs. The logic CLBs now consist of Slice L and Slice M, each support four 6-input LUTs per CLB (reconfigurable into dual 5-input LUTs), a fast ripple-carry-chain, and 8 configurable flip-flops. These CLBs are built efficiently for combinational logic and flip-flops; each LUT is able to propagate its out directly to an output of the CLB (into the interconnect network) without passing other logic. The detailed micro-architecture of a Slice L/M CLB is shown in Figure 5.11.

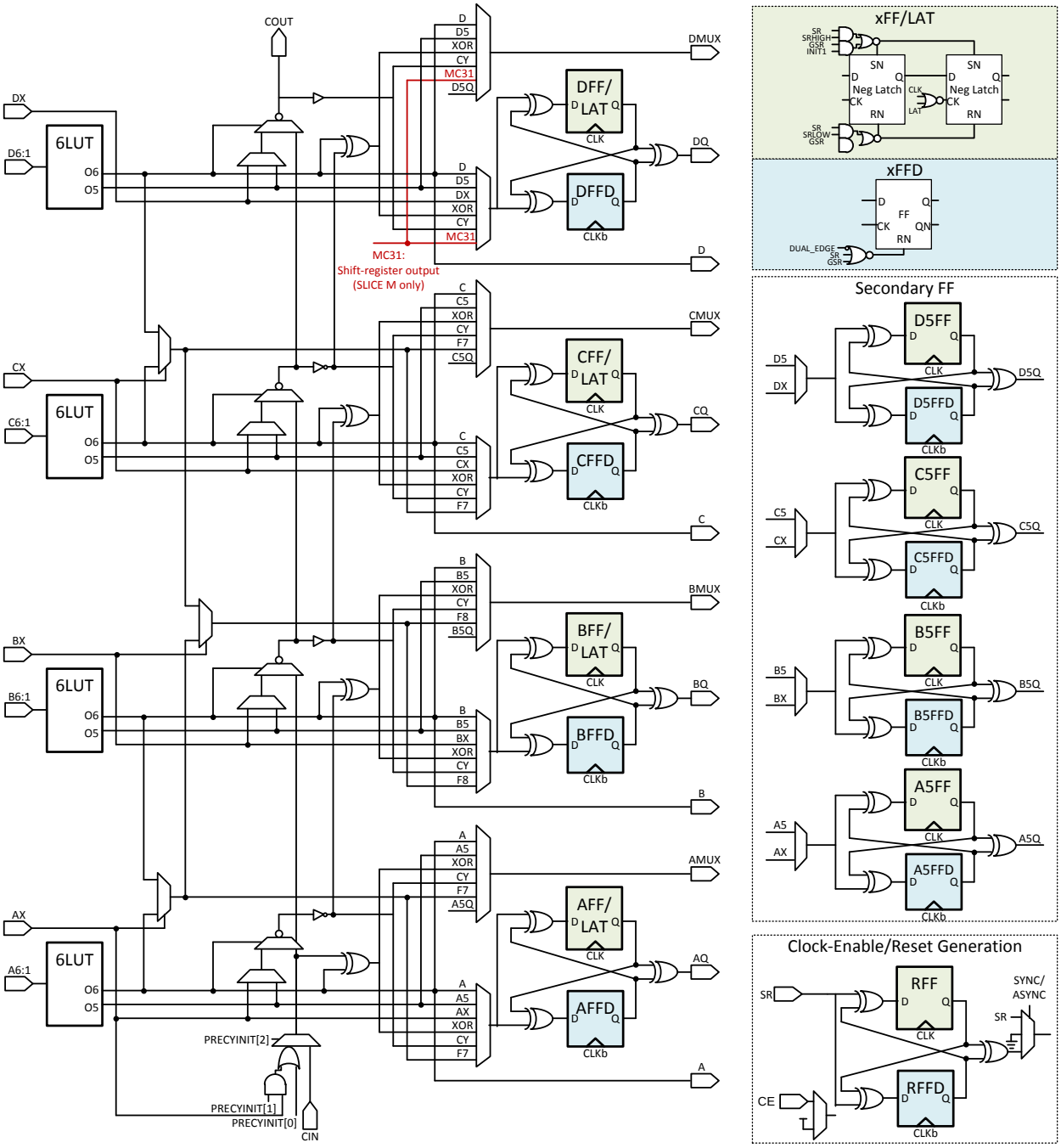


Figure 5.11: Micro-architecture of a Slice L/M CLB with dual-edged clocking.

Figure 5.11 illustrates the programmable dual-edged flip-flops (FFs) used in our CLBs, designed to reduce clocking power, and effectively implement divide-by-two clocks without requiring a separate clock domain. The dual-edge flip-flop is designed using only standard-cell logic to avoid having custom flip-flops as in [Pedram98], which would have to be characterized for timing and performance for our process. By having two clocks running on inverted edges of the clocks, combined with 3 XOR gates, we can efficiently build a dual-edged flip-flop. During synthesis and physical design, commercial CAD tools are able to compute the timing relationships and report timing for this dual flip-flop. To support configurability, the dual-edged flip-flop needs to support more features, such as set/reset for global reset, set/reset for local reset, latch mode, and single-edge mode. These features are implemented, and are shown in the inset on the top-right of Figure 5.11, separated by *xFF/LAT* for the "master" flip-flop running on the positive-edge of the clock, and *xFFD* for the "slave" flip-flop running on the negative-edge of the clock. The "slave" flip-flop always resets to '0' whenever global or local reset is assert. It also remains '0' in single-edge mode. The "master" flip-flop is configurable to set or reset to a '1' or a '0', respectively, and independently configurable for global or local resets. Since the "master" flip-flop is built with two latches, latch mode is supported by simply making the slave latch transparent.

A set of 4 secondary flip-flops are implemented, configurable to store the 5-input LUT outputs (*A5 – D5*) or the auxiliary signals (*AX – DX*). Another dual-edge flip-flop is implemented for reconfigurable synchronous or asynchronous reset (*SYNC/ASYNC*) based on the incoming *SR* signal. Clock-enable signal, *CE*, is configured to drive two clock-gating latches (not shown) that propagate *CLK* and *CLKb* throughout the CLB.

The 6-input LUTs are used to drive the high-speed carry-chain. Unlike our previous CLB

design, no reconfigurable logic is added to the critical path of the carry-chain. Although the carry chain is used to drive CLB outputs and flip-flops, inverters and buffers are inserted to separate the critical-path from its external loading. Each carry stage also inverts the polarity of the carry for performance improvements. With these design considerations, each CLB is able to propagate its carry in under 100ps. The carry chain is also able to accept auxiliary inputs (*AX* – *DX*), which is especially useful for adding partial products in a multiplier. The auxiliary inputs (*AX* – *CX*) also function as select signals for merging 6-input LUTs into 7- or 8- input LUTs.

The Slice L LUTs each contain 64 bit-cells of configuration, useful for building glue logic or read-only memories (ROMs). Slice M CLBs add distributed memory features – each CLB can function as a 256-bit single- or dual-port memory, or a 128-bit shift register, distributed across 4 LUTs inside the CLB. Therefore the LUTs are selectable to propagate their inputs from bit-cells, memory latches, or shift registers (Figure 5.12). Our CLB design even allows each of the 4 LUTs in Slice M to function independently as LUT, memory, or shift register. Similar to commercial FPGAs, the 8-bit write address of Slice M is implemented using other inputs, namely {*BX, CX, D*[6:1]}. Therefore each CLB only has one write port. Since the read ports are implemented through LUT logic, each CLB can have 4 independent read ports, each with 6 bits of address, or use 5-input LUTs to split to 5-bit addresses with 2-bit outputs per LUT. Read address can also be combined across neighboring LUTs into forming two memories with 7-bit read address, or one single memory with 8-bit read address. More memory examples are illustrated in [XilinxV6CLB12], all of which are supported by our CLB design. Distributed memories allow for asynchronous read-access, because the read operation is performed using LUTs. If synchronous read is desired, the user can enable the output flip-flops of the CLB.

To reduce area usage and share resources, write-address decoder is separated into 3

stages. The first stage decodes 3 bits *WA1*, *WA2*, and *WA3* (Figure 5.12, left), requiring 8 logic combinations total. Another parallel stage decodes *WA4*, *WA5*, and write-enable signal *RAMWE*. These two stages are then AND'ed into a 32-bit memory decoder. The top-level decoders for the upper 3 bits are formed by 8 complex logic gates, each controlling half a LUT of memory. Address bit *WA6* is OR'ed with RAM32 modes, because 32-bit RAM modes do not require *WA6*; similarly, *WA7* and *WA8* are enabled only when used.

The memory cell design is shown in the inset of Figure 5.12. It is shared with shift-registers: each memory cell is configurable for outputting two memory bits and one shift-register. Each memory bit is controlled by a latch, and a write is executed when both the lower 5-bit decoder and the upper 3-bit decoder are enabled.

The shift-register is configurable for dual-edge or single-edge operations. It is implemented as 4 latches, 2 of which are shared with the memory cell. The upper 2 latches are only enabled for dual-edge modes, and output a '0' otherwise. Due to its larger footprint, each LUT only support 32 bits of shift registers, configurable as a single 32-bit or two 16-bit shift registers. The 32-bit shift registers can be concatenated with the shift registers from adjacent LUTs, forming a 128-bit shift register per CLB. The final shift-register output from LUT D is signal *MC31*, which can be selected as a CLB output. This allows a CLB to concatenate its shift register with other CLBs to form even longer shift registers.

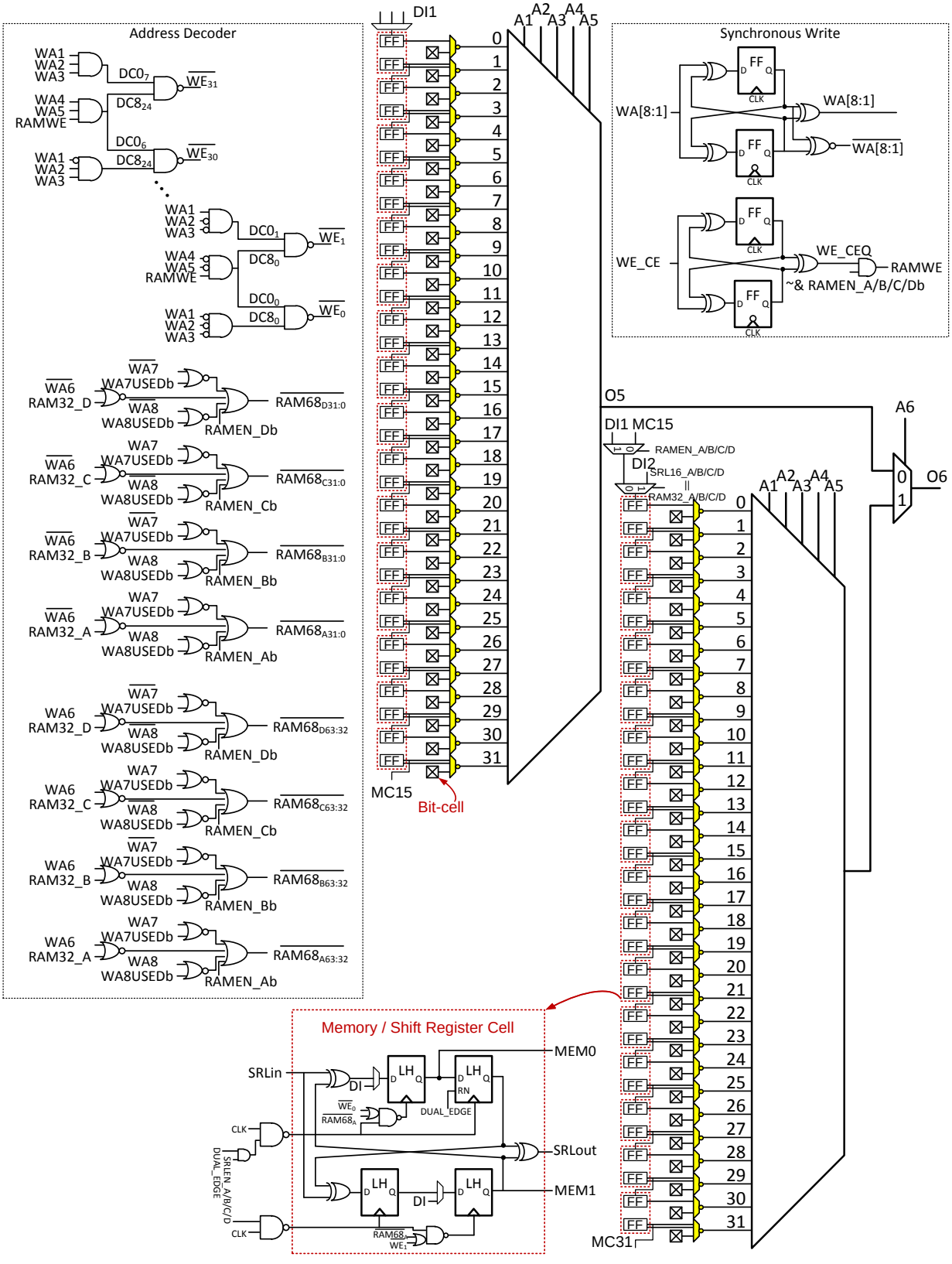


Figure 5.12: Slice M microarchitecture of the memory and shift-register logic.

5.4 Medium-Grained CLBs for the 16K-LUT FPGA

Although Slice L and M are effective for basic arithmetic and small distributed memories, these fine-grained reconfigurable blocks have their limitations. More intensive arithmetic such as wide multiplications and multi-input additions would require a more dedicated DSP accelerators, and larger memories would require a more dedicated memory IP. These requirements lead us to the next granularity of reconfigurable blocks, medium-grain DSP and Block RAM CLBs.

Our previous DSP CLB were designed in-house, thus no commercial FPGA synthesis tools can easily utilize these CLBs. We therefore adopted an ASIC synthesis flow for our FPGA macros, and have coded Synopsys DesignWare macros to automatically translate arithmetic into our DSP CLB, but the results were less than satisfactory. For this DSP CLB, we designed the DSP accelerators to be compatible with the DSP48E1 accelerators from Xilinx Virtex-6 and 7, as shown in Figure 5.13 [XilinxV6DSP11]. We can therefore use commercial FPGA synthesis to map to DSP CLBs. This DSP CLB has no LUTs, and is a dedicated DSP accelerator, reconfigurable for various addition/subtraction, multiplication, and Boolean operations. Various pipeline modes are implemented to support a maximum operation frequency of 800 MHz, and all pipelines stages have programmable latencies of 0, 1 or (sometimes) 2 clock cycles. In our implementation, all pipeline registers support dual-edge flip-flops, making them timing-compatible with Slice L/M.

As shown in the left half of Figure 5.13, the datapath first integrates a pipelined adder for inputs *A* and *D*, feeding into a 25×18 bit multiplier that multiplies with input *B*. Note that the inputs of the multiplier can also be selectable to *ACIN* and *BCIN*, which is locally propagated to *ACOUT* and *BCOUT*. This is very useful in some applications (e.g. filters) where one of the

multiplier's input needs to be constant, and this local propagation allows such operation without using extra interconnect resources. The operation modes can be changed during chip operation by changing the *INMODE* selections, which is connected to the interconnect network.

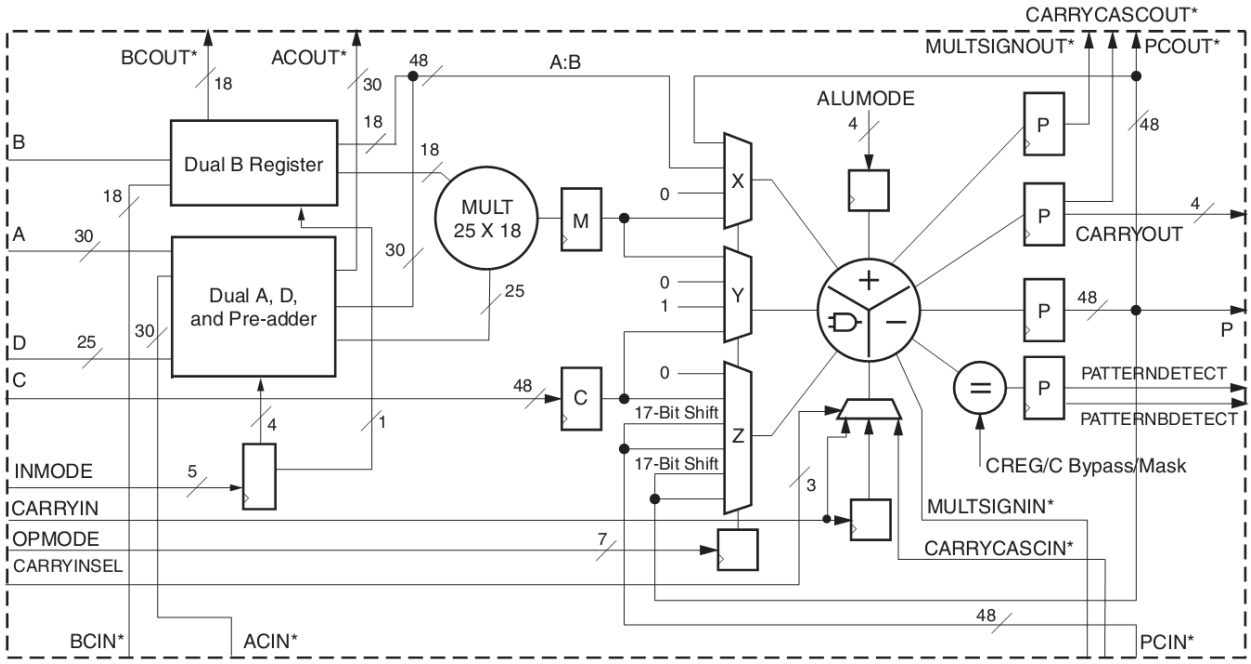


Figure 5.13: Architecture of a commercial FPGA DSP accelerator [XilinxV6DSP11]..

The right half of Figure 5.13 illustrates a 3-input datapath, operating on local inputs *X*, *Y*, and *Z*. These inputs can come from the interconnect network (inputs *A*, *B*, *C*, and *D*), the multiplier output, or locally-propagated input *PCIN* from the previous ALU. Carry-out information are also locally propagated in *MULTSIGNIN* and *CARRYCASCIN*, allowing the ALU to correctly compute data when carried across multiple DSP CLBs. This allows the ALU support wider multiplication and arithmetic operations. Operations such as addition, subtraction, Boolean operations, and pattern detections are supported. The ALU may function as a single 48-bit operator, or function as a single-instruction, multiple-data (SIMD) ALU for two 24-bit operations or four 12-bit operations.

