1-1-2008

The effect of multi-bit correlation on the design of routing resources in field programmable gate arrays

Ping Chen
Ryerson University

TK
7895
.G36
C54
2008

THE EFFECT OF MULTI-BIT CORRELATION ON THE DESIGN OF ROUTING RESOURCES IN FIELD PROGRAMMABLE GATE ARRAYS

by

Ping Chen

Bachelor of Applied Science

Beijing University of Aeronautics and Astronautics, 1996

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2008

THE EFFECT OF MULTI-BIT CORRELATION ON THE DESIGN OF ROUTING RESOURCES OF FIELD PROGRAMMABLE GATE ARRAY

Master of Applied Science, 2009

Ping Chen

Electrical and Computer Engineering
Ryerson University

ABSTRACT

The large arithmetic-intensive applications increasingly implemented on field-programmable gate arrays (FPGAs) challenge FPGA architects to design FPGAs that can efficiently transport large amount of multi-bit wide signals in the data-path circuits of these applications. In this work, we investigate the area efficiency of two FPGA multi-bit aware routing architectures – the sparse and the enhanced sparse architectures, and compare them with the conventional and the configuration memory sharing architectures. We found that the sparse and enhanced sparse architectures are 6-10% more area efficient than the conventional architecture. Our data also show that while the configuration memory sharing architecture can achieve the highest level of theoretical area savings for multi-bit transportation, it performs poorly for circuits with 50% or less multi-bit signals. These results suggest that FPGA architects should look beyond conventional architectures in order to create more efficient routing architectures for modern FPGAs.

ACKNOWLEDGEMENTS

Many people have helped and supported me in this long and intense endeavor that took to get here.

I would like to first address my thanks to my family in China. I would like to thank my father, whom I will always give credit for every achievement in my life even though he could not even witness my graduation from high school. I would like to thank my mother, who always offers unconditional help and support to every decision I have made even when the decision was to bring me to the other side of the earth from her. I would like to thank my sister, who is different from me in many ways but has influence on me in all the ways, in a good way.

I would like to express my gratitude to the many helpful suggestions, endless patience and great support from my supervisor, Dr. Andy Ye, who has given me the best career opportunity I've received so far in Canada. He is a bit "difficult" when it comes to writing, a skill I often overlooked in my research, but now when I look back, I really appreciate his push and enjoy the progress I have made.

The two year study in Ryerson would have been much more stressful than it was without my classmates, who shared pressure and happiness with me in the hall way of the engineering building during class breaks or in the subway on our way back home. I would like to thank all my classmates, in particular, Sebastian and Theepan.

I would like to thank my husband, Mark Stoodley, for his company to me during those tiring weekends when I struggled for deadlines in the lab, for the warm meals he prepared for me after I came home after night classes in cold winter nights, for the knowledge he shares with me on programming and research, and for his patience to wait for me to find my direction in this new continent. Running into him is my best luck.

Finally, I would also like to thank my parents in law, whose house is the warmest in Toronto, and my brother in law's family, who are always a lot of fun to stay with. I am grateful and happy to be the person I am today and I would like to thank all the people who make it happen.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Field programmable gate arrays (FPGAs) are integrated circuits that can be programmed to implement virtually any digital circuit. FPGAs were developed from mask-programmable gate arrays (MPGAs). MPGAs, however, are programmed using integrated circuit fabrication to form metal interconnections. FPGAs, on the other hand, are programmed via electrically programmable switches. FPGAs are similar to the traditional programmable logic devices (PLDs) but can achieve much higher level of integration than PLDs [1].

Over the years, FPGAs have developed rapidly to programmable logic devices that can accommodate more than a half million logic elements and tens of millions of memory bits. These devices can be clocked at over 600MHz to perform high speed tasks [2][3] .

FPGAs are optimized for implementing digital hardware algorithms and have the added advantage of being able to change their functionalities in a fraction of second. Being both hardware-oriented and programmable, FPGAs provide a unique blend of performance and flexibility, which has proven essential in many applications. The programmability of FPGAs, however, has made their design particularly challenging. Typically, only 25% of FPGA area is actually used to perform computation while the remaining 75% is used for the *routing resources* which connect the computing elements together [1]. Due to their vast area consumption, the design of the routing resources is as important as the design of the computing elements.

As the logic capacity of FPGAs increases, more and more arithmetic-intensive applications are implemented on FPGAs. Consequently, there has been a corresponding increase in the variety of FPGA computing elements. From a mere collection of logic blocks, FPGAs now can include digital signal processors, multipliers, multi-bit addressable memory cells, and even processor cores [2][3]. One of the common characteristics of these new computing elements is their multi-bit design, where each element is designed to process several bits of data at a time.

While the input and output pins of a conventional logic block carry independent bits of information, the input and output pins of a multi-bit processing element are logically organized to represent multiple-bit wide data. In this organization, pins that represent a datum are often used at the same time. Similarly, routing resources are routinely used to transport multiple-bit wide data from a common source to a common destination.

To transport a multiple-bit wide datum (a *multi-bit signal*), one can either treat the datum as a set of independent signals (*single-bit signals*) and transport these signals individually through conventional routing resources, or view the entire datum as a single coherent unit and transport the unit collectively through a set of specialized routing resources.

Conventional routing tools treat signals independently and route them individually. In this thesis, we call routing architectures associated with the conventional routing tools as the *conventional architecture* and the routing tracks in these architectures the *single-bit routing tracks*. The conventional routing tools can be represented by the Versatile Place and Route (*VPR*) tool [4], a successful FPGA CAD tool widely used in academia. VPR assumes a routing architecture that distributes switches from logic block I/O pins to routing tracks as uniformly as possible such that each I/O pin has an equal chance to connect to any of the routing tracks. Consequently, a multi-bit signal is often broken into a sequence of single-bit signals – the correlation among these signals is ignored and the regularity of the multi-bit signal is destroyed.

To preserve the regularity of multi-bit signals, we can transport a multi-bit signal as a single unit on a set of specialized routing tracks. We call the specialized routing tracks *multi-bit routing tracks*. The routing architectures containing multi-bit tracks are called *multi-bit aware routing architectures*. The multi-bit tracks and logic block I/O pins are all grouped into buses. Within a bus, each routing track or a logic block I/O pin is assigned to a unique bit position. An I/O pin can only be connected to multi-bit tracks that are at the same bit position and vice versa. Compared to the routing tracks in the conventional architecture, a multi-bit track can connect to fewer logic block I/O pins. This reduced connectivity results in sparser switch patterns which consume less active area. At the same time, sparser switches also lower the flexibility of the routing tracks. As a result, more tracks might be needed to route a particular circuit. Therefore, the area efficiency of multi-bit aware routing architectures compared to conventional architectures depends on the tradeoff between the sparseness of the routing switches and the utilization of the routing tracks. Investigation into the sparser switch pattern design may lead to more area efficient FPGA routing architectures, but we are not aware of any previous work in this direction.

Even though the commercially available general purpose FPGAs have included data-path oriented features such as processors and multipliers, these features are mainly aimed at improving the performance of specific arithmetic functions through heterogeneous architectures. Recent research on commercial FPGAs center on improving the performance or lowering the power consumption of FPGAs through more advanced logic block and routing circuitry design [5][6][7]. The routing architectures in that research, however, are still based on the conventional routing architectures. We are not aware of any industrial research to investigate either multi-bit aware routing architectures or the design of automated CAD tools that can capture and maintain data-path regularity.

Many academic researchers have studied data-path oriented field programmable architectures, which are typically designed for arithmetic-intensive applications. These studies, however, either have not empirically reported on the area efficiency of the routing architectures or have focused on substantially different logic and routing architectures from those found in conventional FPGAs. The field programmable architectures investigated in previous research can be classified into 4 classes: 1) processor-based architectures; 2) static Arithmetic Logic Unit (ALU)-based architectures; 3) dynamic ALU-based architectures, and 4) Look Up Table (LUT)-based architectures. Examples of the processor-based architectures include PADDI-1[8], PADDI-2 [9], RAW machine [10], and REMARC [11] architectures. Instead of processors, the static ALU-based architectures are built on arrays of arithmetic logic units. This class of architectures can be represented by Colt [12], DreAM [13], and PipeRench [14] architectures. Unlike the static ALU-based architectures, which can be only configured by configuration memory bits, the dynamic ALU-based architectures can also be configured by the data from the computation process in the architectures. Several dynamic ALU-based architectures are the RaPiD [15], MATRIX [16], and Chess [17] architectures.

The above 3 classes of architectures are constructed from arrays of processors or ALUs, which are considerably more complex than the conventional LUT-based logic blocks and therefore have very different routing demands and routing resources. This difference limits the researches on these architectures from benefiting from the current work on FPGA routing architectures. The fourth class of the architectures – the LUT-based architectures such as Garp [18], the mixed-grain FPGA [19], and DP-FPGA [20][21] architectures – is the closest to conventional FPGA architectures. In the Garp architecture, however, the track segments remain unconnected to each other instead of being connected through routing switches as in the conventional architectures. This feature severely limits the possible applications of this architecture. The mixed-grain FPGA architecture study defined the basics of its routing architecture, which is closer to conventional architectures than the Garp architecture, but this work didn't measure the area efficiency by actually placing and routing benchmark circuits on the architecture. Like the mixed-grain FPGA architecture, the DP-FPGA architecture study did not empirically report on the area efficiency of its routing architecture. In fact, this work did not define a complete set of its building blocks, so there is not yet a CAD flow for the DP-FPGA architecture.

