

A Novel FPGA Architecture Supporting
Wide, Shallow Memories

by

Steven Oldridge

B.Eng. University of Victoria, 2000

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

Master of Applied Science

in

The Faculty of Graduate Studies

Department of Electrical and Computer Engineering

We accept this thesis as conforming to the required
standard:

The University of British Columbia

April 2002

© Steven Oldridge, 2002

Abstract

A Novel FPGA Architecture Supporting Wide, Shallow Memories

The FPGAs of today are being used to implement large, system-sized circuits. Systems often require significant memory resources, and vendors have responded to these needs by embedding block memories onto their FPGAs. A proportion of these user circuits are communications-based, which have slightly different requirements from standard systems. Most notably, these circuits often contain wide data signals that must be buffered and processed efficiently and quickly. Wide signals also often originate from recently developed high-speed I/O that allow FPGAs to communicate with higher frequency circuits.

This thesis investigates an architecture designed to implement wide, shallow memories on an FPGA. Under the proposed architecture, existing configuration memory normally used to control the connectivity pattern of the FPGA is made user accessible. Typically, not all the switch blocks in an FPGA are used to transport signals. By adding only a modest amount of circuitry, the configuration memory in these unused switch blocks can be used to implement wide, shallow buffers and other similar memory structures. The size of FPGA required to implement a benchmark circuit that makes use of the wide, shallow memories, is 20 percent smaller than a standard memory architecture. In addition, the benchmark circuit is on average 40 percent faster in the proposed architecture.

To get access to this memory additional circuitry must first be added. The extra transistors required will result in an area and speed overhead, even for circuits that do not make use of the new wide, shallow memories. The overhead of this circuitry was measured to be a 36 percent increase in overall FPGA area, and a 5 percent increase in critical path delay.

Table of Contents

Abstract	ii
List of Figures	v
List of Tables	vii
Acknowledgements.....	viii
Chapter 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Objectives	5
1.3 Thesis Organization.....	6
Chapter 2 Background and Previous Work.....	7
2.1 The Characteristics of Programmable Logic	7
2.2 An Overview of FPGA Architectures	9
2.2.1 Logic Resources	9
2.2.2 Routing Resources.....	12
2.2.3 Input/Output Resources	17
2.2.4 Memory Resources.....	18
2.2.5 Configuration Resources	22
2.3 Computer Aided Design Algorithms	25
2.3.1 Synthesis and Logic Block Mapping	25
2.3.2 Placement	27
2.3.3 Routing.....	29
2.4 Focus and Contributions of this Thesis.....	32
Chapter 3 Circuitry to Enable Wide, Shallow Memories.....	34
3.1 The Motivation Behind Switch Block Memories.....	34
3.2 Baseline Programmable Connection.....	36
3.3 Enhanced Architecture.....	38
3.3.1 Enhanced Programmable Bi-Directional Connection.....	39
3.3.2 Memory Addressing and Control.....	41
3.4 Area Overhead due to the Enhanced Programmable Connection.....	42
3.5 Speed Overhead and Transistor Sizing.....	44
3.6 Chapter Summary.....	48

Chapter 4 CAD Algorithms for Wide, Shallow Memories	49
4.1 Synthesis and Mapping Issues	50
4.2 Placement Issues.....	50
4.2.1 Placement of Wide, Shallow Memories	51
4.2.2 Placement Constraints.....	53
4.2.3 Orientationbased Placement	56
4.3 Routing Issues.....	57
4.3.1 Connecting to Switch Block Memories	58
4.3.2 Potential problems with the existing routing algorithm.....	59
4.3.3 Equivalence-based Routing	61
4.4 Chapter Summary.....	63
Chapter 5 Results	65
5.1 A Benchmark Suite for Switch Block Memories	66
5.1.1 Benchmark Circuit Description.....	67
5.1.2 Limiting Factors of Implementation.....	69
5.2 Area Results.....	70
5.2.1 Area Methodology	71
5.2.2 LUTbased Memory	74
5.2.3 Block Memory	74
5.2.4 Switch Block Memory	75
5.2.5 Comparing Memory Architectures	75
5.3 Timing Results.....	78
5.3.1 Timing Methodology	78
5.3.2 LUTbased Memory	79
5.3.3 Block Memory	80
5.3.4 Switch Block Memory	81
5.3.5 Comparing Memory Architectures	89
5.4 Chapter Summary.....	89
Chapter 6 Conclusions and Future Work	91
6.1 Conclusions	91
6.2 Future Work.....	93
6.2.1 Developing Wide, Shallow Benchmarks.....	93
6.2.2 Algorithmic Improvements.....	93
6.2.3 Architectural Improvements	94
6.2.4 Long Term Research	95
References	96

List of Figures

Figure 1.1: High-speed I/O Circuit	2
Figure 1.2: Wide, Shallow Memory Implemented on Block Memories	3
Figure 2.1: A 4-Input Lookup-Table	10
Figure 2.2: Inside a Lookup-Table FPGA Logic Element (Simplified)	11
Figure 2.3: Inside a four logic element FPGA Logic Block	11
Figure 2.4: Island-Style Routing Architecture	13
Figure 2.5: Staggered segmentation of routing tracks	15
Figure 2.6: Wilton Switch Block	16
Figure 2.7: Programmable Connection for a Single Track	16
Figure 2.8: SRAM Based Switch Styles	17
Figure 2.9: Xilinx Virtex-II Floorplan	21
Figure 2.10: Memory Block of the Flex 10 K	21
Figure 2.11: Xilinx Virtex Family Configuration Columns	23
Figure 2.12: FPGA CAD Flow	25
Figure 2.13: Simulated Annealing Pseudo Code	28
Figure 2.14: Pathfinder Algorithm Pseudo Code	32
Figure 3.1: Baseline Bi-directional Programmable Connection	37
Figure 3.2: Preventing the Switch Block From Switching	37
Figure 3.3: Shared Intermediate Staging Point	38
Figure 3.4: Enhanced Programmable Connection (EPC)	40
Figure 3.5: Control Circuitry	41
Figure 3.6: Architectural Parameters	43
Figure 3.7: Baseline Circuit Transistor Sizes	44
Figure 4.1: A Typical FPGA CAD Flow	50
Figure 4.2: Pseudo Code for Constraint Strategy One	54
Figure 4.3: Switch Block Memory Constraint Strategy One	54

Figure 4.4: Desired placement of logic blocks that connect to/from the SBM	57
Figure 4.5: Connecting to a Specific Memory Pin	59
Figure 4.6: Assignment of Data Lines in Switch Block Memories	62
Figure 5.1: A 2 x 2 Crossbar Switch (Version A)	68
Figure 5.2: A 2 x 2 Crossbar Switch Without Memories (Version B)	69
Figure 5.3: A Simple FPGA Constructed From a Single Tile	71
Figure 5.4: Architectural Parameters	72
Figure 5.5: Area of the Benchmark Circuit	76
Figure 5.6: Example Route of the Benchmark Circuit	87
Figure 5.7: Example Critical Path of the Benchmark Circuit	87
Figure 5.8: Forced Circular Route	88

List of Tables

Table 2.1: Logic Blocks Required to Implement Various Memories	19
Table 2.2: Aspect Ratios of the Stratix Family Memory Blocks	20
Table 2.3: Total Memory Resources	20
Table 3.1: Transistor Sizes Tested	45
Table 3.2: Timing information	46
Table 3.3: Benchmark Circuit Results	47
Table 5.1: Number of Minimum-Width Transistors for Each Component	73
Table 5.2: 2 x 2 Crossbar Area Results Summary	76
Table 5.3: Projected 4 x 4 Crossbar Area Results	77
Table 5.4: LUT-based Timing Results	80
Table 5.5: Placement Strategy #1 Timing Results	82
Table 5.6: Placement Strategy #2 Timing Results	83
Table 5.7: Placement Strategy #3 Timing Results, Edge Distance = 1	84
Table 5.8: Placement Strategy #3 Timing Results, Edge Distance = 2	84

Acknowledgements

First and foremost special thanks must go to my supervisor Dr. Steve Wilton. His knowledge of state of the art research in both academia and industry made him an invaluable resource in the direction of this thesis. The opportunities provided to me, to learn and present my research abroad gave my Masters a depth beyond simply researching, and are greatly appreciated. Beyond that, Steve served as an example of how to be a superb supervisor, a skill I hope to pick up as I continue my academic career.

I would like to thank the members of the FPGA research group: Ernie, Tony, Kara, Danna, Julien and Martin, for their opinions, help and support on this thesis. They, along with Gary, Ashish, Graeme, Mike, and especially Marwa kept me sane throughout my stay at UBC and helped make the long hours in the lab bearable, if not fun. Their friendships will be missed in my next academic setting.

I would also like to thank Christian, Matt, and Trevor for helping me have a life beyond school, and Graeme, Dory, and of course Anna, for giving me a creative outlet through RENf, for helping me choose my font (Goudy Old Style), and for loving me unconditionally, respectively and collectively. Truer friends cannot be found.

Finally I would like to thank my parents Dennis and Sue Oldridge for their constant encouragement and support. Without their guidance and love I would not be where I am today.

Funding for this thesis was provided by Micronet, and the British Columbia Advanced Systems Institute. Like thousands of grads before me, I am indebted to them.

Chapter 1

Introduction

1.1 Motivation

In the past decade, Field Programmable Gate Arrays (FPGAs) have grown significantly and are now capable of implementing system-sized integrated circuits. Because of their ability to be configured into virtually any digital circuit in a matter of minutes, FPGAs are implementing designs that would have traditionally been implemented on Application Specific Integrated Circuits (ASICs). This shift to programmable logic is motivated by the time and money costs associated with ASIC fabrication, and the larger designs enabled by the growing capacity and capabilities of FPGAs.

As FPGAs have become more capable, on-chip user storage has become essential. In the past few years, FPGAs have been used to implement large systems, and these systems often require efficient high-speed buffers, tables, and other memory functions. Traditional memory resources (described below) are available to handle these requirements.

Improved FPGA I/O capabilities in the past year, however, are creating opportunities for development of a new type of system [1,2]. Since the introduction of high-speed I/Os that allow a signal in the Giga Hertz (GHz) range to be processed at FPGA speeds, users have

been able to implement high bandwidth circuits on an FPGA that are capable of interfacing with circuitry operating at a much higher frequency. Input signals are time de-multiplexed using a serial to parallel converter, which creates multiple, slower frequency signals on the FPGA. Highspeed output is accomplished by combining multiple FPGA signals using a Parallel to Serial Converter. In Figure 1.1, a user circuit containing a high-speed signal is shown.

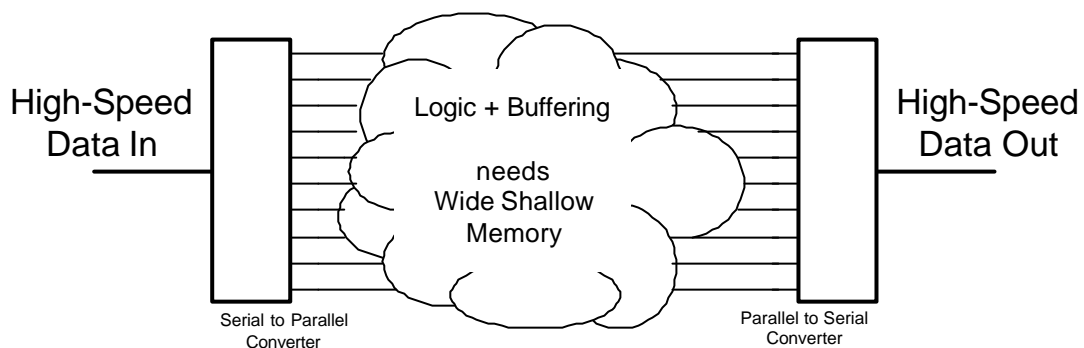


Figure 1.1: High-speed I/O

The wide signals that these I/O create often contain related data that must be buffered and manipulated together, especially in communication circuitry where wide data paths are common. This has created a demand for wide memories, and especially wide, shallow memories that can be used to buffer the communications signals throughout the FPGA.

In the past, vendors have supported user memory requirements by embedding large Random Access Memory (RAM) blocks [1,3,4,5,6]. The RAM blocks provide a flexible yet dense method of storage. The configurability of the RAM blocks allows a physical memory of a fixed size to be configured to a number of different aspect ratios (the ratio of depth to

width). However, if the desired user memory has a width or depth that does not correspond to one of the available aspect ratios, then part of the fixed size physical memory is not used. This means that the implementation of the user memory is not as dense; the unused portion of the memory still counts towards the total size of the FPGA required to implement the circuit. Unfortunately, common FPGA aspect ratios are not well suited to the implementation of wide, shallow memories. For the purposes of this thesis we consider a memory wider than 64 bits, and less than 32 bits deep to be a wide, shallow memory. In most modern FPGA families [1,3,4,5], 4Kbit configurable block RAMs are available that provide configurable aspect ratios from between 4K x 1 and 128 x 32. Wide memories must be constructed by cascading several blocks, and memories shallower than 128 bits can not be implemented densely, because of the unused portion. In the case of the Stratix family [3], a large 512K MegaBlock memory is available that is capable of implementing memories up to 144 bits wide, unfortunately its minimum depth is 4Kbits. If a shallow memory is implemented, most of the physical memory is not used. In addition, only four to twelve of these MegaBlocks are available, depending on the specific device chosen, which limits the number of wide memories that can be implemented. Figure 1.2 demonstrates a 128-bit wide, 8-bit deep memory (shown in grey) implemented using 4K block memories, and the same memory implemented using a 512K MegaBlock.

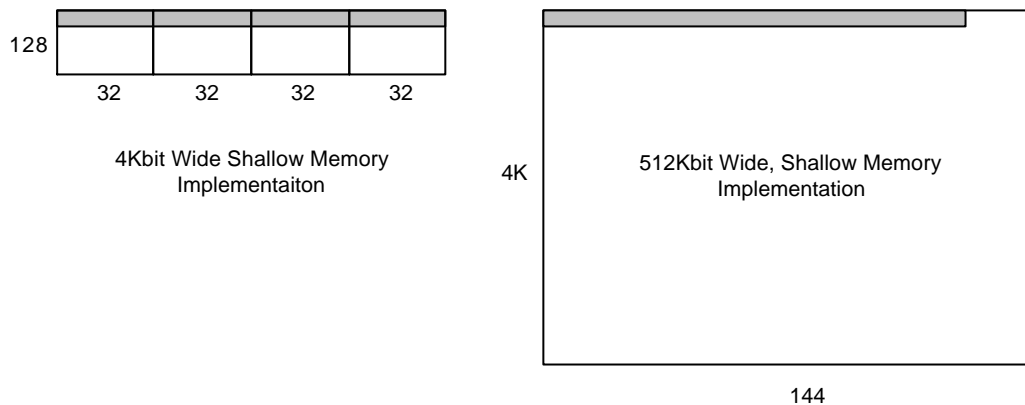


Figure 1.2: Wide, Shallow Memory Implemented on Block Memories

Another potential method for supporting wide, shallow memories is employed by Xilinx and Lattice Semiconductor (in the ORCA product line, formerly produced by Lucent) [4,5]. In their devices each 16-bit lookup-table (LUT) within a logic block can be configured as a 16-bit RAM. Several logic blocks can be combined to implement wider memories that are as deep as is needed. The combination of these small individual memories into a larger one requires more than just the LUT-based memories; additional logic blocks are needed to glue the individual LUTs together. The high overhead involved in using LUT-based memories to implement large user memories led FPGA vendors to put block memory resources onto the FPGA. As will be shown, the implementation of LUT-based wide, shallow memories is also inefficient.

1.2 Objectives

The research objectives of this thesis are as follows:

1. Develop a novel architecture that can efficiently implement wide, shallow memories.
2. Create computer aided design (CAD) tools that can map user circuits to this architecture.
3. Use the CAD tools to evaluate the efficiency of the architecture.

In our novel architecture we provide the user with access to existing memory on the FPGA through additional circuitry. Under a standard architecture, this memory stores a part of the configuration pattern required to implement a user circuit. However, often some of this memory simply specifies that a part of the FPGA is not being used. By adding a modest amount of circuitry, we can gain access to that memory.

To properly evaluate the new architecture, Computer-Aided Design (CAD) tools must be developed. An existing Place and Route tool [8,23] was modified to handle the new architecture, and several placement and routing options were devised. These options must be compared against each other, before the switch block memory architecture can be compared to the LUT-based and block memory-based architectures.

1.3 Thesis Organization

In the next chapter we present an overview of existing FPGA architectures and CAD tools as the body of previous work. Chapter Three describes the enhanced architecture. In Chapter Four, the Computer-Aided Design (CAD) algorithms required to map circuits to the new architecture are discussed. Chapter Five evaluates the CAD algorithms and measures the benefits of the proposed memory architecture by comparing it to alternative memory architectures. The final chapter summarizes the conclusions of the thesis and proposes several areas of future work.

Chapter 2

Background and Previous Work

In this chapter FPGAs will be introduced, along with a discussion of the advantages and disadvantages of FPGAs when compared to nonflexible implementation media such as ASICS or full custom design. FPGAs provide post-fabrication flexibility; a breakdown of the components that enable this functionality, with a focus both on past memory architectures and on configuration memory will also be presented. Finally the algorithms that enable the efficient programming of the architectures will be discussed.

2.1 The Characteristics of Programmable Logic

Over the past six years, FPGAs have grown in logic capacity from the tiny, thousand gate systems [10,11,12,13], to the three million gate systems claimed today [1,3]. This dramatic increase in capacity has allowed FPGAs to expand from their original role of implementing small glue logic at the circuit board level, to the implementation of entire systems within a single chip.

Throughout this shift, the flexibility of FPGAs, has given them a growing advantage over less flexible options, such as Mask Programmable Gate Arrays (MPGAs) or Application Specific Integrated Circuits (ASICs). With the ability to be configured in seconds, and the

ability to be reconfigured virtually an unlimited number of times, FPGAs avoid the process of fabrication that MPGA and ASIC designs face. By circumventing fabrication, a process that costs \$300,000 in 0.18 μ technology and \$750,000 in 0.13 μ [48], FPGAs have a much lower fixed, or Non Recurring Engineering (NRE) costs, offering a low-to-medium volume price advantage. Perhaps more importantly, the immediate reprogrammability of FPGAs allows for very quick design iterations, allowing mistakes to be corrected within hours or days. For an ASIC, each fabrication run takes approximately four months from the time of submission to get the chip back, and a design that fails testing must go through a second four-month iteration. In a market where the first arriver gains significant advantage, the cost in time-to-market of the first layout and fabrication run, let alone a possible second or third, has been enough to incite many companies to move their circuit design onto an FPGA.

Despite the advantages of FPGAs, ASICs still hold a significant share of the chip design market. Although their initial NRE costs are greater than those of an FPGA, the cost to produce a single chip is dramatically less. This means that in higher volumes (anywhere from 10K to 100K units) the ASIC solution is more cost efficient. In addition, because ASICs are implementing only the logic and routing resources required to perform the specific function they are designed for, they are able to fit more logic into a given die area. Typically, a circuit implemented in an FPGA is about 10 times larger than the equivalent circuit implemented as an ASIC. Their specificity and increased density also lead to much faster solutions, with ASICs running at speeds 3 times faster than those of FPGAs [8].

Still, improvements in Computer Aided Design (CAD) tools that compile a circuit description to an FPGA, as well as the recent addition of fixed functionality high density cells such as multipliers [1,3,4,6] are starting to shrink the gap in density and speed, making FPGAs an attractive solution.

2.2 An Overview of FPGA Architectures

Although there are many different FPGA architectures commercially available, they are all built from the same functional components:

1. Logic resources capable of flexibly implementing user logic functions.
2. I/O resources for communication with offchip signals.
3. Memory resources for on-chip user storage.
4. Routing resources used to flexibly connect logic and memory resources together.

In addition to these functional resources, FPGAs also have a backbone of configuration memory and circuitry that enable the loading and maintenance of a circuit configuration.

Each of these resource types will be described within this section.