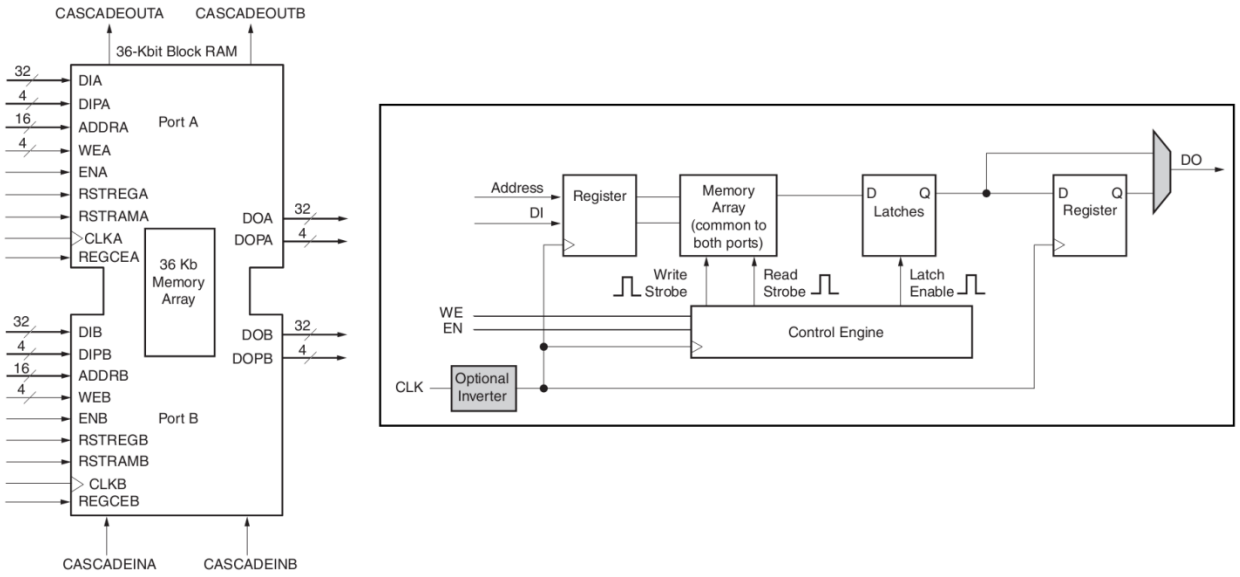
From mapping Block RAM designs on our previous chip, we noticed a similar

inconvenience with the mapping flow. The Block RAM CLBs were highly reconfigurable, and support many modes, but none of these modes can be mapped automatically. Because the ASIC synthesis flow will not map designs onto our BRAM CLBS, every Block RAM must be instantiated manually, and synthesized as a black-box. The suitable BRAM modes must also be configured by the user. For this design, we decided to design a BRAM CLB that is compatible with the RAMB36 designs from Xilinx Virtex-6 and 7 [XilinxV6BRAM11]. Not only does this resolve the issue of mapping automation, allowing commercial FPGA synthesis tools, it also provides a much larger memory (36Kb instead of 1Kb). Because the number of address bits is only a log of the number of elements, the number of I/O bits remains manageable.

Figure 5.14 shows the top-level architecture of the 36 Kb BRAM and a datapath for one of its ports. A total port width of 36 bits is supported. Virtex-6 BRAMs allow 4 bits to store the error-correction codes (ECC) for the other 32 bits of data. Our implementation does not support ECC, and all 36 bits are used for data storage. The reconfigurable BRAM supports memory sizes of 32Kx1b, 16Kx2b, 8Kx4b, 4Kx9b, 2Kx18b, 1Kx36b, or 512x72b, and the two ports can be configured for different memory sizes. Although in 512x72b mode, the dual-port memory can only function as a simple dual-port, because all 72 bits of data inputs (from port A and B) are used for writing, and all 72 bits of data outputs are used for reading. Byte-wise masking is provided, although the granularity is 9-bits per “byte” (8 bits from *DIA/B* and 1 bit from *DIPA/B*) due to the 36-bit data width. In 32Kx1b mode, the BRAM can be locally concatenated with its neighbor into forming a larger, 64Kx1b BRAM. Therefore the address input can be up to 16 bits.

Unlike the distributed memory from SLICE M, the BRAM CLB supports only synchronous read and synchronous write (Figure 5.14b), and an additional pipeline register may be added to the output to improve timing. This pipeline register has its dedicated reset

(*RSTREGA/B*) and clock enable (*RSTCEA/B*) signals. The read/write operation of each port is synchronous to its own clock input. When attempting to read and write from the same address, the user may select the output port to output the memory data before or after the overwrite process, or remain unchanged.



a) b)

Figure 5.14: A commercial dual-port block RAM and its a) block architecture and b) datapath [XilinxV6BRAM11].

In our implementation, we used a memory-compiled IP of 512x72b, dual-port, 8-transistor SRAM. The dual-port IP supports independent clock domains for each port, making it suitable for this reconfigurable implementation. For other memory sizes, additional address decoding is used to mask the data to perform the read and write operations on proper memory locations. Due to the limitation on the memory IP, the BRAM CLB can only operate on one clock edge. However, if dual-edged clocking is necessary, the two ports of the memory can run on two opposite edges of the clock, and external logic can be used to transition this BRAM into a dual-edged, single-port BRAM.

5.5 Coarse-Grained CLBs for the 16K-LUT FPGA

Since this chip primarily targets high-throughput communication applications, we have integrated two coarse-grain accelerators. The first block is a 16-core, highly-efficient communications DSP accelerator, reconfigurable to perform many common communications algorithms very efficiently. The 16-core architecture is illustrated in Figure 5.15. Core-to-core communications utilize both local, fast-path interconnects running vertically and horizontally, as well as a 4-stage hierarchical interconnect network spanning the 16 cores.

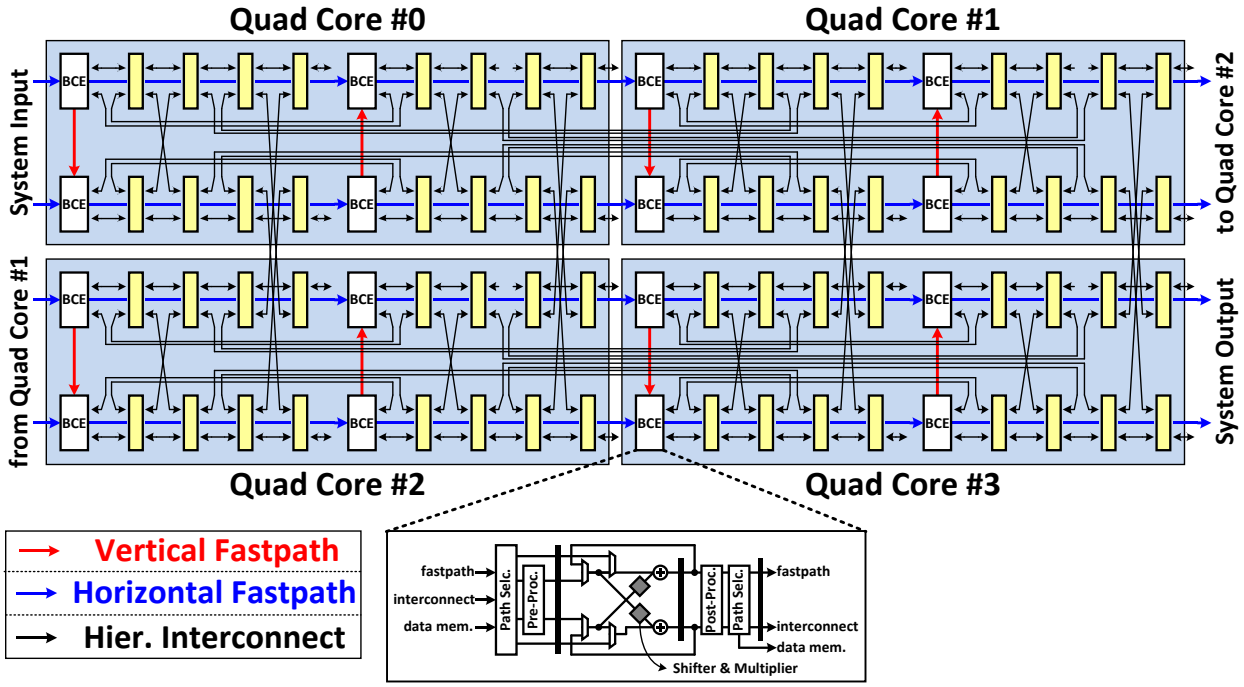


Figure 5.15: Core schematic and interconnect architecture of a 16-core DSP processor.

Each core is realized using radix-2 butterfly architecture, performing 2×2 matrix computations, called a butterfly-computation element (BCE). This provides the versatility for various fundamental 2×2 operations, e.g. permutation, CORDIC, multiplication-and-accumulation (MAC), unitary transformation, etc. Higher level of integration such as multi-stage pipeline is achievable with multiple cores. Each BCE is designed to be run-time reconfigurable

by leveraging the processor-style instruction set architecture (ISA). However, the overhead from traditional control loops (fetch-decode-operate-save) can be reduced since the output of each core is not stored back, but mostly transmitted to neighboring cores. The complexity of instruction fetch and decode can also be greatly reduced by using a specific instruction set combined with bit-cell-level programming. The concept of "real-time instruction redefinition" is therefore established: the same command can cause totally different operations in different cores based on their bit-cell programming.

At a first glance, the 2×2 butterfly design does not seem to relate to many communication applications. But as shown in Figure 5.16, many commonly used applications, such as the lattice filter, 2-way FIR filter, and zero-forcing (ZF) or minimum-mean-square-error (MMSE) signal equalization can all be mapped very efficiently. Even more complex algorithms such as QR factorization (where Q is an orthogonal matrix and R is an upper triangular matrix) and breadth-first sphere detector (SD) can be decomposed into multiple 2×2 BF stages. Though seemingly unrelated, these functions for spectrum shaping, channel factorization, and signal detection are neatly unified into a common architecture. This is therefore called a "universal DSP" (UDSP).

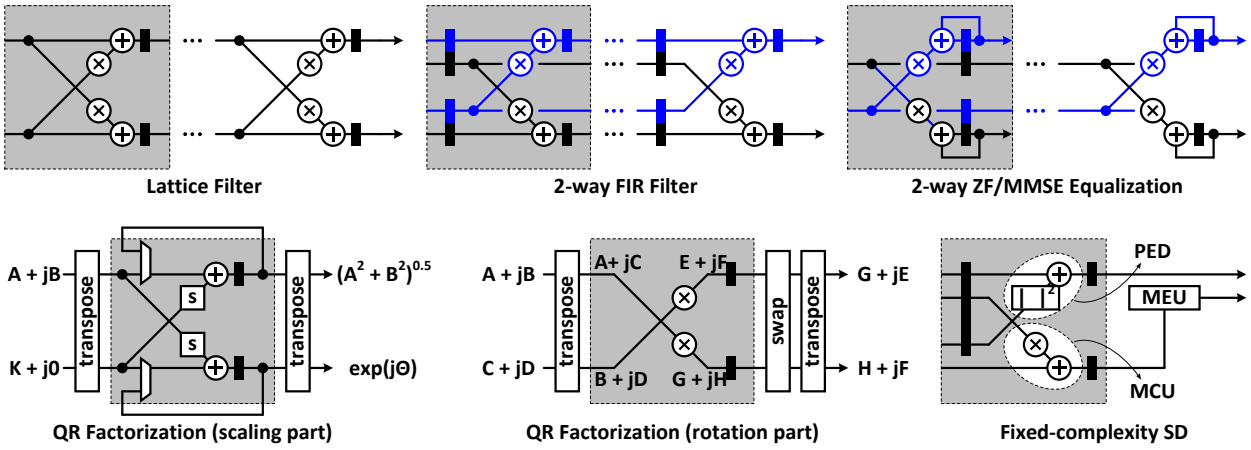
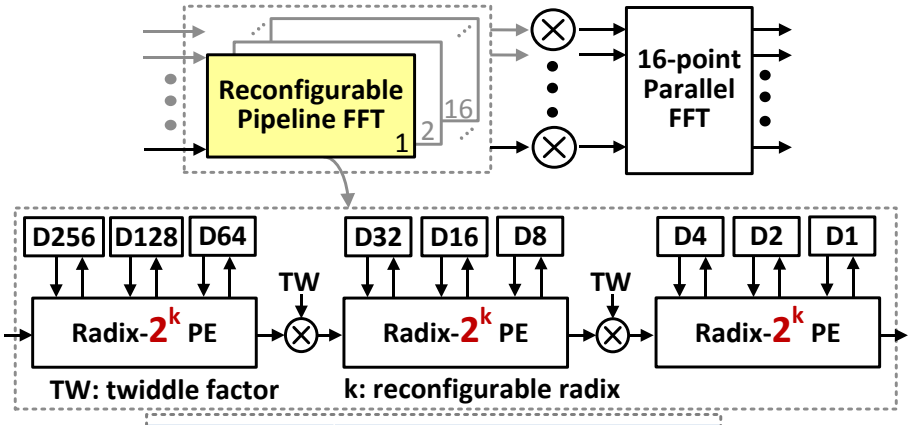


Figure 5.16: Example communication applications of the DSP processor.

A key enabler for this technology is efficient interconnects: three layers of interconnects, the core's internal feedback, the 2-D unidirectional fast-path, and the radix-2 hierarchical network are implemented. Each layer of interconnect deals with the corresponding scale of datapath, from the folding architecture for equalization and QR factorization, the pipelined architecture for filters and SDs, to the signal broadcasting for inter-core communications.

The other coarse-grain reconfigurable block we implemented is a 64 – 8192 point reconfigurable Fast Fourier Transform (FFT) processor. Because FFT processors, especially those with fine resolution, require large memories, this FFT processor is designed to utilize the BRAM and distributed Slice-L memories on the FPGA, instead of having its dedicated memories. This resource sharing reduced the area of the FFT processor by more than 50%. To achieve high energy efficiency, the FFT processor is designed with 16 parallel cores, each with reconfigurable pipelines (Figure 5.17).

The 16 FFT cores can each support frequency resolutions from 64 to 512 points. It is implemented as a 3-stage pipelined FFT to further ease the timing requirements and improve energy efficiency. Extensive analysis in radix factorization is performed in [Yang12] to determine the optimal radix per pipeline stage that minimizes the energy and area requirements. The 16 parallel cores are processed by a final 16-point FFT to realize a 64 to 8192 point FFT processor. An extensive exploration and illustration of radix factorization, energy-area minimization, and parallel FFT are documented in [Yang12, Yu12].



FFT size	Pipeline FFT Radix (k)		
64	-	-	4
128	-	-	8
256	-	4	4
512	-	4	8
1024	-	8	8
2048	4	4	8
4096	4	8	8
8192	8	8	8

Figure 5.17: The FFT architecture and radix factorizations of different FFT resolutions.

5.6 Macro-based Chip Integration for the 16K-LUT FPGA

Due to the large size of the chip, and the very limited man-power available, this chip is designed using macro-based hierarchical physical design. In the previous 2048-LUT FPGA, only 2 types of SMs are used, forming 3 different types of SM macros. In this design, over 10 types of SMs are used, forming into 7 different types of SM macros with 10 different physical footprints. As a result, we can no longer create individual SMs and integrate a collection of them into a physical layout. The area discrepancies between the individual SMs will cause gaps in the integrated layout, causing wasted area.

To maximum area utilization, each of the 7 SM macros is realized as a complete layout,

including all SM stages with their bit-cells, muxes, and output buffers. Each SM macro is designed to support 5 inputs and 2 outputs from the CLB, although some support 3 or 2 inputs and 2 outputs, depending on the CLB it is tied to. To not cause gaps in physical integration, all SM macros are designed to have a fixed width, and all SM macros residing in the same horizontal row have the same height as well. This ensures a gap-less physical integration.

An example SM macro layout is shown in Figure 5.18. The top half of the SM layout is filled by an array of bit-cells. Each of the BC is an optimized 6T SRAM cell, as previously described in Section 4.4. The true and complement bit-lines (*BL* and *BLb*) runs horizontal, and the word-line (WL) runs vertically within the cell, and all the bit-lines and word-lines of an SRAM array are connect through physical placement. In addition, the bit-lines and word-lines also connect to the boundaries of the SM macro, so any SM macros placed adjacent to it will have its bit-lines and word-lines easily connected.

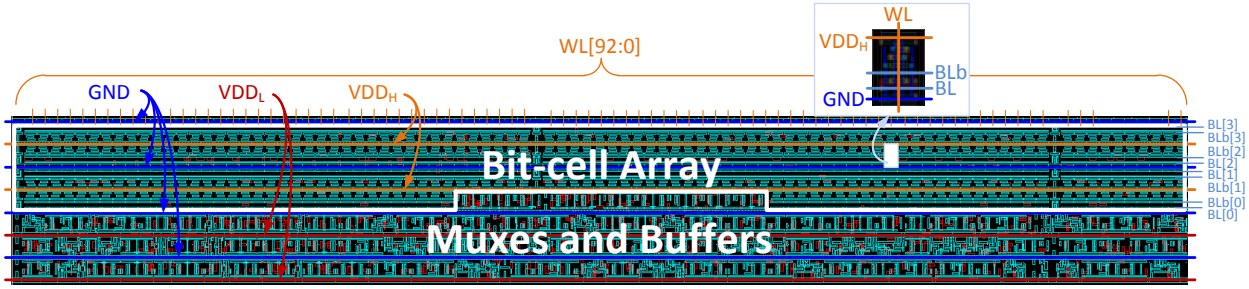


Figure 5.18: An example physical design of a SM macro.

The SM macro is separated into two voltage domains, VDD_H for the bit-cells, and VDD_L for the muxes and buffers. However, it is common for the bit-cell arrays to fill up an integer multiple of rows, because the width of the SM macro is fixed across all macros. The result is a U-shaped bit-cell array, as shown in Figure 5.18. This leaves an area gap in the VDD_H domain. To not waste area, we place pass-gate muxes in this region. Pass-gate muxes do not have buffers, nor do they have any power-gating signal (which requires VDD_L). These pass-gates can be safely

placed in VDD_H domain. The PMOS body of these gates is tied to VDD_H, but since the gate voltages are driven by bit-cells, the gate voltages are also in VDD_H, causing no leakage issues.

As previously described in Section 4.5, the true and complement outputs of the bit-cells are driven directly from the bit-cell to the gate of the pass-gate muxes. This causes routing congestion inside the SM macro; for example, an 8-input static mux with power-gating requires 9 bit-cells, which is 18 true-and-complement control signals in a very small area. As a result, no automated placement is used. The bit-cell array is designed with spatial locality in mind. Instead of assigning a bit-line and word-line to each bit-cell, and placing it in an array, we first analyze the muxes each bit-cell is tied to. The bit-cells are then arranged to minimize routing distance and minimize wire cross-over. For example, if a mux has 9 different bit-cell inputs from left to right, it is best to arrange the bit-cells in the array in the same order to simplify routing. Additionally, if an SM input is driving 4 muxes, for example, it is best to place these 4 muxes locally, and not on opposite sides of the SM macro. As a result, the bit-cells associated with these muxes must also be placed close together.

Once the bit-cell array is determined, each mux and buffer is then placed manually. The location of each mux is chosen to minimize routing distance and minimize wire cross-over. The buffers are also placed close to the mux output. In the end, routing is performed automatically using commercial routing tools. Timing characterization is then analyzed, and the design is iterated if necessary. The SM macros are all designed with layout density of 98% or higher.

The chip integration hierarchy is illustrated in Figure 5.19. Starting from the SM macro, it is integrated with a corresponding CLB into one CLB macro. The CLB macro includes all the SM macros required to network its inputs and outputs. A heterogeneous collection of CLB macros are then integrated into a tile, and 9 heterogeneous tiles are integrated in the top level.