Based on the DP-FPGA architecture, the work in [22] developed the MB-FPGA architecture. This work carefully defined the detailed routing architecture for the MB-FPGA architecture, and has designed a full CAD flow for the architecture. This work also routed a set of benchmark circuits on the MB-FPGA architectures and found 8% routing area saving over the conventional routing architectures. The MB-FPGA routing architecture is multi-bit aware routing architectures and has multi-bit tracks that can be grouped together for multi-bit transportation. The routing architecture, however, has configuration memory sharing among the switches within a bus, so the effect of the switch pattern design was mixed together with the effect of configuration memory sharing. From the experimental results of this work, therefore, we can not decide how much of the area savings can be attributed to the sparser switch pattern design on the multi-bit tracks and how much is from sharing configuration memory.

1.1. Thesis Objectives

The motivation of this thesis is to investigate the sparser switch patterns independently. In this research, we first separate the sparser switch patterns from configuration memory sharing to generate a new routing architecture called the *sparse architecture*, and then we improve upon this routing architecture to produce the *enhanced sparse architecture*. The goal of this work is to look beyond the conventional architectures and search for more area efficient FPGA routing architectures that can be used to implement data-path oriented applications.

1.2. Thesis Contribution

This thesis evaluates two new FPGA multi-bit aware routing architectures: the sparse and the enhanced sparse architectures. The research on these two routing architectures greatly expands the FPGA routing architecture design space previously explored by the conventional architecture. Our thesis shows that the conventional architecture only explored a very small fraction of the total design space.

In the research, we routed a set of benchmark circuits on these two new routing architectures and measured their area efficiency, track counts, and performance. We found that the sparse and the enhanced sparse architectures are more area efficient than the conventional architecture with only a slight increase in track count, and are as area efficient as the configuration memory sharing architecture with significant decrease in track count. We also found that the performance of these two new architectures is as good as that of the conventional and the configuration memory sharing architectures.

1.3. Thesis Organization

The thesis is organized as follows. The general FPGA architecture and the two existing routing architectures – the conventional and the configuration memory sharing architectures will be described in Chapter 2. The two new routing architectures that this research targets – the sparse and the enhanced sparse architectures – are introduced in Chapter 3. To understand the tradeoffs between the routing density and the routing flexibility in these routing architectures, Chapter 4 then compares the routing resource density of all four routing architectures and estimates the routing flexibility required by the sparse and the enhanced sparse architectures to maintain their area efficiency. Before we experimentally evaluate the area efficiency of these two routing architectures, we select, in Chapter 5, the routing tools for the evaluation. Chapter 6 presents the results of our evaluation. Finally, in Chapter 7, we summarize the thesis and propose future work.

CHAPTER 2

BACKGROUND

In this chapter, we describe the background material for the remainder of the thesis. Section 2.1 introduces the general FPGA architecture, including the structure of logic blocks and the routing architecture. We will then review two earlier routing architectures in Sections 2.2 and 2.3, respectively: 1) the conventional architecture generated by the VPR tool, and 2) the configuration memory sharing architecture. The first routing architecture acts as a baseline in our research, while the second one is the building block of the new routing architectures described later in this thesis.

2.1. General FPGA Architecture

FPGA's programmability can be realized through various programming technologies such as: anti-fuses, erasable programmable read-only memories (EPROMs) and *static random access memories* (SRAMs). The FPGA logic blocks can be built on one or more of the followings: transistor pairs, basic gates, wide fan-in AND-OR structures, multiplexers and *Look Up Tables* (LUTs) [1]. SRAM based FPGAs whose logic blocks are built on LUTs are the main stream in the current FPGA market, and also the focus of our research. In this section, we will introduce the architecture of these FPGAs.

2.1.1. Overview of the General FPGA architecture

The general architecture of an FPGA is shown in Fig. 2.1. The *logic blocks* are the basic programmable computing elements in an FPGA. The input or output pins of the logic blocks are connected to the routing tracks through the *connection blocks* and the routing tracks are connected together through the *switch blocks*. The switch and the connection blocks are the two building blocks for the FPGA *routing architecture*. The I/O blocks exchange signals between the FPGA and external circuits. Since the remainder of the thesis does not deal with the I/O blocks, we do not describe them in detail here. In the remaining part of this section, we will give further details on the structures of the logic blocks and the routing architecture.

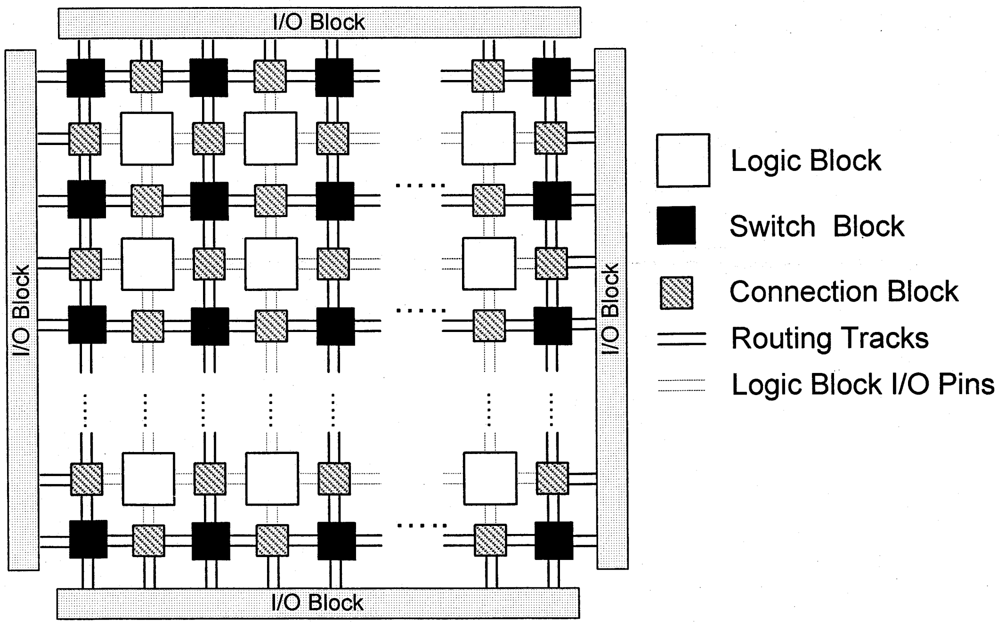
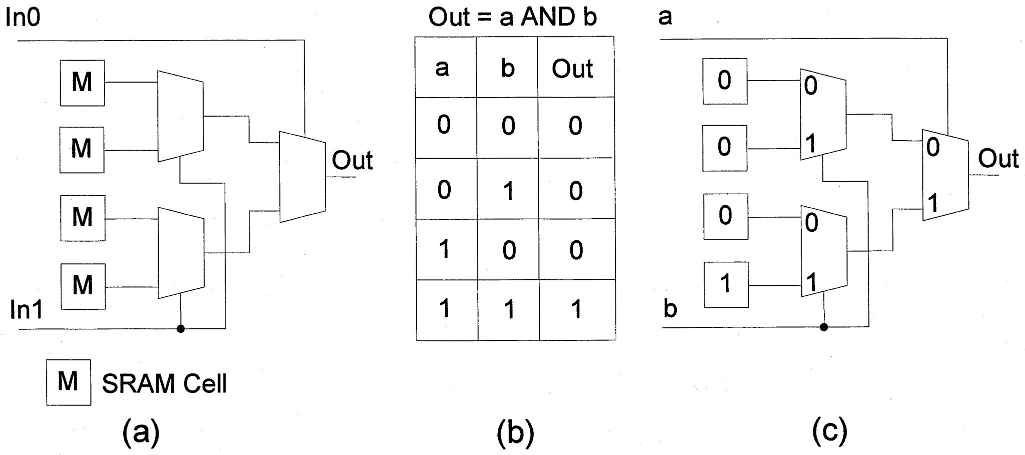


Fig. 2.1 Overview of the General FPGA Architecture

2.1.2. *Common Logic Block Structures.*

The logic blocks are based on Look Up Tables (LUT). A LUT, paired with a register forms a *Basic Logic Element* (BLE). Several BLEs are then connected together to become a cluster. Finally, one or several clusters form a logic block. In this section, we will introduce the structure of the logic block in details, starting from its basic building blocks – LUTs.



a	b	Out
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 2.2 2-Input LUT (2-LUT)

LUTs are combinational circuits consisting of K inputs, a set of multiplexers and SRAMs, through which LUTs can be programmed to realize any K input combinational logic functions. An example of a 2-input LUT (2-LUT) is shown in Fig. 2.2. The structure of the LUT is illustrated in Fig. 2.2 (a) while Fig. 2.2 (b) and (c) show how the LUT works as an AND gate. To map a 2-input AND gate to a 2-LUT, we need to store all possible outputs of the AND gate in the SRAM cells of the 2-LUT. These outputs are shown in the 3rd column of the truth table shown in Fig. 2.2 (b). Correspondingly, the binary values "0 0 0 1" are stored in the SRAM cells of the 2-LUT as shown in Fig. 2.2 (c). In Fig. 2.2 (c), the digit "0" or "1" on each input line of multiplexers denotes the value of the selection signal that is needed for the corresponding input line to be selected to be the multiplexer output. For example, if the selection signal *a* is 1 and the selection signal *b* is 0, the lower input to the multiplexer in the second stage and the upper input to the multiplexer in the first stage are selected. As a result, the stored content in the 3rd SRAM is delivered to the output of the 2-LUT. This output matches the output in the 4th row of the truth table shown in Fig. 2.2 (b). The remaining 3 cases in the truth table can be verified in the same way.