2.2.1 Logic Resources

In order to be able to flexibly implement the logic required by a user circuit, an FPGA must contain logic resources that are capable of efficiently implementing many different functions. The majority of FPGAs use lookup tables (LUTs) in their logic blocks. Each LUT contains a small memory that is used to hold a logic function. The inputs to the LUT control multiplexers that select which bit is accessed, allowing any possible

combination of inputs to produce the desired output, as shown in Figure 2.1. The number of inputs dictates the size of the memory; a k-input lookup table (k-LUT) requires 2^k bits of memory. The majority of commercial FPGAs use 4LUTs as their basic logic resource [1,3,4,5,6,10,12].

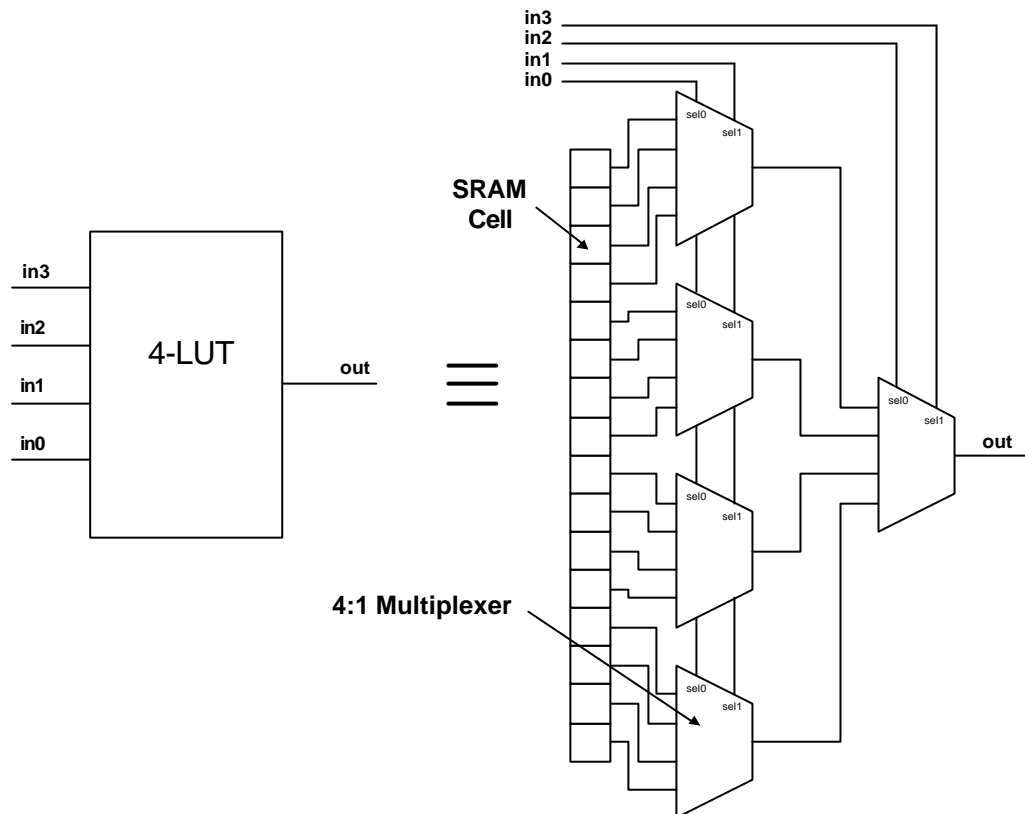


Figure 2.1: A 4-Input Lookup-Table [4]

Any value can be stored in the memory, allowing a kinput LUT to implement any logic function of kinputs. By combining this k-LUT with a storage element and a multiplexer as shown in Figure 2.2, the basic Logic Element (LE) can implement either combinational or sequential logic.

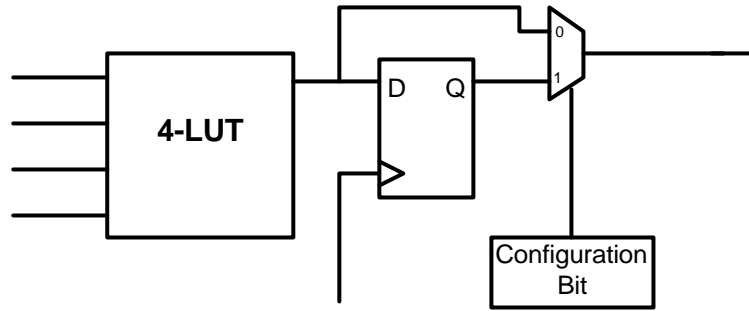


Figure 2.2: Inside a Look-Up-Table FPGA Logic Element (Simplified)

The LE's are clustered into logic blocks, in order to take advantage of shared inputs. A cluster of N logic elements typically has less than $N * K$ distinct inputs and contains local programmable interconnect that allows the output of a given LE to be fed to the input of another local LE. This local programmable interconnect allows fast connections to be made between logic elements that share a cluster. Figure 2.3 illustrates the clustering scheme described above.

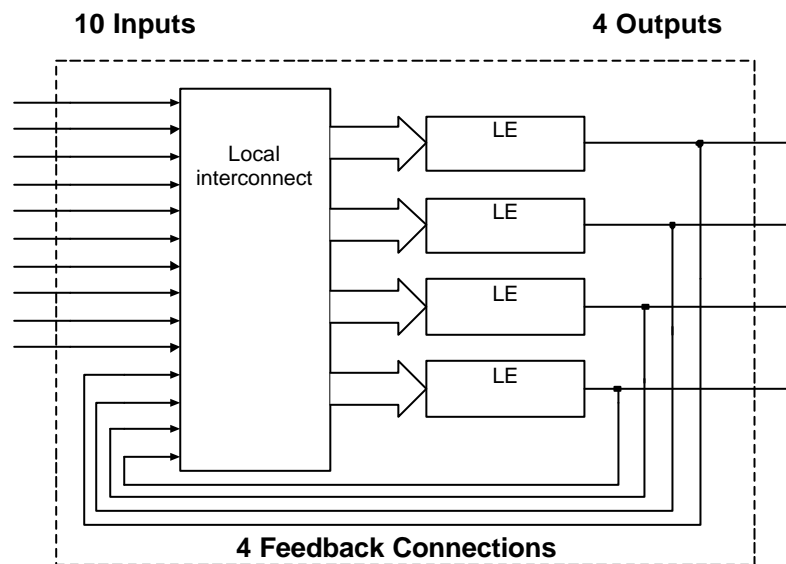


Figure 2.3 Inside a four logic element FPGA Logic Block

In addition to the flexible LEs, many new FPGAs also contain fixed logic that implements a fixed function densely and quickly. The multiplier blocks found in Xilinx's Virtex-II Family [1] are the beginning of a trend to combine programmable and fixed logic resources on a single chip.

2.2.2 Routing Resources

The routing resources of an FPGA provide a programmable connection scheme that links the logic resources to each other, and to the memory and I/O resources. In the past, the *island-style* architecture used by Xilinx [10], Lattice and Vantis, and the *hierarchical* architecture used predominantly by Altera [12] were the dominant routing architectures. Current FPGAs almost exclusively use a combination of the two, opting for local interconnect resources within the logic clusters, and island-style routing between them [1,3,4,5,6].

2.2.2.1 Local Interconnect

The local interconnect provided within a logic cluster serves two purposes. First, it allows each cluster input pin to connect to a number of internal logic element input pins. The added flexibility that this provides makes the task of routing the circuit, described in Section 2.3, much easier because multiple paths into each logic element's inputs are available. Secondly, local interconnect within a cluster provides fast connections between the logic elements being clustered. These fast internal connections allow larger logic functions to be efficiently implemented by combining multiple LUTs. The same thing can

be accomplished by combining LUTs in different clusters, however the routing delay between clusters is significant, leading to slower circuits

The flexibility of the internal interconnect is provided by an interconnect matrix [14,15,16] that allows external (cluster) input pins and the logic element outputs to access internal (logic element) input pins. The level of connectivity in a given FPGA architecture can be *full*, where every cluster input and feedback path can reach every logic element input, or *partial*, where a pattern of limited connectivity is used, tradingoff flexibility for reduced area and capacitance. Connectivity is provided using programmable pass-transistor and multiplexer connections.

2.2.2.2 Global Interconnect

The routing between logic blocks utilizes an island-style architecture, with logic clusters as islands, surrounded by a sea of interconnect as shown in Figure 2.4.

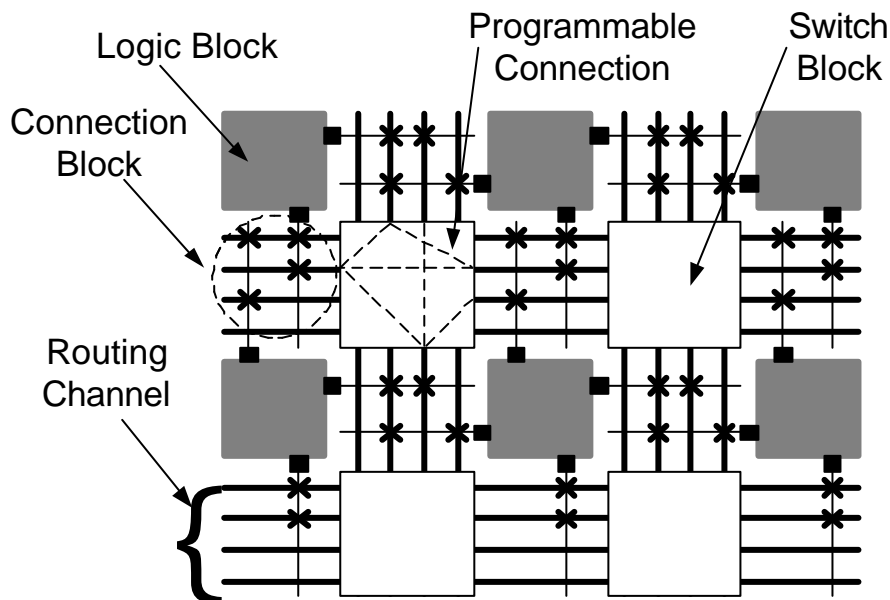


Figure 2.4: Island-Style Routing Architecture (From [17])

The basic island style architecture can be broken into three components:

1. Routing channels: Groups of metal wires that run horizontally and vertically throughout the FPGA.
2. Connection blocks: Programmable connections between the logic blocks and the adjacent routing channels.
3. Switch blocks: Programmable connections between routing channels.

Routing channels made up of long horizontal and vertical wires cross the FPGA in a checkerboard pattern, providing fixed routing resources that allow signals to travel across the FPGA. The channel width of an FPGA refers to the number of metal wires, or tracks, in each channel. The segmentation length of the tracks, measured in the number of logic clusters that the wires span before a programmable connection is needed to continue the connectivity, varies between architectures. Segmentation lengths of between four and eight are common [4,6]. The starting cluster of the segmented track is staggered in order to increase the routing flexibility. In this thesis we will focus on an architecture with a segmentation length of four, with one quarter of the tracks starting/terminating at each switch block. Figure 2.5 demonstrates this staggered segmentation concept for a channel width of 12.

Connection blocks, shown in Figures 2.4 and 2.5, are adjacent to the logic blocks on all four sides. They provide the programmable connections between the logic blocks and the

routing tracks. Each input pin is able to connect to a certain percentage of tracks in the adjacent channel.

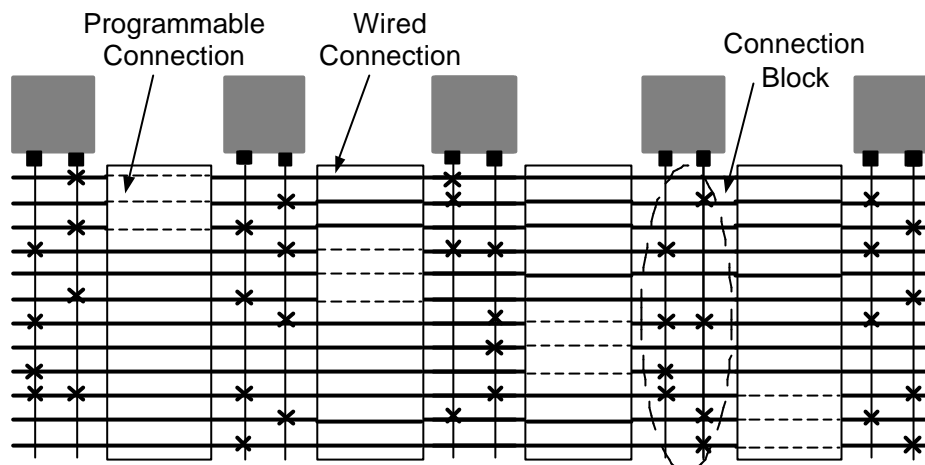


Figure 2.5: Staggered segmentation of routing tracks

Switch blocks occur at the intersection of every horizontal and vertical channel. They provide programmable connections between the horizontal and the vertical channels, as well as horizontal-to-horizontal (and vertical-to-vertical) connectivity at the end of a segment. Fully connecting the tracks within the incident channels is not feasible because too many switches would be required; each switch consumes chip area and adds capacitance to the incident tracks. Thus, several switch block topologies have been developed [16,17,18,19,20] in which many of the potential connections have been removed. In this thesis we will focus on the Wilton switch block topology [20] shown in Figure 2.6. Each dotted line in the figure represents a programmable connection between two routing tracks.

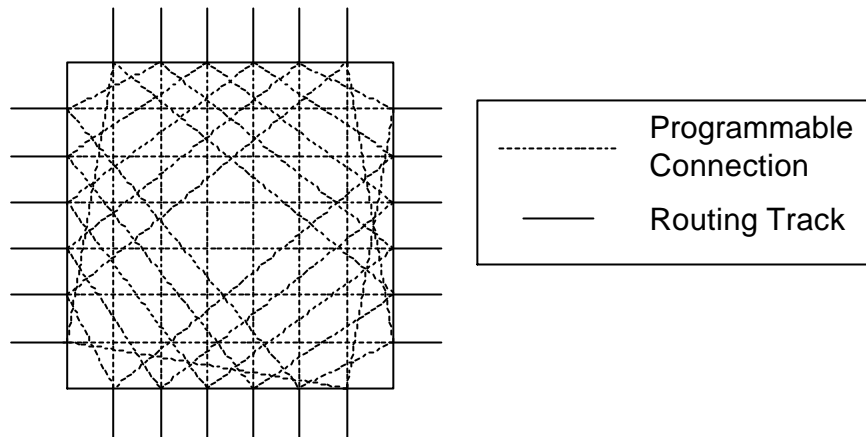


Figure 2.6: Wilton Switch Block (from [20])

The staggered segmentation concept applied to the Wilton switch block leads to two distinct possibilities for connectivity for each track; either the incoming track ends and requires a switch to continue on in the same direction as in Figure 2.7a (which shows the programmable connections for a single track of the switch as bi-directional arrows) or the track passes through the switch block, making the connection automatically with a metal wire as in Figure 2.7b.

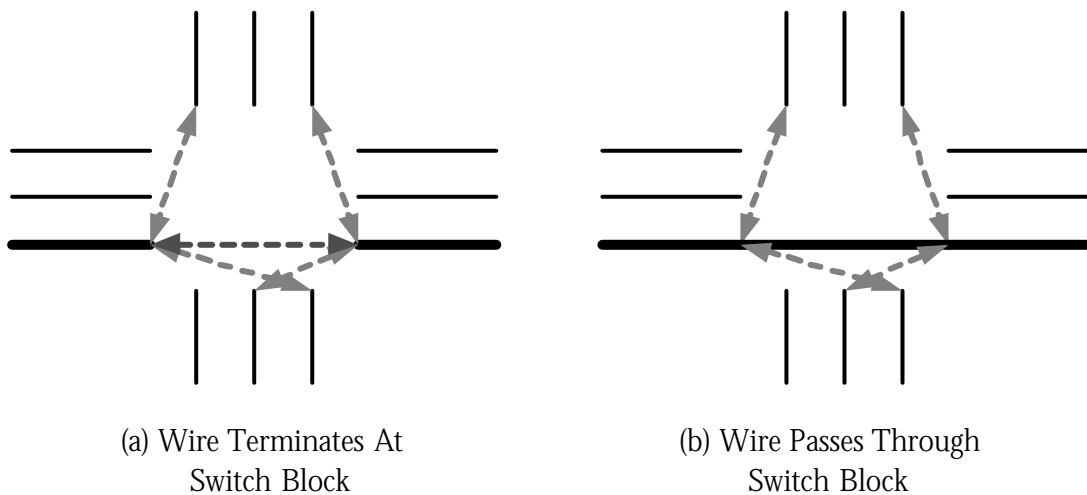


Figure 2.7: Programmable Connection for a Single Track in the Switch Block

The majority of FPGAs today use SRAM cells [1,3,4,5,6] to control the pass transistors, multiplexers, and tri-state buffers that make up the routing connections as shown in Figure 2.8. The notable exception to this is the antifuse technology seen in many Actel FPGAs [24, 25] which is incompatible with the switch block memory architecture proposed in this thesis, since it does not use configuration memory. Section 2.2.5 explores the configuration resources of FPGAs in more detail.

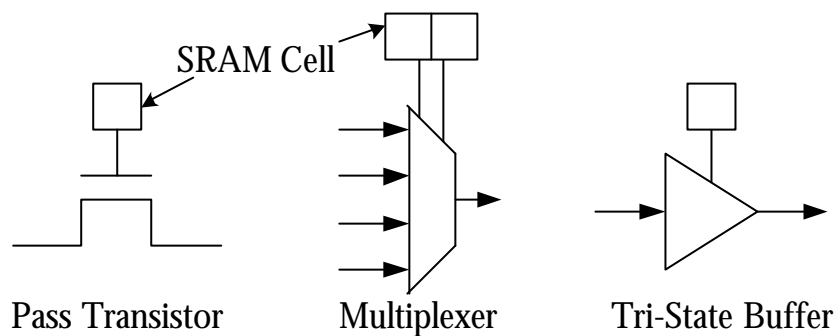


Figure 2.8: SRAM-Based Switch Styles

2.2.3 Input/Output Resources

In order to interface with off-chip signals, FPGA I/O blocks are programmable in a number of ways. Each I/O block is capable of acting as either an input, an output, or as a bi-directional signal carrier. Most FPGAs feature variable voltage levels, supporting as many as 20 different I/O standards [1,3,4,5] on a per-I/O block basis in order to allow the integration of multiple chips that work at different voltage levels.

Some vendors also include high-speed I/Os in order to allow the FPGA to communicate with systems that have higher operating frequencies than the FPGA [1,2]. Input signals are

time de-multiplexed using a serial-to-parallel converter, which creates multiple, slower frequency signals on the FPGA. High-speed output is accomplished by combining multiple FPGA signals using a parallel-to-serial converter. In addition to allowing the FPGA to communicate with higher speed circuits, these I/Os also increase the bandwidth of the FPGA. Xilinx achieves a 3.125 Gbps data rate per I/O [1] by converting a 1-bit wide serial stream into a 78-bit wide parallel stream. Outgoing signals use the reverse, a parallel to serial conversion, to match external chip speeds.

2.2.4 Memory Resources

The memory resources of FPGAs can be categorized as either fine-grained or coarse grained. Finegrained memory architectures consist of a large number of small memory elements distributed across the FPGA. Early architectures, such as the Xilinx 4000 family, provided users with access to the configuration memory in the logic elements' lookup tables. Recall that a k LUT is a read only memory with k address lines. By adding write circuitry and address pins, datain pins, and a write enable pin these read only memories can be converted into RAM. In addition to the circuitry required to enable writing to this memory, FPGA vendors often include circuitry within their logic blocks that helps with the grouping of the smaller LUT memories into a larger user memory, and that allows the memory to take on different configurations such as carry cascade chains or dual port capability [1,3,4,5]. By combining these fine-grained memory elements, users were able to implement memories of arbitrary depth and width. The cost (in logic blocks) of implementing a memory consists of both the logic blocks directly implementing the

memory, as well as those needed to implement the address decoder and output multiplexers [20]. As Table 2.1 demonstrates [26], the number of logic blocks required to implement a reasonably sized memory using a Xilinx 4000 family part is significant. If a design requires several large memories they quickly fill the logic blocks available, leaving very little room on the chip to implement logic.