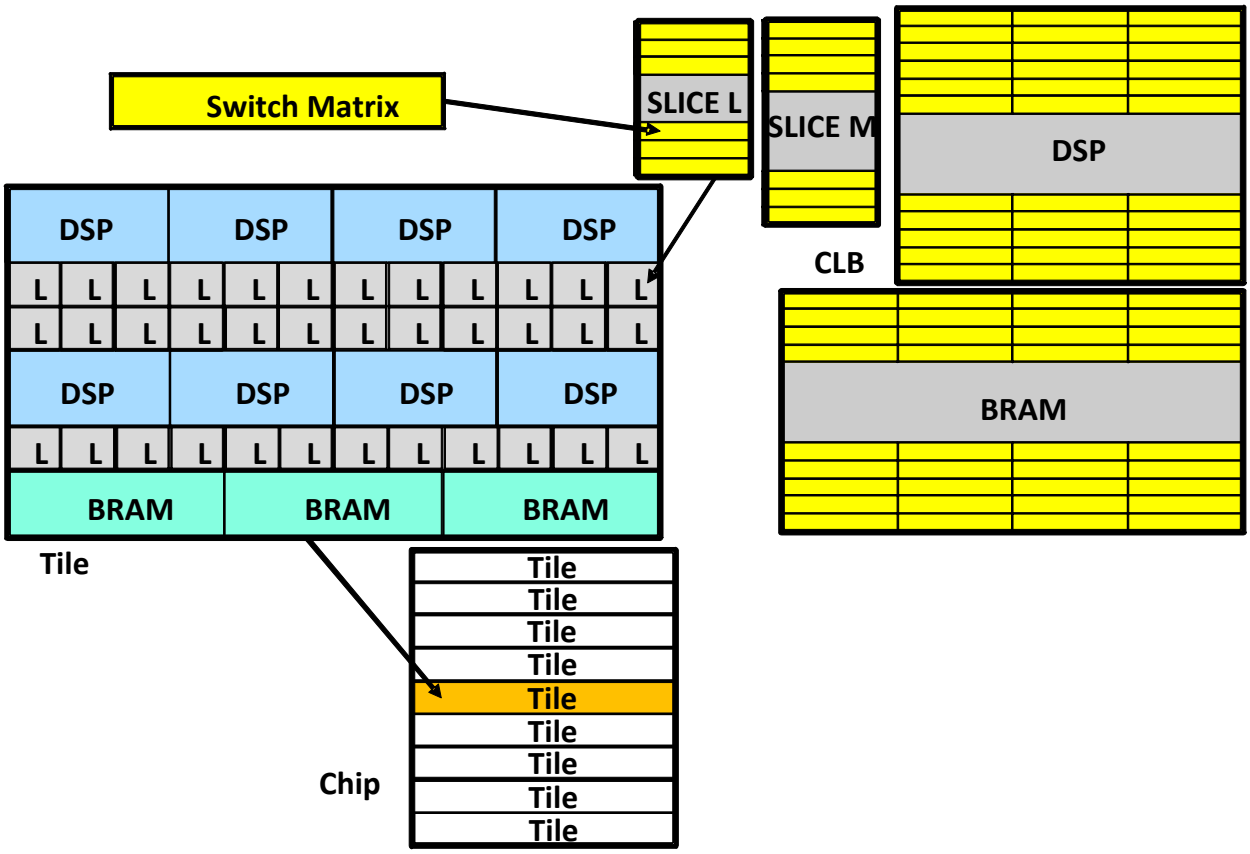
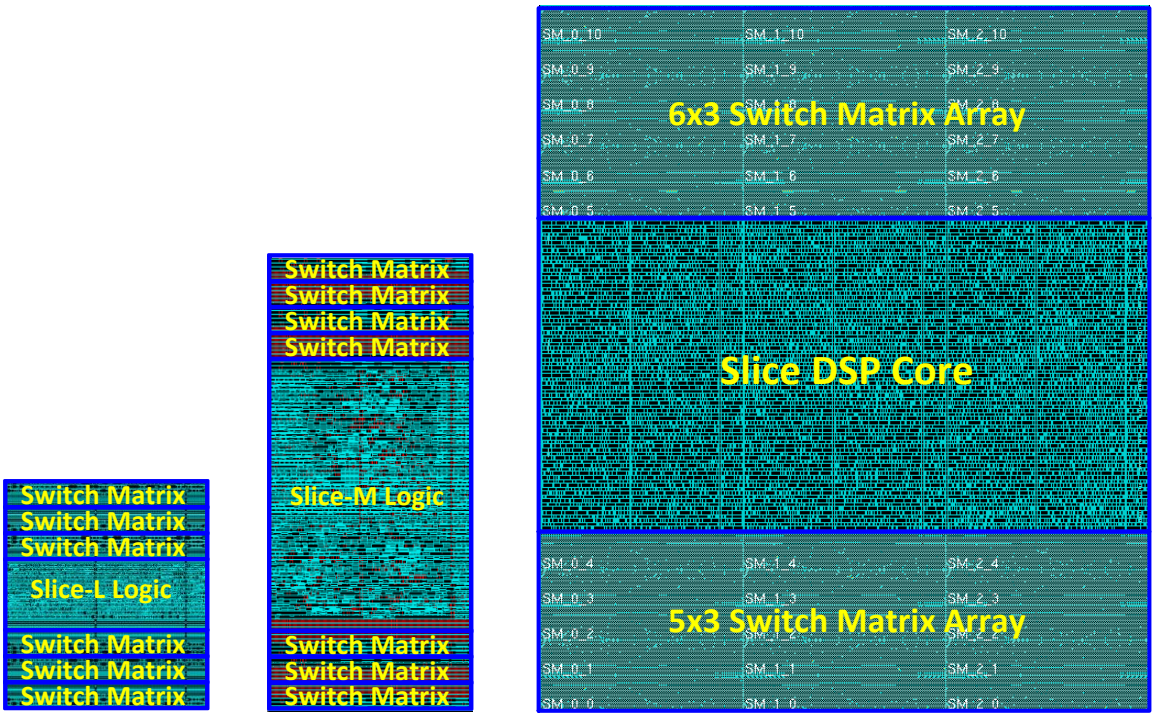


Figure 5.19: Illustration of the hierarchical design methodology used for chip integration.

Depending on the interconnect requirements of each CLB, different number of CLBs are instantiated to integrate with the CLB into one macro. Example layouts of these CLBs are shown in Figure 5.20. A Slice L CLB requires 30 inputs and 12 inputs, therefore requiring 6 SM macros (5-input, 2-output each). A Slice M CLB requires 7 SM macros. The DSP CLB requires 33 SM macros due to its large input requirement. A BRAM CLB has large output requirements (72), requiring 36 SM macros, but its input requires are smaller (124), and are composed with 28 3-input, 2-output SMs and 8 5-input, 2-output SMs. The 36 SM macros are best arranged in a 4x9 array, but as shown in Figure 5.20d), the physical footprint of the 512x72b memory IP is far too wide for 4 SM macros; it is approximately the width of 8 SM macros. Therefore, 2 BRAM CLBs are integrated together, their SM macros are placed side-by-side, forming an 8x9 array of SM

macros, and the two memory IPs are stacked vertically. The area in the middle is used by the control logic for both BRAMs.



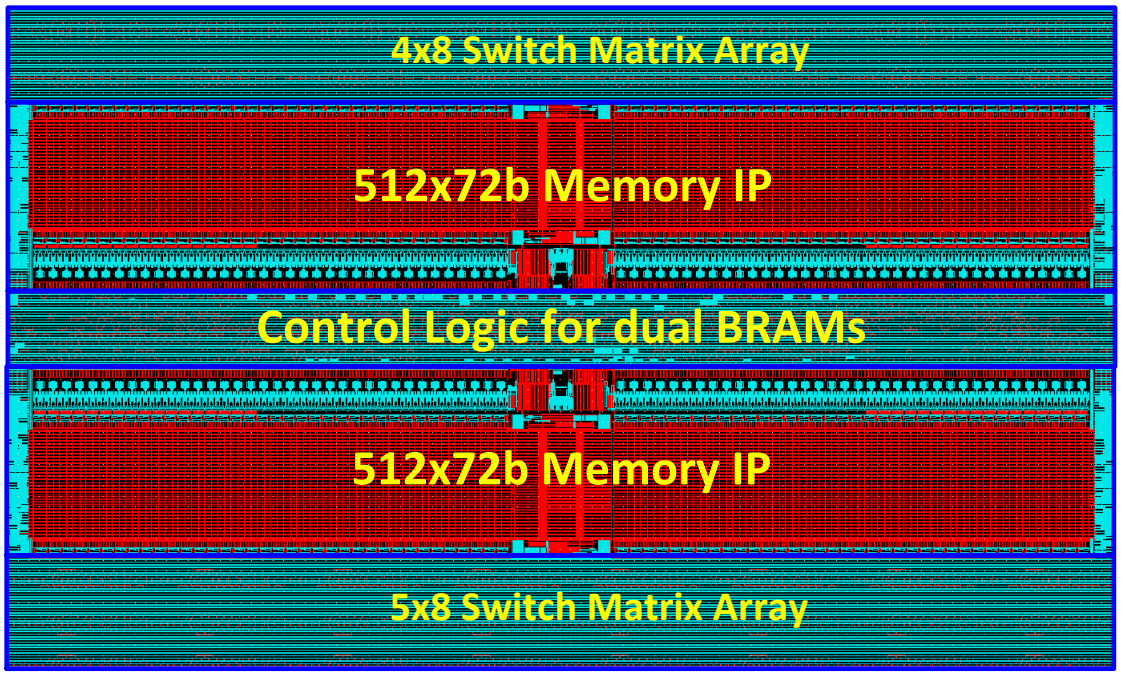


Figure 5.20: Layout examples of a) Slice L, b) Slice M, c) DSP, and d) BRAM CLBs and SMs.

By integrating each CLB with its SMs, the actual I/Os of each CLB is completely abstracted away. The only remaining I/Os are the I/Os of the SM macro that each CLB connects to. This creates a slightly more systematic I/O netlist for tile-level and top-level integrations. Since the interconnect architecture is optimized by our software mapping tool, we coded the tool to automatically generate the interconnect Verilog associated with every tile. Since each tile integrates hundreds of CLBs with thousands of SMs, each tiles has 50,000 to 100,000 interconnect signals. This automated Verilog creation method is robust, error-free, and ensure the created Verilog to be an exact match to the optimized interconnect architecture.

The top-level architecture of the 16K-LUT FPGA is shown in Figure 5.21. It is divided into 9 tiles of heterogeneous CLBs. Unlike the 2048-LUT FPGA design, the WL scan chains do not run inside every CLB, instead, there is only one set of BL scan chain and one set of WL scan chain, propagating across the entire chip. To avoid hold-time issues, the scan flip-flops are implemented as master-slave latches, running on a non-overlapping clock *P1* and *P2* for maximum robustness in timing and reliability. Inside each tile, the BL and WL signals are buffered to meet transition time requirements for electromigration. The buffering overhead is far less than the area of implementing a separate scan chain inside every CLB. Because the BL and WL are routed to the boundaries of every SM macro, which are then routed to the boundaries of every CLB macro, connecting the BL and WL signals in the tile level and the top level requires minimal effort.

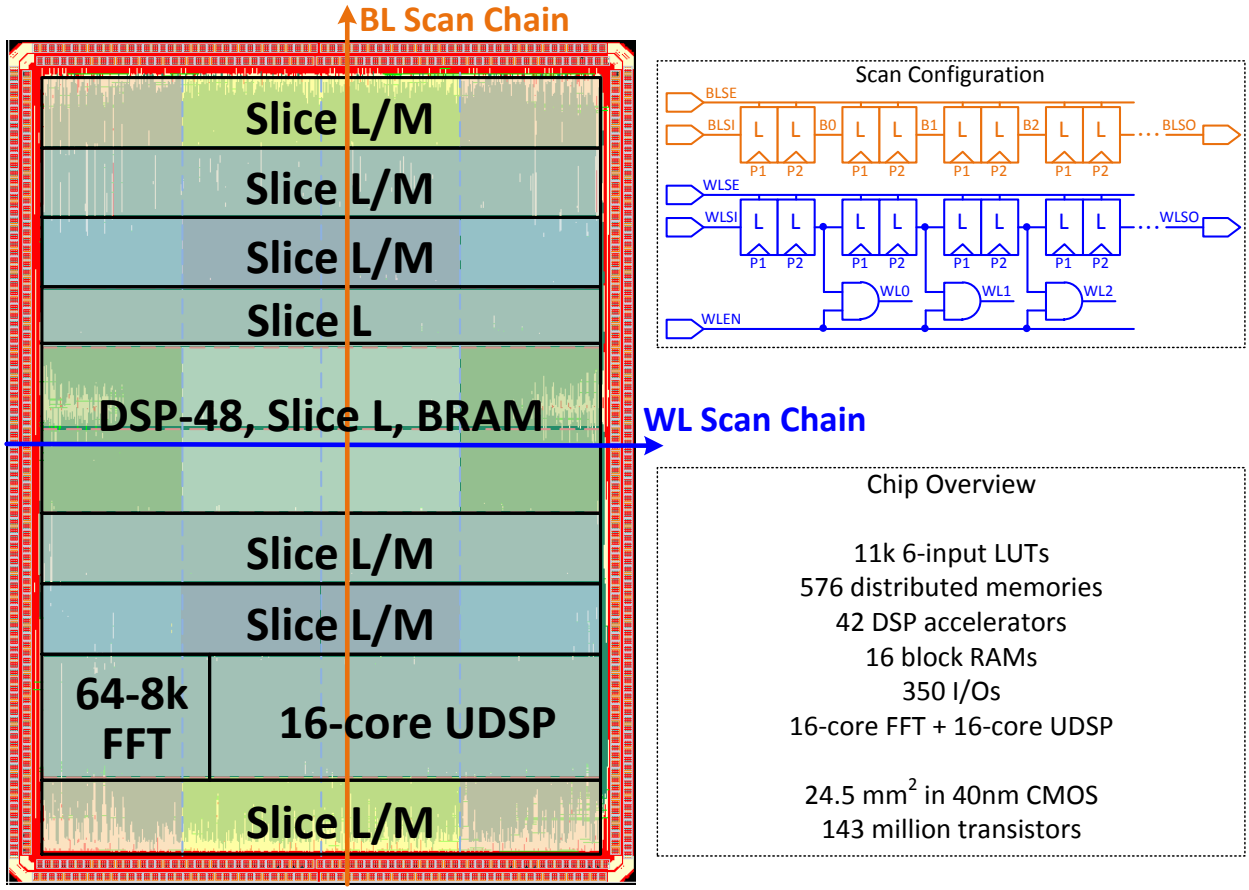


Figure 5.21: Top level CLB and SM architecture, illustrating scan chain for BL and WL.

As shown in Figure 5.21, the heterogeneous integration in the top level requires various CLB and SM combinations. The colors in Figure 5.21 illustrate the different SM macros used for interconnects, matching the colors used in Figure 3.16 for interconnect illustration. A total of 16K SM macros are used.

Even with a 10x interconnect complexity, the overall interconnect area is 52%, still maintaining a logic to interconnect ratio of 1:1. This 16K FPGA has demonstrated the O(N·log$N$) the scalability of the hierarchical interconnect network.

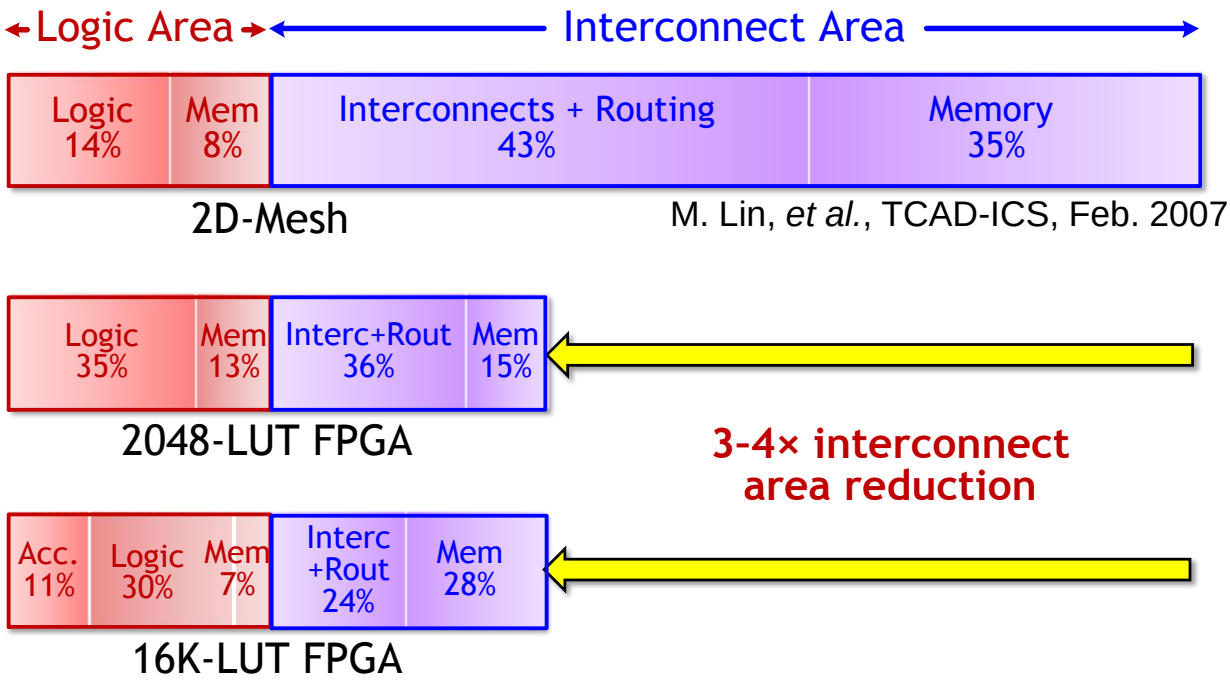


Figure 5.22: Area impact of our two FPGAs: a 1:1 logic-to-interconnect ratio.

Overall, this FPGA integrates 11,008 6-input LUTs in 2752 CLBs, 2176 of which are Slice L, and 576 are Slice M. Following conventional FPGA scaling, this 11K 6-input LUT FPGA is approximately a 16K 4-input LUT FPGA. A total of 350 configurable data I/Os are implemented. There are 42 DSP CLBs and 16 BRAM CLBs, all integrated in the center tile. The 16-core parallel FFT and the 16-core universal DSP macros are located near the bottom. Overall, this chip occupies 24.5 mm^2 in 40nm CMOS technology, and is comprised of 143 million transistors.

CHAPTER VI

Software Flow and Design Mapping

6.1 Overview of FPGA Software Mapping Flow

A key feature of reconfigurable hardware is software programmability. Unlike dedicated chips, FPGA chips are accompanied by complete software flows to enable users the map their designs onto the FPGA. At a high level, FPGA mapping and ASIC physical design do not appear too different, they even use similar terminology, from logic synthesis to gate placement, followed by interconnect routing. Although a few algorithms do apply to both flows, in reality the two software flows are designed very differently, and are generally not inter-compatible.

Figure 6.1a) shows a general software flow for mapping user design onto commercial FPGAs. The user supplies a hardware-description language (HDL), generally in Verilog or VHDL, and feeds into the tool. Modern FPGA tools are very complete, and require the users to perform very few manual preparations to have a mappable design. Unlike ASIC synthesis, FPGA synthesis often does not require a user-supplied timing constraint or timing library. The user simply selects the FPGA platform to be targeting, and the tool applies the timing libraries automatically, and also synthesizes its logic into the appropriate LUTs for the target FPGA. In case timing constraints are not provided, the tool would generally attempt to create the fastest-possible design.