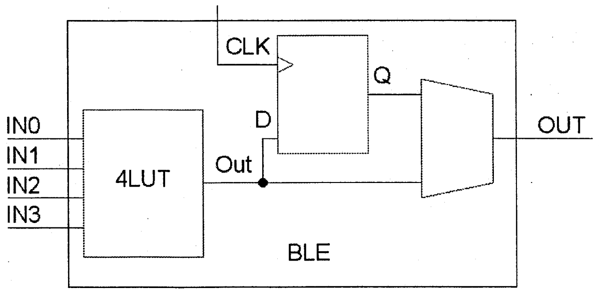


Fig. 2.3 A Basic Logic Element (BLE)

LUTs are the major programmable resources in FPGA logic blocks. A LUT is often paired with a sequential circuit to form a higher level logic unit. In a typical FPGA, a LUT and a register are wired together to form a Basic Logic Element (BLE) as shown in Fig. 2.3. In the figure, 4 BLE inputs are directly connected to a 4-input LUT and the output of the BLE can either be connected to the registered version of the 4-LUT output or be directly connected to the output of the 4-LUT.

Several BLEs are then connected by a set of local routing resources to form a cluster. Fig. 2.4 shows a cluster with 3 inputs and 2 outputs. This cluster has two 2-input BLEs whose inputs can be connected to any of the inputs or outputs of the cluster through a set of multiplexers controlled by SRAM cells. These inputs to the cluster are therefore *logically equivalent* and so are the outputs.

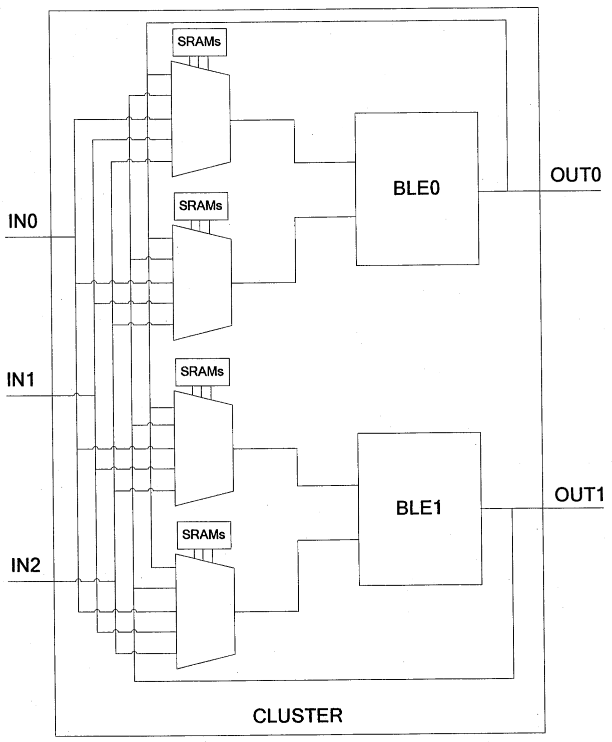


Fig. 2.4 A 3-Input 2-Output Cluster.

The conventional FPGAs assume that each cluster directly forms a logic block. The data-path oriented FPGAs, on the other hand, group a set of clusters into a logic block.

2.1.3. *Common Routing Architecture Structure*

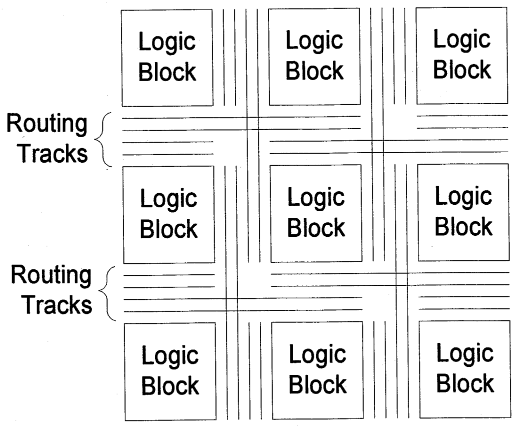
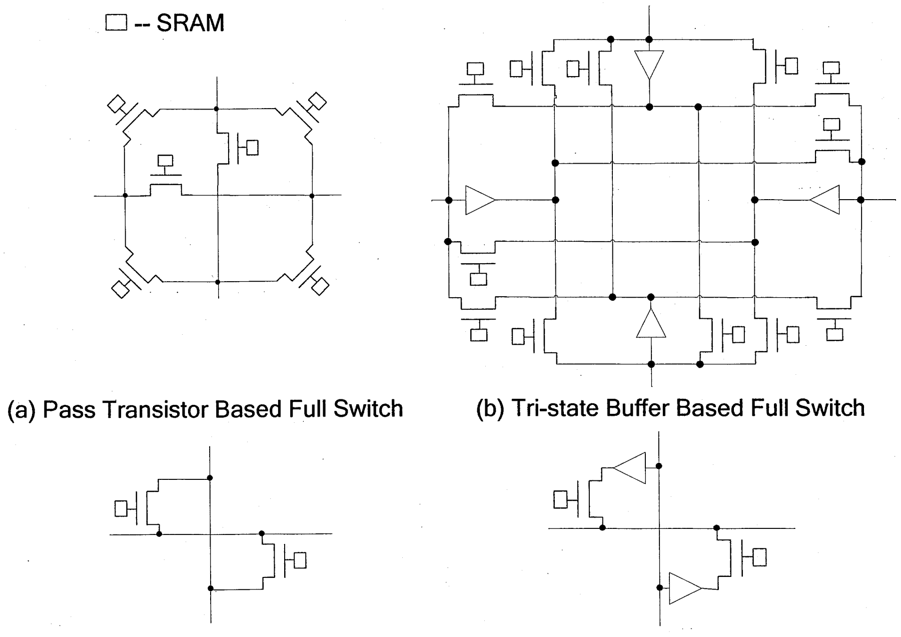


Fig. 2.5 Routing Segments of Length 2

The logic blocks are connected through programmable routing resources in an FPGA. FPGA routing resources include routing tracks and programmable routing switches, which are grouped into the switch and connection blocks.

The routing tracks are organized into *routing channels*, which run both horizontally and vertically between the logic blocks. A routing track is broken into *track segments*. A track segment with length N spans N logic blocks. Fig. 2.5 shows a simplified routing architecture with 2 horizontal and 2 vertical routing channels with the track segment length of 2. The track segments are connected together through switch blocks and to the logic block I/O pins through connection blocks. In the remainder of this section, we describe the structure of the switch blocks and the connection blocks respectively.



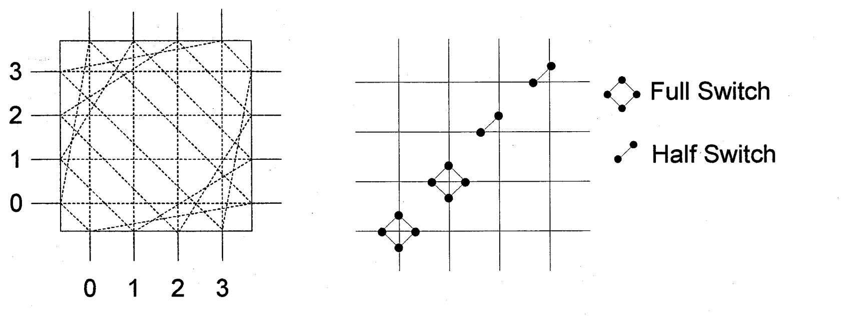
(a) Pass Transistor Based Full Switch (b) Tri-state Buffer Based Full Switch

(c) Pass Transistor Based Half Switch (d) Tri-state Buffer Based Half Switch

Fig. 2.6 Full Switches and Half Switches Employed in Switch Blocks

Fig. 2.6 shows two types of switches that are employed in a switch block – the full and the half switch. A full switch connects 2 horizontal routing track segments with 2 vertical routing track segments. The switch can be constructed out of pass transistors or tri-state buffers as shown in Fig. 2.6 (a) and (b). A half switch connects two perpendicular routing tracks that continue through a switch block. A pass transistor based half switch is illustrated in Fig. 2.6 (c) and a buffer based half switch is shown in Fig. 2.6 (d).