User Memory	Logic Blocks used for Storage	Logic Blocks used for glue logic	Total Logic Blocks
128 x 8	32	10	42
256 x 4	32	15	47
1K x 1	32	29	61
4K x 1	128	117	245

Table 2.1: Logic Blocks Required to Implement Various Memories (From [26])

To meet the growing need for large memories, FPGA vendors have embedded larger, coarse-grained memory resources on-chip. These memories directly implement the decoder and multiplexer logic and amortize the area overhead of this circuitry across many bits, allowing for denser and faster memory implementations. In order to be useful in a wide variety of circuits, coarse-grained memories are designed to be as flexible as possible without significant impact on area and delay [20]. This flexibility is achieved by allowing the user to program the aspect-ratio of the memories, trading the depth of a fixed size memory for width. The aspect ratio options for the three different sizes of memory block (512 bits, 4K bits, 512K bits) available in Altera's Stratix family [3] is shown in Table 2.2.

512 Bits Total	4K Bits Total	512K Bits Total
512 x 1	4K x 1	64K x 8
256 x 2	2K x 2	64K x 9
128 x 4	1K x 4	32K x 16
64 x 8	512 x 8	32K x 18
64 x 9	512 x 9	16K x 32
32 x 16	256 x 16	16K x 36
32 x 18	256 x 18	8K x 64
	128 x 32	8K x 72
	128 x 36	4K x 128
		4K x 144

Table 2.2: Aspect Ratios of the Stratix Family Memory Blocks [3]

In addition to a programmable aspect ratio, flexibility is also provided by allowing several coarse memory elements to be combined to implement even larger memories. The total memory resources of three modern FPGA Families are listed in Table 2.3 (this list is not meant to be comprehensive, but rather, representative of the state of the industry today).

Family	Block Type(s)	Quantity	Total Bits
Altera Stratix (2002) [3]	(512, 4K, 512K)	(94 – 1118, 60 – 520, 1 – 12)	920K – 10 Meg
Xilinx Virtex-II (2001) [1]	(18K)	(4 – 168)	72K – 3 Meg
Lattice ORCA 4 (2000) [5]	(9K)	(8 – 16)	74K – 147K

Table 2.3: Total Memory Resources

Physical memories are embedded at fixed locations on-chip, often with all memories grouped together in a row or column. Figure 2.9 shows the floorplan of the Xilinx Virtex-II architecture. The programmable connections between the memories and the routing resources are similar to the connection blocks described in Section 2.2.2. In addition,

most FPGA memory blocks contain local interconnect in order to increase the input flexibility to the memory. Figure 2.10 illustrates the memory block used in Altera's FLEX 10K Family [6].

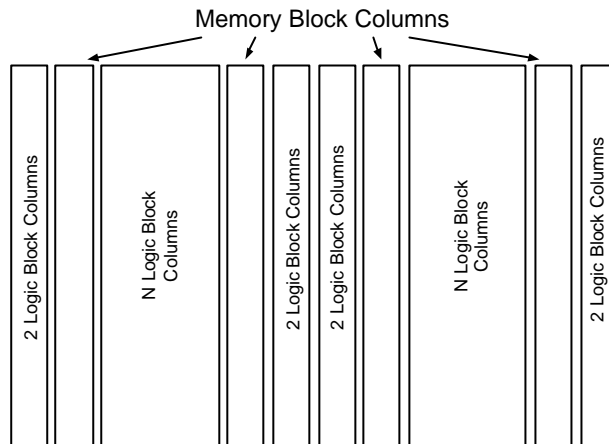


Figure 2.9: Xilinx Virtex-II Floorplan (From [1])

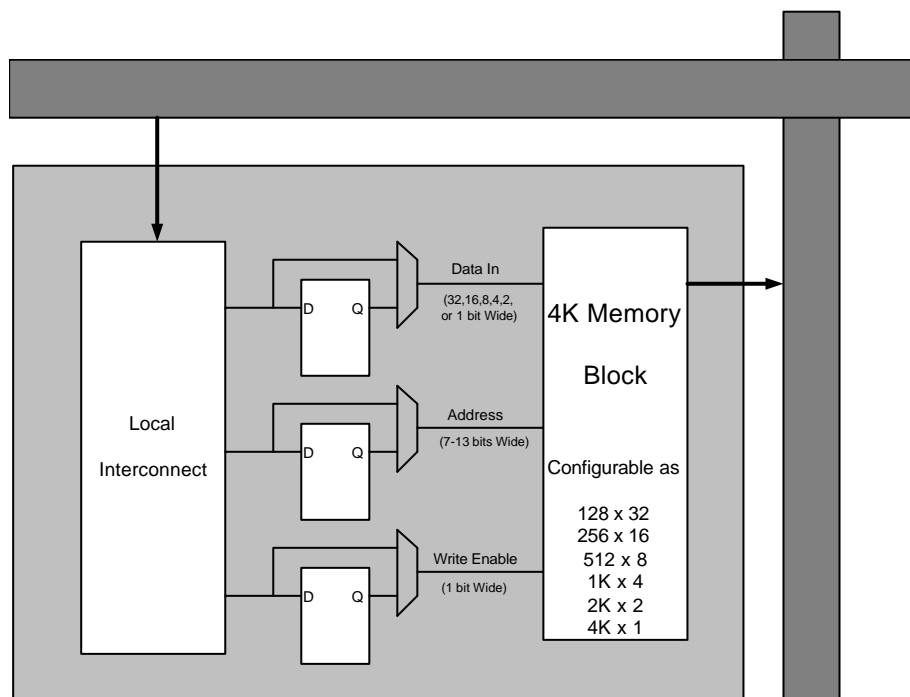


Figure 2.10: Memory Block of the Stratix Architecture [3]

2.2.5 Configuration Resources

To make FPGAs programmable, the user must be able to modify the circuit configuration. As mentioned in Section 2.2.2, most FPGAs use Static RAM (referred to as configuration memory) to maintain the configuration values of the circuit. Setting the values of all of the SRAM cells configures the FPGA to implement a specific circuit. These SRAM cells are located in logic and memory blocks (as shown in Figure 2.1 and 2.10) and in the programmable routing (as shown in Figure 2.8). In modern FPGAs, a majority of the configuration bits are located in the routing resources. The configuration bits are connected directly to the gates of programmable connections, controlling whether each gate is open or closed, or in the case of a multiplexer, which input is selected. By programming specific gates open or closed, paths can be established between the logic and memory resources.

The string of values that will configure the entire FPGA is called a bitstream. These values are calculated by the computer-aided design algorithms discussed in Section 2.3. In order to be able to access the configuration bits, FPGAs are equipped with a configuration backplane [27], made up of wires and circuitry that are solely used for programming the FPGA. This portion of the chip is inaccessible to the user, except through the configuration mechanisms provided. The configuration mechanism is usually an external device (often a computer), which provides the configuration values, and the appropriate control signals, which tell the configuration backplane where those values need to be stored [28,29].

The configuration memory of Xilinx's Virtex family is divided into columns that run vertically in the FPGA. The configuration memory is then further divided into frames of a fixed number of bits. In many cases a specific frame is padded with empty slots because it is not always physically convenient (or possible) to connect all of the column wires to a configuration bit in order to make a full frame. An example of the configuration columns of Xilinx's Virtex Architecture [27] is shown in Figure 2.11. The number of frames per configuration column also varies. This variability, along with the padding of empty slots, adds flexibility to the configuration backbone.

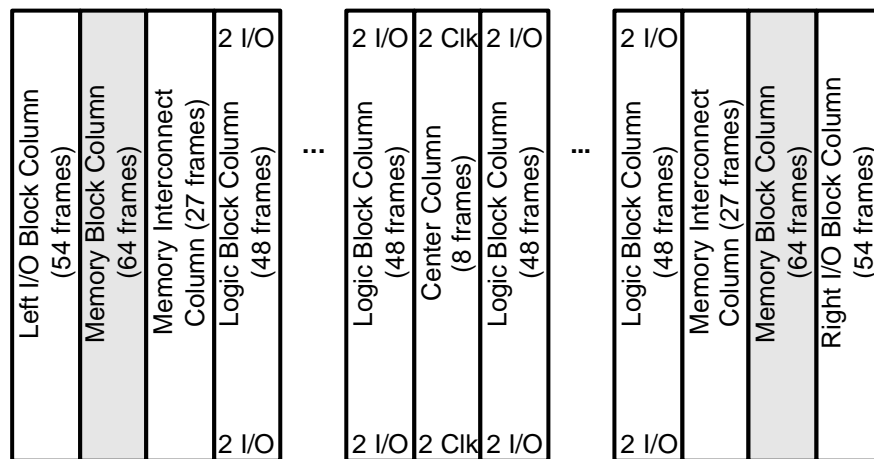


Figure 2.11: Xilinx Virtex Family Configuration Columns

The column locations correspond to the physical locations of logic, routing, memory, and I/O resources. Configuration bits on an FPGA are not grouped as a single central memory, but are distributed throughout the device, near to the resources they control.

Under the Virtex family, five column types occur:

1. The I/O Block Column connects to the configuration bits that control the I/O blocks on either the left or right edge of the chip.
2. The Memory Block Column connects to the configuration bits that control the aspect-ratio of the block memory, as well as the initial data to be stored.
3. The Memory Interconnect Column connects to the configuration bits controlling the block memory interconnect.
4. The most frequent column is the Logic Block Column which connects to the configuration memory for the logic and routing resources, as well as to the I/Os at the top and bottom of the chip in that column.
5. Finally the Center Column connects the configuration backbone to the configuration bits that control the on-chip clock distribution network.

In order to differentiate which configuration bits each frame corresponds to, the configuration memory is addressed. Addressing occurs at multiple levels, first to determine which column of the chip this particular portion of the bitstream belongs to, and second to determine which frame within the column. Each frame is loaded sequentially into the FPGA until all frames have been configured. Once fully programmed, partial reconfiguration of specific portions of the chip is possible. The discontinued XC6000 family [30] allowed for reconfiguration of logic, connection and switch blocks individually, however today's architectures only support reconfiguration of a whole column of configuration memory.

2.3 Computer Aided Design Algorithms

Software tools are needed to configure an FPGA. Clearly it is infeasible to hand-calculate the millions of specific values for the configuration memory in the FPGA. Instead, through a number of steps, a hardware description of the circuit to be implemented, usually in VHDL or Verilog, can be automatically transformed by the computer-aided design (CAD) tools into the desired bitstream. Those steps are:

1. Synthesis and Mapping, the process of optimizing and mapping the hardware description into logic block and memory block-sized pieces.
2. Placement, where the location of each of these logic blocks is determined
3. Routing, which involves the assigning of specific routing resources to all signals.

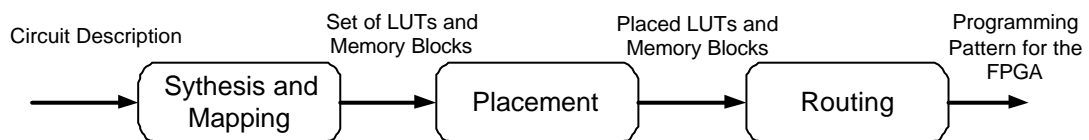


Figure 2.12: FPGA CAD Flow

2.3.1 Synthesis and Logic Block Mapping

The Synthesis and Mapping tool takes a hardware description language (HDL) specification of the circuit to be implemented as its input. First, the circuit is synthesized into basic logic functions, in preparation for optimization. After a technology-independent optimization [31,32,33] stage, where the component level description is optimized to remove redundant hardware and simplify the logic as much as possible, technology-dependent mapping occurs

[34,35,36,37]. This involves taking the optimized component description of the circuit and partitioning it into k-LUT-sized groups, where k is chosen based on the specific FPGA that this circuit is to be implemented on. Once the circuit has been mapped to LUTs, these are grouped into logic blocks of NLUTs, again based on the architecture of the specific FPGA. Clustering algorithms must simultaneously balance constraints including the maximum number of inputs and logic elements per cluster, while attempting to minimize the number of signals between logic blocks[17]. By minimizing the number of external connections between clusters, the circuit can better take advantage of the faster internal routing resources.

While the logic contained in the HDL description is mapped to logic block-sized clusters, the user memories are also mapped to memory block-sized elements. The width and depth of each user memory determine how many physical memory blocks are required in order to instantiate it. The aspect ratios of the memory blocks are chosen to be the minimum that will accommodate the depth of each user memory. If a user memory is deeper than the available memory blocks, multiple blocks are used to meet this depth requirement and the extra logic needed to glue those memories together is added to the circuit. Once the aspect ratio of the memory blocks has been decided, the width of the user memory dictates how many physical memory blocks are needed.

2.3.2 Placement

Once the HDL description has been reduced to a list of blocks and a netlist describing the connectivity between those blocks, a placement algorithm is used to determine where each block should be placed on the FPGA. Placement algorithms use several strategies in order to determine the best locations for the blocks. Simple placers put logic clusters that are connected to each other close together, in order to minimize the length of the wires required to route between them. On an FPGA the routing wires only travel horizontally and vertically throughout the chip, so the horizontal plus vertical (Manhattan) distance between connected clusters is used. Alternatively, placement algorithms can optimize based on:

- Bounding Box of nets[38], a modification of Manhattan distance to better suit nets that have multiple sinks.
- Wiring density (routability-driven placement)[17], which attempts to prevent signal congestion by spreading clusters evenly throughout the chip.
- Circuit speed (timing-driven placement)[17,39], which estimates critical paths at each iteration in an effort to provide a faster solution.
- Power (power-driven placement), which uses a power model to estimate the power consumption of the configuration.

Several algorithms are available to optimize these goals. They can be broken into three classes: (1) mincut (partition-based) [40,41,42,43]; (2) analytic [44,45,46]; and (3) simulated annealing-based placement tools [8,17,20,38,39]. As in [17], we focus on the simulated annealing-based placement tool because it is much easier to add new optimization goals

and constraints to a simulated annealing-based placement tool, than to a min-cut or analytic placement tool.

Simulated annealing is a heuristic optimization algorithm based on the industrial annealing processes where molten metal is gradually cooled, in order to produce high quality metal objects [47]. Figure 2.13 shows the pseudo-code for a generic simulated annealing-based placer. A cost function is designed based around one or more of the optimization goals listed above, and is used to evaluate the quality of a placement. In a purely timing-driven placement the cost function could, for example, consist of the sum over all nets of each net's delay, multiplied by its criticality (a measure of how close the net is to being the slowest).

```
Current_Placement = Random_Placement();
Temp = Get_Initial_Temperature();

While ( Anneal_Finished() == False)
{
    For ( Num_Inner_Loop_Moves ) Repeat
    {
        New_Placement = Try_Swap ( Current_Placement );
        ? Cost = Cost( New_Placement ) - Cost( Placement );
        R = Random( 0, 1);
        If ( R < e-? Cost / Temp ) // Note that if ? Cost is negative this path will
        { // automatically be taken.
            Placement = New_Placement;
        }
    }
    Temp = Update_Temperature ( );
}
```

Figure 2.13: Simulated Annealing Pseudo Code [47]

At the beginning of the anneal, a random placement is generated. The algorithm then performs many random swaps of the logic block positions, gradually improving the placement through a sophisticated acceptance schema. If a swap improves the cost of the placement (by decreasing it) then it is accepted automatically. If the cost of the placement is increased by the swap, then there is still a chance that the move will be accepted. The probability of accepting such a seemingly bad move is $e^{-\Delta \text{Cost} / \text{Temp}}$. A negative delta cost, which represents an improvement in score, will have a probability of acceptance greater than 1 and will be automatically accepted. A positive delta cost reduces this probability to less than 1, decreasing the chance that these seemingly bad moves are accepted. The temperature parameter T is used to control the reduction in probability of acceptance. By starting with a very high temperature and slowly reducing it, the algorithm will initially accept nearly any move, but will eventually begin to accept fewer and fewer bad moves. This gradual decrease allows the algorithm to escape local minima in the cost function that would trap a greedy algorithm designed to only accept moves that improve the cost. The final stage of annealing at temperature $T=0$ is a greedy minimization.

The initial temperature, and the changes made to the temperature over the course of the anneal are referred to as the annealing schedule and are discussed in detail in [17].

2.3.3 Routing

FPGA routing is the process of assigning specific routing resources to each signal, or net, within the user circuit. Using a list of nets (netlist) provided by the mapping tool, and a list

of logic block locations (a placement) provided by the placement tool, the routing tool attempts to find paths for all of the signals. The resources available to route between logic and memory blocks of the specific FPGA architecture being targeted are usually represented by a directed graph [49,50,51], referred to as the routing resource graph. Nodes of the routing resource graph consist of the logic block input/output pins and the wires of the FPGA, while edges correspond to programmable connections.

Routing a net corresponds to finding a path between the starting node (source) and the end nodes (sinks) in the graph. Nets need resources to be assigned to them exclusively because the resources represent a physical path that an electrical signal will take in the chip. In order to prevent two nets from using the same node, most FPGA routers use congestion avoidance schemes [49,50] to prevent overuse of a specific resource. In timing-driven routing schemes the router additionally tries to give nets on or near the critical path (the slowest path in the circuit) advantage over nets with more timing slack. As Wilton shows in [52], timing-driven routing is essential for circuit speed related measurements to be meaningful.

Two different groups of routing algorithms exist. Two-step routers [53] first perform a global routing stage [54,55], assigning routing channels to each of the nets, and then a detailed routing stage [53,56,57], where the nets are assigned to specific wires within those channel segments. Combined global-detailed routing algorithms [49,50,56,58] find complete paths from source to sink(s) in one step. At the detailed routing level, both types

use maze routing algorithms [57,59], which employ Dijkstra's shortest path [60] to find the lowest cost path between a net source node and a net sink node. Many routing iterations are carried out, where some (or all) of the nets that have previously been routed are ripped up and rerouted in a different order, in an attempt to resolve competition for resources (routability-driven) or improve speed (timing-driven).

The Pathfinder algorithm [50] improves upon this concept by initially allowing resources to become overused, and gradually resolving the congestion by introducing a historical congestion cost to nodes that makes overused nodes more expensive in the next iteration. This eliminates the dependence on the order in which nets are routed. The pseudo code of the Pathfinder algorithm is demonstrated in Figure 2.14. The cost of using a node n as part of a connection (i,j) [17] is:

$$\text{Cost}(n) = \text{Crit}(i,j) * \text{Delay}(n) + [1 - \text{Crit}(i,j)] * [\text{Delay}(n) + \text{historical}(n)] * \text{usage}(n) \quad (1)$$

The first term is delay sensitive: the closer a connection is to being on the critical path, the more priority is given to delay. The second term is congestion sensitive; to the delay of the node we add a historical congestion cost of the node, which is increased every iteration that the resource is overused. These are multiplied by the current usage of the node, which is 1 if using this node will not cause overuse, and otherwise is increased as a function of the number of routing iterations and the overuse. Using this function, the delay and congestion can both be optimized simultaneously, with the focus shifting from delay to

congestion as the criticality of the connection decreases. This means that connections on or close to the critical path are optimized for speed, because their cost to use resources is lower, while connections with more slack avoid areas of congestion, taking slower paths.

```

Criticality ( i,j ) = 1 for all nets i and sinks j;
while (overused resources exist)
{
    for (each net, i)
    {
        rip-up routing paths and update resource usage;
        for (each sink, j, sorted by criticality)
        {
            add connectable nodes from net source node to path
            queue, sorted by cost (connectable node);
            while (sink(i,j) not found)
            {
                remove lowest cost node (lc_node) from path queue;
                add connectable nodes from lc_node to path queue,
                sorted by cost (connection to node) + cost (lc_node);
            }
            store routing path for sink j and update resource usage;
        }
    }
    update historical congestion; // Used by cost( );
    perform timing analysis and update Criticality (i,j);
}

```

Figure 2.14: Pathfinder Algorithm Pseudo Code [50]

2.4 Focus and Contributions of this Thesis

This thesis proposes a new memory architecture for FPGAs that supports wide, shallow memories. Previously devised LUT-based [1,4,5,10,11,12,13] and BlockRAM-based [1,3,4,5,6] architectures common in modern FPGAs are inefficient at implementing wide,

shallow memories. In Chapter 3 the circuitry needed to enable the proposed architecture will be examined and the circuitry's impact on existing user designs will be estimated.