Once synthesized into LUTs for the target FPGA, LUT packing is performed. LUT packing is designed to efficiently “pack” LUTs into CLBs, which generally consist of 4 LUTs and their supporting logic. Depending on the target application, LUT packing may be performed to minimize inter-CLB routing (connection-driven), or to minimize critical-path delay (timing-

driven), or a combination of the two [Marquardt99]. Once the LUTs are packed into CLBs, the CLB is ready for placement and routing.

Logic gate placements are generally performed to utilize spatial locality, either based on connections or timing. In other words, gates with many communication between them, or are lies on a critical path, should be placed closely to each other. Such task is generally realized as hierarchical partitions, where a large network of gates is partitioned into 2 or more sections with the goal of minimizing the total communication between the sections (min-cut). The partition scheme can be applied hierarchically to reduce the large network into small clusters of gates, which is then mapped locally onto CLBs.

Routing is executed once the gates are placed. It is generally executed in two stages, global routing and detailed routing. Global routing is first performed to route all connections onto the interconnect network while disregarding routing resource conflicts. This not only provides a best-case timing of the design, the routing conflicts also serve as an indication of routing congestion (e.g. how many nets are trying to use the same routing resource). Detailed routing is then executed to resolve routing conflicts to create a fully routable design where each routing resource is only occupied by one net. In an FPGA, routing resource is often scarce, and extensive research is done in this field to resolve routing conflicts.

From the fully routed design, the tool has full visibility into the configuration of each CLB, and the path of each route. From this information, it is then able to determine the proper configuration for all the bit-cells. The bit-cells are then written and created into a bitstream, which is used to program the target FPGA to perform the configured design.

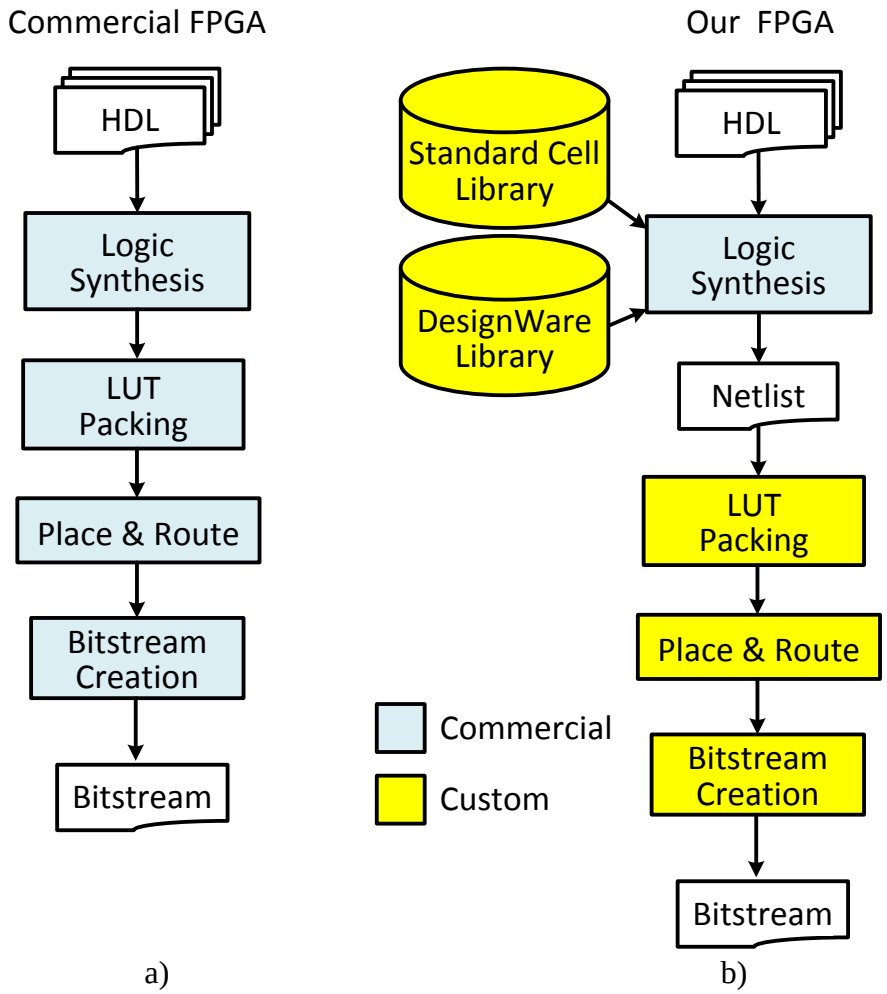


Figure 6.1: Software mapping flow of a) commercial FPGA tools and b) our flow.

Our FPGAs are designed to serve a similar user experience (Figure 6.1b). However, our first 2048-LUT FPGA employed custom CLBs that are not compatible with commercial FPGAs. This required us to create a custom synthesis flow. To avoid writing our own FPGA synthesis tool, we designed custom a standard-cell library of our LUTs and a custom DesignWare® library for our DSP blocks to perform logic synthesis using ASIC synthesis tools. The resulting netlist is then sent into our custom software tool for packing LUTs into CLBs. The details of our custom synthesis and packing flow are discussed in Section 6.2.

Since our hierarchical FPGA has a different chip architecture from commercial FPGAs,

and our netlist consists of custom CLBs, commercial software for logic placement cannot be used. We therefore developed a custom gate-placement flowing using hierarchical partitioning techniques. Partition is performed hierarchically, following the radix boundaries of our FPGA, until the sub-partitions are small enough to be directly mapped to the appropriate CLBs. The placement details are discussed in Section 6.3.

A custom router is also developed. Global wiring is generally based on a shortest-path algorithm [Nair87], and the hierarchical architectures allows for very deterministic global routing. Our detailed routing was first based on a "rip-up-and-reroute" approach to resolve routing conflicts [Dees81], but it poses a timing penalty, and because the routing order of the nets matter, the router does not produce consistent results. We then adopted a negotiation-based, timing-driven algorithm called Pathfinder [McMurchie95], and modified it to improve runtime for our applications. Bitstream is then created based on the placed-and-routed design. The router details are discussed in Section 6.4.

6.2 FPGA Synthesis and LUT Packing

For our first, 2048-LUT FPGA, the CLBs are not compatible with the CLBs in commercial FPGAs, disallowing us from using a commercial synthesis flow. To avoid writing a custom synthesis tool, we create a custom standard-cell library to use ASIC synthesis tools to map HDL onto our LUTs. The standard-cell library is created to contain 2-, 3- and 4-input LUTs, summing to around 4,000 different types of cells. Each LUT is assigned a constant area and a constant logic delay. A snapshot of a synthesized netlist is shown in Figure 6.2. Each LUT configuration maps to a unique standard cell name, so the software tool can determine the LUT mapping, and the connections are described by the net names.

For arithmetic functions, Synopsys DesignWare® libraries are created to automatically instantiate DSP CLBs to perform additions and subtractions. For memories that require block RAM instantiations, the user needs to instantiate BRAM CLBs in their HDL, else the synthesis tool will realize the memory array using flip-flops from the CLB resources.

```
…
LUT4_0x0x_x0x0_0xx0_x00x_1 U85(.A (n28), .B (n9), .C (mult3_N3), .D(n53),
     .Y (mult3_DP_OP_7J4_127_4559_n160));
LUT4_0000_1 U86(.A (mult1_N1), .B (mult1_N5), .C (mult1_N2), .D(mult1_N3),
     .Y (n65));
LUT3_010_100_1 U87(.A (mult1_DP_OP_7J4_125_9370_n214),
     .B(mult1_DP_OP_7J4_125_9370_n146), .C (mult1_N7), .Y (n66));
LUT4_1xx1_x0x1_xx01_1 U88(.A (mult1_N6), .B (n65), .C (n66), .D(mult1_N18),
     .Y (mult1_N25));
LUT3_010_100_1 U89(.A (mult2_DP_OP_7J4_126_8975_n217),
     .B(mult2_DP_OP_7J4_126_8975_n135), .C(mult2_DP_OP_7J4_126_8975_n133),
     .Y (n67));
LUT3_011_1 U91(.A (mult4_add_x_26_1_n1), .B (mult4_n6), .C(mult4_n4), .Y
     (n69));
LUT2_01_10_1 U92(.A (mult4_n7), .B (n69), .Y (M_OUT4[2]));
…
```

Figure 6.2: A snapshot of a synthesized netlist using our custom standard-cell library.

From the synthesized netlist, the software tool can “pack” LUTs into CLBs, which consist of 4 LUTs per CLB. The packing algorithm we implemented is based on the “timing-driven packing” proposed in [Marquardt99]. As expected, the interconnect delay for intra-CLB LUTs are much shorter than the delay for inter-CLB LUTs, these delay are added in addition to the internal logic delay of each LUT. Based on these 3 delay models (which is can be just 1 constant value for each model for simplicity, and inter-CLB delay would remain unknown until place-and-route is complete), the delay for each path can be modeled. Based on the delay of each path, divided by the maximum path delay, each net is assigned a net-criticality ratio between 0 and 1. In reality, the affinity between two LUTs is often not only defined by the timing-critically of the net, but also the number of connections between the two LUTs. Therefore a second ratio,

called “net attraction” is defined as the number of nets shared by the two LUTs, divided by the total number of nets they can accommodate, which also results in a ratio between 0 and 1. The total attraction between two LUTs is then computed as the weighted-sum of the net-criticality ratio and the net-attraction ratio. The weight essentially trades off delay minimization with net sharing.

Packing is not only useful in realizing a CLB netlist that can be used for place-and-route, it also effectively reduces the number of gates (by up to 4x in our case), which significantly reduces the problem size for placement algorithms [Betz97].

Although the synthesis and packing flow is functional, from our experience mapping our first 2048-LUT FPGA, the synthesis results were not always ideal. ASIC synthesis tools always perform technology-independent mapping, and then try to allocate suitable standard-cells from the technology library to map the appropriate functions. Most of the ASIC algorithm assumes that small, simple CMOS gates are fast, and complex gates are slow. This is intrinsically different from FPGA synthesis [Sangiovanni93], where a logic gate of arbitrary function is realizable, but adding every LUT to the critical path can significantly impact timing. ASIC synthesis can also impact packing results, for example, a multiplexer can be integrated into the CLB by directly using the F7 or F8 logic gates, but the ASIC synthesis tools often implement the multiplexer as LUTs, which can no longer packed as efficiently.

From our mapping experience, ASIC synthesis tools often results 50% or more gates on the critical path than FPGA synthesis. Additionally, although 2048-LUTs are still reasonable to map, large designs (10,000 LUTs or more) often cause our ASIC synthesis tool to struggle, and sometimes crash unexpectedly. These concerns have motivated us to migrate to a CLB design that is compatible with commercial FPGA synthesis tools.

The updated software-mapping flow for our new 16K-LUT FPGA is shown in Figure 6.3. By using commercially-compatible CLBs, we are able to utilize commercial FPGA synthesis tools such as Synopsys Synplify Pro for logic synthesis. The synthesized netlist can be fed into commercial FPGA place-and-route tools, which create a benchmark comparison for our design, and also outputs packet netlist and place-and-route results. In Mode 1 of our new design flow, the tool is able to parse the packed netlist from the commercial tools and perform place-and-route onto our FPGA architecture. This allows for a direct comparison between commercial FPGA and our FPGA while mapping the same netlist. In Mode 2, the synthesized netlist is processed to instantiate of our coarse-grain CLBs, which are used as accelerators and kernels. This new netlist is then fed into our place-and-route flow to perform LUT packing, as well as place-and-route. The accelerator/kernel insertion tool is still being developed.

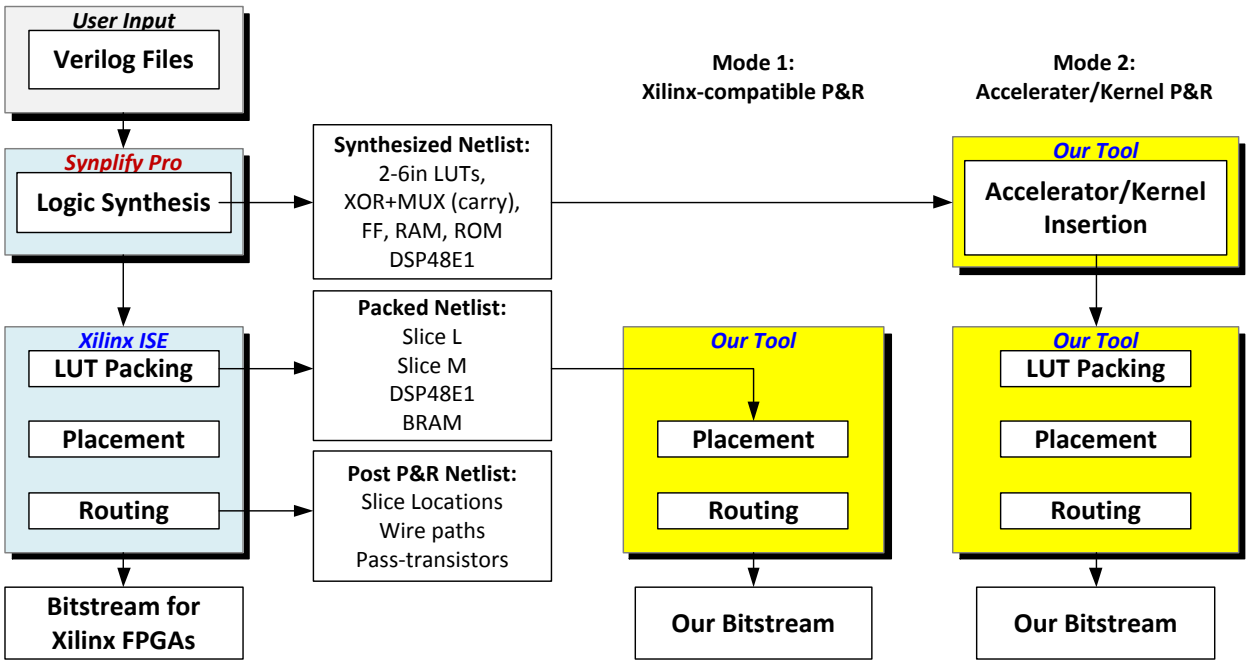


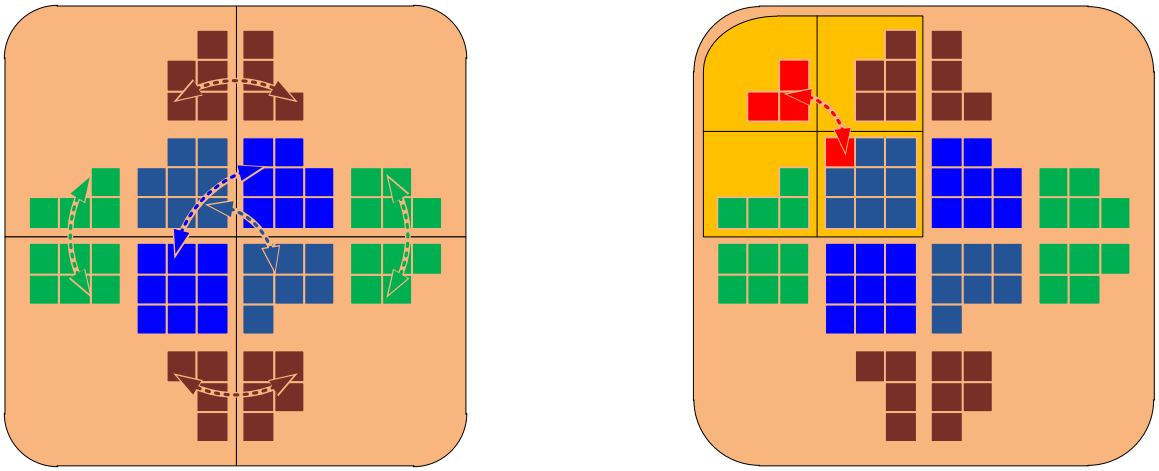
Figure 6.3: The updated software mapping flow for our new FPGA.

The updated software mapping flow allows us to utilize mature, commercial synthesis tools for mapping designs onto our FPGA. The quality of results is significantly improved

compared to using ASIC synthesis tools. The Mode 1 even allows us to utilize the LUT packing tool from commercial FPGAs, which is made possible because our new FPGA is CLB-compatible with that of Virtex 6 and 7. The CLB netlist still needs to be placed and routed onto our FPGA, which is covered in the next two sections.

6.3 FPGA Partitioning and Placement

For large VLSI designs, partitioning is an essential step towards reducing the problem size. In case of large problems, clustering is one effective method to further reduce the problem size by combining individual gates into larger "clusters", then perform partition on the clusters, followed by a coarsening phase of unclustering the gates. Partitioning can be performed hierarchically to form successively-smaller partitions, eventually converging to partitions small enough to place individual gates, and then the placement process can be completed. Figure 6.4 illustrates the concept of hierarchical partitioning. Figure 6.4a) shows a chip top-level divided into 4 quadrants, where partitioning is performed to minimize the number of wire crossing the horizontal, vertical, and diagonal boundaries (min-cut). The gates that have wires crossing these boundaries are shown. Figure 6.4b) illustrates the next step, where the hierarchical partition tool descends into a quadrant, one quadrant at a time, and performs the min-cut partition on the quadrant. The gates that have wires crossing the new partition boundary are shown. In our implementation, we have modified the min-cut algorithm to not only count the number of nets, but also to weigh the wire cost by the timing-criticality of the net. This allows for a timing-driven and connectivity-driven partitioning.



a) b)

Figure 6.4: Hierarchical partitioning performed on a) top-level, and b) one quadrant.

In the partitioning algorithm, only the gates whose wires cross the partition boundary contribute to the cost function, and the goal of a min-cut partition is to reduce the total cost. In the field of computer-aided-design (CAD), the most commonly-used algorithms are the Kernighan-Lin [Kernighan70] algorithm and the modifications proposed by Fiduccia-Mattheyses [Fiduccia82]. The KL algorithm finds the best pair of gates, from two partitions, to swap between the two partitions, and repeat until the cost is minimized. The FM algorithm modified the KL algorithm to move only one gate at a time, and to keep a sorted "bucket list" of cost functions for more efficient memory management. Krishnamurthy then proposed an improved algorithm based on FM partitioning to provide "foresight" into moving gates [Krishnamurthy84]. The algorithm not only chooses the gate that will reduce the total wire cost, it is able to "foresee" wire costs. For example, if two gates need to be moved in order to reduce a wire cost, these two gates are assigned to a second-order "bucket list"; the same applies for third-order and higher. Therefore, even if multiple gates have the same effect on the first-order wiring cost, the second order "bucket list" can be used as tie-breakers, and the third order can be a tiebreaker for the second order. Although many other algorithms exist, such as simulated annealing [Bui89] or

ratio cut [Wei91, Hagen92], surveys have shown the techniques based upon KLFM can still produce better results [Hauck95].

Although partitioning is generally applied to bisections, called bipartition, multi-way partitioning has been introduced by Sanchis in [Sanchis89]. The first multi-way partitioning is to reduce the wire cross *any* boundary, thus wires with gates residing in 4 partitions, for example, are treated with the same cost as wires with gates residing in just 2 partitions. Sanchis later proposed a different cost function in [Sanchis93] where reducing the wire cost from 1 partition will still reduce the total wiring cost, which is more applicable for FPGA applications.

When hierarchical partitioning is used, it is best to keep the gates with boundary-crossing nets closer to the boundary, as shown in Figure 6.4. This reduces the wiring length for these gates. In Figure 6.4b), partitioning is performed on the quadrant shown, and it would be best for the gates in blue to remain in the lower right partition, because that results in the shortest wire-lengths when they need to communicate diagonally in the top level. To implement such location preference, a concept called “terminal propagation” from [Dunlop85] is implemented. Terminal propagation adds bias to a partition when the gate has higher-level nets closer to that partition.