Using the full and half switches, a switch in a switch block connects an incoming track to a set of outgoing tracks. The topology of switch blocks decides the exact set of outgoing tracks that an incoming track can connect to. Among several common topologies used for switch blocks [23] [24] [25], we will describe the *disjoint topology* in detail since the topology was shown to be one of the most efficient and is used in many studies on the conventional architecture [4] [23] [26] and the configuration memory sharing architecture [27].



(a) The Disjoint Topology (b) A Switch Block with The Disjoint Topology

Fig. 2.7 Switch Block

Fig. 2.7 (a) shows the connections in a switch block that employs the disjoint topology. In this figure, a dashed line represents a connection between two track segments. An incoming track in the switch block is always connected to 3 outgoing tracks of the same index. For example, as shown in Fig. 2.7 (a), the horizontal track at index 1 entering the switch block from the left can connect to 1) the vertical track at index 1 leaving the switch block from the top, 2) the horizontal track at index 1 leaving the switch block from the right or 3) the vertical track at index 1 leaving the switch block from the bottom. Fig. 2.7 (b) shows actual construction of a switch block with the disjoint topology. The switch block connects two perpendicular channels that contain 4 tracks each. Two of these 4 tracks continue through the switch block while the other two end. To connect the ending tracks, we need two full switches shown in Fig. 2.6 (a) or (b). To connect the continuing tracks, two half switches shown in Fig. 2.6 (c) or (d) are used instead.

The other types of building blocks for the FPGA routing architecture are the connection blocks, which are used to connect either the routing tracks to the logic input pins (the *input connection blocks*), or the logic output pins to the routing tracks (the *output connection blocks*). The input connection blocks are built from multiplexers while output connection blocks are built from tri-state buffers. Fig. 2.8 shows a simple example of an input connection block and an output connection block. In the input connection block, the SRAMs connected to the select inputs of the multiplexers select the routing track to be connected to a logic block input. In the output connection block, the SRAM cell on the base of a pass transistor turns on the pass transistor to connect a logic block output to a routing track.

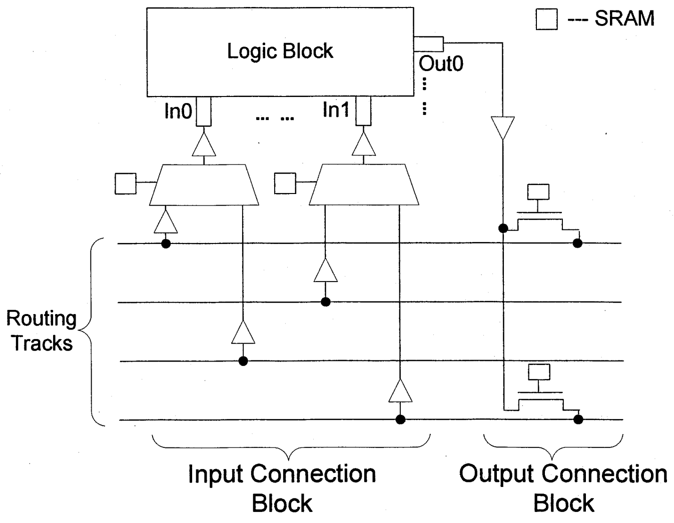


Fig. 2.8 Connection Block.

More details about the connection and the switch block design will be described in the next two sections where we introduce the two routing architectures studied in previous research: the conventional and the configuration memory sharing architectures.

2.2. Conventional Architectures

In the section, the conventional architecture generated by VPR is introduced. We will first describe the structure of the routing architecture, and then explain how the architecture is modeled in VPR.

2.2.1. Structure of the Conventional Architecture

Similar to the general routing architecture introduced in Section 2.1.3, the building blocks of the conventional routing architecture are the switch blocks and the connection blocks. The switch blocks utilized by VPR are tri-state buffer based and employ the disjoint topology. The design of the switch blocks are described in Section 2.1.3.

The connection blocks are more complex than the switch blocks. The switch patterns in the connection blocks are generated by the algorithm shown in Fig. 2.9. In this figure, F_c is the fraction of routing tracks per channel that can be connected to a logic block input or output (I/O) pin, P is the number of logic block I/O pins, and W is the number of routing tracks per channel. The algorithm generates a switch pattern matrix containing the index of a routing track that the jth connection of the ith logic block I/O pin connects to.

```
IF logic block I/O pins are output pins THEN
        C = ceil (Fc * W)
ENDIF
IF logic block I/O pins are input pins THEN
        C = floor (Fc * W)
ENDIF
step = W / P / C
increment = W / C
FOR (i = 0; i < P; i++)
        FOR (j = 0; j < C; j++)
                switch_pattern[i][j] = floor(step * i + increment * j)
        ENDFOR
ENDFOR
```

Fig. 2.9. Algorithm to Generate Switch Patterns in Connection Blocks

Fig. 2.10 shows 4 examples of the connection blocks that can be generated by the algorithm with varying F_c values. In the figure, a dot represents a switch in the connection blocks. A vertical line is a logic block I/O pin and a horizontal line is a routing track. There are a total of 4 I/O pins and 8 routing tracks in each of the switch patterns shown in the figure. When F_c is 0.25 as shown in Fig. 2.10 (a), two of the 8 tracks are connected to each logic block I/O pin via routing switches. When F_c is 1.0 as shown in Fig. 2.10 (d), all the tracks are connected to each logic block I/O pin.

F_c, and P are described in the *architecture file* [28] and W can be either specified by a command line parameter input to VPR or decided by the routing process in VPR according to routing requirements. VPR generates the specified FPGA architecture according to the command line parameters and the architecture file.

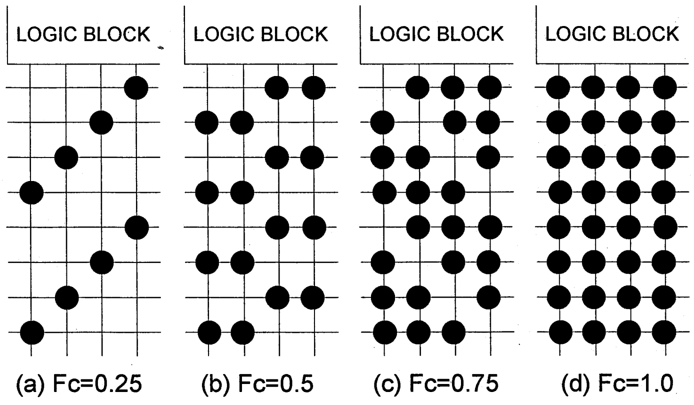


Fig. 2.10 Connection Blocks in The Conventional architectures

2.2.2. Routing Resource Modeling in VPR

Routing resources in the conventional architecture include *sources*, *sinks*, input or output pins of logic blocks, routing tracks and routing switches. All output pins from a logic block are logically equivalent and originate from a common source. Similarly, all the input pins to a logic block are also logically equivalent and terminate to a common sink.

Routing resources in VPR are modeled as a directed graph called the *routing resource graph*, or *rr-graph* [4]. In an rr-graph, the *nodes* are used to represent sources, sinks, logic block I/O pins and routing tracks. Routing switches that connect logic block I/O pins to routing tracks or routing tracks together are modeled as *edges* in the rr-graph.

To generate an rr-graph, VPR first generates a switch pattern using the algorithm shown in Fig. 2.9. VPR then converts the pattern along with the assigned switch block topology into connections between the nodes of the graph. During the conversion, the specific circuit level details such as the buffer or pass-transistor based switches are also modeled according to the architecture file.

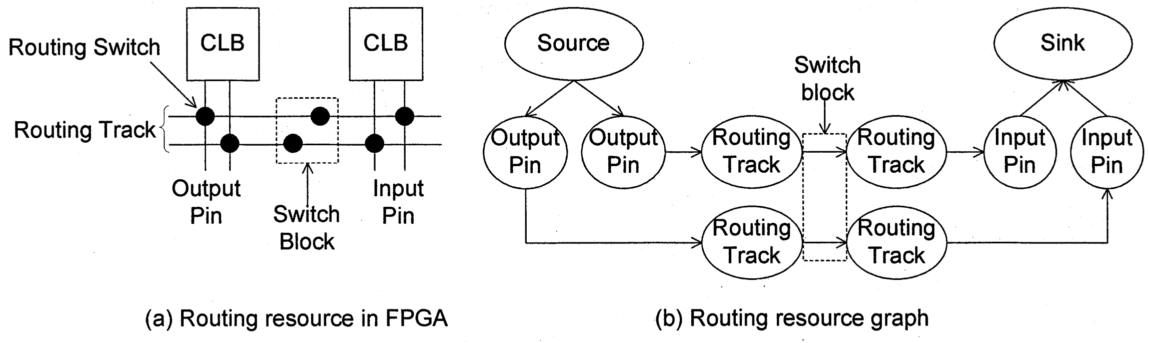


Fig. 2.11 Routing Resource Graph in VPR

Fig. 2.11 shows an example of the routing resource graph. The routing resources shown in Fig. 2.11 (a) are represented by the routing resource graph shown in Fig. 2.11 (b). In this figure, all the routing resources except routing switches are modeled as nodes that are represented by white circles, and the connection between two nodes is modeled as a directed edge.