In Chapter 4 the CAD algorithms that were developed in order to fully evaluate the circuitry will be described. These algorithms were created by modifying an existing tool to recognize and take advantage of the new memories. Chapter 5 contains the results of the new architecture and comments on their significance.

The contributions of this thesis are:

1. A novel memory architecture that supports wide, shallow memories.
2. CAD tools to support the proposed architecture, allowing further evaluation.
3. Experimental evaluation of the proposed architecture.

Chapter 3

Circuitry to Enable Wide, Shallow Memories

In this chapter, we will present an enhancement to the circuitry of switch blocks in order to enable the use of their configuration bits as wide, shallow user memories. We will then evaluate the impact these modifications have on speed and area, comparing our results to existing architectures.

3.1 The Motivation Behind Switch Block Memories

On-chip user storage has become an essential component of today's high-density FPGAs. FPGAs are often used to implement large systems, and these systems require high-speed buffers, tables, and other memory functions. Typically, vendors support these requirements by embedding large Random Access Memory (RAM) blocks [1,3,4,5,6]. The RAM blocks provide a very dense method of implementing storage, but they are not well suited to wide, shallow memories, which are essential to many switching and DSP applications. Block memories typically provide between 8 and 32 bits per access, meaning wide memories must be constructed by cascading several blocks. In addition, the RAM blocks in current devices are embedded at a fixed location on the chip, with all the addressing and data signals routed to and from that point.

One method for supporting wide, shallow memories is employed by Xilinx and Lucent [1,2]. In their devices, the 16-bit lookup-tables within each logic element can be configured as a RAM. Since each logic block can be configured as RAM individually, the memories can be placed close to the attached logic, and several logic blocks, each configured as a RAM, can be combined to implement wider structures. Unfortunately, the amount of storage is limited to 16 bits per lookup-table, which may not be sufficient for some applications.

In this thesis, we present an alternate implementation of on-chip memory. In our architecture we reuse the configuration memory within each switch block, providing the user with access to this memory through additional circuitry. Normally, this memory is used to store the connection patterns required to implement the user circuit. Typically, however, not all the switch blocks in an FPGA are used to transport signals. We show that, by adding only a modest amount of circuitry, the configuration memory in these unused switch blocks can be used to implement wide, shallow buffers and other similar memory structures.

Utilizing unused switch blocks in this way has a number of advantages. The amount of storage within each switch block is significant; an FPGA with 128 tracks/channel contains 1088 configuration bits within each switch block. As we will show, the structure of the switch blocks leads to a natural implementation of wide, shallow memories. In addition, since the switch blocks are evenly distributed across the FPGA, the memories created from

their configuration bits are not restricted to fixed locations on the device. This allows memories to be placed close to the logic that uses them, reducing routing delay. In order to realize these advantages, however, it is important that the additional circuitry does not slow the switch block unacceptably, or result in a significant area overhead.

3.2 Baseline Programmable Connection

In this thesis, we focus on island-style FPGAs [17]. Each horizontal and vertical channel consists of 128 parallel tracks, each track spanning four clusters (although the concept can be applied to FPGAs of any channel width and segment length). At the intersection of each horizontal and vertical channel is a Wilton switch block [20], shown in Figure 3.1 (for clarity, the figure only shows six tracks per channel, however the extension to 128 tracks per channel is clear). We assume that the tracks within each channel are staggered so that one quarter of the tracks start/terminate at each switch block. As shown in Fig. 3.2, the tracks that terminate at a switch block can be connected to one track in each of the other three incident channels, while the tracks that pass through a switch block can be connected to two tracks in each of the two perpendicular incident channels.

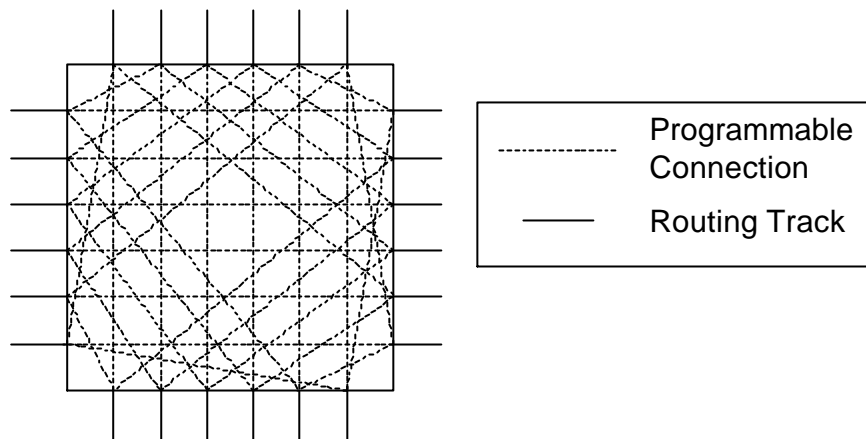


Figure 3.1: Wilton Switch Block (Repeated from Figure 2.6)

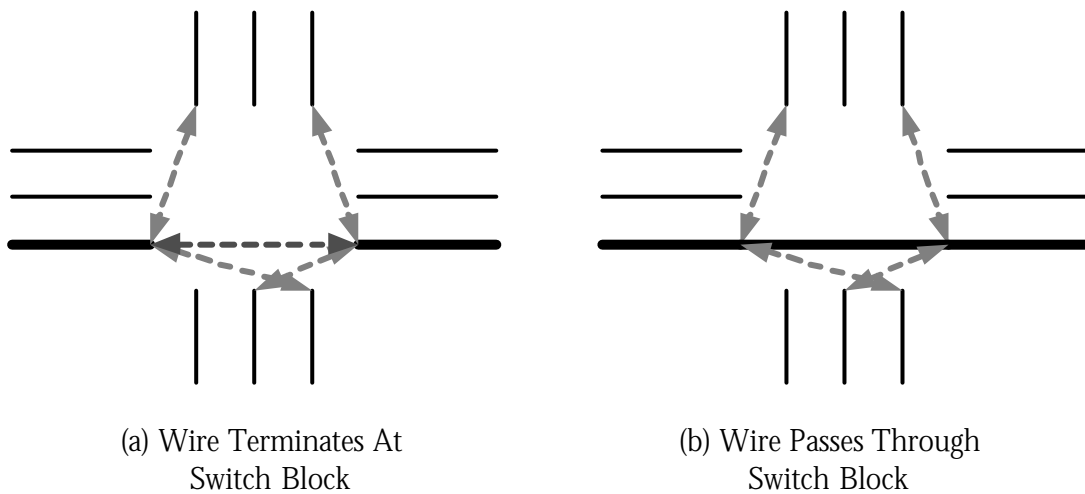


Figure 3.2: Programmable Connections within Switch Block (Repeated from Figure 2.7)

Each programmable connection within the switch block is a bi-directional re-powering programmable switch, containing buffers and two configuration bits, as shown in Fig. 3.3. The value stored in each bit is set during configuration; once the chip is configured, the values in these bits do not change.

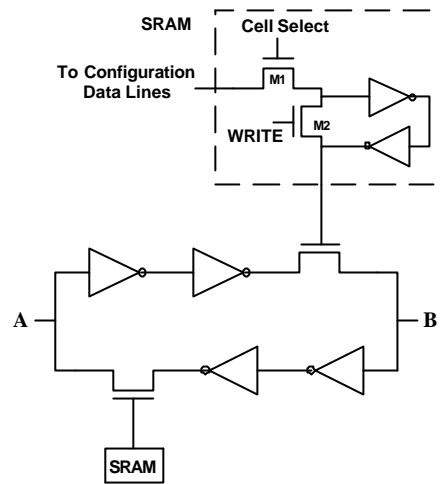


Figure 3.3: Baseline Bi-directional Programmable Connection

3.3 Enhanced Architecture

In the enhanced architecture, each switch block can be optionally used as a memory with 8 words of up to 124 bits each. Rather than adding new memory bits to the switch block (as in [61]), our approach is to provide circuitry to allow the user to write and read the configuration memory corresponding to all diagonal connections within the switch block. Assuming 128 tracks per channel, there are 512 diagonal programmable connections within each switch block. We group these programmable connections into 128 groups of 4 connections each; each group implements a single bit position of the memory. Since each connection contains two configuration bits, each group contains 8 bits of storage. By providing appropriate select circuitry, any one of these 8 bits can be read or written to; thus, the switch block acts as a 8 x 128 bit memory. As described below, to allow for address and control lines, our architecture can only use up to 124 of the 128 groups, thus, the maximum memory size we can support in a single switch block is 8 x 124.

When used as a memory, the switch block can be connected to the incident tracks in one of two modes. In the first mode, the data input for each group is taken from a track in the horizontal channel incident to the switch block, and the data output of each group is supplied to a track in the vertical channel incident to the switch block. In this mode, the address bits and write enable signal are selected from four tracks in the horizontal channel as described in Section 3.3.2. In the second mode, the role of the horizontal and vertical channels are swapped; the data inputs, address bits, and write enable signal are taken from tracks in the vertical channels and the data output bits are supplied to tracks in the horizontal channels. In both modes, the non-diagonal connections are closed, meaning that connections to the horizontal channels can come from either the right or the left side of the switch block, and connections to the vertical channels can come from above or below.

3.3.1 Enhanced Programmable Bi-Directional Connection

The key to our architecture is an Enhanced Programmable Connection element (EPC). As Figure 3.4 shows, the EPC provides access to its two configuration memory bits by linking each configuration memory cell to a horizontal or vertical track through pass transistor circuitry. Each of the two configuration bits in the EPC are connected to an intermediate staging point through pass transistors M3 and M4. These transistors are controlled by lines CA and CB, the values of which are generated by the address control circuitry, as described in Section 3.3.2. All four connections within a group share the same staging point. From this intermediate staging point, pass transistors M5 and M6 connect to a vertical and a

horizontal track respectively, with these transistors controlled by the MemVert or MemHorz lines (discussed in Section 3.3.2) which allow access to the memory from either the vertical or horizontal direction.

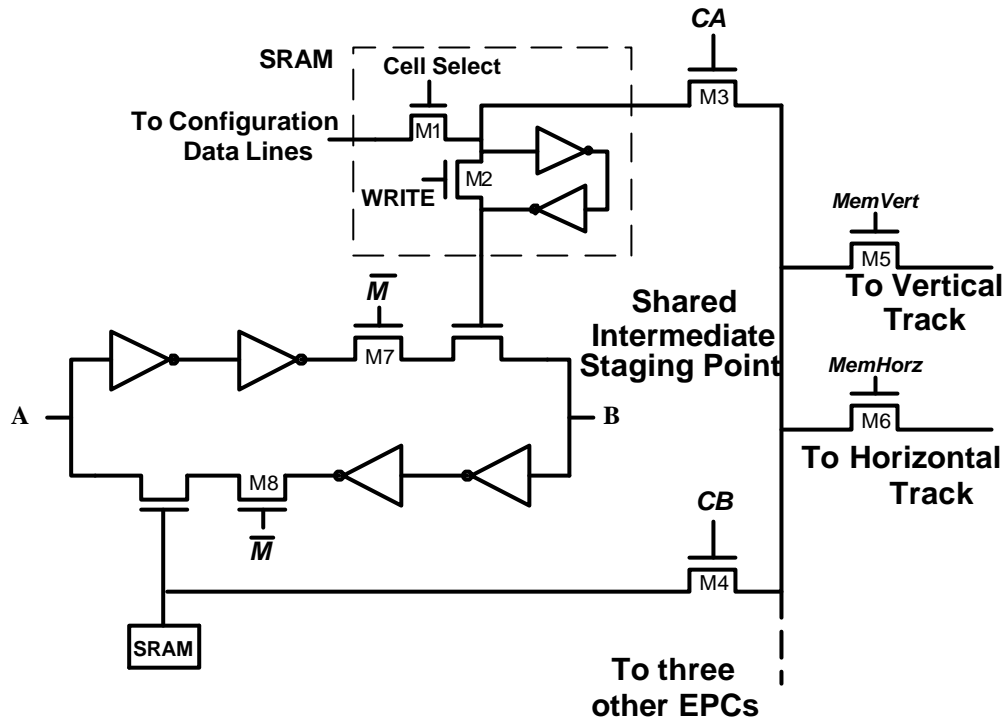


Figure 3.4: Enhanced Programmable Connection (EPC)

Additional pass transistors M7 and M8 are used to isolate the two sides of the programmable connection when the switch block is used as a memory. When not used as a memory, transistors M7 and M8 are closed, allowing the circuit to operate as a normal programmable connection.

3.3.2 Memory Addressing and Control

The eight word memory created by the grouping requires three address bits, as well as a read/write control signal. Rather than tying these inputs to four specific incident tracks, the address and control inputs are chosen from the entire set of incident tracks in either the horizontal or vertical channel, as shown in Figure 3.5. Each address and control line can be selected from one quarter of the incident tracks; those tracks chosen to supply address and control information are, of course, not used for data. The input lines to the control circuitry are shared between the horizontal and vertical incident tracks using four two-input multiplexers (only shown for line A2 in Figure 3.5). The address and control signals are fed to a 3-to-8 decoder; the eight outputs of the decoder are used to drive the CA and CB lines of one of the four programmable connections in each group.

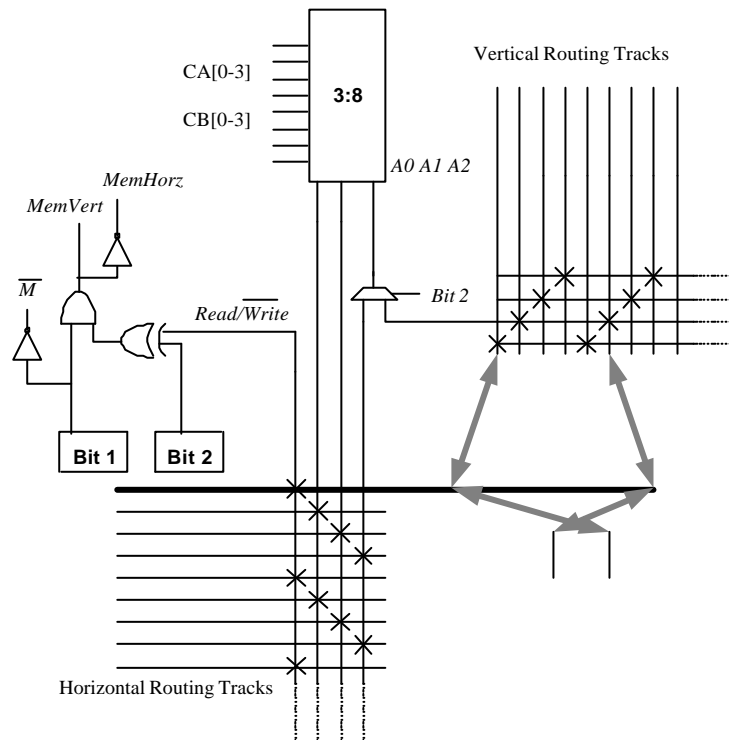


Figure 3.5: Control Circuitry

In addition to this circuitry, two configuration bits are needed per switch block. The first configuration bit is used to indicate whether the switch block is acting as a memory or as a routing switch. This bit is connected through an inverter to M7 and M8 of the EPCs. The second configuration bit indicates whether the data inputs are connected to the horizontal or vertical channel (the data outputs are connected to the other), and is used both to control the previously mentioned multiplexers, and with the read/write signal to produce the MemVert and MemHorz signals. These signals control the connection between the intermediate staging point of Figure 3.4 and the channels incident to the switch block. Note that, if the switch block is used as a memory, the intermediate staging point is connected to either the horizontal or vertical channels (depending on whether a read or write is occurring), but never both simultaneously. When not used as a memory the intermediate switching point is isolated.

Finally, transistor M2, which directly enables the writing of the memory cell, must share its control with both the configuration circuitry and the read/write line. This allows the bit to be written during configuration as well as when it is written to by the user.

3.4 Area Overhead due to the Enhanced Programmable Connection

The enhanced architecture presented in Section 3.3 will provide a large amount of user memory. However, the extra transistors required in the EPC will result in an area and speed overhead, even for circuits that do not make use of the new wide, shallow memories. In this section (and in the next), we quantify this area and speed overhead; an evaluation of

the benefits of the new architecture will be delayed until Chapter 5, after several CAD tool issues are discussed.

In order to quantify the overhead, we have assumed a “typical” FPGA, as described in Figure 3.6. Definitions of the terms in this figure can be found in [17].

Logic Blocks: 10 x 4-LUTs, 22 inputs, 10 Feedback connections
Connection Blocks: Each input connects to 25% of tracks in channel ($F_c = 0.25$)
Switch Blocks: Wilton Switch Block (Figure 2.6)
Routing Channel: Segmentation Length Four Channel Width 128
I/O Resources: 8-24 I/O Per Column at FPGA Perimeter

Figure 3.6: Architectural Parameters

In terms of area, replacing each programmable connection with an EPC requires an extra 4 transistors per connection. Since the original connection require 12 transistors, this leads to a 30 percent increase in switch block size. Of course the FPGA also contains logic blocks and connection blocks; the sizes of these blocks are not affected. To measure the overall area increase in the size of the FPGA, a detailed area model was used [17]; for the architecture in Figure 3.6, each logic block consists of 7830 transistors, each connection block consists of 1760 transistors, and each switch block consists of 23936 transistors. Adding the enhanced programmable connections increases the number of transistors in

the switch block to 31264 transistors. This translates into an estimated overall area increase of 21 percent overhead due to our enhancement.

The above numbers assume that every switch block is enhanced in this way. Of course, it is possible to enhance only some switch blocks, leading to a smaller area overhead. This will not be pursued in this thesis, yet is an interesting avenue for future work.

3.5 Speed Overhead and Transistor Sizing

The speed overhead will depend on the sizes of the new transistors in the EPC. Wide transistors add larger capacitance to routing wires; on the other hand, wide transistors are desired in order to speed up memory access. This section describes how the transistor sizes were determined; in doing so, we will estimate the memory access time as well as the speed overhead for circuits which do not use the memory.

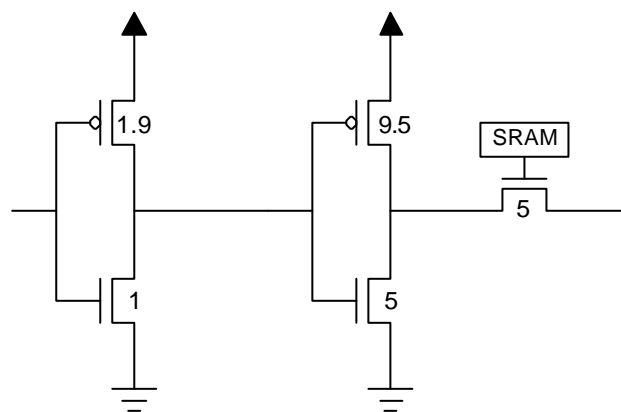


Figure 3.7: Baseline Circuit Transistor Sizes

Values for the sizes of the baseline circuit's transistors, shown above in Figure 3.7, were taken from [17]. Extensive experimental testing was carried out in order to determine the best sizes for each of the added transistors. In order to reduce the number of transistor size combinations considered, reasonable sizes were chosen based on initial estimates, and all but one were fixed in order to determine the best size for that transistor. In all cases where the initial estimates turned out to be incorrect, the prior experiments were re-run with the new transistor sizes. This did not often occur; predictions as to which transistor size would be most effective were fairly accurate. Table 3.1 shows the options from which the size for each of the transistors to be added was chosen.