For large-scale partition problems, clustering is an indispensable tool. Since the partitioning problem is NP-complete, the iterative algorithms are heuristic methods. When the problem size gets large, these algorithms tend to remain in local optima and often fail to find a global optimum [Cong93]. Additionally, it has been shown that FM-based method is more effective on gates on average have at least 6 nets [McMurchie95]. Such constraint is not always met in regular circuits, especially when 4-input LUTs dominate. Once clustering is performed, all these issues can be resolved – the problem size is reduced, and each gate cluster has far more than 6 nets on average. Many clustering techniques has been proposed over the years [Schuler72,

Garbers90, Hagen292, Cong93, Ding01], and they are generally used as a pre-partition process. After partitioning of the clustered gates are done, the unclustering step can be performed in one step, or be gradually "uncoarsened" into smaller clusters, performing additional FM iterations between each uncoarsening step [Karypis98].

Although clustering is indispensable for many partitioning algorithms, we do not find it very effective once LUT is packed into CLBs. In our new FPGA, each CLB can have anywhere from 42 to hundreds of connections. Although not all connections are utilized, the number of nets per gate is far more than 6, and FM-based methods work very effectively. LUT packing also reduces the effective number of CLBs to no more than 3,000, which is well-manageable by our software tool even without additional clustering.

In our partitioning implementation, we implemented a KLFM-based, iterative min-cut partitioning algorithm with terminal propagation. Both 2-way and multi-way partition are considered, and generally 2-way partitioning produces better results, and does not increase routing wire length when used with terminal propagation. Multi-way partition is still necessary because parts of the radix-3 architecture require 3-way partition at those hierarchies. Random-walk-based clustering was implemented initially, but is not used for partitioning CLBs after LUT packing. The only clustering that is implemented is to cluster CLBs that reside on a carry chain. In these cases, the CLBs must be placed in a particular order to ensure proper carry propagation between CLBs. These CLBs are clustered into a large cluster, and is only unclustered when the hierarchically-partitioned area of the next hierarchy is too small to fit the entire cluster. Upon unclustering, the CLBs are placed at the current partition. In many of our mapped designs, carry-chain CLBs can account for as much as 50% of all CLBs. As a result, further clustering has proved unnecessary. For the non-clustered gates, placement is performed when partitioning has

reached the bottom hierarchy, resulting in a partition size of 1 CLB. The corresponding CLB is then placed in the current partition.

6.4 FPGA Routing

Due to the large overhead of interconnect area, FPGA routing is performed on very limited routing resources. In our hierarchical FPGA design, the interconnect architecture is also designed to provide just sufficient routing resources to avoid area waste. As a result, FPGA routing places large emphasis on the quality of the software router. The router need to not only resolve all routing congestions, minimize critical-path, and complete the task in a reasonable (hours of less) run-time even for large designs.

As shown in Figure 6.5, the hierarchical interconnect architecture was implemented to have many path diversities, therefore improving connectivity. However, not all paths result in the same timing performance, as illustrated by the routing preferences. It is generally preferred to travel the shortest routings, using fast-path whenever possible, to reduce overall interconnect capacitance. But in the case of routing congestion, re-routing must be done, and some nets may be required to take non-preferred routes.

Modern routers generally employ global routing before detailed routing. The purpose of global routing is to provide a best-case timing performance of the design, and to estimate routing congestion. Being agnostic to routing congestion, the router is able to perform global routing very quickly, such as using the shortest-path algorithm [Nair82] and [Nair87]. In our hierarchical interconnects, the hierarchical architectures allows for very deterministic global routing. The router may utilize fast-path to perform no branching on the upward path, make a U-turn at the required hierarchy, and the downward path is very deterministic (computed by the radix-2

boundaries).

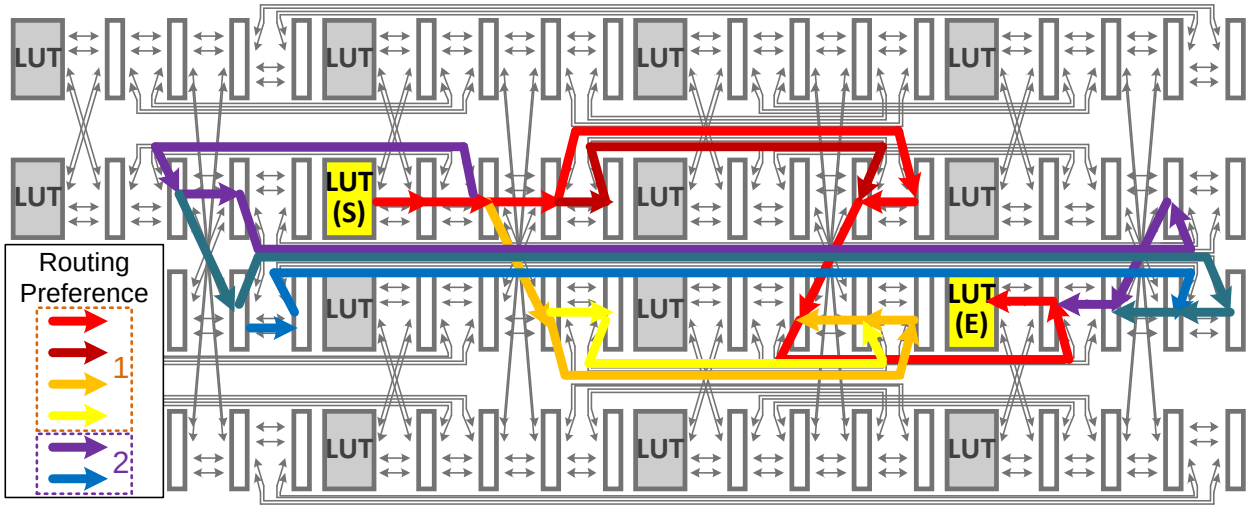


Figure 6.5: A routing-preference example for a point-to-point connection, LUT (S) to LUT (E).

Global routing gives the router valuable information, such as timing feasibility and routing congestion, but all congestions must be resolved for the design to be realizable. The initial version of our router employs rip-up-and-reroute detailed routing to resolve routing congestions [Dees81]. However, the algorithm we implemented was not timing-driven, and is dependent on the routing order of the nets. Therefore the routing results often have inconsistent timing, and sometimes fail to converge. Unsatisfied with our routing results, we implemented a new routing algorithm to the PathFinder router [McMurchie95, Ebeling95].

The PathFinder is a negotiation-based router that iteratively improves routing congestion by de-touring the lesser-performance-critical gates. It is able to incorporate global routing and detailed routing into a unified algorithm. The first iteration of the router is performed only based on interconnect delay, and not routing congestion, resulting in a minimum-delay design with many routing conflicts. However, the router does not attempt to rip-up the conflicting nets, instead it reroutes the design iteratively, but each successive iteration places a higher cost on routing conflicts. Eventually, the cost of routing through a faster, congested net will outweigh the

cost of routing through a slower, non-congested net, and the slower route will then be applied. Although the above approach resolves first-order congestion, it requires the second net to be rerouted when the resources are occupied by the first net. However, a good routing algorithm should be agonistic to the routing order of the nets, because sometimes the second net may not be able to re-route itself, but the first net could. A history factor is therefore implemented: every time a routing resource is congested, a history coefficient is incremented on the net. The history coefficient also contributes to the routing costs, and eventually, the cost of a congested resource may be high enough that the first net will not occupy that routing resource anymore, and the second net can then occupy the space.

Based on our implementation of the algorithm, the routing time has been significantly decreased, as shown in Table VI.I, and many previously-unroutable designs are now routable. This router has not only accelerated the mapping process, its improved quality-of-result has allowed us to further prune the interconnect network and realize a smaller interconnect area.

Design	Original Router	PathFinder Router
ex5p	> 2 hours	6 minutes
ex1010	> 1 day (not feasible)	41 minutes
clma	> 1 week (not feasible)	3 hours

Table VI.I: Routing time of our original router vs. PathFinder-based router.

Based on the improved routing algorithm, we further modified the PathFinder router for concurrent delay optimization and run-time improvements. The original PathFinder paper [McMurchie95] has suggested a congestion/delay-based router that computes the cost function as a weighted-sum of the interconnect delay and the total wiring cost. Note the wire cost not only include interconnect delay, but also costs from the congestion history and the current routing

congestion of the routing resource. The suggestion in [McMurchie95] is to compute the critical-path coefficient of each net, normalized to the worse-case delay. The coefficient is then used to weight the non-timing-critical routes to have a larger cost impact from routing congestion, while the timing-critical nets to have little cost impact from routing congestion, but mostly from routing delay. Using this method, timing-critical nets are routed without much regard for routing congestion, and the non-timing-critical nets would then detour around the critical nets to avoid occupying their routing resources. Although the paper suggested the coefficient can be as large 1, we find that to not always be feasible. A coefficient of 1 causes the router to be completely agnostic to routing congestion, thus two conflicting nets both with coefficient of 1 will never be resolved. We have limited the maximum coefficient to 0.99.

The PathFinder algorithm is based on Nair's algorithm from global routing [Nair87], which is a breadth-first search. To ensure routing quality, we cannot convert to a depth-first search, but reducing the search radius will significantly reduce its routing cost. As a result, at each intermediate node, the minimum cost from the intermediate node to the final destination is added to the current cost. This prevents the router from searching for too large of a radius before converging to a destination. This modification is called A* routing [McMurchie95, Tessier98], and has speed up the run-time of PathFinder by as much as 2x, as confirmed by our implementation.

In our implementation, we made key modification to the PathFinder router to further improve run-time. We noticed that the majority of the nets are not on the critical-path, and can be routed without timing-driven algorithms. We therefore use a non-timing driven algorithm to arrive at a feasible routing solution more quickly, and then enable timing-driven mode. The first iteration is still global routing, which is used to compute the minimum routing delay. Starting

from the second iteration, we completely ignore net delay in routing – all nets are assigned a delay cost of 0, unless a routing conflict occur. As a result, the router will always avoid routing conflict whenever a possible, because a routing conflicts results in a non-zeros wiring cost. Many intermediate nodes will then have a cost of 0. To avoid excessive search radius, the node that are the closest to the destination are chosen as the "wavefront" [Nair87] for the next iteration. This allows the router to converge to a feasible result very quickly. Once a feasible design is routed, timing is computed, and the critical-path coefficient is applied to each net. Timing-driven PathFinder is then applied on the net that exceed the coefficient threshold: for example, if the minimum-routable timing is 5 ns, but the current timing is 7 ns, the coefficient threshold is set to 71%; all nets with a critical-path coefficient greater than 0.71 is then re-routed. From our experience, this two-stage approach of congestion-followed-by-timing routing has produced equally good results as the original PathFinder algorithm, but in a faster runtime.

In more recent years, Boolean Satisfiability (SAT) approaches has been suggested for routing. There are efficient SAT solvers available, as listed in [Nam04], and unlike PathFinder, SAT-based routing simultaneously considers all routes, which allows higher degrees of freedom and potentially better routing results [Nam04]. However, the FPGA interconnect routing problem is often too large for SAT solvers, and the router in [Nam04] is limited to detailed routing, and unable to choose a different route when global-re-routing is required. We initially decided not to consider SAT-based routing for our FPGA, but a recent paper [Gort11] suggests a good compromise. PathFinder-based routing can converge very quickly to an almost-feasible solution, but spends most of its run-time resolving the few-remaining conflicts. Paper [Gort11] suggested using PathFinder to perform global routing and coarse detailed routing, where multiple tracks are routed together as a "coarse" track, and then use SAT-based formulations for detailed routing.

Although the coarse routing technique may not always apply to our hierarchical routes, the concept of a two-stage PathFinder-followed-by-SAT routing approach is interesting to consider.

6.5 Bitstream Generation

Based on the placement and routing results, the software tool has complete knowledge of the mapped design. The function of each LUT and the configuration of each CLB are determined by the netlist, and the switch matrix (SM) configurations are derived from the interconnect routing. We implemented bit-accurate information about the bit-cell mapping in all CLBs and SMs, as well as the word-line (WL) and bit-line (BL) information of each bit-cell. The tool then configures the individual bit-cells based on the place-and-route results, and power-gates the unutilized blocks when possible. The output bitstream is in a 2D-array following the WL and BL structure implemented on the chip. The testbench is able to stimulate the scan logic to shift in one row of BL at a time. The details of the testbench and measurements are in Chapter VII.

CHAPTER VII

Test Infrastructure and Measurement Results

7.1 Matlab Simulink-based Testing Infrastructure

Unlike testing dedicated chips, bringing up a reconfigurable hardware generally involves an extensive configuration process before the chip is able to execute data-processing tasks. Even our first "small" FPGA of 2048 LUTs involves close to 300,000 configuration bits, all of which must be set properly for correct functionality. When configuration bits are set improperly, it is often very difficult to isolate the exact location of the programming fault, making debugging process very difficult. This not only places a heavy emphasis on a proper bitstream generation process from the software tool, but also an importance on an easy-to-use, easy-to-configure testbench.

Due to the large number of configuration bits, the initial chip-bring-up process requires extensive iterations between bitstream generation and chip programming. It is therefore desirable to have the two processes in an integrated platform. We therefore employed a Matlab Simulink-based testing platform using the Interconnect Break-Out-Board (IBOB) developed by UC Berkeley and Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) [IBOB10]. The IBOB hardware is programmable using Matlab Simulink, as shown in Figure 7.1, and communicates to our 2048-LUT FPGA using 2 high-speed ZDOK+ connectors, each supporting 40 single-ended or 20 differential I/Os. The IBOB is reconfigurable through a Xilinx Virtex-II-Pro FPGA, capable of up to 300 MHz operation when fully pipelined. The IBOB platform is responsible for the scan-configuration for programming our FPGA, as well as the input generation and output capture.

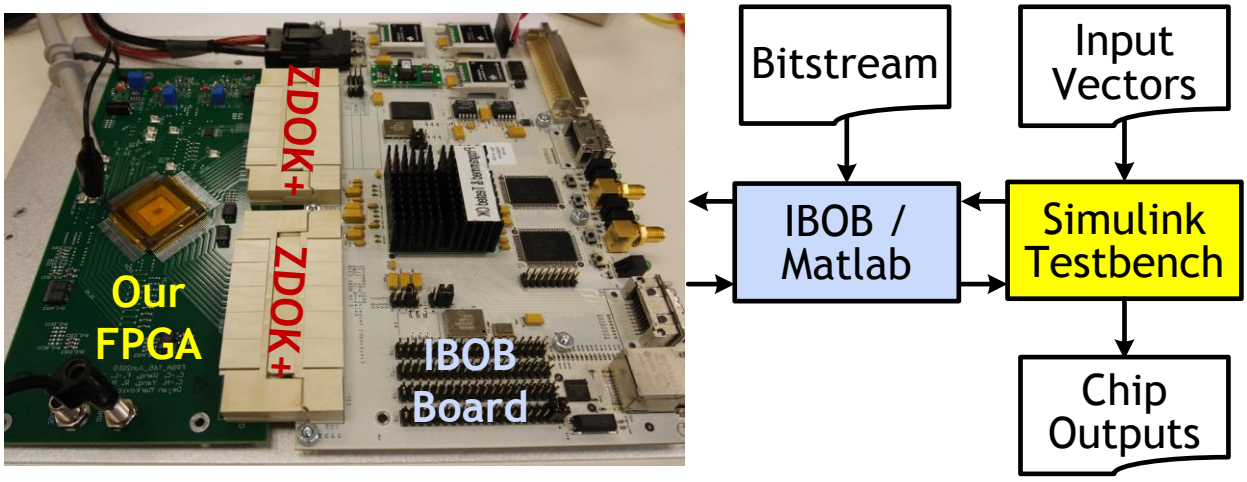


Figure 7.1: A IBOB platform use for Matlab Simulink-based testing infrastructure.

The Matlab-based testing platform provides a unified testing environment, because our software mapper is also developed in the Matlab environment. The bitstream can then be directly written to the block RAMs on the IBOB from within the Matlab environment. Similarly, input test vectors are generated in the Matlab environment, and written onto the IBOB block RAMs. The IBOB block RAMs are also used for captures the outputs of our FPGA, which can be read back into the Matlab environment for data analysis. The configuration of the IBOB platform is executed completely in the Matlab Simulink environment, as shown in a sample Simulink testbench in Figure 7.2. Because it operates on a Xilinx FPGA, most of the building blocks are compatible with Xilinx System Generator blocks, which are Simulink blocks that can be simulated in Simulink, as well as mapped onto a Xilinx FPGA. These blocks are colored in blue in Figure 7.2. In addition to the Xilinx FPGA, the IBOB integrates many peripheries such as clock source, on-board block RAMs, Ethernet interface, GPIO headers, and ZDOK+ connectors. To support a seamless integration with Simulink, IBOB support built-in Simulink blocks, shown in yellow in Figure 7.2 These IBOB interface blocks can be simulated in Simulink, and when mapped, can also instantiate the corresponding IBOB peripheral blocks.

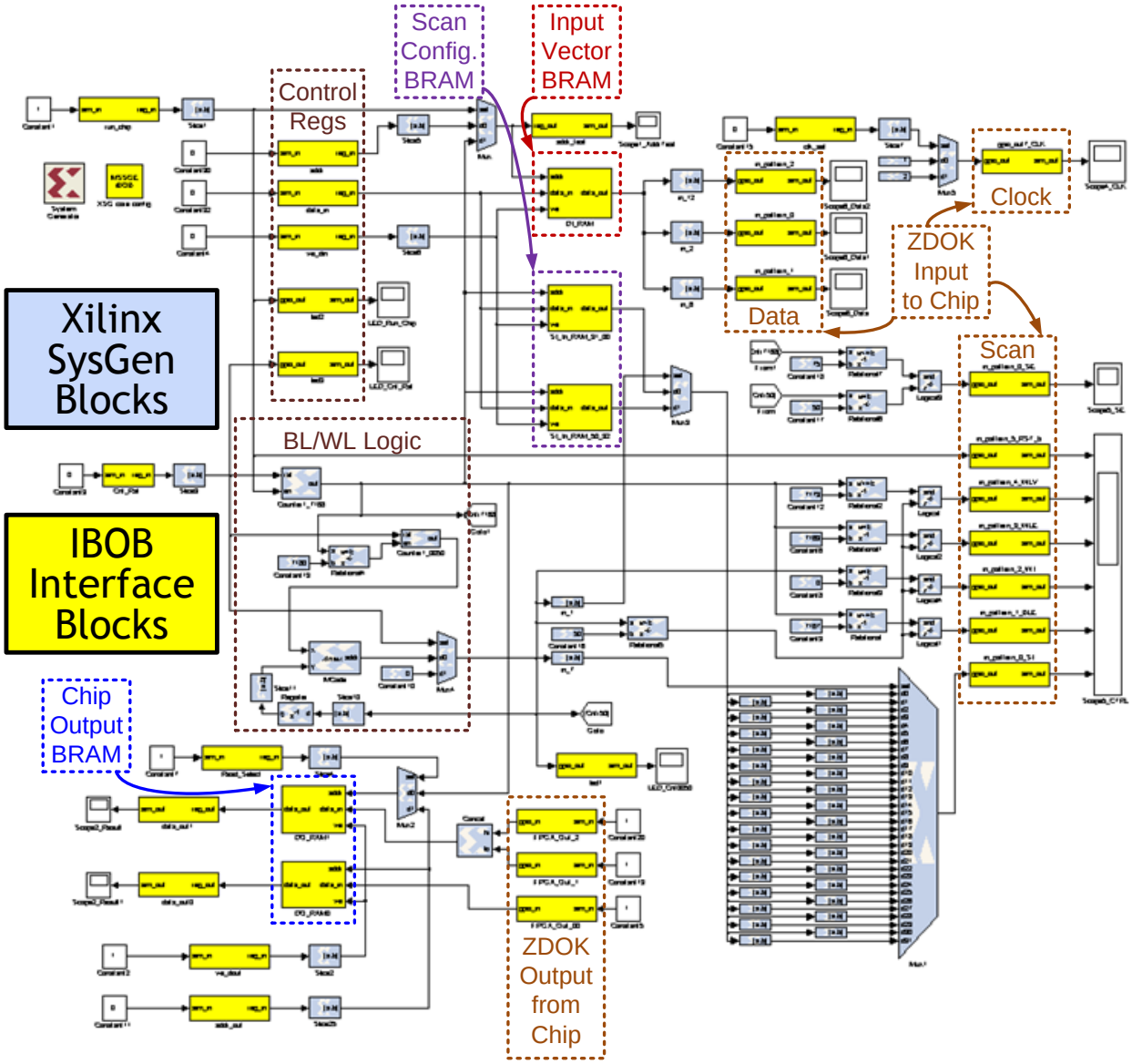


Figure 7.2: An example IBOB Simulink testbench for chip configuration and testing.