2.3. Configuration Memory Sharing Architecture

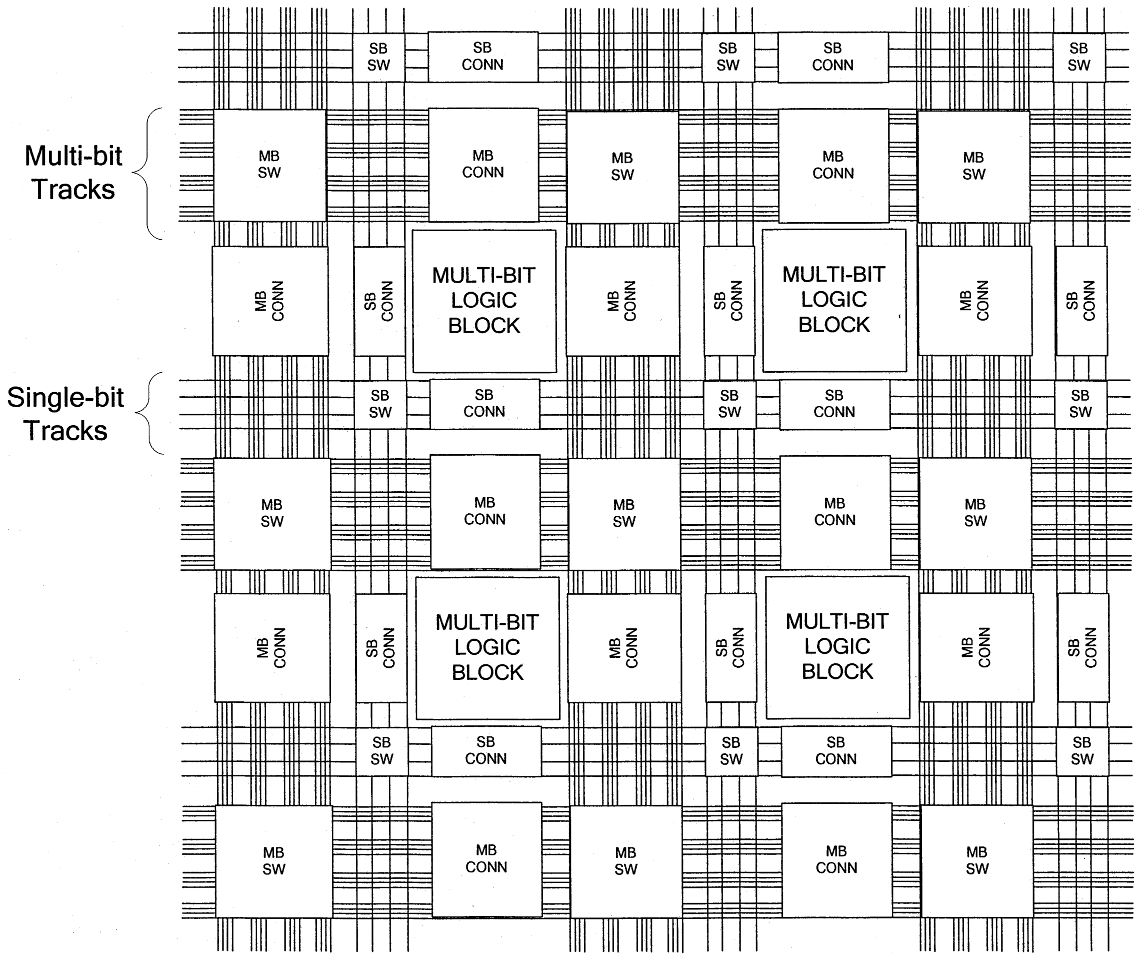


Fig. 2.12 FPGA with Configuration Memory Sharing Architecture

Another routing architecture we will describe in details is the configuration memory sharing architecture. This architecture is one of the multi-bit aware routing architectures, which are designed to take advantage of the large amount of multi-bit wide signals existing in data-path oriented applications. Most of the earlier studies on data-path oriented FPGAs have focused on the design of logic blocks [20][21]. The work in [22][27]

[29] investigated the design of the multi-bit aware routing architectures. This work, however, focused primarily on the configuration memory sharing aspect of the multi-bit aware routing architecture designs. In this section, we briefly review the configuration memory sharing architecture studied in his work because it forms the basis of this thesis.

2.3.1. FPGA with Configuration Memory Sharing Architecture

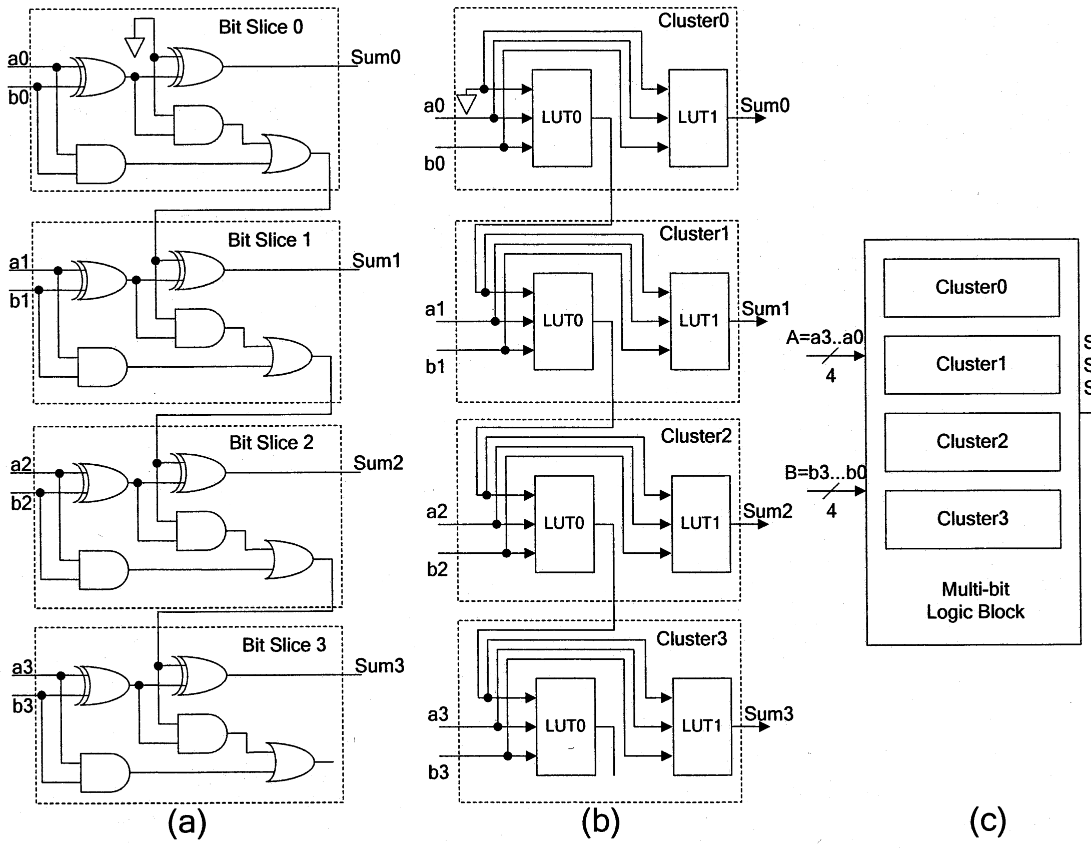


Fig. 2.13 Map a 4 bit Wide Adder to a Multi-bit Logic Block

Fig. 2.12 shows an FPGA with the configuration memory sharing architecture. The FPGA is constructed out of *multi-bit* logic blocks. These blocks are connected by two types of routing tracks – the single-bit and the multi-bit tracks. A set of single-bit tracks can be connected either to another set of single-bit tracks by a *single-bit switch block* (SBSW) or to a set of logic block I/O pins by a *single-bit connection block* (SBCONN). One major new feature of this routing architecture is the addition of multi-bit routing tracks specialized for transporting multi-bit signals. The multi-bit routing tracks are grouped into *routing buses*. A routing bus can be connected either to another routing bus by a *multi-bit switch block* (MBSW) or to a set of logic block I/O pins by a *multi-bit* *connection block* (MBCONN).

In the remainder of the section, we will describe each of the building blocks – the multi-bit logic blocks, the switch blocks and the connection blocks. The switch blocks include both multi-bit and single-bit ones and so do the connection blocks.