Transistor	Sizes Measured
M1,M2	5, 7, 9, 10 , 11, 13, 15
M3,M4	1 , 2, 3, <u>4</u>
M5,M6	1 , 2, 3, 5, 7, <u>10</u>

Table 3.1: Transistor Sizes Tested

Using this range of sizes, along with the given sizes for the baseline circuit, we measured the point-to-point path delay and memory access time using HSPICE. To measure the delay, we simulated a path consisting of three length-four segments. The memory access time was measured by writing a value to the memory, and reading that value out. Table 3.2 shows the point-to-point path delay and the memory access time for a number of tested sizes.

Transistors (M1,M3,M5)	Routing Delay Increase (%)	Memory Access Time
(10,1,1)	11.6%	2.5ns
(5,2,1)	16%	2.27ns
(5,10,2)	16%	1.17ns
(10,4,10)	20%	0.85ns

Table 3.2: Timing information

Compared to the baseline architecture, the routing path of the best sizing option considering both area and delay (shown in **bold** in Table 3.1) was 11.6 percent slower in the enhanced architecture. This slowdown is mainly due to the parasitic capacitances of the added transistors. The corresponding memory access time was 2.5 ns, not including routing into and out of the memory. This corresponds to an operating frequency of 400MHz, faster than the 4K and MegaRAM blocks from the Stratix Architecture [3]. If we instead optimize for memory access time, the best sizing option (underlined in Table 3.1) produced a 0.85 ns memory access time, and a 20 percent impact on path delay. Since the path delay overhead affects all circuits we will use the path delay optimized sizes in the rest of this thesis.

The average speed degradation for an FPGA architecture containing EPCs rather than baseline connections was found to be 5.2 percent. Measurements were made by comparing the critical paths of a set of circuits, using both a modified and a baseline architecture file. The architecture file describes the FPGA architecture in detail, allowing CAD tools to build an accurate model of the FPGA. Sixteen large benchmark circuits were placed and routed by the academic version of Versatile Place and Route (VPR) [8] using the two

different architecture files. Table 3.3 shows the percentage degradation in speed along the critical path for all the circuits. The resistance and capacitance values for a single programmable connection were required to create the modified architecture file. These values were determined in HSPICE [9] using the point-to-point path delay optimized transistor sizes (10,1,1). In two cases, the architecture containing EPCs is faster than the one containing baseline connections. This is possible as a result of the non-deterministic nature of the CAD algorithms that are used to place and route the benchmark circuits.

Circuit	Baseline Critical Path	EPC Critical Path	Ratio
Pair	12.4 ns	11.4 ns	0.92
apex1	12.4 ns	15.9 ns	1.28
Cps	10.6 ns	11.8 ns	1.11
C6288	33.2 ns	33.6 ns	1.01
apex3	12.8 ns	13.2 ns	1.03
C7552	17.1 ns	17.6 ns	1.03
il0	22.9 ns	23.3 ns	1.02
ex5p	15.1 ns	15.3 ns	1.02
Spla	18.7 ns	20.1 ns	1.07
Pdc	25.6 ns	23.8 ns	0.93
apex4	14.6 ns	15.8 ns	1.08
Tseng	16.8 ns	16.9 ns	1.00
Bigkey	7.8 ns	8.1 ns	1.05
s38417	19.7 ns	19.7 ns	1.00
s298	25.9 ns	31.8 ns	1.23
Frisc	28.0 ns	29.4 ns	1.05
Total			1.05

Table 3.3: Benchmark Circuit Results

Using the actual transistor sizes also allows us to increase the accuracy of our area model by measuring the total number of minimum width transistor sizes [62] rather than simply the number of transistors. The minimum width transistor model is a more accurate measure of the actual size overhead of an FPGA containing EPCs because wider transistors take up

proportionally more area. Examining a single switch block we see that the number of minimum transistors increases from approximately 24000 to 36000 minimum width transistors, an increase in switch block size of 50 percent. This leads to a total area overhead of 36 percent.

These impact numbers demonstrate the effect EPCs will have on existing circuits that do not contain wide, shallow memories. In Chapter 5 we will look at the benefits of the proposed architecture for circuits that make use of switch block memories.

3.6 Chapter Summary

In this chapter, we have presented a new programmable connection element for use in an FPGA. The new architecture allows the user to write to and read from the configuration memory in unused switch blocks, providing efficient wide, shallow memories. The technique has an area overhead of 36 percent, and reduces the system performance of the FPGA by 5.2 percent. Although these numbers are not negligible, for many switching and DSP applications the ability to implement these wide memories may outweigh the area and speed penalties.

The use of these new switch block memories presents interesting CAD problems. In the next chapter we will explore these problems and we will present potential solutions.

Chapter 4

CAD Algorithms for Wide, Shallow Memories

This chapter presents enhancements for FPGA computer-aided design algorithms that allow these algorithms to target devices containing the switch block memories described in Chapter Three. Each of the major CAD stages, synthesis and mapping, placement, and routing, will be examined and solutions to the issues raised in each case will be addressed.

In order to make our algorithmic proposals concrete we have implemented our enhancements on two existing tools: Versatile Packer (VPACK) [17] and Versatile Place & Route (VPR) [8]. VPACK takes an optimized circuit description and converts it to target a specific FPGA, packing user logic into logic blocks and user memories into physical memory arrays that exist on the FPGA. VPR allows a researcher to create FPGA architectures of various capabilities and sizes, and place and route user circuits (from VPACK) in order to test each architecture.

As shown in Figure 4.1 (repeated from Figure 2.12), the CAD flow consists of three major stages: (1) synthesis and mapping, (2) placement, and (3) routing. In this chapter we will consider all three stages. Section 4.1 will discuss synthesis and mapping, Section 4.2 will discuss placement, and Section 4.3 will discuss routing.

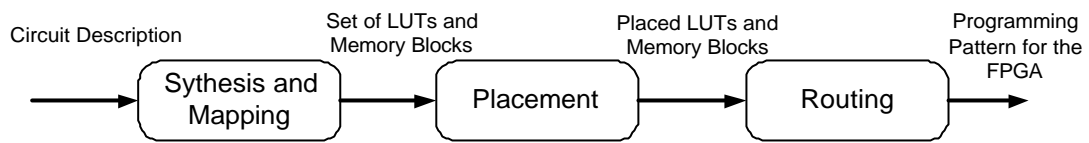


Figure 4.1: A Typical FPGA CAD Flow

4.1 Synthesis and Mapping Issues

The first major step shown in Figure 4.1 is synthesis and mapping. The synthesis and mapping tool takes a description of a circuit (in the form of a hardware description language (HDL) specification or a schematic) and transforms it into a set of logic blocks and memory blocks that implement the circuit. When targeting the proposed architecture, this algorithm must determine, for each user memory, whether it will be mapped to wide, shallow memories or to block RAM memories. We have not investigated how this determination should best be made, but postulate that it should depend on the aspect ratio and size of the user memory, as well as the dimensions of the block RAM resources on the target FPGA. In our experiments, we performed this memory allocation manually, however the VPACK tool had to be modified slightly to read and write circuits containing wide, shallow memories.

4.2 Placement Issues

Placement algorithms are used to assign a physical location on the FPGA to each logic and memory block. When fixing their location, several optimization goals can be used including minimizing wire length, wire density, delay, and power consumption. These

goals are measured through a cost function that evaluates the quality of each potential placement. As described in Section 2.3, VPR's placement algorithm starts with a random placement, and uses simulated annealing to find a good (though not necessarily optimal) solution. In order to accommodate the placement of switch block memories, several issues need to be addressed.

4.2.1 Placement of Wide, Shallow Memories

The user memories have been mapped into switch block memory-sized blocks during the mapping stage of the CAD flow. The physical location of the user switch block memories on the FPGA must still be decided. Every switch block can potentially be configured as a memory; thus there are many potential locations where the user memory blocks could be placed. This is a similar situation to logic block placement, therefore simulated annealing is a likely candidate to determine where user switch block memories should be located on the FPGA. In order to implement this, the cost function used by the placement algorithm to evaluate a placement needs to incorporate the change in cost of swapping the location of a user switch block memory.

The placement of wide, shallow memories could be accomplished with a two-step placement algorithm, where either the logic or the memory blocks are first annealed into position and locked in place, and then the blocks of the remaining type are placed onto the FPGA in the best possible configuration. Unfortunately in most designs there are numerous connections between the memory blocks and the logic blocks, meaning that the

placement of one type of block affects the other. In order to find a good placement for both logic and memory a heterogeneous placement is needed, where logic blocks and switch block memories are placed simultaneously. With a slight adaptation, nets that are associated with memory blocks can be evaluated using the same cost function as ordinary nets. In the case of a Manhattan distancebased cost function, this means that when a memory swaps to a new location, each of the nets that target or originate from that memory need to have their bounding boxes re-evaluated. For a timing-driven cost function, the delay of each net that connects to the memory would need to be re-evaluated. Manhattan distancebased placement of switch block memories has been implemented in a modification to VPR that attempts to swap switch block memory locations every 1000th swap, and attempts to swap logic blocks the other 999 times. Determining the best ratio between logic block swaps and switch block memory swaps over the course of a placement is left as future work; the 1000:1 ratio was chosen because it provides ample opportunity for switch blocks to swap, while not significantly affecting the number of logic block swaps, but it was not optimized.

Rather than start with a completely random placement of Switch Block Memories (SBMs), we attempt to distribute them evenly across the device based on the pseudo-code in Figure 4.2 which will be described in detail in Section 4.2.2. Experimental results using this heterogeneous placement algorithm will be discussed in Chapter 5.

4.2.2 Placement Constraints

Rather than allowing the CAD tool to position memory blocks anywhere, some guidelines are needed to help reduce congestion of the available routing resources near the memories. In a user circuit requiring memories that are nearly as wide as each routing channel this becomes extremely important, because almost all of the available routing tracks near a memory will be needed to route the memory's data and control lines. Several independent constraints were developed to attempt to prevent congestion near the memories.

The first modification to the placement algorithm was designed to restrict which switch blocks are available to implement memories. Valid memory locations are chosen based on the pseudo-code in Figure 4.2, which chooses positions equidistant from each other and from the edge of the circuit. Num_Mems, a measure of the number of memories in the circuit, is rounded up, so that the square root produces an integer value and the memory locations are even distributed evenly throughout the FPGA. An example of the valid switch block memory locations is shown in Figure 4.3 for a circuit with 59 memories mapped onto a 20 x 20 tile FPGA.

```

Delta_Position = FPGA_Width /  $\lceil \sqrt{\text{Rounded\_Num\_Mems}} \rceil$  ;

X = Delta_Position;
Y = Delta_Position;

For (Rounded_Num_Mems)
{
    Position(X,Y) = Valid_Choice;
    X += Delta_Position;
    if (X >= FPGA_Width)
    {
        X = Delta_Position;
        Y += Delta_Position;
    }
}

```

Figure 4.2 : Pseudo Code for Constraint Strategy #1.

This pseudo-code is also used to generate the initial placement for all other constraint strategies because it ensures a valid starting point. The block locations made available by this algorithm help reduce congestion problems that occur when two user memories are placed close together.

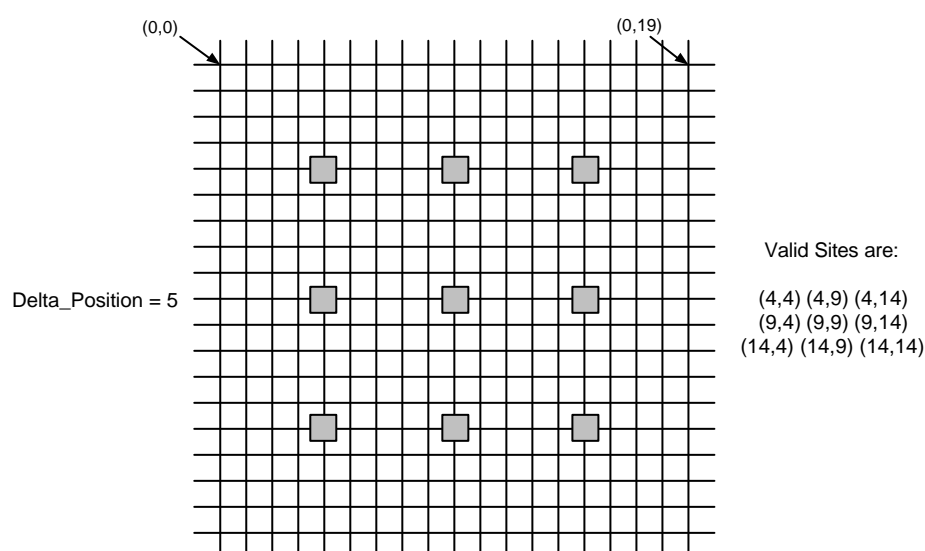


Figure 4.3 : Switch Block Memory Constraint Strategy #1.
5-9 SBMs on an 20x20 grid FPGA, Initial placement

A second, more flexible placement constraint strategy was designed. Under this strategy, memories are forced to maintain a minimum Manhattan distance from each other, by preventing swaps to locations that are too close to another memory. Maintaining the distance between memories by also swapping the memory(s) whose perimeter is being violated is complex; if multiple memories must be moved to accommodate the new location, where these memories should go is unclear, especially if moving them causes other memories to have to move. Because of the number of possible situations this is left as future work.

A third strategy was introduced to take into account the decrease in routing resources near the edge of the FPGA. Recall from Section 2.2.2 that a segmentation length of four provides wired connections that span four logic blocks. Near the edge of the chip however, this pattern changes because there are no logic blocks to span on one side. The switch blocks near the edge of the chip are likely poor choices for switch block memories, because of the significant demand on routing resources by switch block memories. We still wish to maintain a minimum distance between memories, so the third strategy follows the rules of the second, while also maintaining a minimum distance from the edge of the FPGA.

These three strategies placement strategies were implemented in VPR and were evaluated independently. Experimental results will be presented in Chapter 5.

4.2.3 Orientation-based Placement

The final issue in placement that must be considered is whether the data input accesses the memory from the horizontal channel and outputs from the vertical or vice-versa, a concept referred to as the orientation of the switch block memories. Using a Manhattan distance based cost function will not produce the best possible placement in this case because it does not differentiate between horizontal and vertical distance. Changing the orientation of the memory (for example, from horizontal input to vertical input) will not change the cost of the placement when using the existing cost function. If we want to differentiate between the two options, a weighting needs to be given to influence the likelihood that logic blocks with input nets will be placed closer to the input channel and those with outputs will be placed closer to the output channel. Figure 4.4 demonstrates the desired placement for the logic blocks that connect to and from the switch block memory, assuming the memory is configured to take input from the horizontal channel. In the case of timing-driven placement, this occurs automatically, because nets that connect to the memory inputs will be faster if they connect to the proper channel more quickly. Orientation-based placement has not been implemented.

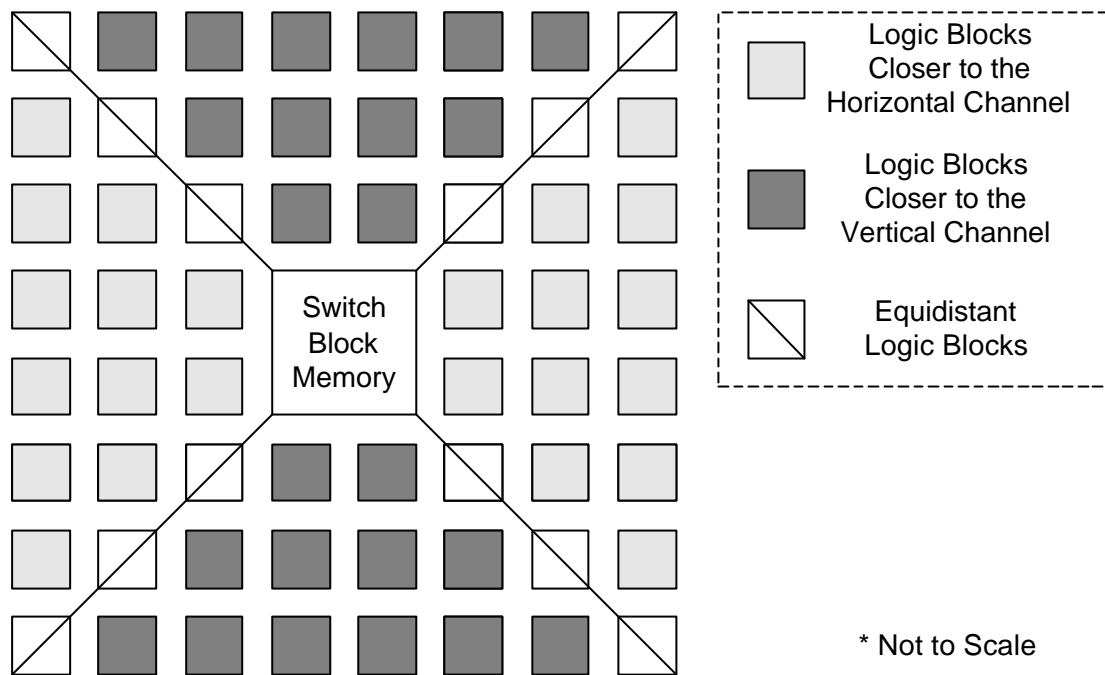


Figure 4.4 : Desired placement of logic blocks that connect to and from the SBM

4.3 Routing Issues

Routing is the process of connecting all of the nets in the netlist from their source block to their sink block(s), using the routing resources provided by the FPGA. A directed graph, referred to as the routing resource graph, describes the connectivity pattern of the FPGA; its nodes correspond to the logic / memory block pins and the routing wires, and its edges correspond to the programmable connections between those resources. With the placement complete we know the locations of the logic and memory blocks, allowing the start and end points of the nets to be located on the graph. The Pathfinder algorithm [50] can then be used to find paths for all nets.

The addition of switch block memories to the architecture changes the pattern of programmable connections available when routing. In particular, the programmable connections that connect to and from the memory must be added to the routing resource graph that represent the FPGA. The basic maze router employed by the Pathfinder algorithm may not be sufficient to route to switch memories. The potential problems of routing to and from switch block memories, along with the solutions to these problems are presented below.

4.3.1 Connecting to Switch Block Memories

In order to allow the nets in the netlist to connect to and from switch block memories, the routing resource graph must be modified. Because routing occurs after placement, the location of the switch block memories on the FPGA is known. This allows us to create a graph that models the switch blocks in those locations as memories, rather than one where all switch blocks have the memory connections and switch block connections.

In addition to adding connections to the graph that allow the routing tracks to connect to the memory, the switch blocks that are implementing memories must also be stripped of their normal routing connections. This is important because the values being stored in the memory will be arbitrary, and as mentioned in Section 3.3 they cannot be relied upon to route a signal. By removing the connections from the routing resource graph we are effectively modeling the M7/M8 transistor in Figure 3.2 that is used to prevent connectivity.

4.3.2 Potential problems with the existing routing algorithm

Recall from Section 2.2.2 that an incoming track to a switch block only connects to a certain number of perpendicular tracks. Fully connecting the tracks in incident channels takes up too much area and adds capacitance to the tracks. These limited connection patterns cause a problem when accessing the memory. In order to connect to a specific input pin on the memory, which is only accessible from a single adjacent track, we must approach the switch block from a limited number of tracks. Figure 4.5 shows in bold the tracks which are able to make a connection to the memory input pin. For clarity, we have shown a channel width of five, which because of its narrowness, has multiple connections to the same adjacent track. For a wider channel width, four perpendicular tracks can make the desired connection at each switch block. For a wide channel width with a segmentation length of four there are a total of 18 switches able to access a given track. Although the number of possible connections may seem reasonable, it is important to remember that in a user memory that is nearly as wide as the channel there will be competition amongst all nets attempting to access the memory.