As shown in Figure 7.2, the entire testbench can be first simulated in Simulink, and many output scopes are placed to verify the logic functionalities. This is especially for the control logic of bit-lines (BL) and word-lines (WL). The testbench just scan in 7168 bits of the bitstream, pause BL scan chain, increment WL scan chain by one bit, perform the write, and perform the same operation 50 times to configure all 50 WLs. The IBOB block RAM (BRAM) has a maximum depth of 8192 by 32 bits, therefore 2 BRAMs are instantiated to contain all

configuration bits. Although the BL and WL bits reside on a 2D array, it is serialized line-by-line to fit into a 32-bit data width. The BL and WL logic is responsible for computing the correct location of each serialized bit during scan configuration.

The input BRAM on the IBOB is used to store test vectors. Because the I/O pins on our FPGA are reconfigurable, each I/O pin corresponds to a different I/O on the mapped design. For example, if the mapped design has a 16-bit input, our software tool will utilize 16 different I/O pins on the FPGA to accept these inputs, where each I/O pin is tied to a ZDOK+ connector pin on our testboard. Therefore the user must specify the input pin mapping from the IBOB BRAM to the ZDOK+ connector, else the testbench will not stimulate the correct input pins on our FPGA.

The same scenario applies for outputs of the chip, which is determined during design mapping to map to a set of I/O pins. These chip pins are tied to ZDOK+ connector pins on the testboard. The user needs to specify in the testbench the ZDOK+ pins based on a pin-out mapping table. The output bits are then stored in the on-board IBOB BRAM.

Before the chip is powered up, the Simulink testbench is first mapped to a bitstream, and the IBOB testboard FPGA is configured via standard JTAG interface. Configuration bits and input test-vectors are then written into the BRAM on the IBOB via a serial interface, while IBOB control registers are used to hold the chip in its reset state before power-up. The scan mode is then initiated to write to all bit-cells on our FPGA, one row at a time, controlled by the BL/WL logic on the IBOB. Once configuration mode is complete, the configuration registers are set to place the chip out of reset mode and start normal operation.

For designs that require a more elaborate testbench, the Reconfigurable Open Architecture Computing Hardware (ROACH) [ROACH13], also developed by CASPER,

provides an upgraded capacity using a Xilinx Virtex-5 FPGA. The ROACH blocks in Simulink are very similar to IBOB blocks, and a separate PowerPC processer is placed on board to allow for much faster interface between the BRAMs/FIFOs on the FPGA and the external devices, such as Matlab, using Ethernet instead of serial port. In terms of hardware capability, the same two ZDOK+ connectors on our testboard also fits onto a ROACH platform.

7.2 Measurement Results of our 2048-LUT FPGA

Many designs are mapped onto our FPGA to verify functionality and performance, even successfully mapping designs-of-interest from commercial companies. Table VII.I illustrates the measurement results from 4 key designs we have mapped. Like any hardware, area efficiency is maximized when most of the blocks are utilized (thus why "dark silicon" is inefficient). Our chip achieves 16.4 GOPS/mm^2 when all Logic and DSP CLBs are utilized; executing 175 16b accumulators at 370MHz. Since a 16b adder can be implemented with 2 DSP CLBs but requires 4 Logic CLBs to propagate the carry chain, the DSP adders are faster, reaching 400MHz. However, the area efficiency of the DSP-only accumulator is lower, because the logic CLBs are mostly idle, additionally, energy efficiency of the DSP-only CLBs are also lower due to idling CLBs. Leakage is well-controlled even without power gating. A 1.08 GOPS/mW is attainable with only 112 DSP accumulators active and most of the Logic CLBs idle.

Accumulators are implemented because the IBOB testbench is only capable of reaching 300 MHz even with a fully-pipelined testbench, it is a limitation of both the FPGA hardware and the FPGA PLL. The ROACH platform [ROACH13] is unable to run faster due to its Block RAM timing, which is un-pipelined. Since we are unable to toggle any inputs beyond 300 MHz, the inputs are held constant, and our FPGA is tied to an external SMA clock source to perform the

accumulator operation. Without BRAM capabilities for data capturing, the functionality of the accumulator is examined on the oscilloscope. Performance is hindered by equipment limitations due to a 0.25ns input-clock jitter at 400MHz.

Result / Design	Resource Utilization			Performance			
	Logic (256)	DSP (224)	BRAM (16)	V_{DD} (V)	Power (mW)	Freq. (MHz)	GOPS /mW
175 L/DSP 16b Acc.	256	224	0	1.0 0.5	179 8.6	370 55	0.36 **1.13**
112 DSP 16b Acc.	4	224	0	1.0 0.51	123 6.2	**400** 60	0.57 1.08
32-tap 16b FIR	132	209	0	1.0 0.56	120 10.2	274 50	0.21 0.45
2x2 MIMO 64-pt FFT	196	93	10	1.0 0.78	82.7 26.5	83 40	0.05 0.07

Table VII.I: Key measurement results from our 2048-LUT FPGA chip.

A 32-tap finite-impulse-response (FIR) filter is mapped to utilize most of the DSP CLBs, configured for a 16-bit datapath. The FIR filter achieves 274MHz due to longer routing, but interconnect delay is still under 50%. Fast-Fourier-Transform (FFT) hardware generally require extensive memory usage to implement the delay lines. A 2×2 MIMO 64-point is mapped to exercise most of the BRAM CLBs, as well as most of the Logic CLBs for FIFO control and butterfly implementations. With many control signals and a critical path of 11 CLBs, the FFT achieves 83MHz.

The energy efficiency (GOPS/mW) shown is computed for 16-bit arithmetic operations. Not surprisingly, the FPGA produces higher efficiency when mapping more arithmetic-heavy designs, such as accumulators and filters, while efficiency is significantly lower for control-heavy designs such as FFT and microcontrollers. The energy-delay curve and power-breakdown

of the FPGA when mapping the 175-accumulator design are shown in Figure 7.3. Minimum-energy point for the accumulator occurs at 0.5V, which happens when the leakage increments caused by a slower design offsets the energy reduction. For other designs where leakage occupies a larger percent of the total power, the minimum energy-point occurs at a higher VDD.

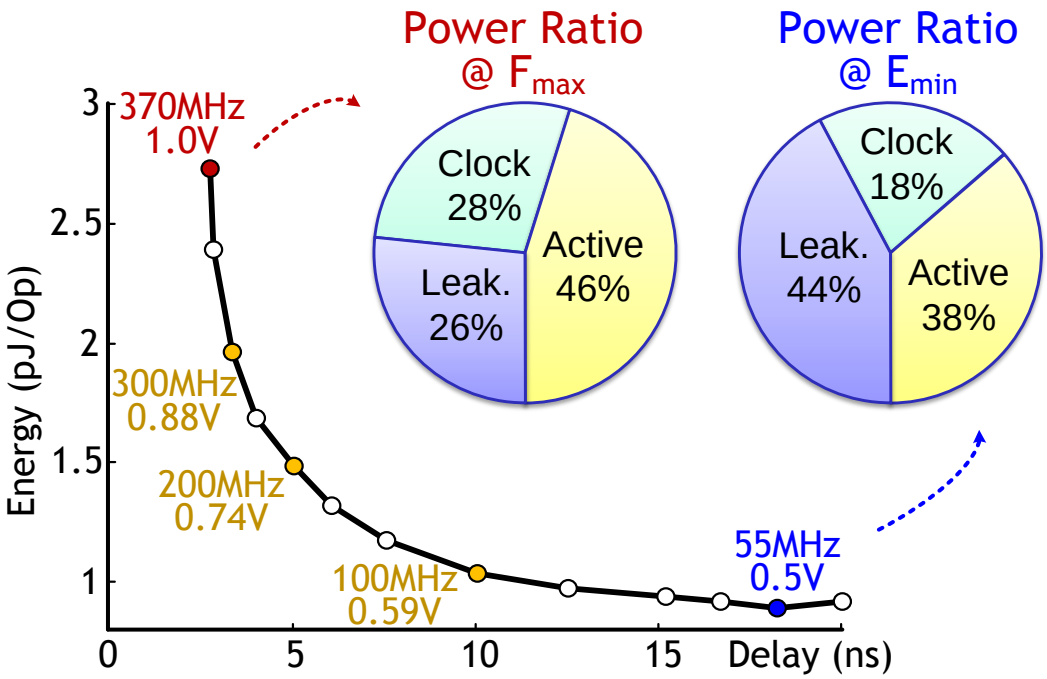


Figure 7.3 – Energy efficiency and power ratio at maximum frequency and minimum energy.

This chips achieves a maximum energy efficiency of 1.13 GOPS/mW. In comparison, Intel's work [Agarwal10] has no interconnects and lower energy efficiency, though the full-custom CLB in 32-nm LVT is 2.5x faster. It achieves 2.6 GOPS/mW at 0.34V for 8b operations, which is 0.65 GOPS/mW for 16b (2 CLBs per operation at half the speed). In comparison, our 65nm chip achieves 1.7x the energy even including interconnect power and delay (Figure 7.4). Among commercial FPGAs, the highest-reported efficiency is 0.05 GOPS/mW from an Altera Straix IV [George11]. In comparison, our chip is 22x more energy efficient even with an older process technology.

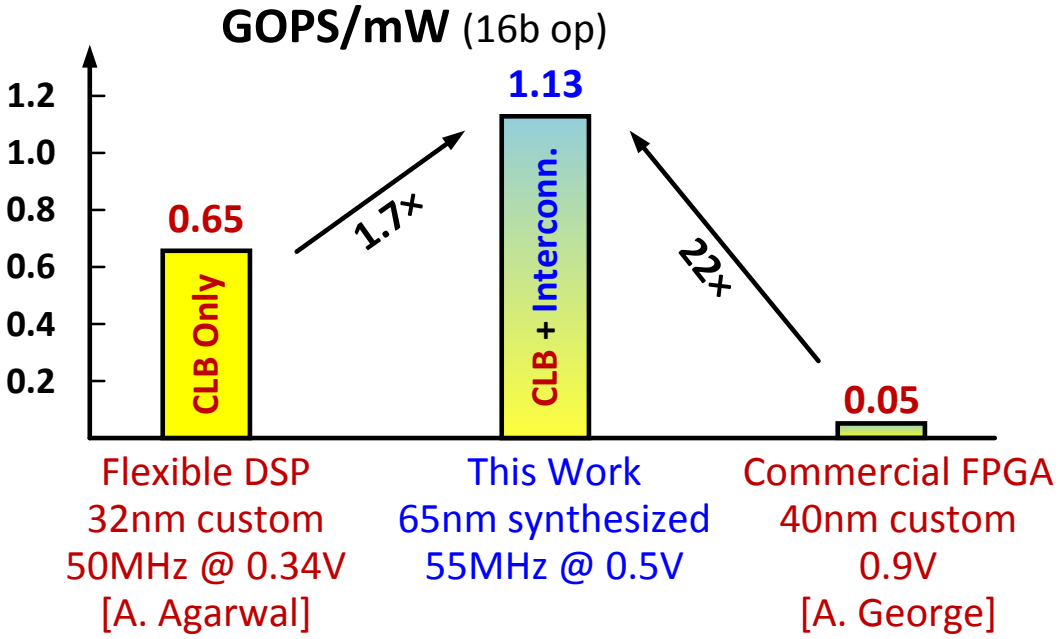


Figure 7.4 – Comparison of energy efficiencies against state-of-the-art reconfigurable hardware.

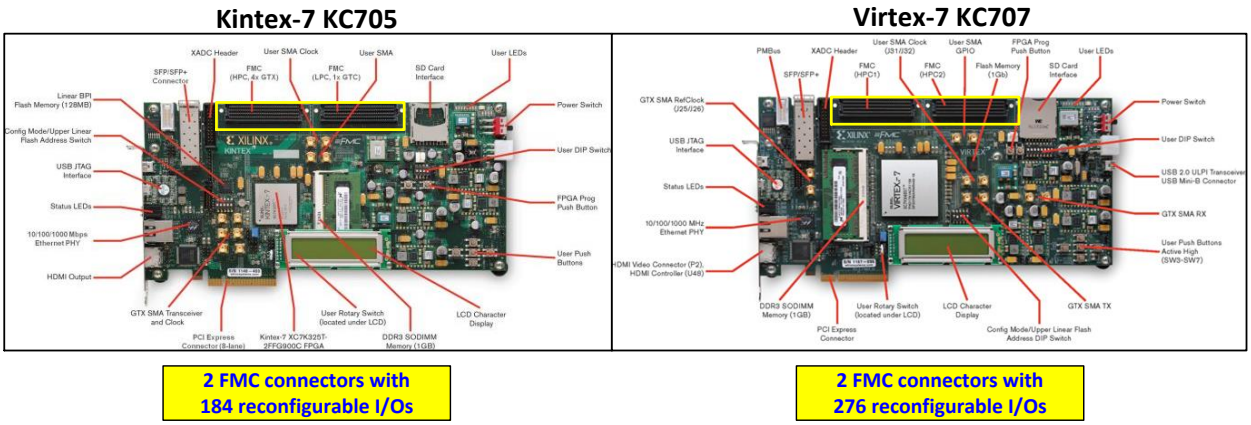
Although our FPGA is far more efficient than commercial FPGA today, it can only maintain its efficiency when it is mapping designs that effectively utilize its architecture. Based on our experience in mapping designs onto our FPGA, computation-intensive designs such as finite-impulse-response (FIR) filters and arithmetic blocks such as parallel accumulators are highly efficient when mapped onto our FPGA. However, control-heavy designs such as Fast-Fourier Transform (FFT) cannot be mapped very efficiently. As shown in Table VII.I, the measured energy efficiency of a FFT is up to 16x lower than the maximum efficiency of our FPGA.

Fortunately, because many of these inefficient blocks are core blocks, we can implement some of these designs as coarse-grain accelerator cores on our FPGA instead of using the fine-grained DSP CLB. These observations motivated us to implement the 64 – 8192 point programmable FFT and the 16-core Universal DSP accelerator on our 16K-LUT FPGA. We have also decided to implemented coarser, medium-grain DSP CLBs to implement arithmetics

and multiplications more efficiently than fine-grain DSP CLBs. Even though medium- and coarse-grain accelerators are still not as efficient as fully dedicated chips, embedding them on the FPGA enables us to push the efficiency gap much closer.

7.3 Updated Testing Infrastructure

Our 16K-LUT FPGA requires even higher performance and a larger number of I/Os from the testing platform. Therefore the IBOB and ROACH boards, supporting 80 singled-ended data pin through ZDOK+, are no longer sufficient for our testing. In search for a new hardware, we would also like to preserve the compatibility with Matlab Simulink interface to simplify testbench development. The Xilinx evaluation platforms for based on the newest 7-series FPGAs are an ideal candidate. Two suitable platforms are shown in Figure 7.5. The Virtex-7 platform offers more than 50% additional capacity, and provides 276 reconfigurable I/Os through its two FPGA-Mezzanine-Card (FMC) connectors. For our applications, we decided to go with the Kintex-7 platform for our initial testing. It is less than half the price of the Virtex-7 platform, and offers more than sufficient capacity for our testbench purposes. Although our chip has 350 programmable I/Os, our initial board design only bonded 180 I/O pins due to limitations of chip-on-board bonding. All Xilinx 7-series FPGAs are marketed with high-performance DSPs to run up to 600 MHz [Mehta12].



a) b)

Figure 7.5 –Xilinx evaluation platforms – a) Kintex-7 KC705 and b) Virtex-7 KC707.

Based on our choices of Kintex-7 KC705 evaluation board, we designed a new testboard to interface to the FMC connectors on the test platform. Due to the high pin-count requirement of this FPGA even when bonding just a subset of I/Os (194 signal pins and 150 power/ground pins), there is no commercially-available package that offers enough performance (500 MHz or more), enough pin-counts, and a small-enough cavity to avoid long bond wires. As a result, we migrated to a chip-on-board solution, where the chip is mounted and bonded onto the board without a package (Figure 7.6).

Chip-on-board wire-bonding poses many design limitations on the PCB. The board needs to be small enough to fit inside the wire-bond machine, thus constraining our size (to 6 inches by 2.5 inches in this case). The backside of the board needs to have no components when bonding. The components on the top must also not exceed a height a constraint. Due to the height of the FMC connectors, the board must be assembled after the chip is bonded and protected in a hermetic seal.

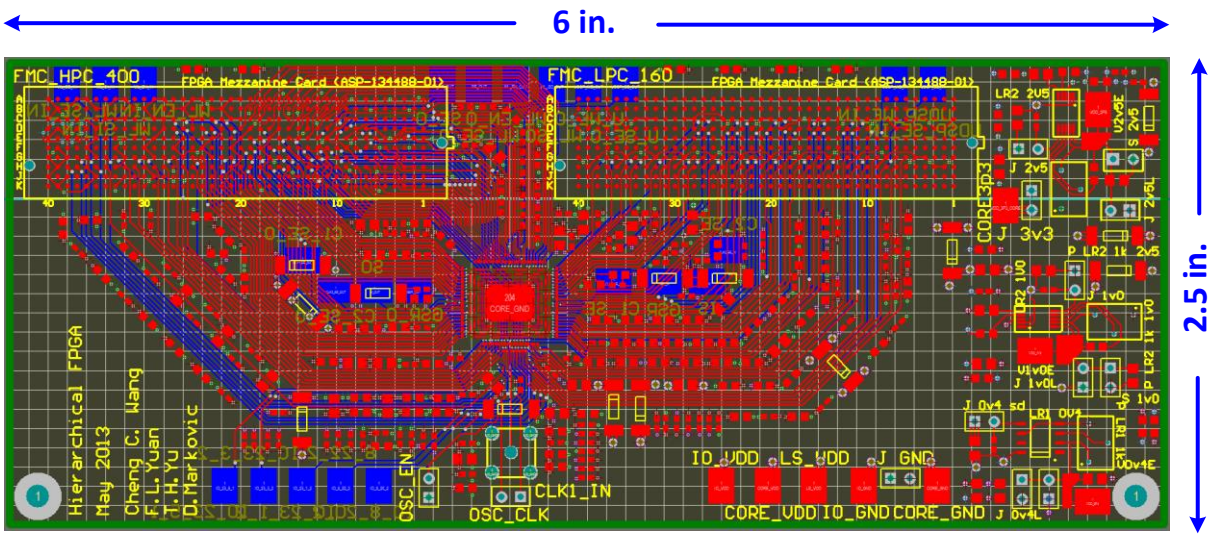


Figure 7.6 – Board layout of the chip-on-board testboard with two FMC connectors.