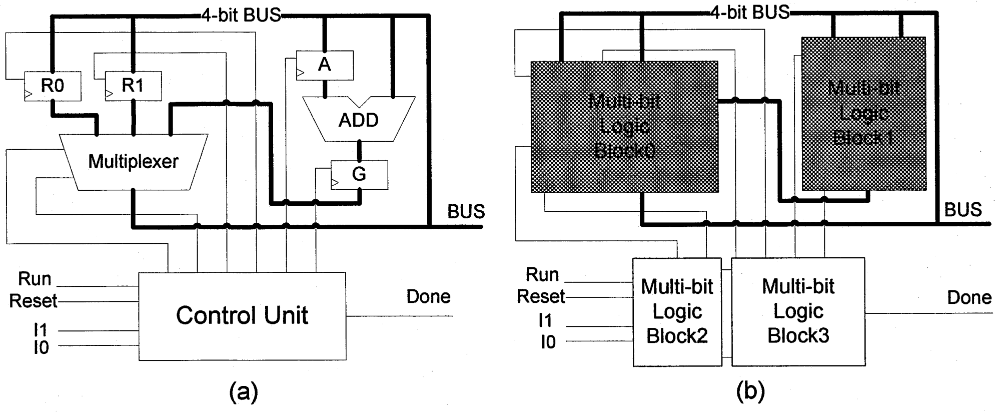


Fig. 2.14 A Simple 4 bit Wide Processor

Each multi-bit logic block contains *M* identical clusters and is designed to handle an *M*-bit computation. Fig. 2.13 shows an example of 4-bit adder mapped to a 4-bit wide multi-bit logic block. Fig. 2.13 (a) shows each individual bit slice of the 4-bit adder. Each of these slices can be implemented in a cluster as shown in Fig. 2.13 (b). Four of the clusters are grouped into a 4-bit wide multi-bit logic block that can perform a 4-bit addition as shown in Fig. 2.13 (c). In the 4-bit wide multi-bit logic block, every 4 input or output pins are grouped into an *input bus* or an *output bus*. In the example shown in Fig. 2.13 (c), there are two input buses used to input two 4-bit wide datum, A and B, to the adder and an output bus to output a 4-bit wide result, SUM, from the adder.

The multi-bit logic blocks are connected through both of the single-bit and multi-bit tracks. The multi-bit tracks are specialized for transporting multi-bit signals in data-path applications while single-bit tracks are specialized for single-bit signals. As an example, Fig. 2.14 shows a simple 4-bit wide processor. The data-path of the processor is shown in Fig. 2.14 (a). It contains an adder, a multiplexer, and several 4-bit wide buses. Besides the data-path, the processor also includes a control circuit. The control circuit sends out single-bit control signals to the data-path based on external commands. To map the processor onto an FPGA, we first mapped the processor into a set of multi-bit logic blocks as shown in Fig. 2.14 (b). These multi-bit logic blocks are then mapped onto a

4-bit wide FPGA with the configuration memory sharing architecture as shown in Fig. 2.15. In the example, the single-bit signals are routed on single-bit tracks while the multi-bit signals are routed on multi-bit tracks that are grouped into 4-bit wide buses.

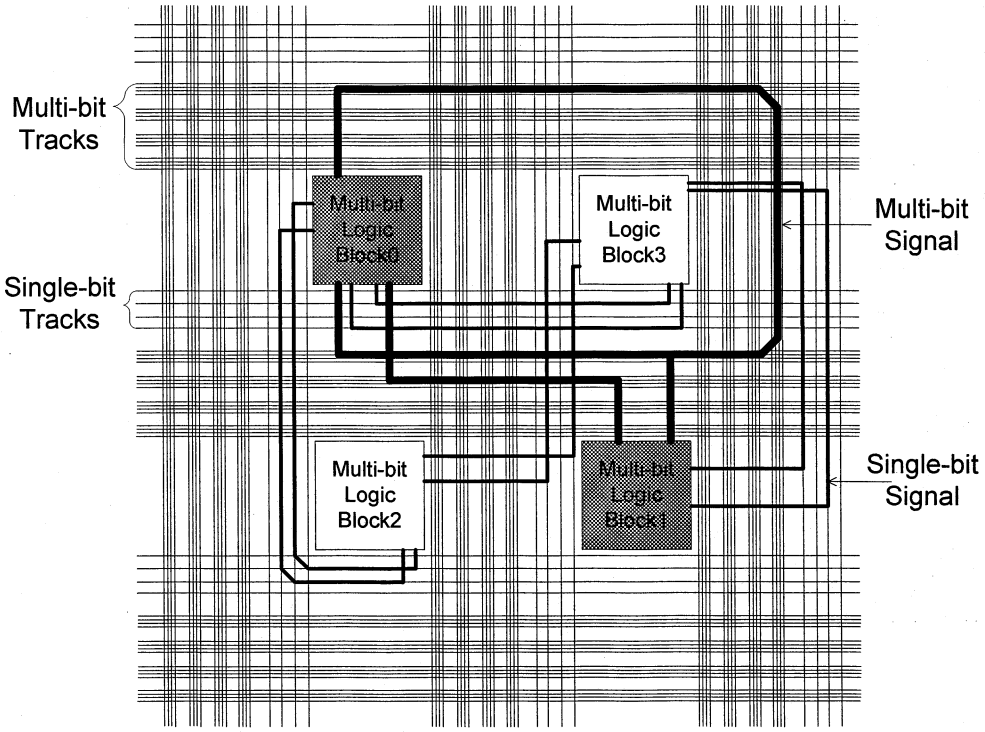


Fig. 2.15 A Processor Mapped Onto an FPGA with Multi-bit aware routing architecture

The multi-bit and single-bit tracks are connected together via the multi-bit and the single-bit switch blocks respectively. The single-bit switch blocks employ the same switch block design as described in section 2.2. The multi-bit switch blocks, on the other hand, is constructed out of a set of full and half switches that share configuration memory. A 4-bit wide switch block with the disjoint topology is shown in Fig. 2.16. Unlike the conventional switch blocks, however, switches in the multi-bit switch blocks are grouped into 4-bit wide groups and each group shares a single set of configuration memory. Note that without configuration memory sharing, each full switch would require 12 bits of configuration memory and each half switch would require 2 bits of configuration memory. With configuration memory sharing, 12 bits of configuration memory control the 4 full switches in this example. Only 2 bits of configuration memory control the 4 half-switches.

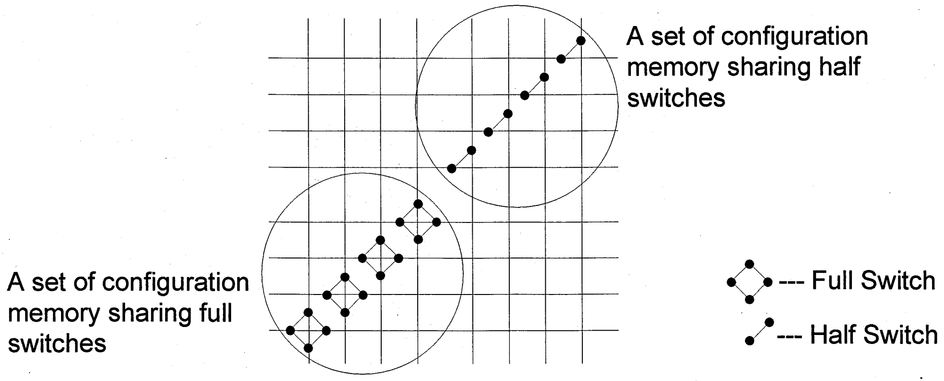

A set of configuration memory sharing half switches
A set of configuration memory sharing full switches
--- Full Switch
--- Half Switch

Fig. 2.16 Multi-bit Switch Block in the Configuration Memory Sharing Architecture

Similar to the switch blocks, the multi-bit aware routing architecture utilize two types of connection blocks – the single-bit and the multi-bit connection blocks. The single-bit connection blocks employ the same design methodology as described in section 2.2. The multi-bit connection blocks, on the other hand, is generated by operating the algorithm shown in Fig. 2.9 on buses instead of individual bits. In this case, F_c is the percentage of routing buses per channel that can be connected to a logic block input/output bus, P is the number logic block input/output buses, and W is the number of routing buses per channel.

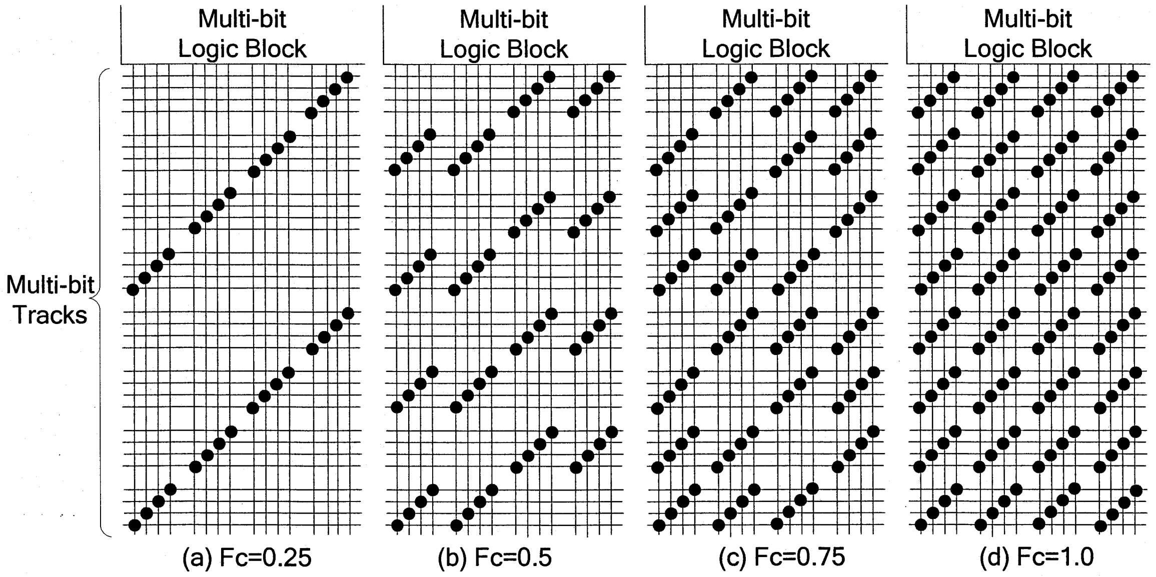


Fig. 2.17 Multi-bit Connection Blocks

Fig. 2.17 shows 4 examples of multi-bit connection blocks generated by the algorithm described above with varying F_c value. The bus width of these connection blocks is 4. In the figure, a black dot represents a routing switch. The vertical lines are I/O pins of multi-bit logic blocks and the horizontal lines are multi-bit tracks. Every 4 input or output pins are grouped into an input or output bus and every 4 routing tracks are grouped into a routing bus. When F_c is 0.25 as shown in Fig. 2.17 (a), each I/O bus connects to 2 of the 8 routing buses through two sets of switches. When F_c is 1.0 as shown in Fig. 2.17 (d), every I/O bus can connect to any of the routing buses.