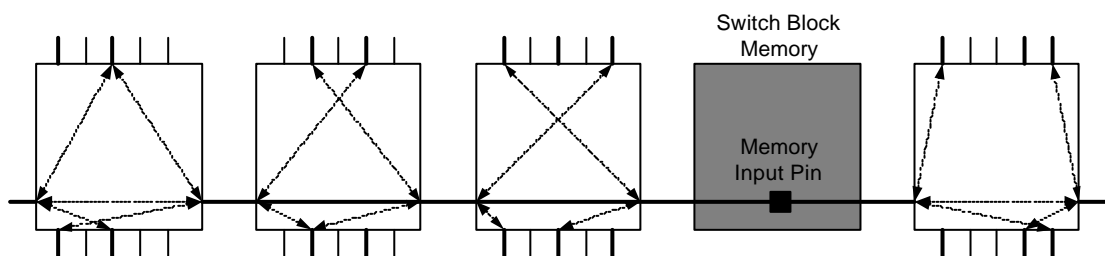


Figure 4.5 : Adjacent Tracks Able to Connect to a Specific Memory Pin
Segmentation Length 4, Channel Width 5, Wilton Switch Block

The implications of this become clearer when we examine the maze routing algorithm. As the maze routing algorithm searches for a path between the two nodes in the graph, it expands a wavefront that exists at the edge of the area that has already been searched. A speed enhancement to the original Pathfinder algorithm [50] uses A* to direct the search along a path within the graph, moving in the direction of the target block rather than expanding around the source in all directions. This is accomplished by making a conservative (best case) estimate of the cost (measured in delay, wire-length, or number of nodes) required to reach the target, and expanding only along paths that could potentially achieve that cost. Once all potential paths fail to reach the target in the estimated cost, the estimate is revised and the algorithm continues iterating until it reaches its target.

As mentioned above, the number of paths available to reach the target node decreases as the algorithm approaches the input pin of the memory. Comparatively, when routing to a logic block input node, we have several options to make a connection. First, connection blocks provide flexibility as to which track in the channel connects to the logic block, allowing us to enter the channel from a number of tracks and still be able to connect to the logic block on a specific input pin. In addition to connection blocks, local interconnect is available to connect any of the logic block inputs to any input of the LUTs, making all of the logic block inputs equivalent. This gives the router flexibility when making a connection. In the case of switch block memories a single track is the connection point.

If a user memory is nearly as wide as the channel, the overcrowding of the routing tracks near the memory could make it difficult for the router to connect all of the memory inputs. When the direct paths to an input pin are unavailable, an indirect path must be taken, adding delay to circuits that utilize these memories. Worse yet, if all paths become congested then the routing algorithm may not be able to find a solution. The effects of this restrictive routing will be shown in Chapter 5. Remedying this with the addition of a connection block or local interconnect is not feasible; it would require more transistors to implement this additional circuitry than it would to embed a wide, shallow memory onto the FPGA. Instead, a solution that takes advantage of the uniform nature of memory is proposed in the next section.

4.3.3 Equivalence-based Routing

Examining a switch block memory, we see that each input/output data pin in the memory performs the same function; it provides access to/from eight SRAM cells. In the mapping of the user's circuit to the FPGA, these data pins have been assigned to specific connections. Figure 4.6 shows an eight-bit wide memory. By mapping the input connection DataIn0 to bit 0 in the memory, we are only able to connect to it from track 0 in the input channel. We map all of the input connections up to the width of the memory, W , which can only be connected to from track W . The same is true for output connections from the memory, DataOut0- W .

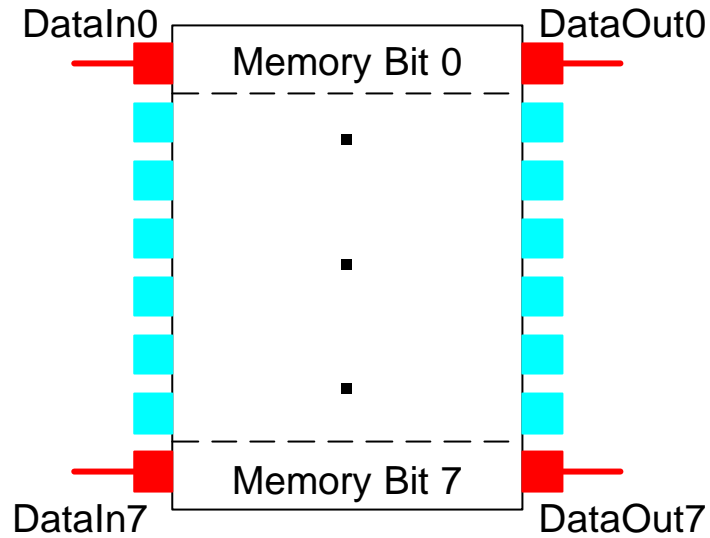


Figure 4.6 : Assignment of DataLines in Switch Block Memories

Given that each data pin pair of the memory is equivalent, this assignment of each input to a specific input pin is not strictly necessary. The assigning of input pins is done to guarantee that each bit of the user memory has an associated bit in the physical switch block memory. By removing these initial assignments and replacing them with a set of connections to be routed, the routing algorithm can connect to any available memory input pin, giving it more routing options. Overuse of the data pins in the memory is automatically avoided by the existing routing algorithm, because each datapin can only be connected to from only a single track, which the routing algorithm will not allow to become congested.

When assigning memory data pins, additional constraints must be considered. While each of the input data pins in Figure 4.6 can be associated with any of the memory connection nets, it is essential that the output pins are associated with the proper output connection

DataIn0 must access the same SRAM bits that DataOut0 does. If the assignment of DataIn0 is being made during the route then the routing algorithm does not initially know which resource DataOut0 starts from on the routing resource graph. This information is only known after the corresponding input net has been routed. In order to resolve this problem each of the output nets associated with a switch block memory are not routed until the corresponding input net has connected to the memory. Once the input connection has been assigned to a pin, the router knows which graph node to start routing the output nets from.

4.4 Chapter Summary

In this chapter we have described the modifications and improvements to the CAD tools that allow these algorithms to target FPGAs containing wide, shallow memories.

The placement of wide, shallow memories is accomplished using a heterogeneous simulated annealing algorithm, which swaps the locations of both switch block memories and logic blocks in the same iteration. Three placement constraints were established to help avoid congestion in the routing channels near the switch block memories.

Routing to and from wide, shallow memories is enabled through a modification to the routing resource graph, the description of the FPGA architecture used by the router. The inflexibility of the routing resources near the switch block memories is a potential problem

in finding a valid routing solution. Equivalence-based routing, which delays the mapping of the user memory's input pins until routing, is presented as a solution.

In the next chapter we will evaluate the options of the CAD tools. We will then evaluate the switch block memory architecture by comparing it to other memory architectures.

Chapter 5

Experimental Results

In this chapter, we present the results of our experiments. In order to evaluate the proposed architecture and CAD tools, a benchmark suite with circuits that make use of wide, shallow memories is needed. Unfortunately we do not have access to such circuits, forcing us to create our own benchmark. Section 5.1 describes a circuit that contains wide, shallow memories. The benchmark circuit allows for the evaluation of circuits that contain wide, shallow user memories on different physical memory architectures.

In this chapter we will determine:

- The area of the benchmark circuit when mapped onto FPGAs with the following memory architectures:
 - the LUT-based memory architecture,
 - the block memory architecture, and
 - the switch block memory architecture.
- The best placement strategy for the placement of switch block memories, chosen from the three strategies described in Chapter Four.

- The best routing algorithm, original or equivalence-based, for routing circuits onto an architecture containing switch block memories.
- The average critical path (timing) of the benchmark circuit on FPGAs with the following memory architectures:
 - the LUT-based memory architecture, and
 - the switch block memory architecture.

5.1 A Benchmark Suite for Switch Block Memories

When designing a benchmark circuit or set of circuits, the goal is to make it as representative of the range of user circuits as possible. If the set of circuits is sufficiently representative, then after measuring the effect of various options and architectures on the benchmarks, the averaged results of the experiments can be extrapolated to determine how real user circuits will be affected.

As previously mentioned, one of the main motivations for embedding wide, shallow memories onto FPGAs is the large number of communications circuits that are being developed. Ideally, we would like a suite of such circuits in order to properly evaluate the proposed architecture. Unfortunately, the availability of real-world commercial circuits is, and will continue to be, an issue in academic benchmarking; companies are rarely willing to donate their older circuits, much less their more recently developed ones. As a result academics are forced to use either old benchmarks like MCNC'91 or ISPD'98 [63], or look within the academic community for more recent circuits.

None of the established benchmark circuits or academic circuits available contain wide, shallow memories, forcing us to create our own benchmark. Time and technology constraints meant that rather than a suite of representative circuits, only a single benchmark circuit could be created. This circuit allows an initial evaluation of the switch block memory architecture, and will prove that the algorithms developed in Chapter Four can place and route circuits onto FPGAs supporting this architecture. However, it does not allow for a robust evaluation of the average effect on real user circuits. The creation of a more complete benchmark suite to fully evaluate the proposed architecture is left as future work.

5.1.1 Benchmark Circuit Description

The circuit chosen to test the switch block memory architecture is a crossbar switch [64,65]. A 2 x 2 crossbar is shown in Figure 5.1. Signals enter the circuit at the inputs (A and B), and are sent to one of the outputs (1 or 2) based on an address tag that is a part of the incoming data. Because data destined for the same output could arrive at both of the inputs simultaneously, the data is buffered in order to prevent data loss. The crossbar in Figure 5.1 is fully buffered, meaning that every input-output combination has its own buffer. For the 2 x 2 crossbar, this means that four memories are required to buffer the incoming data. More generally, for an N x M crossbar, N*M memories are required.

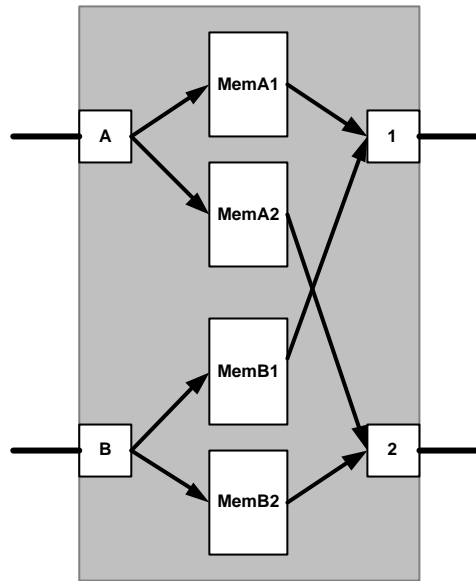


Figure 5.1: A 2 x 2 Crossbar Switch (Version A)

Our intent in using this circuit is to stress the limits of the proposed architecture, so a very wide input data width of 120 bits has been chosen. While this memory width may seem unreasonable when using conventional I/O, the development of high speed I/Os [1,2] described in Section 2.2.3 make this bandwidth feasible. As mentioned in Chapter 1, the wide outputs of this I/O on the FPGA are a part of the motivation behind the switch block memory architecture.

In order to enable mapping of the circuit to architectures containing LUT-based, block-based, and switch block-based memories, two HDL models of the circuit were devised. Version A, shown above in Figure 5.1, includes the user memories, which will be mapped using VPACK into LUT-based memories. Version B, leaves the memory for a later manual mapping step and instead targets the memory inputs and outputs to the FPGA I/O so that

they can be easily found. Figure 5.2 shows Version B of the circuit, in which the memory is accessed through FPGA I/O.

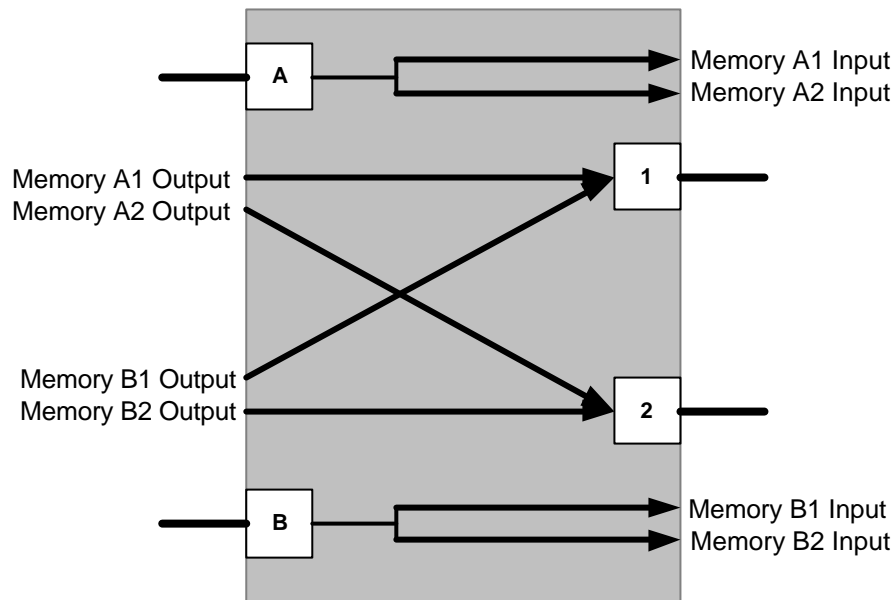


Figure 5.2: A 2 x 2 Crossbar Switch Without Memories (Version B)

5.1.2 Limiting Factors of Implementation

When dealing with wide input paths, a limiting factor in their implementation is the number of I/O available, especially in Version B of the circuit, where the memory is also accessed through the FPGA I/O so that it may be hand mapped later on. This limits the size and width of the crossbar that could be synthesized. To convert the VHDL description of the circuit to a netlist that could then be optimized and packed, the synthesis portion of Altera's MaxPlus+II software [66] was used. The output of the synthesizer (in .edif [67] format) was then converted to a VPACK [17] compatible format. Unfortunately, while MaxPlus+II is capable of outputting the synthesized circuit to a file, it was designed to do this as part of a larger CAD flow. The requirements of MaxPlus+II reflect this; a target

Altera device must be chosen to even begin the flow. Since this is not absolutely necessary for synthesis (the circuit can be optimized independently of the device it will eventually target), the maximum number of I/O in Altera's Flex10K family artificially limits the size of the crossbar that can be implemented. Although the I/O limitation affects how large a crossbar we can implement, the uniform nature of the crossbar allows us to extrapolate our results to larger circuits, thus helping us draw conclusions.

5.2 Area Results

To measure the improvement in density obtained by the proposed architecture, the pre-synthesized test circuit was mapped onto FPGAs with each of the three different memory types: LUT-based, memory block, and switch block. In order to meaningfully compare the three architectures a common measure of area is required. Section 5.2.1 details the methodology used in measuring and comparing these architectures.

The area model used to compare the three memory architectures is based on the minimum-sized FPGA, measured in minimum-width transistors, capable of implementing the test circuit. Unlike a commercial FPGA, where the logic and memory resources are fixed at the time of fabrication, the minimum-sized FPGA provides the exact logic and memory resources required by the circuit. However, the minimum-sized FPGA that the benchmark circuit can be implemented on is a measure of the density of that implementation. If the circuit can be implemented on a smaller minimum-sized FPGA by using an alternate memory architecture, then it can fit in a smaller portion of the commercial FPGA, allowing

bigger circuits to be implemented using the remaining portion. Equivalently, if the circuit can be implemented on a smaller FPGA, it can potentially fit in a smaller device within the same FPGA family, saving money for circuit designers.

5.2.1 Area Methodology

On an island-style FPGA, “islands” of logic blocks are separated into rows and columns by a “sea” of interconnect. The logic resources and routing resources are identical from one row or column to the next. Figure 5.3 demonstrates a simple FPGA constructed from a single tile, consisting of one logic block, two connection blocks, a switch block, and a portion of the horizontal and vertical routing channels. The tile has been replicated to create a 10 column x 10 row FPGA.

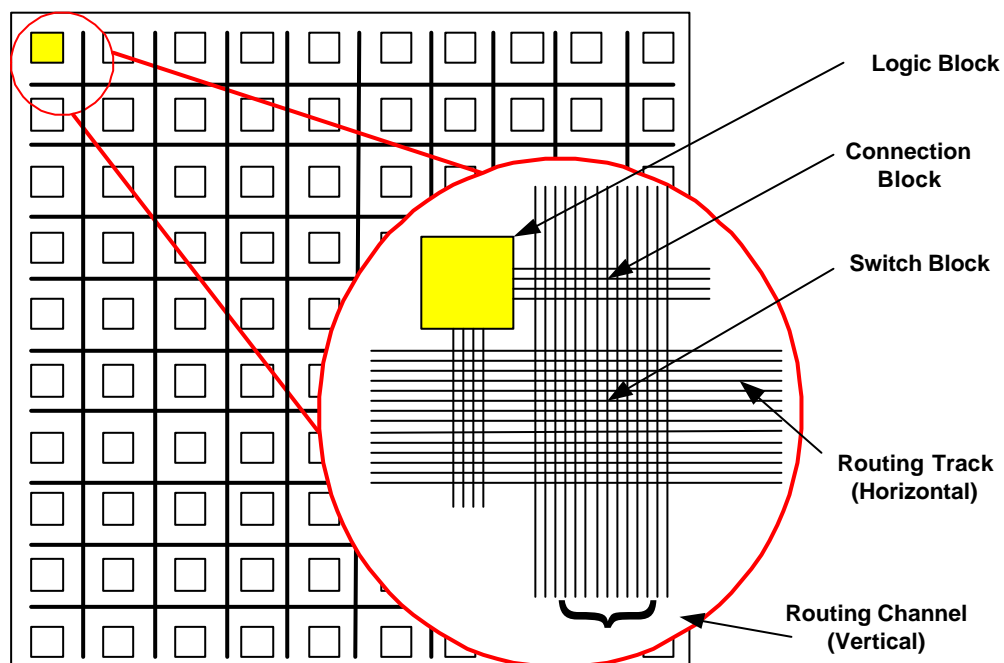


Figure 5.3: A Simple FPGA Constructed From a Single Tile

The *size* of the FPGA required to implement the benchmark circuit is expressed as the minimum number of tile columns and rows. The *area* of a single tile consists of four components:

1. Logic Block Area: The number of minimum-width transistors in a logic block.
2. Connection Block Area: The number of programmable connections in the connection block times the number of minimum-width transistors per connection.
3. Switch Block Area: The number of programmable connections in the switch block times the number of minimum-width transistors per connection.
4. Memory Area: The area taken up by memory varies depending on the memory architecture and is discussed in detail below.

In order to determine the area of each of these components information about the FPGA architecture is needed. The architectural parameters shown below in Figure 5.4 are common to all three memory architectures that were compared.

Logic Blocks: 10 x 4-LUTs, 22 inputs, 10 Feedback connections
Connection Blocks: Each input connects to 25% of tracks in channel ($F_c = 0.25$)
Switch Blocks: Wilton Switch Block
Routing Channel: Segmentation Length Four
I/O Resources: 8-24 I/O Per Column at FPGA Perimeter

Figure 5.4: Common Architectural Parameters

Table 5.1 shows the number of minimum-width transistors for each component that makes up a tile's area. These values were calculated using the area model from [17]. The area of switch blocks are expressed as the number of minimum-width transistors per track, because the required channel width of the FPGA varies significantly between memory architectures.

	Minimum Width Transistors
10 x 4-LUT Logic Block	7830
Connection Block	1840
Switch Block	187 / track
Switch Block (EPC)	255 / track

Table 5.1: Number of Minimum-Width Transistors for Each Component

Expressing the area of an FPGA as the total number of minimum width transistors required to implement the test circuit is useful, however, the size of the FPGA (in columns and rows) presents a more understandable view of the FPGA. In order to be able to accurately compare architectures based on FPGA size, the tile for that particular architecture must be normalized to a *base tile* area. For the purposes of this thesis, the base tile has a channel width of 128 tracks, and uses the architecture described in Figure 5.4. Using Table 5.1, the number of minimum-width transistors in the base tile is 35,446.

In the LUT-based memory architecture, no additional area is needed to account for the memory, since all memory is packed into logic blocks. In the case of block memories, the size of a configurable 4K memory block and the associated routing is estimated to be 2.5 times the size of a tile containing a logic block, or 88,615 minimum-width transistors. Finally, for switch block memories, the additional circuitry of the enhanced programmable

connection increases the number of transistors per track to 255, as Table 5.1 reflects resulting in a larger tile size.