7.4 Measurement Results of our 16K-LUT FPGA

As of May 2013, the chip testing of the new FPGA is not yet complete, though we have fully debugged our 9-Mb bitstream generation process, and verified all standalone blocks on the chip for functionality. To estimate actual performance, we chose the most conservative timing estimations from chip design (0.8V, slow-slow corner, 125ºC) and added the timing information into our software mapper to perform a preliminary timing estimation. The propagation delay for signal path can be determined by tracing the interconnect path that the signal travels, and summing the delays for each wiring segment (from one SM to the next SM) to compute the net delay. The power information, however, depends on the activity factor of the mapped design, and cannot be easily estimated by modeling. The FPGA mapping results are compared with the post-place-and-route timing from a Xilinx Virtex-6 LX75T, also in 40-nm technology. Although LX75T is the smallest Virtex-6 offered, it still has more than 4 times the number of LUTs than our FPGA, and 7 times the number of DSP elements, along with much more BRAM in both

18Kb and 36Kb variants. Measurement results from their ASIC counterparts, when available, are also included.

The performance estimations of our chip, compared with post place-and-routes timing estimations from Xilinx Virtex 6 and ASIC implementation are shown in Table VIII. Even with very rudimentary software tools for place and route, along with a non-full-custom chip design, we are able to achieve performance similar to commercial FPGAs. We are actively improving our place-and-route software to achieve better quality-of-results. Based on our prior chip results, we can expect 10 – 20x improvements in power consumption and energy efficiency compared to commercial FPGAs. The ASIC implementation of the 2x2 MIMO 256-pt FFT is from [Yuan08] , and the Digital Baseband Process is from [Nanda12].

Result / Design	Resource Utilization				ASIC		Virtex-6	Our Performance			
	Slice M (576)	Slice L (2176)	DSP (42)	BRAM (16)	Freq. (MHz)	Process Tech.	Freq. (MHz)	V_{DD} (V)	Power (mW)	Freq. (MS/s)	GOPS /mW
Viterbi Decoder	0	1010	0	8	– –	– –	193	0.9 0.5	- -	150 -	- -
2x2 MIMO 256-pt FFT	1	2571	4	0	80	180nm	131	0.9 0.5	- -	100 -	- -
Digital Baseband Processor	13	2100	40	0	56	65nm	66	0.9 0.7	- -	30 -	- -
MSP430 Microcontroller	0	425	1	0	25	130nm (ULP)	85	0.9 0.6	- -	50 -	- -
x86-compatible Microprocessor	0	797	1	0	10	3um	98	0.9 0.6	- -	60 -	- -

*Synthesis + P&R Estimates

Table VII.II: Chip performance comparison against commercial FPGA and ASIC implementations, based on design mapping and conservative timing estimations.

Although the ASIC designs are spread across multiple technologies, we may observe that FPGAs performance are within the margins of ASIC designs, at least for these smaller design examples. When design complexity grows, the interconnect delay of commercial FPGAs began to show its effects, especially for control-heavy designs. The 8192-point FFT design

implemented on our FPGA is also mapped onto a large Xilinx Virtex-6 FPGA. Our FFT was designed for 400-500 MHz per core, thus capable of aggressive voltage scaling due to its 16-way parallelism. In contrast, the commercial FPGA can only achieve 54 MHz while taking a very large area footprint. For efficient implementation of FPGAs as ASIC replacements, it is essential to have coarse-grain reconfigurable blocks.

Result / Design	Resource Utilization (Virtex-6)					Our Performance			
	Slice M	Slice L	DSP	BRAM	Freq. (MHz)	V_{DD} (V)	Power (mW)	Freq. (MS/s)	GOPS /mW
8192-point 16-way parallel FFT	893	19362	295	17	54	0.9 0.5	- -	400 40	- -
Spherical Decoder (UDSP)	--	--	--	--	--	0.9 0.5	- -	450 50	- -

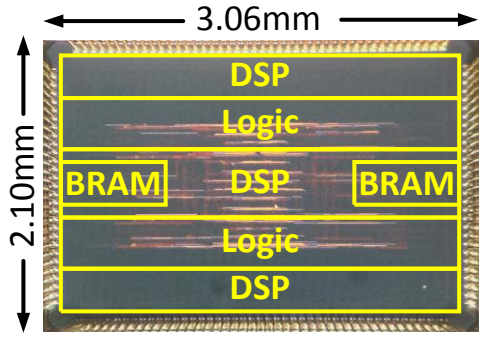
*Synthesis + P&R Estimates

Table VII.III: Coarse-grain accelerator performance against commercial FPGA implementations.

As discussed in Section 5.5, the 16-core Universal DSP processor is capable of performing many common communication applications. A conservative post place-and-route estimate of 450 MHz throughput is achievable. Because the UDSP design is mapped using a reconfigurable instruction-set-architecture (ISA), no HDL is used, thus its hardware-equivalent design is not available.

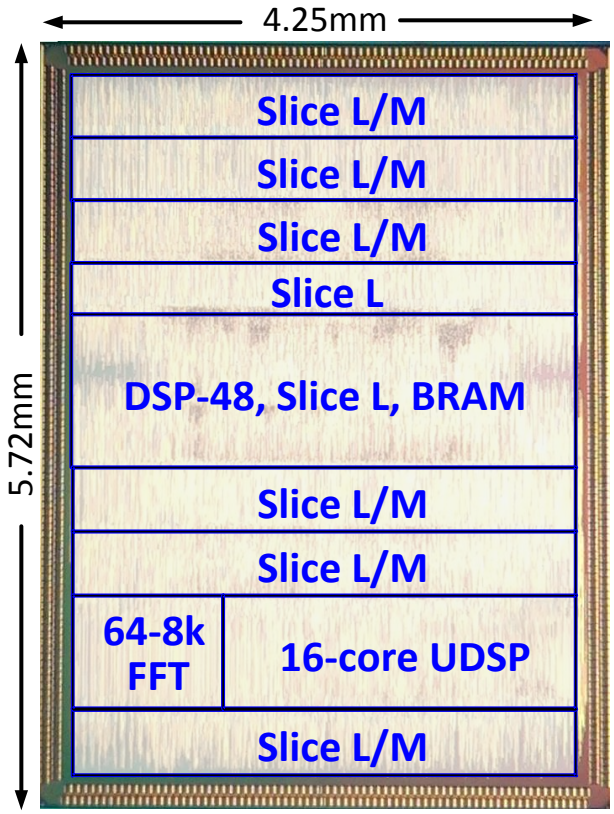
7.5 Chips Summary and Die Photos

This chapter is highlighted the measurement results and performance characteristics of two of our FPGAs: a 2048-LUT FPGA (Figure 7.7a) and a 16K-LUT FPGA with heterogeneous, multi-granularity reconfigurable blocks (Figure 7.7b). There die photos are shown on the same scale.



Technology	65nm 1P9M CMOS
Core V_{DD}	0.34 to 1.0V
Frequency	40 to 400 MHz
I/Os	75 bidirectional
Core Size	2.52mm × 1.56mm
Gate Count	2.73M
CLB Count	256 Logic, 224 DSP
Block RAMs	16 128×8b
Config. Bits	297,472b

a)



Technology	40nm 1P10M CMOS
Core V_{DD}	0.3 to 0.9V
Frequency	50 to 1000 MS/s
I/Os	350 bidirectional
Core Size	3.84mm × 5.31mm
Gate Count	35.75M
L/M CLB	2176 Logic, 576 Mem
DSP Accel.	21 DSP
Block RAMs	16 36Kb
Config. Bits	297,472b

b)

Figure 7.7 –Chip photo and summary of a) our 2048-LUT FPGA and b) our 16K-LUT FPGA.

The newer design is not a mere expansion in size. It demonstrates that hierarchical interconnect can scale to large (10x) the complexity without occupying additional area overhead. It also demonstrates the need for coarser-grain reconfigurable blocks, especially for control-

heavy designs. Once matched with a more mature place-and-route software, we believe this hierarchical FPGA can pose significant improvements over commercial FPGAs in performance, along with orders-of-magnitudes improvement in energy efficiency.

CHAPTER VIII

Conclusion and Future Outlook

8.1 Concluding Remarks

Efficiency is important. Efficient hardware has led to unprecedented possibilities of mobile and personal computing. It is driven by the ever-more stringent product requirements on chip features, battery life, and heat dissipation. It is realized by a vertical integration of suitable architecture, advanced circuit design, and easy programmability. Recapping from the introduction, chips today are efficient when implement as dedicated hardware, or ASICs. But ASICs are not programmable, so they are integrated with microprocessors as a system-on-a-chip (SoC). Although today's large SoCs provide efficient, programmable designs, it come at a great price: the growing feature requirements, standard changes, and design fixes require frequent re-design, and the cost of chip design is increasing with every technology generation.

Many companies, especially FPGA companies, have suggested FPGA as a cost-effective replacement for today's ASICs. FPGAs are very flexible, but are orders-of-magnitude less efficient then ASICs. In order to be realized in today's SoCs, large improvements on energy and area efficiencies must be made.

This work noted FPGA interconnects as the bottleneck for its inefficiency: often accounting for over 80% of the chip area, delay, and power. A method for constructing FPGA interconnects using hierarchical network is proposed in architecture, realized in silicon, and tested in mapping user-designs. To demonstrate the scalability of hierarchical interconnects, two chips are demonstrated in this thesis with 10x difference in interconnect complexity, both with interconnects occupying 51-52% of total area – a 3–4x reduction from commercial FPGAs.

Compared to commercial FPGAs today, we have achieved one order-of-magnitude improvements in energy and area efficiencies, approaching the efficiencies of ASIC designs (Figure 8.1).

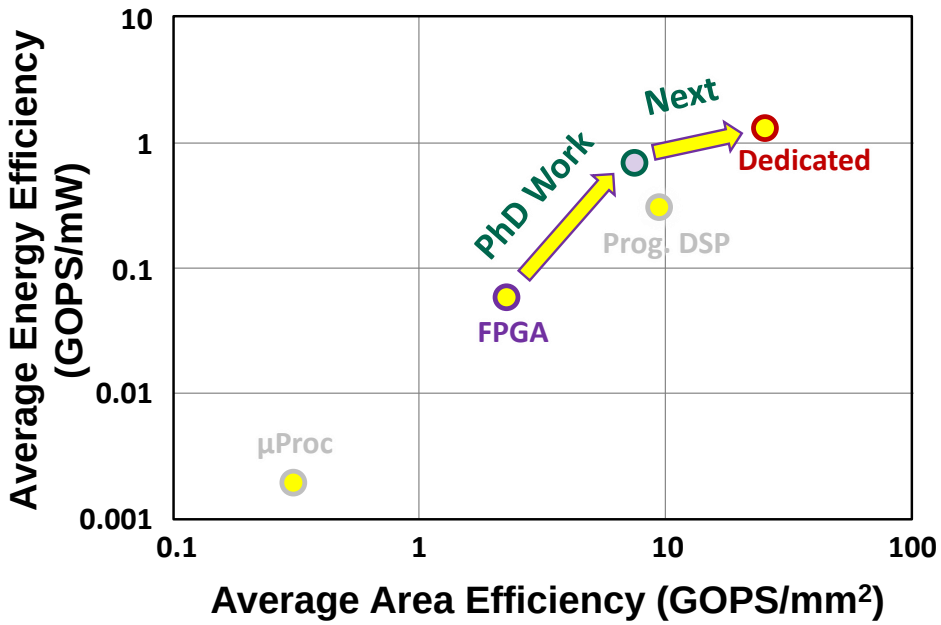


Figure 8.1: Energy and area efficiency from modern VLSI chips and our chips.

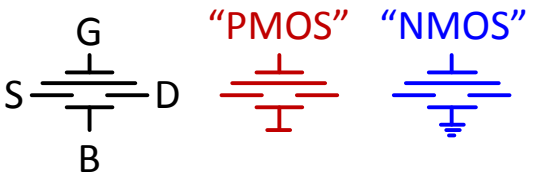
In addition to improvements interconnects, we realized that some arithmetic-heavy designs can be mapped efficiently in FPGAs, while other control-heavy designs cannot. We therefore implemented coarse-grain, reconfigurable accelerators for our applications. Although reconfigurable accelerators are still not as efficient as fully dedicated chips, embedding them on the FPGA enables us to push the efficiency gap even closer. Reconfigurable hardware will never be as efficient as ASICs, but given a small enough penalty in efficiency and performance, we are convinced that the reduced design-time and cost of reconfigurable hardware will make a strong candidate for ASIC replacement.

8.2 Outlook: Nano-Electrical-Mechanical Devices

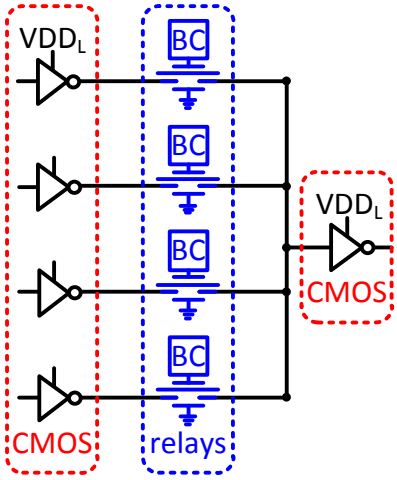
Many people believe CMOS technology is here to stay, at least for the near decade. It is true that there are no promising devices in the near horizon that can provide the speed, density, and robustness of CMOS technologies, which our industry has been heavily investing in for almost 50 years. For FPGAs, however, having fast and dense logic gates are not sufficient; we also need fast and dense interconnect switches. This is where nano-electro-mechanical (NEM) relays can come in.

Micro-electro-mechanical relays are very ideal for interconnect switches. A single relay is able to function as a "NMOS" and a "PMOS" depending on its body voltage (Figure 8.2a) [Chen10, Spencer11]. Although they are considered slow due to their long mechanical delay, because it takes micro-seconds to physically open and close the gate. However, such delay is irrelevant for our static pass-transistor mux (Figure 8.2b), because the gate remains stationary after programming. The same scenario applies for bit-cells: we can construct NEM-based SRAM cells (Figure 8.2c), and for controlling static muxes, the actual SRAM performance is irrelevant.

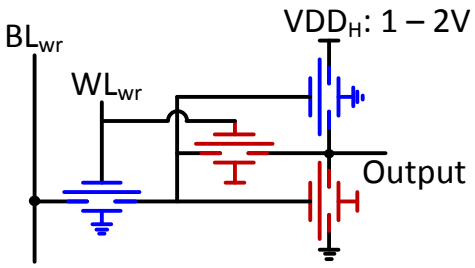
The major benefit of NEM based interconnect is the low on-resistance (< 1kΩ) of NEM switches, which drastically improves the interconnect speed. Since NEM switches provides an ohmic contact, the on-resistance remains constant throughout all regions of transistor operation. The interconnect performance is therefore unaffected by voltage scaling on the core transistors. Additionally, the NEM switch provides 0 off-current, thus drastically reducing the overall chip leakage by at least 2x.



a)



b)



c)

Figure 8.2: NEM relays as a) PMOS and NMOS-equivalent devices, b) a static switch, and c) a SRAM bit-cell (BC).

Another benefit of NEM relays is its simplicity of fabrication: no silicon substrate is required, and fabrication can be done with just 2 metal layers. Therefore, relays can even be "stacked" on top of CMOS designs. Ideally, the CLB and core logic can be implemented in CMOS, and the NEM-based interconnect can be stacked on the relays layer, effectively cutting chip area by 2x! We have already designed relays-based FPGA interconnects (currently in fabrication), and this seemingly far-fetched idea may not be that far into the future!

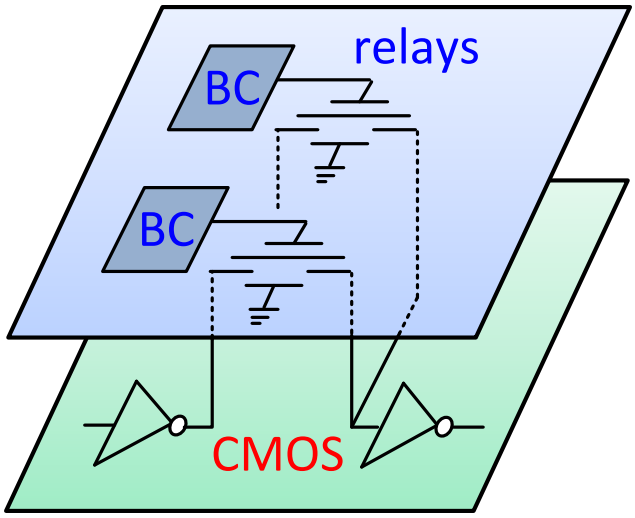


Figure 8.3: A relay-interconnect concept with CMOS logic on the bottom and NEM-interconnects on the top 2 metal layers.

Conclusion and Future Outlook

[Agarwal10]
A. Agarwal , et al, "A 320mV-to-1.2V On-Die Fine-Grained Reconfigurable Fabric for DSP/Media Accelerators in 32nm CMOS," *Proc. IEEE Int. Solid-State Circuits Conf.*, pp. 328-329, Feb. 2010.

[Ahmed10]
S. Z. Ahmed, G. Sassateli, L. Torres, L. Rouge, "Survey of new trends in industry for Programmable hardware," *Int. Conf on FPGA*, Aug. 2010.

[Alfke07]
P. Alfke, "20 Years of FPGA Evolution," *Hot Chips 19*, Aug. 2007.

[Anderson04]
J. H. Anderson, F. N. Najm, "A Novel Low-Power FPGA Routing Switch," *IEEE Custom Int. Circuits. Conf.*, pp. 719-722, Sep. 2004.

[Benes62]
V. E. Benes, "Heuristic Remarks and Mathematical Problems Regarding the Theory of Switching Systems," *Bell Syst. Tech. J.*, vol. 41, pp. 1201-1247, 1962.

[Betz97]
V. Betz, J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *IEEE Custom Int. Circuits Conference*, pp. 551-554, Sep. 1997.

[Betz98]
V. Betz, J. Rose, "How much logic should go in an FPGA logic block?" *IEEE Design & Test of Computers*, vol. 15, no, 1, pp. 10-15, Jan. 1998.

[Bolsens06]
I. Bolsens, "Programming Modern FPGAs," *Int. Forum on Embedded Multiprocessor SoC*, Keynote, Aug. 2006.

[Brown92]
S. D. Brown, *Routing Algorithms and Architecutres for Field-Programmable Gate Arrays*, Ph. D. Thesis, University of Toronto, Jan. 1992.

[Bui89]
T. Bui, C. Heigham, C. Jones, T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms," *Proc. Design Automation Conf.*, pp. 775-778, June 1989.

[Calhoun10]
B. H. Calhoun, J. F. Ryan. S. Khanna, M. Putic, J. Lach, "Flexible Circuits and Architectures for Ultralow Power," *Proc. of the IEEE*, vol. 98, no. 2, pp. 267-282, Feb. 2010.