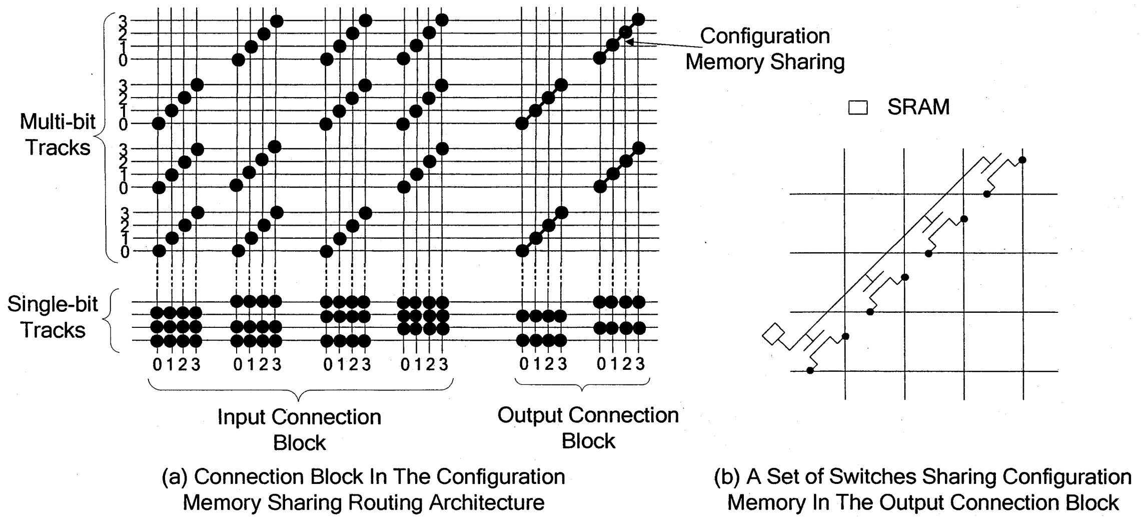

Configuration Memory Sharing
Multi-bit Tracks
Single-bit Tracks
3 2 1 0
0123
Input Connection Block
Output Connection Block
(a) Connection Block In The Configuration Memory Sharing Routing Architecture
□ SRAM
(b) A Set of Switches Sharing Configuration Memory In The Output Connection Block

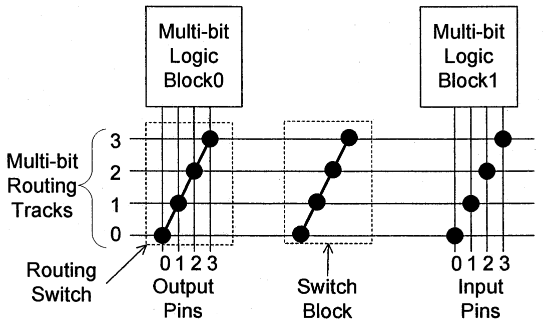
Fig. 2.18 Connection Block in the Configuration Memory Sharing Architecture

Fig. 2.18 (a) shows a complete connection block in the configuration memory sharing architecture. This connection block contains the multi-bit input and output connection blocks as well as the single-bit input and output connection blocks. Each set of switches in the multi-bit output connection block share configuration memory as shown in Fig. 2.18 (b). Each switch in the multi-bit input connection block, on the other hand, is independently controlled by its own configuration memory cells because this switch is an input to a multiplexer which also takes inputs from single-bit tracks to a logic block input pin.

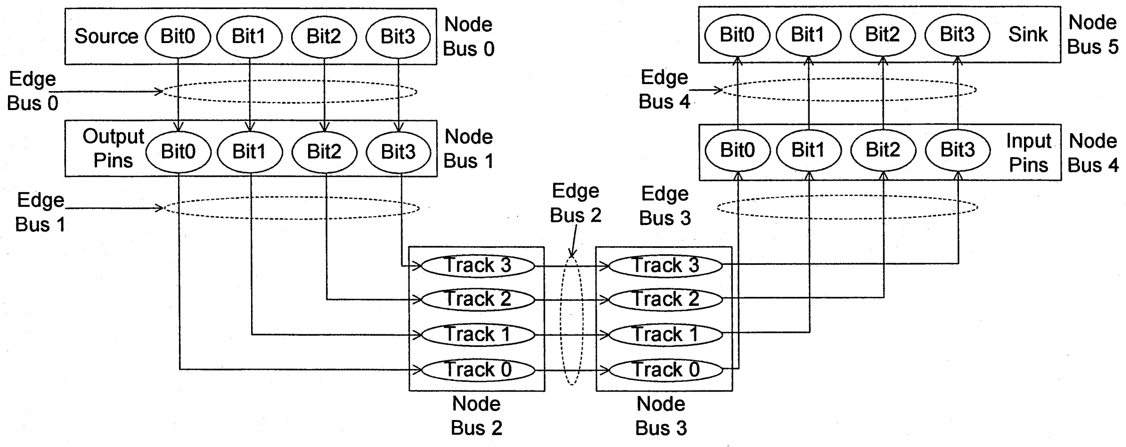
Fig. 2.18 (a) also shows an important feature of the connection block in the configuration memory sharing architecture: the switches in the multi-bit connection block are much sparser than those in the single-bit connection block. One of the goals of this research is to find out how much of the area efficiency of the configuration memory sharing architecture is attributed to the sparser switch pattern in the multi-bit connection block.

2.3.2. *Routing Resource Modeling in CAD Tool*

The routing resources in the configuration memory sharing architecture are modeled by the *multi-bit aware place and route tool* [22][27]. The tool models the routing resources with an rr-graph as well as several auxiliary data structures we describe later in this section.



(a) Routing Resource in the Configuration Memory Sharing Architecture



(b) Routing Resource Graph

Fig. 2.19 Routing Resource Graph for the Configuration Memory Sharing Architecture

The rr-graph used in the multi-bit aware place and route tool can represent multi-bit as well as single-bit routing resources. Fig. 2.19 (a) shows a set of 4-bit wide multi-bit routing resources. In this figure, a logic block output bus is connected to a logic block input bus through routing buses and 3 sets of routing switches. These routing resources are modeled as an rr-graph by the multi-bit aware place and route tool as shown in Fig. 2.19 (b). Unlike the conventional architecture, each node representing a source, sink, logic block I/O pin, or multi-bit track in the rr-graph for the configuration memory sharing architecture belongs to a bus and has a bit position in that bus. These buses are marked as *node buses* in Fig. 2.19 (b). Similarly, each edge representing a switch that connects to sources, sinks or multi-bit tracks also belongs to a bus and has a bit position in that edge bus. These buses are marked as *edge buses* in Fig. 2.19 (b).

The multi-bit aware place and route tool employs two sets of indices to record all the buses in the rr-graph corresponding to: 1) the node buses, and 2) the edge buses. Each node bus is assigned an index, numbered from zero, and the individual nodes comprising that bus are assigned bit positions in the bus. Similarly, each edge bus is assigned an index, also numbered from zero, and the individual edges comprising an edge bus are assigned bit positions in the bus. These indices are stored in the data structures representing the nodes and edges in the rr-graph.

Configuration memory sharing is not represented explicitly in the rr-graph. Instead, an array of integers is used to store whether each numbered edge bus index in the rr-graph uses configuration memory sharing. We call this array the *configuration memory sharing array*. When an integer in this array is one, the corresponding edge bus has configuration memory sharing; otherwise, there is no configuration memory sharing in the edge bus.

CHAPTER 3

The SPARSE AND ENHANCED SPARSE ARCHITECTURES

In this chapter, we introduce two multi-bit aware routing architectures that we investigate in this research: the sparse and enhanced sparse architectures. We developed the sparse architecture from the configuration memory sharing routing architecture and we enhanced the flexibility of the sparse architecture to create the enhanced sparse architecture. We describe the sparse architecture in Section 3.1 and the enhanced sparse architecture in Section 3.2.