5.2.2 LUT-based Memory

Version A of our test circuit was mapped onto an FPGA modeling the architectural parameters in Figure 5.4. No block memory resources were included within the FPGA, forcing the four user memories to be mapped directly to the 16-bit LUTs. The smallest size FPGA that can implement this circuit consists of a 49 x 49 set of logic and routing tiles for the FPGA. Because user logic and memory on this FPGA are less densely packed than the FPGA architectures containing block or switch block memories, the routing resources required are not as significant; the FPGA only requires 25 tracks per channel as opposed to 128 tracks per channel required by the other architectures. This means that the LUT-based tiles area is only 14,345 minimum-width transistors. Normalizing leaves us with a 19.8 x 19.8 base tile-sized FPGA.

5.2.3 Block Memory

Version B of the test circuit was mapped to an architecture with 4K block memories similar to those found in Altera's Stratix architecture. The non-memory portions of the circuit can be implemented on an 8 x 8 base tiled FPGA with a channel width of 128. In order to find the minimum-sized FPGA required to implement the test circuit we need to determine how many memory blocks are required to implement the memory portion. Using the methodology described in Section 2.3.1, the aspect ratio is chosen to meet the

depth requirements of the memory. Using an aspect ratio of 128 x 32 for the 4K blocks, four memories are needed to implement each of the 120 bit wide memories found in the test circuit. Converting the area of the 16 memories required (1.42M minimum-width transistors) into base tile sizes and adding them to the 8 x 8 FPGA required to implement the non-memory portion of the circuit results in a square 10.2 x 10.2 base tile-sized FPGA.

5.2.4 Switch Block Memory

For the switch block memory architecture, the non-memory portions of the circuit can again be implemented on an 8 x 8 FPGA with a channel width of 128. The memories are mapped to four switch blocks and thus take up no extra tiles. However, the extra circuitry to enable switch block memories adds transistors to the switch block component of the tile. The area of a switch block memory tile is 47,191 minimum-width transistors. Translated into logic and routing tiles of the base size, the size of the FPGA required is 9.3 x 9.3 base tiles.

5.2.5 Comparing Memory Architectures

The size of FPGA required to implement the benchmark circuit using each of the three memory architectures is shown in Figure 5.5. The base tile size is superimposed onto each architecture in the lower left corner as a dark gray box, and the size of the FPGA (in base tile size) is listed. Table 5.2 lists this FPGA size, as well as the total number of transistors for the three memory architectures. Even though the switch block memory architecture has significant area overhead, it still provides a denser implementation of wide, shallow

memories than the other memory architectures. Even with only four switch block memories in use out of a possible 64, the architecture is still 22 percent smaller than a block memory-based solution, and 11.4 times smaller than a LUT-based solution. In addition, adding more memories to the FPGA does not increase the area overhead; a circuit with more memories (and the same amount of logic) could be implemented on an FPGA of the same size, increasing the density even more.

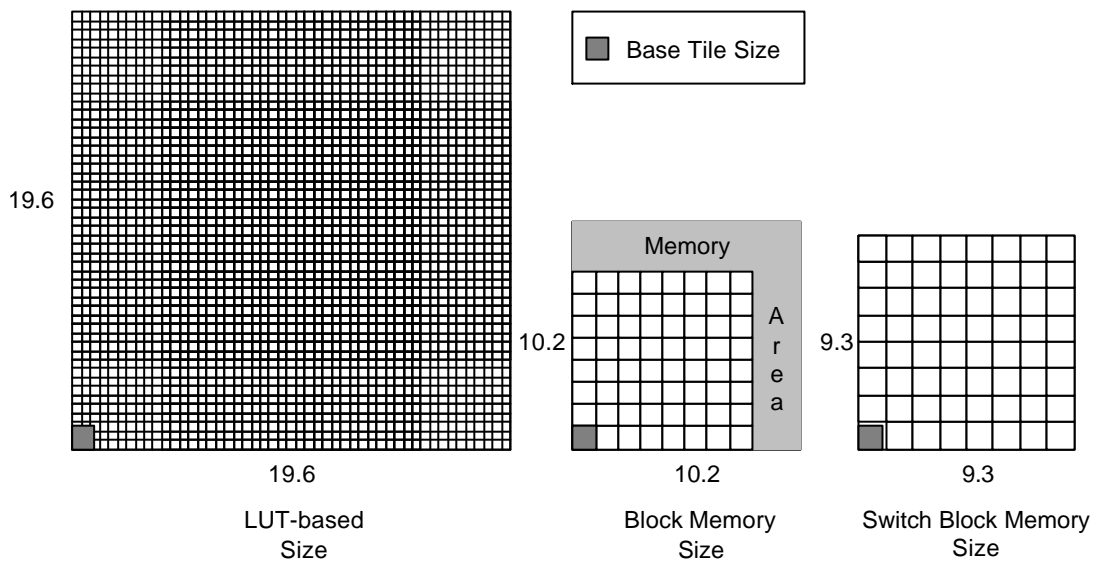


Figure 5.5: Area Comparison of the Three Memory Architectures

	Total # of Minimum-Width Transistors	Minimum Size (in Base Tiles)	Ratio
LUT-based	34.4M	19.6 x 19.6	11.4
Block Memory	3.69M	10.2 x 10.2	1.22
Switch Block	3.02M	9.3 x 9.3	1

Table 5.2: Area Results for the Benchmark Circuit

Using knowledge of the crossbar circuit, we can extrapolate the results that would be obtained if a 4 x 4 crossbar switch had been implemented. The amount of memory and control required by a 4 x 4 crossbar is roughly quadruple that of a 2 x 2 crossbar switch. In the case of LUT-based memories, both the memory and non-memory portion of the circuit are packed into LUTs, so the size of the FPGA required is quadrupled to 98 x 98 LUT-memory tiles, or 39.2 x 39.2 base tiles. In the case of block memory, the 8 x 8 non-memory portion of the array was also quadrupled, however the increased memory requirements mean that 64 blocks of memory are needed to implement the memory portion of the circuit, rather than the 16 needed in the 2 x 2 crossbar case. When translated to base tile size and added to the control logic, we find that a 22.7 x 22.7 base tile-sized FPGA is required. The required size of an FPGA using the switch block memory architecture quadruples to a 16 x 16 array, which after normalization becomes a 18.6 x 18.6 base tile sized FPGA. Table 5.3 shows the extrapolated area results for the 4 x 4 crossbar circuit.

	Total # of Minimum-Width Transistors	Minimum Size (in Base Tiles)	Ratio
LUT-based	138M	39.2 x 39.2	10.9
Block Memory	14.8M	22.7 x 22.7	1.22
Switch Block	12.1M	18.6 x 18.6	1

Table 5.3: Projected Area Results for a 4 x 4 Crossbar Circuit

Comparing the three implementation options again, we see that the area improvements of switch block memories scale roughly linearly as the number of memories in the benchmark circuit grows. For the 4 x 4 crossbar circuit the proposed architecture is 22 percent smaller than a block memory-based solution, and 10.9 times smaller than a LUT-based solution.

These results clearly demonstrate how poorly existing architectures deal with wide, shallow memories. Switch block memories are able to densely implement wide, shallow memories, despite the area overhead from the additional circuitry.

5.3 Timing Results

In this section, the results of the placement and routing of the benchmark circuit onto the LUT-based and switch block memory architectures are discussed. Using only a single benchmark circuit, it is hard to accurately measure the overall timing benefits of the proposed architecture on the different types of circuits that could potentially take advantage of switch block memories. Still, conclusions about the impact that the proposed architecture has on circuits that are similar to the benchmark can be examined, and the placement and routing tools developed for the architecture in Chapter 4 can be verified as working.

5.3.1 Timing Methodology

The benchmark circuit presented in Section 5.1 must be placed and routed onto the two available memory architectures: LUT-based and switch block memory. The resulting critical paths of the circuit's implementation can be compared to determine the timing benefit of switch block memories. The non-deterministic nature of the placement and routing algorithms can cause significant differences in the resulting critical paths, potentially making conclusions drawn from a single placement and routing run error

prone. To overcome the dependence on the random number that initializes the non-determinism, every unique place and route attempt is iterated 10 times, each time with a different initial random seed. A unique place and route attempt is defined as a placement and routing of the benchmark circuit onto an FPGA, with either an algorithmic option or the channel width of the architecture varied.

In some cases the routing algorithm fails to find a valid routing solution. *Routability* is the term given to the number of times that a valid route was found by the routing algorithm and is measured as a percentage. In order for a placement and routing combination to be accepted they must have a routability greater than or equal to 80 percent, that is, the routing algorithm must be able to route at least eight of the ten iterations.

With as many as ten critical paths per unique place and route attempt, there is no longer a single critical path that can be used in the comparison of the CAD tools and memory architectures. Instead, the *average critical path* of the successful iterations, is used when making a comparison between attempts. The tables where these averages are presented also present the range of critical paths.

5.3.2 LUT-based Memory

To provide a valid comparison between the LUT-based memory architecture and the switch block memory architecture an attempt must be made to find the best placement algorithm, routing algorithm, and architectural settings. There are two cost function options, in the

placement of Version A of the benchmark circuit onto the FPGA containing LUT-based memory: bounding box, or timing-driven placement. Once placement is complete, a timing-driven routing algorithm routes the circuit. Channel widths of 19 and 25 were both tested in an attempt to find the best solution. Table 5.4 summarizes the timing results for the four unique place and route attempts using the LUT-based memory architecture.

Channel Width	Bounding Box		Timing Driven			
	Routability (%)	Critical Path Range (ns)	Critical Path Average (ns)	Routability (%)	Critical Path Range (ns)	Critical Path Average (ns)
19	100	13.0 - 15.0	13.9	90	15.0 - 19.3	16.9
25	100	12.9 - 15.3	13.9	100	10.6 - 11.3	10.9

Table 5.4: LUT-based Timing Results

The best average critical path (10.9 ns) of the placement and routing of the benchmark circuit onto an FPGA with the LUT-based memory architecture uses timing-driven placement and a channel width of 25.

5.3.3 Block Memory

Unfortunately, a version of VPR capable of the placement and routing of block memories has not been developed. This prevents us from properly gathering timing results regarding block memory-based architectures and makes a direct comparison of block memory-based architectures to switch block memory-based architectures impossible. However, the access time of switch block memories, even when optimized for routing allows them to operate at speeds of up to 400 MHz. Comparatively, the Stratix architecture's memory operates at 312MHz for the 4K blocks and 300MHz for the MegaRAM blocks [3]. This suggests that

the benchmark circuit could be up to 30 percent faster on an architecture containing switch block memories, depending on the input and output path.

5.3.4 Switch Block Memory

After the placement and routing of Version B of the benchmark circuit to an architecture containing switch block memories, we can compare timing results with the LUT-based architecture. In order to do so, we must first evaluate the various CAD tool options, and choose the best. The combination of each of the three placement strategies with the two routing styles produces six placement and routing combination possibilities. Four channel widths (128, 140, 150, and 160) for the architecture are explored to establish the routability and average critical paths of the switch block memory architecture.

5.3.4.1 Comparison of Placement Strategies

In the first placement strategy, each switch block memory can only be placed in one of four locations. The results obtained using this placement strategy, in combination with both the original routing algorithm and the equivalence-based routing algorithm, are shown in Table 5.5. Given sufficient channel width, this placement strategy is capable of producing a placement with greater than 80 percent routability. The resulting critical paths of the test circuit vary significantly; the best and worst case critical path for the equivalence-based router with a channel width of 140 varies by a factor of 5. This provides significant justification to the parameter variation proposed in [52] when investigating new FPGA

architectures. If only one iteration had been run, the observed results could have been based on a worst case (or best case) implementation.

Channel Width	Original Router			Equivalence-Based Router		
	Routability (%)	Critical Path		Routability (%)	Critical Path	
		Range (ns)	Average (ns)		Range (ns)	Average (ns)
128	0	N/A	N/A	0	N/A	N/A
140	0	N/A	N/A	90	5.4 - 28.7	10.5
150	100	5.6 - 11.3	7.3	100	5.6 - 21.3	10.6
160	100	5.6 - 14.0	6.7	90	6.9 - 13.8	10.2

Table 5.5: Placement Strategy One Timing Results

A close investigation of individual iterations reveals that for the solution with the worst critical path, the control circuitry of the benchmark circuit is very inefficient. This circuitry determines which of the memories are connected to the I/O, and so it connects to the large number of logic blocks that are de-multiplexing the memory outputs and are spread across the chip. The crowded resource usage in the channels near the switch block memories prevents direct connections from being made to a number of these logic blocks. In the best case, the placement allows for a much more efficient routing of the control circuitry.

In the case of the second and third placement strategies, there are two parameters that can be varied:

- Memory Distance: the distance that the algorithm must maintain between switch block memory locations, and
- Edge Distance: the distance that the algorithm must maintain between memories and the edge of the FPGA.

For the second placement strategy, the edge distance parameter is set to zero, allowing switch block memories to be placed at the edge of the FPGA. Table 5.6 illustrates the timing results of the second placement strategy using both the normal and equivalence based routers. In the case of the third placement strategy, the edge distance can be varied. Tables 5.7 and 5.8 present results for an edge distance of one and two, respectively. As mentioned in Section 5.3.1, all possible combinations were repeated ten times using different random number seeds each time. Track width was varied between 128, 140, 150, and 160, and the minimum distance between memories was varied between 3, 5, and 6.

		Original Router			Equivalence Based Router		
		Routability (%)	Critical Path		Routability (%)	Critical Path	
		Channel Width	Range (ns)	Average (ns)	Channel Width	Range (ns)	Average (ns)
Memory Distance = 3 Edge Distance = 0	128	0	N/A	N/A	0	N/A	N/A
	140	0	N/A	N/A	50	5.6 - 22.7	11.6
	150	0	N/A	N/A	40	7.4 - 10.3	8.8
	160	0	N/A	N/A	40	5.7 - 11.5	8.3
Memory Distance = 5 Edge Distance = 0	128	0	N/A	N/A	10	9.3	9.3
	140	0	N/A	N/A	50	5.4 - 9.6	7.6
	150	20	5.7 - 5.8	5.8	60	6.1 - 16.1	9.7
	160	30	5.7 - 19.0	8.4	60	5.6 - 16.9	8.2
Memory Distance = 6 Edge Distance = 0	128	10	13.9	13.9	0	N/A	N/A
	140	10	17.9	17.9	20	5.5 - 6.1	5.8
	150	10	5.5	5.5	30	6.1 - 13.1	8.6
	160	10	7.4	7.4	40	5.5 - 16.2	8.3

Table 5.6: Placement Strategy Two Timing Results (Edge Distance = 0)

Analyzing the results from Table 5.6, we observe that the routability, or number of times a valid routing solution for the benchmark circuit is found, is significantly affected by the decrease in available routing resources around the edge of the FPGA.

		Original Router			Equivalence Based Router			
		Routability	Critical Path		Routability	Critical Path		
		Channel Width	(%)	Range (ns)	Average (ns)	(%)	Range (ns)	Average (ns)
Memory Distance = 3 Edge Distance = 1	128	0	N/A	N/A	10	13.1	13.1	
	140	10	9.4	9.4	70	6.5 - 16.4	8.6	
	150	50	5.47 - 20.1	10.6	90	5.5 - 13.7	8.5	
	160	50	6.5 - 19.4	11.9	70	5.5 - 24.8	12.7	
Memory Distance = 5 Edge Distance = 1	128	20	5.7 - 7.4	6.6	10	8.1	8.1	
	140	20	6.1 - 7.0	6.6	80	5.6 - 10.2	7.8	
	150	50	5.4 - 11.4	6.9	90	5.4 - 14.9	9.5	
	160	50	5.4 - 6.1	5.8	80	5.4 - 14.4	8.6	
Memory Distance = 6 Edge Distance = 1	128	10	5.6	5.6	0	N/A	N/A	
	140	10	5.6	5.6	100	5.7 - 24.0	8.7	
	150	20	5.6 - 6.3	6.0	90	5.4 - 21.3	9.2	
	160	20	5.7 - 5.8	5.7	90	5.5 - 11.9	6.2	

Table 5.7: Placement Strategy Three Timing Results (Edge Distance = 1)

		Original Router			Equivalence Based Router			
		Routability	Critical Path		Routability	Critical Path		
		Channel Width	(%)	Range (ns)	Average (ns)	(%)	Range (ns)	Average (ns)
Memory Distance = 3 Edge Distance = 2	128	20	6.0 - 12.3	9.1	0	N/A	N/A	
	140	20	5.7 - 13.3	9.5	40	6.8 - 11.8	6.6	
	150	50	7.9 - 20.5	14.3	50	5.7 - 23.0	10.5	
	160	40	6.3 - 19.5	11.2	40	5.8 - 10.0	7.8	
Memory Distance = 5 Edge Distance = 2	128	10	5.4	5.4	0	N/A	N/A	
	140	10	6.5	6.5	100	6.0 - 12.8	8.2	
	150	40	5.5 - 5.9	5.7	90	5.6 - 14.0	7.3	
	160	40	5.4 - 5.8	5.5	30	8.9 - 16.2	12.2	
Memory Distance = 6 Edge Distance = 2	128	0	N/A	N/A	0	N/A	N/A	
	140	0	N/A	N/A	100	5.5 - 12.4	7.7	
	150	60	5.4 - 10.7	7.1	80	5.4 - 9.7	7.0	
	160	60	5.5 - 8.0	6.3	90	5.4 - 34.0	10.9	

Table 5.8: Placement Strategy Three Timing Results (Edge Distance = 2)

Examining the results from Tables 5.7 and 5.8, we observe that the routability increases as the distance between memories increases. As the minimum distance between the memories and the edge of the chip increases beyond the initial edge distance of 1, however, the routability is the same or worse. This is attributed to the fact that within the test circuit, the memory is connected directly to the I/O of the FPGA, making a closer

connection desirable. Based on our routability observations, memory distances of 5 and 6 and edge distances of 1 and 2, in combination with the equivalence-based router were the four best choices. Of those four, a minimum memory distance of 6 and a minimum edge distance of 1 had the best average critical path, 7.8 ns for a channel width of 140.

Comparing placement strategies one and three, we observe a similar level of routability between each of their best options; both strategies are over 80% routable using the equivalence-based router. The critical path of placement strategy three, however, is more consistent at finding solutions with shorter critical paths. Strategy three with a minimum memory distance of 6 and a minimum edge distance of 1 has an average critical path that is 24 percent faster than the best variant of strategy one, making it the best overall placement strategy for switch block memories.

5.3.4.2 Evaluation of the Equivalence-based Router

Throughout the examination of placement strategies, the equivalence-based router provided the only solutions with sufficiently acceptable routability. If we compare across the entire set of place and route attempts, equivalence-based routing was 2.5 times more likely to produce a successful route than the normal router. For the best placement strategy (strategy three, edge distance = 1, memory distance = 6), the equivalence-based router was able to route 28 out of a possible 40 iterations of the test circuit, and 10 of the unroutable solutions were at the minimum possible track width. The original router was

only able to route 18 of the 40 iterations using the placement strategy that produced its highest routability (strategy three, edge distance = 0, memory distance = 5).

The average critical path of the benchmark circuit routed by the normal router is faster than the equivalence-based routing algorithm in most cases. This is most likely because the strict routing requirements near the memories force the normal routing algorithm to find only good solutions. Examining the cases where the equivalence-based routing algorithm produces a better solution than the original, we find further evidence that the strict routing requirements of the original routing algorithm cause routing problems. These poor routing solutions chosen by the original routing algorithm can give insight into the potential problems. Figure 5.6 shows an example route of the benchmark circuit where the solution was worse than the equivalence-based router's solution. The density of the used routing (shown in black) is particularly intense on the right half of the FPGA. Figure 5.7 shows the critical path of the circuit.