[Carlson04]
I. Carlson, S. Andersson, S. Natarajan, A. Alvandpour, "A high density, low leakage, 5T SRAM for embedded caches," in *Proc. European Solid-State Circuits Conf.*, pp. 215–218, Sep. 2004.

[Chen10]
F. Chen, et al, "Demonstration of Integrated Micro-Electro-Mechanical Switch Circuits for VLSI Applications," in *Proc. IEEE Int. Solid-State Conference*, pp. 26-27, Feb. 2010

[Clos53]
C. Clos, "A Study of Non-blocking Switching Networks," *Bell Syst. Tech. J.*, vol. 32, pp. 406-424, 1953.

[Cong93]
J. Cong, M. Smith, "A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design," *Proc. Design Automation Conf.*, pp. 755-760, Jun 1993

[Curd07]
D. Curd, "Power Consumption in 65nm FPGAs", *White Paper: Virtex-5 FPGAs WP246*, Xilinx Inc., Feb. 2007.

[Dally04]
W. J. Dally, B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[Dees81]
W. Dees, R. Smith, "Performance of Interconnection Rip-Up and Reroute Strategies," *Proc. Design Automation Conf.*, pp. 382-390, June 1981.

[DeHon99]
A. DeHon, "Balancing Interconnect and Computation in a Reconfigurable Computing Array," *ACM Int. Symp. on FPGA*, pp. 69-78, Feb. 1999.

[DeHon00]
A. DeHon, "Compact, Multilayer Layout for Butterfly Fat-tree," *Proc. ACM Symp. on Parallel Algorithms and Archi.*, pp. 206-215, 2000.

[DeHon04]
A. DeHon, "Unifying Mesh- and Tree-Based Programmable Interconnect," *IEEE Trans. on Very Large Scale Int. Systems*, vol. 12, no. 10, pp. 1051-1065, Oct. 2004.

[Devlin11]
B. Devlin, M. Ikeda, K. Asada, "A 65nm Gate-Level Pipelined Self-Synchronous FPGA for High Performance and Variation Robust Operation", *IEEE J. of Solid-State Circuits*, vol. 46, no. 11, pp. 2500-2513, Nov. 2011.

[Ding01]
C. H. Q. Ding, et al, "A Min-max Cut Algorithm for Graph Partitioning and Data Clustering," *Proc. Int. Conf. on Data Mining*, pp. 107-114, 2001.

[Donovan10]
J. Donovan, "ARM CTO warns of dark silicon," *EE Times*, March 22, 2010 http://www.eetimes.com/electronics-news/4136890/ARM-CTO-warns-of-dark-silicon

[Duato02]
J. Duato, S. Valamanchili, L. Ni, *Interconnection Networks, An Engineering Approach*, Morgan Kaufmann, 2002.

[Dunlop85]
A. E. Dunlop, B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on Computer-Aided-Design*, vol. CAD-4, no. 1, pp. 92-98, Jan. 1985.

[Ebeling95]
C. Ebeling, L. McMurchie, S. A. Hauck, S. Burns, "Placement and Routing Tools for the Triptych FPGA," *IEEE Trans. on Very Lage Scale Int. Systems*, vol. 3, no. 4, pp. 473-482, Dec. 1995.

[Fiduccia82]
C. M. Fiduccia, R. M. Mattheyses, "Partitioning Very Large Circuits Using Analytical Placement Techniques," *Proc. Design Automation Conf.*, pp. 175-181, June 1982.

[Garbers90]
J. Garbers, H. J. Promel, A. Steger, "Finding Clusters in VLSI Circuits," *Proc. Int. Conf. on Computer-Aided Design*, pp. 520-523, Nov. 1990.

[George11]
A. Geroge, H. Lam, G. Stitt, "Novo-g: At the Forefront of Scalable Reconfigurable Supercomputing," *IEEE Computing in Science and Eng.*, vol. 13, no. 1, pp.82-86, Jan. 2011.

[Gort11]
M. Gort, J. H. Anderson, "Reducing FPGA Router Run-Time Through Algorithm and Architecture," *IEEE Int. Conf. on Field Prog. Logic and App.*, pp. 336-342, Sep. 2011.

[Goulding11]
N. Goulding-Hotta, et al., "The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future," *IEEE Mirco*, vol. 31, no. 2, Mar. 2011.

[Greenberg88]
R. I. Greenberg, C. E. Leiserson, "A compact layout for the tree-dimensional tree of meshes," *Appl. Math Lett.*, vol. 1, no. 2, pp. 171-176, Feb. 1988.

[Hagen92]
L. Hagen, A. B. Kahng, "New Spectral methods for Ratio Cut Partitioning and Clustering," *IEEE Trans. on Computer-Aided-Design*, vol. 11, no. 9, pp. 1074-1085, Sep. 1992.

[Hagen292]
L. Hagen, A. B. Kahng, "A New Approach to Effective Circuit Clustering," *Proc. Int. Conf. on Computer-Aided Design*, pp. 422-428, Nov. 1992.

[Hauck95]
S. Hauck, G. Borriello, "An Evalution of Bipartitioning Techniques," *Chapel Hill Conf. on Adv. Researc in VLSI*, pp. 383-402, Mar. 1995.

[IBOB10]
IBOB: Interconnect Break-out Board, Center for Astronomy Signal Processing and Electronics Research. http://casper.berkeley.edu/wiki/IBOB

[Karypis98]
G. Karypis, V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs,", *SIAM J. of Sci. Computing*, vol. 20, no. 1, pp. 359-392, Jan. 1998.

[Kernighan70]
B. W. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partition of Electrical Circuits," *Bell Systems Tech. Journal*, vol. 49, no. 2, pp. 291-307, Feb. 1970

[Kleinrock77]
L. Klienrock, F. Kamoun, "Hierarchical Routing for Large Networks", *Computer Networks*, vol. 1, pp. 155-174, 1977.

[Konda08]
V. Konda, "VLSI layouts of fully connected generalized networks," *WO 2008/147928*, World Intellectual Property Organization, Dec. 2008.

[Krishnamurthy84]
B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks," *IEEE Tran. On Computers*, vol. c-33, no. 5, pp. 438-445, May 1984.

[Kurafuji11]
T. Kurafuji, et al., "A Scalable Massively Parallel Processor for Real-Time Image Processing," *IEEE J. of Solid-State Circuits*, vol. 46, no. 10, pp. 2363-2373, Oct. 2011.

[Kuon07]
I. Kuon, J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Trans. on Computer-aided Design of Int. Circuits and Systems*, vol. 26, no. 2, pp. 203-215, Feb. 2007.

[Kuon207]
I. Kuon, R. Tessier, J. Rose, "FPGA Architecture: Survey and Challenges," *Found. and Trends in Elec. Design Automation*, vol. 2, no. 2, 2007.

[LaFrieda10]
C. LaFrieda, B. Hill, R. Manohar, "An Asynchronous FPGA with Two-Phase Enable-Scaled Routing", *Proc. of IEEE Int. Symp. On Async. Circuits and Systems (ASYNC)*, May 2010.

[Lai97]
Y.-T. Lai, P.-T. Wang, "Hierarchical Interconnection Structures for Field Programmable Gate Arrays,", *IEEE Trans. on Very Large Scale Int. Systems*, vol. 5, no. 2, pp. 186-196, June 1997.

[Landman71]
B. S. Landman, R. L. Russo, "On a Pin Versus Block Relationship For Partitions of Logic Graphs," *IEEE Trans. on Computers*, vol. c-20, no. 12, pp. 1469-1479, Dec. 1971.

[Lee06]
Interconnect Driver Design for Long Wires in Field-Programmable Gate Arrays, M.S. Thesis, University of British Columbia, June 2006.

[Leiserson85]
C. E. Leiseron, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Trans. on Computers*, vol. 34, no. 10, pp. 892-901, Oct. 1985.

[Leiserson96]
C. E. Leiseron, et al, "The Network Architecture of the Connection Machine CM-5," *J. of Parallel and Distributed Computing*, vol 33, no. 2, pp. 145-158, Mar. 1996.

[Lemieux04]
G. Lemieum, E. Lee, M. Tom, A. Yu, "Directional Single-Driver Wires in FPGA Interconnect," *Inc. Conf. on Field Prog. Tech*, pp. 41-48, Dec. 2004.

[Lewis05]
D. Lewis, et al, "The Stratix II Logic and Routing Architecture", *ACM Int. Symp. on FPGA*, pp. 14-20, Feb. 2005.

[Lin07]
M. Lin, A. El Gamal, Y.-C. Lu, S. Wong, "Performance Benefits of Monolithically Stacked 3-D FPGA," *IEEE Tran. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp.216-229, Feb. 2007.

[Lin09]
M. Lin, A. El Gamal, "A Low-Power Field-Programmable Gate Array Routing Fabric," *IEEE Tran. On Very Large Scale Int. Systems*, vol. 17, no. 10, pp.1481-1494, Oct. 2009.

[Manohar06]
R. Manohar, "Reconfigurable Asynchronous Logic," *Proc. of IEEE Custom Int. Cricuits Conf.*, Sept. 2006.

[Manuel07]
P. Manuel, W. K. Qureshi, A. William, A. Muthumalai, "VLSI layout of Benes networks,", *J. of Discrete Math. Sci. & Cryptography*, vol. 10, no, 4, pp. 461-472, 2007.

[Marquardt99]
A. R. Marquardt, *Cluster-Based Architecture, Timing-Driven Packing and Timing-Driven Placement for FPGAs*, M.S. Thesis, University of Toronto, 1999.

[McMurchie95]
L. MMurchie, C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," *ACM Int. Symp. on FPGAs*, pp. 111-117, Feb. 1995.

[Mehta12]
N. Mehta, *Xilinx Redefines Power, Performance, and Design Productivity with Three Innovative 28nm FPGA Families: Virtex-7, Kintex-7, and Arteix-7 Devices*, WP373, Xilinx, Inc., Oct. 2012.

[Merritt13]
R. Merritt, "FPGAs add comms cores amid ASIC debate,", *EE Times*, Mar. 2013.

[Minev09]
P. B. Minev and V. S. Kukenska, "The Virtex-5 Routing and Logic Architecture," *Annual J. of Electronics*, pp. 107-110, 2009.

[Moore93]
G. Moore, International Solid-State Circuit Conference, February 10, 2003.
Intel Keynote Transcript,
http://www.intel.com/pressroom/archive/speeches/moore20030210.htm

[Mrabet06]
H. Mrabet, Z. Marrakchi, P. Souillot, H. Mehrez, "Performances improvement of FPGA using novel multilevel hierarchical interconnection structure," *IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 675-679, Nov. 2006.

[Nair82]
R. Nair, S. J. Hong. S. Liles, R. Villani, "Global Wiring on a Wiring Routing Machine,", *Proc. Design Automation Conf.*, pp. 224-231, 1982.

[Nair87]
R. Nair, "A Simple Yet Effective Technique for Global Wiring,", *IEEE Trans. on Computer-Aided Design*, vol. CAD-6, no. 2, pp. 165-172, Mar. 1987.

[Nakajima06]
M. Nakajima, et al, "A 40GOPS 250 mW Massively Parallel Processor Based on Matrix Architecture," *Proc. IEEE Int. Solid-State Circuits Conf.*, pp. 1616-1617, Feb. 2006.

[Nalam11]
S. Nalam, B. Calhoun, "5T SRAM with Asymmetric Sizing for Improved Read Stability," *IEEE J. of Solid-State Circuits*, vol. 46, no. 10, pp. 2431-2442, Oct. 2011.

[Nam04]
G.-J. Nam, F. Aloul, K. A. Sakallah, R. A. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," *IEEE Tran. on Computers*, vol. 53, no. 6, pp. 688-696, June 2004.

[Nanda12]
R. Nanda, D. Marković, "Digitally Intensive Receiver Design: Opportunities and Challenges", *IEEE Design & Test of Computers*, vol. 29, no. 6, pp. 19-25, Dec. 2012.

[Noda07]
H. Noda, et al, "The Design and Implementation of the Massively Parallel Processor Based on the Matrix Architecture," *IEEE J. of Solid-State Circuits*, Vol 42, No. 1, pp. 183-192, Jan. 2007.

[Pedram98]
M. Pedram Q. Wu, X. Wu, "A new design of double edge tripped flip-flops," *Asia and South Pacific Design Auto. Conf.*, pp. 417-421, Feb. 1998.

[Rabaey03]
J. M. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits, Second Edition*, Prentice Hall, 2003.

[Rahman04]
A. Rahman, V. Polavarapu, "Evealuation of Low-Leakage Design Techniques of Field Programmable Gate Arrays," *ACM Int. Symp. on FPGA*, pp. 23-30, Feb. 2004.

[ROACH13]
ROACH: Reconfigurable Open Architecture Computing Hardware, Center for Astronomy Signal Processing and Electronics Research. http://casper.berkeley.edu/wiki/IBOB

[Rose93]
J. Rose, A. El Gamal, A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proc. of IEEE*, Vol. 81, No. 7, pp. 1013-1029, July 1993.

[Ryan10]
J. F. Ryan, B. H. Calhoun, "A sub-threshold FPGA with low-swing dual-VDD interconnect in 90nm CMOS," *IEEE Custom Int. Circuits. Conf.*, pp. 1-4, Sep. 2010.

[Saban12]
K. Saban, "Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth and Power Efficiency", *Xilinx White Paper: Virtex-7 FPGAs*, WP380, Dec. 2012.

[Sanchis89]
L. A. Sanchis, "Multiple-Way Network Partitioning," *IEEE Trans. on Computers*, vol. 38, no. 1, pp. 62-74, Jan. 1989.

[Sanchis93]
L. A. Sanchis, "Multiple-Way Network Partitioning with Different Cost Functions," *IEEE Trans. on Computers*, vol. 42, no. 12, pp. 1500-1504, Dec. 1993.

[Sangiovanni93]
A. Sangiovanni-Vincentelli, A. El Gamal, J. Rose, "Synthesis Methods for Field Programmable Gate Arrays," *Proc. of the IEEE*, vol. 81, no. 7, pp. 1057-1083. July 1993.

[Scholo12]
C. R. Scholottmann, S. Shapero, S. Nease, P. Hasler, "A Digitally Enhanced Dynamically Reconfigurable Analog Platform for Low-Power Signal Processing," *IEEE J. of Solid-State Circuits*, vol. 47, no. 9, pp. 2174-2184, Sept. 2012.

[Schuler72]
D. M. Schuler, E. G. Ulrich, "Clustering and Linear Placement," *Proc. Design Automation Conf.*, pp. 50-56, 1972.

[Spencer11]
M. Spencer, F. Chen, C. C. Wang, et al, "Demonstration of Integrated Micro-Electro-Mechanical Relay Circuits for VLSI Applications," *IEEE J. of Solid State Circuits*, vol. 46, no. 1, pp. 308-320, Jan. 2011.

[Sperling12]
E. Sperling, "Designing at 28nm and Beyond," *Chip Design Magazine*, Mar. 2012.

[Sutherland99]
I. Sutherland, B. Sproull, D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, Morgan Kaufmann, 1999.

[Taylor12]
M. B. Taylor, "Is Dark Silicon Useful?" *Proc. of Design Automation Conf.*, pp. 1131-1136, June 2012.

[Tessier98]
R. Tessier, "Negotiated A* Routing for FPGAs," *Proc. Canadian Workshop on Field Prog. Devices*, vol. 6, 1998.

[Tessier00]
R. Tessier, H. Giza, "Balancing Logic Utilization and Area Efficiency in FPGAs", *Int. Workshop on Field-Prog. Logic and App*, pp. 535-544, 2000.

[Teifel04]
J. Teifel, R. Manohar, "Highly Pipelined Asynchronous FPGAs," *ACM Int. Symp. On FPGAs (FPGA'04)*, Feb. 2004.

[Teifel204]
J. Teifel, R. Manohar, "Asynchronous Dataflow FPGA Architecture," *IEEE Trans. on Computers*, vol. 53, no. 11, pp. 1376-1392, Nov. 2004.

[Tsu99]
W. Tsu, et al, "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array," *Int. Symp. on FPGA*, pp. 125-134, Feb. 1999

[Wang11]
C. C. Wang, F.-L. Yuan, H. Chen, D. Marković, "A 1.1 GOPS/mW FPGA Chip with Hierarchical Interconnect Fabric," *Proc. Int. Symp. on VLSI Circuits*, pp. 136-137, June 2011.

[Wang13]
C. C. Wang, D. Marković, "A Radix-3 Network Architecture For Boundary-Less Hierarchical Interconnects", March 2013, US Application No. 61/786,676

[Wang13b]
C. C. Wang, D. Marković, "Fine-Grained Power Gating in FPGA Interconnects", March 2013, US Application No. 61/791,243

[Wei91]
Y.-C. Wei, C.-K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs,", *IEEE Trans. on Computer-Aided-Design*, vol. 10, no. 7, pp. 911-921, July 1991.

[Wong04]
D. Wong, "Interconnection network for a field programmable gate array," *US 6,693,456 B2*, United States Patent, Feb. 2004.

[Wu80]
C.-L. Wu, T.Y. Feng "On a Class of Multistage Interconnection Networks," *IEEE Trans. on Computers*, vol. c-29, no. 8, pp. 694-702, Aug. 1980.

[XlinxV408]
Virtex-4 FPGA User Guide, UG070, Xilinx, Inc., Dec. 2008.

[XlinxV506]
Virtex-5 Platform FPGA Family Technical Backgrounder, Xilinx, Inc., May 2006.

[XlinxV512]
Virtex-5 FPGA User Guide, UG190, Xilinx, Inc., Mar. 2012.

[XilinxV6CLB12]
Virtex-6 FPGA Configurable Logic Block User Guide, UG364, Xilinx, Inc., Feb. 2012.

[XilinxV6DSP11]
Virtex-6 FPGA DSP48E1 Slice User Guide, UG369, Xilinx, Inc., Feb. 2011.

[XilinxV6BRAM11]
Virtex-6 FPGA Memory Resources User Guide, UG363, Xilinx, Inc., Apr. 2011.

[XilinxXC99]
XC4000E and XC4000X Series Field Programmable Gate Arrays, Product Specification, Xilinx, Inc., May 1999.

[Yang99]
Y. Yang, G. M. Masson, "The Necessary Conditions for Clos-Type Nonblocking Multicast Networks", *IEEE Trans. on Computers*, vol. 48, no. 11, pp. 1214-1227, Nov. 1999.

[Yang12]
C.-H. Yang, T.-H. Yu, D. Marković, "Power and Area Minimization of Reconfigurable FFT Processors: A 3GPP-LTE Example," *IEEE J. Solid-State Circuits*, vol. 47, no.3, pp. 757-768, Mar. 2012.

[Yu12]
T.-H. Yu, C.-H. Yang, D. Čabrić, D. Marković, "A 7.4 mW 200 MS/s Wideband Spectrum Sensing Digital Baseband Processor for Cognitive Radios," *IEEE J. Solid-State Circuits*, vol. 47, no. 9, pp. 2235-2245, Sep. 2012.

[Yuan08]
F.-L. Yuan, Y.-H. Lin, C.-F. Wu, M.-T. Shiue and C.-K. Wang, "A 256-Point Dataflow Scheduling 2x2 MIMO FFT/IFFT Processor for IEEE 802.16 WMAN," *Proc. Asian Solid-State Circuits Conf.*, pp. 309-312, Nov. 2008.