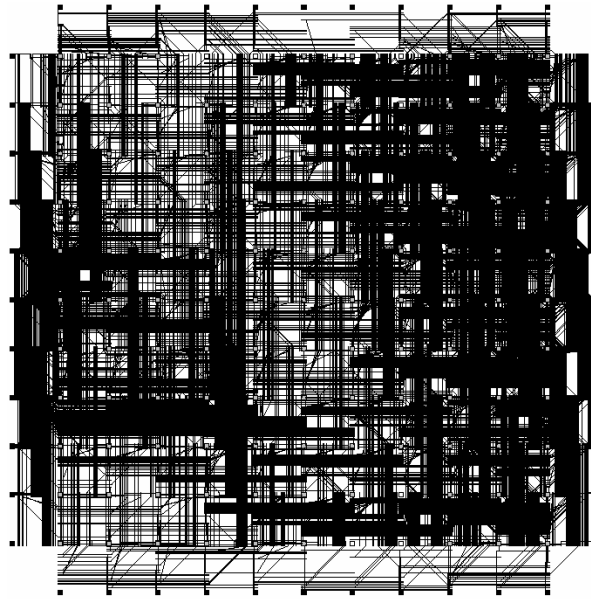


Figure 5.5: Example Route of the Benchmark Circuit

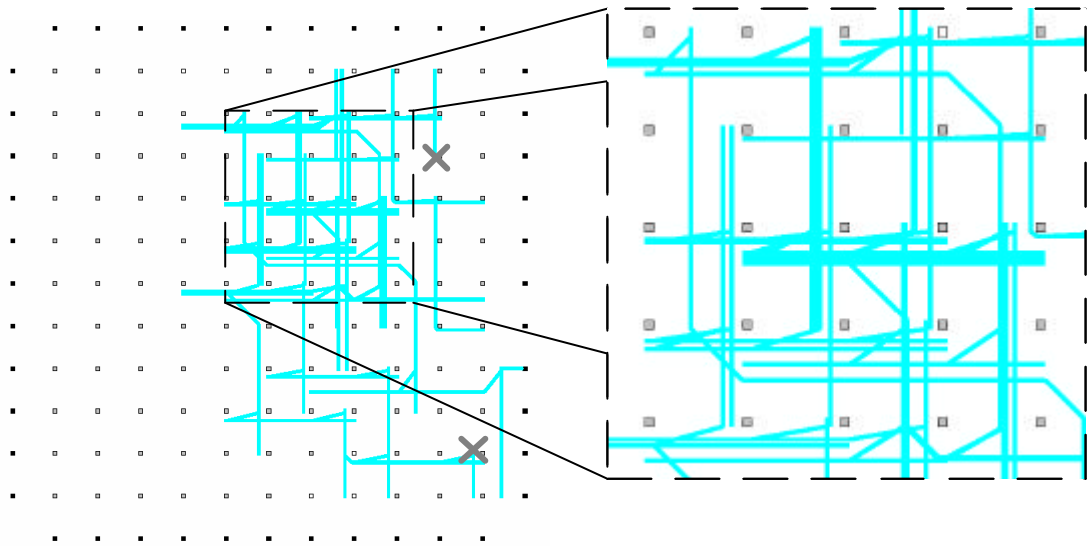


Figure 5.7: Example Critical Path of the Benchmark Circuit

The net shown in Figure 5.7 connects the I/O pin at the right edge of the FPGA to the two memories, each shown as a dark grey X. This routing of the critical path is very inefficient; within the expanded section of the net the path circles around a number of

resources in order to access the memory, so a large number of them could potentially end up on or near the critical path. In most cases, this problem would cause the route to fail because the critical paths use up all the available routing resources, leaving none for the rest of the nets. However, a connection to the desired track could be found by circling through other switch blocks until the router finds a free resource that can connect to the desired track. The critical path of the benchmark circuit has been measured up to five times slower than the best solution's critical path in such cases. Thus, because of the increased routing flexibility, which not only greatly increases routability but also avoids circular critical paths, the equivalence-based routing algorithm was chosen.

5.3.5 Comparing Memory Architectures

The average critical path using the best placement and routing algorithms and architectural options for the implementation of the benchmark circuit onto an FPGA with the LUT-based memory architecture is 10.9ns. Comparatively the average critical path of the of the best set of options for the switch block memory-based architecture is 7.8 ns, 40 percent faster.

5.4 Chapter Summary

In this chapter the area and timing results of a benchmark circuit implemented on the switch block memory-based architecture were compared with the benchmark's implementation on LUT and block memory-based architectures.

The results of experimentation show:

- The mapping of the testbench circuit onto an FPGA using the switch block memory architecture took 22 percent less area than the same circuit mapped onto a block memory-based FPGA, and 11.4 times less area than the same circuit mapped onto a LUT memory-based FPGA
- The best placement strategy for the placement of switch block memories was placement strategy three, where memories must maintain a distance of at least five switch blocks from each other, and a distance of at least one switch block from the edge of the FPGA. For a channel width of 140 the benchmark circuit had an average critical path of 7.8 ns when this strategy was used in combination with the best routing algorithm.
- The best routing algorithm, equivalence-based routing was 2.5 times more likely to produce a successful routing solution than the normal router.
- The placement and routing of the benchmark circuit onto an FPGA containing switch block memories was 40 percent faster than the same circuit placed and routed onto an FPGA containing LUT memory, using the best placement and routing options available for each memory architecture.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In the past ten years, the capacity of FPGAs has grown to such an extent that they are now being used to implement entire systems, and represent an alternative to System On a Chip (SOC) and Application Specific Integrated Circuit (ASIC) development. Logic and memory resources on the FPGA have reached levels such that they are capable of providing a complete, one-chip solution. I/O capabilities have also followed this trend, most notably with the introduction of highspeed I/O that allow GHz range signals to access an FPGA running at a lower frequency. High speed I/O has also allowed users to implement high bandwidth circuits on an FPGA. The high speed interface is accomplished by time demultiplexing an incoming signal to create a wide set of signals on the FPGA. These signals are often related, especially in communications circuitry, and if so must be buffered in wide, shallow memories as they are processed throughout the FPGA. This thesis is the first academic research towards an FPGA architecture that supports wide data signals. As I/O bandwidth on FPGAs increases, and more circuits begin to take advantage of these high speed I/Os, the need for efficient wide data blocks, including wide, shallow memories, will grow.

This thesis examines an FPGA architecture that allows switch blocks to be used as wide, shallow memories capable of buffering wide data paths. Chapter 3 explores the circuitry that must be added to the FPGA in order to implement these switch block memories, and measures the impact that these additions have on existing user circuits. The additional circuitry makes an FPGA 36 percent larger, assuming that every switch block contains the circuitry. As well, the critical path of circuits that are implemented on an FPGA with the switch block memory-based architecture are 5 percent slower.

Chapter 4 discusses the computer-aided design algorithms required to implement switch block memories. Three placement strategies which limit the physical placement of switch block memories were proposed, to help avoid overcrowding of the nearby routing resources. An alternative routing algorithm, equivalence-based routing, which has more flexibility when routing circuits containing switch block memories, was also proposed to alleviate congestion

Finally, Chapter 5 evaluates the benefits of switch block memories by comparing them to alternative memory architectures. A benchmark circuit was developed to make this comparison, and was implemented on the various architectures. When implemented on an FPGA using the switch block memory architecture, the benchmark circuit was 22 percent smaller than the solution provided by the nearest competing memory architecture.

In addition, the benchmark circuit was on average 40 percent faster on an FPGA using the switch block memory architecture than on one using the LUT-based memory architecture.

6.2 Future Work

This thesis is the first published research towards supporting user circuits with wide data paths. This section discusses possible extensions to the proposed architecture that will enhance this support. Developing benchmarks, algorithmic improvements, and architectural improvements are all important components of FPGA research and are discussed in turn below.

6.2.1 Developing Wide, Shallow Benchmarks

In order to fully evaluate the switch block memory architecture, and other architectures designed to support wide data paths, a more complete benchmark suite, one that contains a large number of circuits, must be developed. These circuits should be representative of the set of user circuits that need wide, shallow memories. With this benchmark suite the impact and benefits of switch block memories could be much more accurately measured.

6.2.2 Algorithmic Improvements

To facilitate the development of benchmark circuits, a synthesis and mapping tool that supports not only wide, shallow memories, but also high-speed I/O needs to be created. Similarly, the placement and routing tool should be modified to support highspeed I/O.

This would allow the circuits containing high-speed I/O to be evaluated by academia, and would simplify research into architectures that support these circuits.

In addition, to properly compare the timing of switch block memories, the development of an academic place and route tool -- most likely a modification to VPR -- that supports block memories is also desirable. Such a tool would allow us to compare the timing of a block memory architecture to the switch block memory architecture. The inclusion of wide, shallow block memories would also allow a comparison to be made between the switch block memory architecture and an embedded wide, shallow block memory architecture. Such a study would likely be of great interest to the FPGA research community.

6.2.3 Architectural Improvements

An evaluation of switch block memory architectures in which only certain switch blocks are capable of becoming memories is also needed. These architectures may offer significant benefits over an architecture where every switch block contains enhanced programmable connections. The area of such an architecture may be significantly reduced because less circuitry is needed, and the impact on timing may also be smaller because the slower switch blocks can be avoided by critical path nets using an intelligent routing algorithm. A measure of the impact for various configurations, along with the decrease in benefits to circuits that make use of wide, shallow memories would allow suitable patterns for the architecture to be found.

The high sensitivity of these architectures to variations in critical path timing, based solely on the random number seed used in the randomization of the CAD algorithms, suggests that all of these evaluations should be carried out using the statistical methods suggested in [52]. This robust style of evaluation is becoming an important part of FPGA research, and is especially important in this situation, as the wide range of critical paths in Chapter 5 clearly shows.

6.2.4 Long Term Research

Logic and memory resources that efficiently implement specific functionality commonly used in system-sized designs, such as multipliers or large memories, are already a part of modern FPGAs. As high-speed I/O becomes increasingly faster and FPGAs are expected to meet higher bandwidth requirements, the width of datapaths in certain classes of user circuits are likely to increase. To meet the needs of these circuits, logic and memory resources that efficiently implement wide datapaths will be necessary additions to future FPGA architectures.

With the addition of these resources, FPGAs will become more like SOCs, in both in capability and in form. This is especially likely in light of projections that future SOCs will include a programmable logic core [68]. Understanding the requirements of wide datapath user circuits, and the issues involved in the heterogeneous architectures that support these circuits, will be critical as FPGA capacities grow.

References

- [1] Xilinx, Inc., Datasheet: Virtex II Pro Platform FPGAs: Functional Description (DS083-2), January 2002.
- [2] Altera Corporation, Application Note: Using High-Speed Differential I/O in Stratix Devices, March 2002.
- [3] Altera Corporation, Datasheet: Stratix Programmable Logic Family Device, February 2002.
- [4] Xilinx, Inc., Datasheet: Virtex-E 1.8V Field Programmable Gate Arrays, September 2000.
- [5] Lattice Semiconductor, Datasheet: ORCA Series 4 Field-Programmable Gate Arrays, August 2000.
- [6] Altera, Datasheet: APEX 20K Programmable Logic Device Family, March 2000
- [7] S. Oldridge and S. Wilton, "A Novel FPGA Architecture Supporting Wide Shallow Memories," IEEE Custom Integrated Circuits Conference, May 2001, pp. 75-78.
- [8] V. Betz and J. Rose, "VPR: A New Packing, Placement, and Routing Tool for FPGA Research," Int. Workshop on Field-Programmable Logic and Applications, 1997, pp. 213-222.
- [9] Meta-Software, HSPICE User's Manual, February 1996.
- [10] Xilinx, Inc., The Programmable Logic Data Book, 1994.
- [11] AT&T Microelectronics, Data Sheet: Optimized Reconfigurable Cell Array ORCA Series Field Programmable Gate Arrays, March 1994.
- [12] Altera Corporation, Datasheet: Flex 10K Embedded Programmable Logic Family, July 1995.
- [13] Actel Corporation, Datasheet: 3200DX Field Programmable Gate Arrays, 1995.
- [14] G. G. Lemieux, P. Leventis, and D. Lewis, "Generating Highly-Routable Sparse Crossbars for PLDs," in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, February 2000, pp. 155-164.
- [15] G. G. Lemieux and D. Lewis, "Using Sparse Crossbars Within LUT Clusters," in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, February 2001, pp. 59-68.
- [16] M.I. Masud, "FPGA Routing Architectures: A Novel Switch Block and Depopulated Interconnect Matrix Architectures," M.A.Sc Thesis, University of British Columbia, December 1999.
- [17] V. Betz, "Architecture and CAD for Speed and Area Optimization of FPGAs," Ph.D. Thesis, University of Toronto, 1998.
- [18] Y. W. Chang, D. Wong, and C. Wong, "Universal Switch Modules for FPGA Design," in ACM Transactions on Design Automation of Electronic Systems, vol. 1, January 1996, pp. 80-101.
- [19] M.I. Masud, and S. J. E Wilton, "A New Switch Block for Segmented FPGAs," in Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications, Glasgow, UK, September 1999, pp. 274-281.
- [20] S. J. E. Wilton, "Architecture and Algorithms for Field Programmable Gate Arrays with Embedded Memory," PhD Thesis, University of Toronto, 1997.

- [21] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
- [22] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proceedings of the IEEE*, July 1993, pp. 1013-1029.
- [23] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, February 1999.
- [24] Actel Corporation, *Datasheet: eX Family FPGAs*, December 2001.
- [25] Actel Corporation, *Datasheet: SX Family FPGAs*, June 2001.
- [26] T. Ngai, "An SRAM-programmable field-reconfigurable memory," *Master's Thesis*, University of Toronto, 1994.
- [27] Xilinx, Inc., *Application Note: Virtex Series Configuration Architecture User Guide*, September 2000.
- [28] Xilinx, Inc., *Datasheet: Virtex Configuration Guide*, September 2000.
- [29] Altera, *Datasheet: Configuration Devices for SRAM-Based LUT Devices*, February 2002.
- [30] Xilinx, Inc., *Datasheet: XC6200 Development System*, August 1997.
- [31] A. Sangiovanni-Vincentelli, A. El Gamal, J. Rose, "Synthesis Methods for Field-Programmable Gate Arrays," *Proceedings of the IEEE*, July 1993, pp. 1057-1083.
- [32] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proceedings of the IEEE*, February 1990, pp. 264-300.
- [33] E. Sentovich et al, "SIS: A system for sequential circuit analysis," *Technical Report UCB/ERL M92/41*, University of California Berkeley, May 1992.
- [34] J. Cong, and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Based FPGA Designs," *IEEE Trans. Computer-aided Design*, January 1994, pp. 1-13.
- [35] J. Cong, S. Xu, "Technology Mapping for FPGAs with Embedded Memory Blocks", *Proc. ACM International Symposium on FPGA*, February 1998, pp. 179-188.
- [36] S. J. E. Wilton, "SMAP: Heterogeneous Technology Mapping for Area Reduction in FPGAs with Embedded Memory Arrays ", *Proc. ACM International Symposium on FPGA*, Monterey, CA, Feb. 1998, pp. 171-178.
- [37] E. Bozorgzadeh, S. Ogreni-Memik, M. Sarrafzadeh, "RPack: Routability-driven Packing for Clusterbased FPGAs, ". *Asia South Pacific Design Automation Conference*, Jan 2001.
- [38] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," *IEEE Trans. on VLSI*, Dec. 1995, pp. 473 - 482.
- [39] A. Marquardt, V. Betz, and J. Rose, "TimingDriven Placement for FPGAs", in *Proceedings of the International Symposium on FPGAs*, 2000, pp. 203-213.
- [40] A. Dunlop and B. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on CAD*, January 1985, pp. 169-173.
- [41] D. Huang and A. Kahng, "PartitioningBased Standard-Cell Global Placement with an Exact Objective," *ACM Symp. on Physical Design*, 1997, pp. 18-25.
- [42] S. Hauck and G. Borriello. "An Evaluation of Bipartitioning Techniques. " *Proc. of the 16th Conf. on Advanced Research in VLSI*, 1995, pp. 383-402.

- [43] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?", in Proc. ACM/IEEE Design Automation Conf., June 2000, pp. 477-482.
- [44] G. Sigl, K. Doll, and F. Johannes, "Analytical Placement: A Linear or a Quadratic Programming and Slicing Optimization," in Proc. ACM/IEEE Design Automation Conf., 1991, pp. 427-432.
- [45] B. Riess and G. Ettl, "Speed: Fast and Efficient Timing Driven Placement," IEEE Symp. on Circuits and Systems, 1995, pp. 377-380.
- [46] A. B. Kahng, "Futures for Partitioning in Physical design", Proc. IEEE/ACM International Symposium on Physical Design, April 1998, pp. 190-193.
- [47] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," Science, May 1983, pp. 671-680.
- [48] EBN Web Article, "Executive Comment: The leap to 0.13-micron poses risks", <http://www.ebnews.com/story/OEG20011126S0042>, November 2001.
- [49] C. Ebeling, L. McMurchie, S.A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," IEEE Trans. on VLSI, December 1995, pp. 473-482.
- [50] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in Proceedings of the International Symposium on FPGAs, February 1995, pp. 111-117.
- [51] S. Nag, R. Rutenbar, "Performance Driven Simultaneous Place and Route for Row-Based FPGAs," in Proc. ACM/IEEE Design Automation Conf., 1994, pp. 301-307.
- [52] A. Yan, R. Cheng, and S.J.E. Wilton, "On the Sensitivity of FPGA Architectural Conclusions to Experimental Assumptions, Tools and Techniques," in Proceedings of the International Symposium on FPGAs, February 2002, pp 147-156.
- [53] G. Lemieux, S. Brown, and D. Vranesic, "On TwoStep Routing for FPGAs," International Symposium on Physical Design, April 1997, pp. 60-66.
- [54] J.S. Rose, "Parallel Global Routing for Standard Cells," IEEE Trans. on CAD, October 1990, pp. 1085-1095.
- [55] Y. Chang, S. Thakur, K. Zhu and D. Wong, "A New Global Routing for FPGAs," International Conference on Computer Aided Design, 1994, pp. 356-361.
- [56] S. Brown, M. Khellah and G. Lemieux, "Segmented Routing for Speed-Performance and Routability in Field-Programmable Gate Arrays," Journal of VLSI Design, Vol. 4, No. 4, 1996, pp. 275-291.
- [57] C.Y. Lee, "An Algorithm for Path Connections and its Applications," IRE Trans. Electron. Comput., Vol. 10, 1961, pp. 346-365.
- [58] M. Alexander, J. Cohoon, J. Ganley and G. Robins, "Performance-Oriented Placement and Routing Based on Multi-Weighted Graphs," European Design Conference, 1994, pp. 259-264.
- [59] E. Moore, "Shortest Path Through a Maze," Proceedings of International Symposium on Switching Circuits, Harvard University Press, 1959, pp. 285-292.
- [60] E. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numer. Math. Vol. 1, 1959, pp 269-271.

- [61] W. Tsu, K. Macy, A. Joshi, et al., "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array," in Proceedings of the International Symposium on FPGAs, February 1999, pp.125-134.
- [62] S. Kang, Y. Leblebici, "CMOS Digital Integrated Circuits, Analysis and Design," McGraw-Hill, 1999.
- [63] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite", Proceedings International Symposium on Physical Design, April 1998, pp. 85-90.
- [64] H. Ahmadi, and W. Denzel, "A Survey of Modern High-Performance Switching Techniques," IEEE Journal on Selected Areas in Communications, September 1989, pp. 1091-1103.
- [65] H. Eggers, P. Lysaght, H. Dick and G. McGregor, "Fast Reconfigurable Crossbar Switching in FPGAs", Proc. 6th International Workshop on FieldProgrammable Logic and Applications, 1996, pp. 297-306.
- [66] Altera Corporation, Datasheet: MAX+PLUS II, Programmable Logic Development System & Software, January 1998.
- [67] EDIF Steering Committee, "EDIF Electronic Design Interchange Format Version 2.0," Electronic Industries Association, 1987.
- [68] S.J.E. Wilton, R. Saleh, "Programmable Logic IP Cores in SoC Design: Opportunities and Challenges", in the IEEE Custom Integrated Circuits Conference, San Diego, CA, May 2001, pp. 63-66.