3.1. The Sparse Architecture

Compared to the conventional architectures, the configuration memory sharing routing architecture not only requires fewer configuration memory bits, but also employs sparser switch patterns in the multi-bit connection blocks. To isolate the effects of the sparser switch pattern in the multi-bit connection block from the effects of fewer configuration memory bits, we introduce the sparse architecture, which has the same arrangement of switches as the configuration memory sharing routing architecture but allows the switches to be individually controlled. Fig. 3.1 and Fig. 3.2 show the switch block and the connection block, respectively, in the sparse architecture.

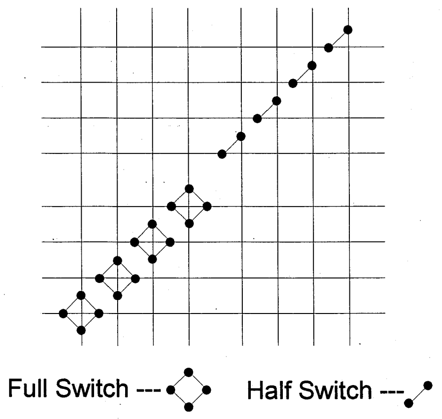
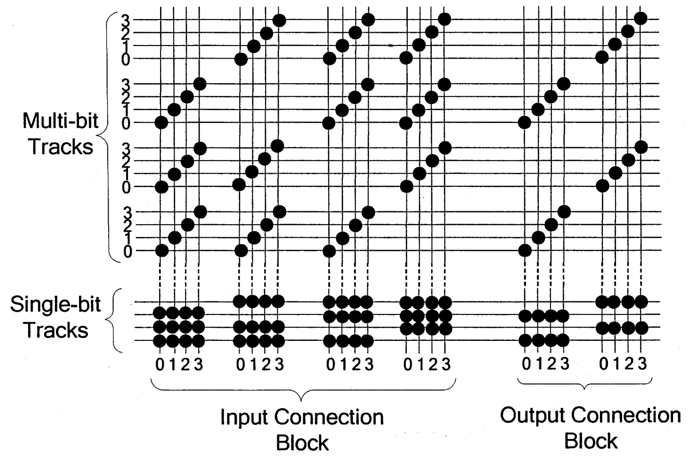
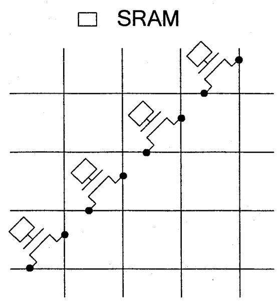


Fig. 3.1 Switch block on multi-bit tracks in the sparse architectures



(a) Connection Block In The Sparse Routing Architecture



(b) A Set of Switches In The Output Connection Block

Fig. 3.2 Connection Block in the Sparse Architecture

We call the sparse architecture's switch pattern, which connects two buses using a set of switches in a diagonal arrangement, the *sparse switch pattern.* Similar to the switch patterns in the configuration memory sharing architecture, each switch in the sparse switch pattern only connects logic block I/O pins to the routing tracks at the same bit position. But unlike the configuration memory sharing routing architecture, each bit in a bus in the sparse architecture can be connected to the corresponding bit in another bus without affecting the connections in the other bits in the same bus.

We modeled the sparse architecture with the multi-bit aware place and route tool. To represent the sparse architecture in the tool, we need to use the rr-graph and the configuration memory sharing array. The rr-graph for the sparse architecture is the same with that for the configuration memory sharing. The members in the configuration memory sharing array, however, are not set to constants as in the original multi-bit aware place and route tool.

```
#Multi-bit Architecture

  ⋮

CMSSw  0
CMSConnIn  0
CMSConnOut  0

  ⋮
```

(a) Sparse

```
#Multi-bit Architecture

  ⋮

CMSSw  1
CMSConnIn  0
CMSConnOut  1

  ⋮
```

(b) Configuration Memory Sharing

Fig. 3.3 Configuration Memory Sharing Setting in the Architecture File

We set the values of the configuration memory sharing array according to 3 variables representing whether there is configuration memory sharing in the switch blocks, the input connection blocks, and the output connection blocks. These 3 variables pick up their values from the architecture file, which can now include 3 new keywords, as shown in Fig. 3.3: *CMSSw*, *CMSConnIn,* and *CMSConnOut*. The parameter following the keyword *CMSSw* decides whether there is configuration memory sharing in the switch blocks. Value 1 or 0 for this parameter sets or clears, respectively, the configuration memory sharing in the switch blocks. The other two keywords work in the same way: *CMSConnIn* for configuration memory sharing in the input connection blocks and *CMSConnOut* for configuration memory sharing in the output connection blocks.

With the multi-bit aware place and route tool modified to handle these 3 new keywords in the architecture file, we can then model either the sparse architecture with the settings shown in Fig. 3.3 (a), or the configuration memory sharing routing architecture with the settings shown in Fig. 3.3 (b). These new keywords simplify exploring the multi-bit aware routing architectures.

3.2. The Enhanced Sparse Architecture

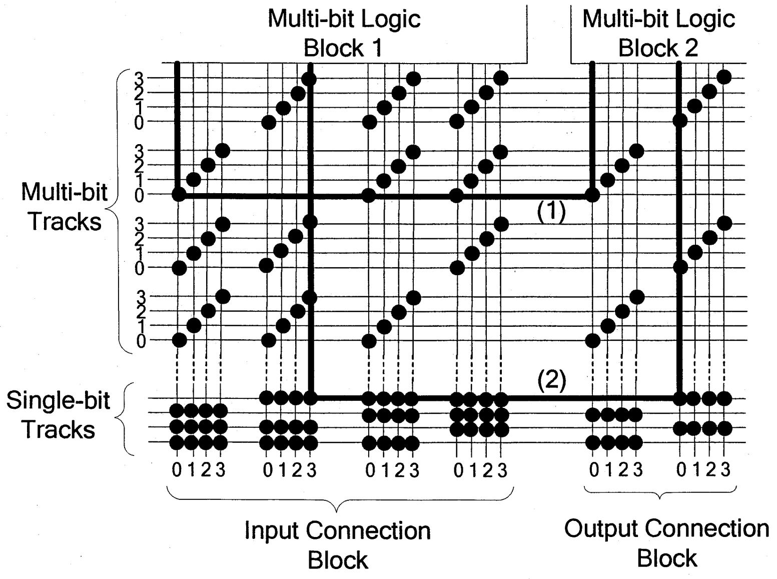


Fig. 3.4 Signals Routed in the Connection Blocks in the Sparse Architecture

In the sparse architecture, only signals from logic block output pins to the logic block input pins at the same bit position can be routed through the multi-bit tracks. For example, Fig. 3.4 shows an input and an output connection block in the sparse architecture. The thick lines in the figure represent signals routed through these connection blocks. As shown, the logic block I/O pins at bit position 0 can be only connected to the multi-bit tracks at bit position 0, so signal 1 from bit 0 to bit 0 can be routed on the multi-bit tracks. Signal 2, on the other hand, cannot be routed on the multi-bit tracks because this signal connects bit 0 to bit 3. In this case, single-bit routing tracks are needed even though there are empty multi-bit tracks.

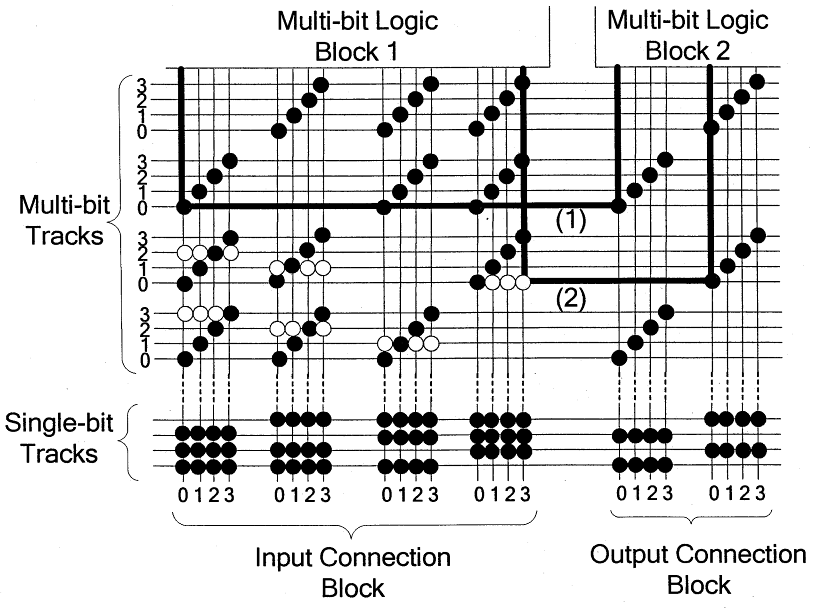


Fig. 3.5 Signals Routed in the Connection Blocks in the Sparse Architecture with Extra Routing Switches

If we sparsely add switches to the multi-bit connection blocks to connect logic block I/O pins to routing tracks at different bit positions, as shown in Fig. 3.5, signal 2 can be routed on the multi-bit tracks. In this figure, a white dot represents one of these *added switches* on multi-bit tracks. With these added switches, the multi-bit connection blocks become more flexible. We call the sparse architecture with added switches on multi-bit tracks the *enhanced sparse architecture*.

In this research, we selectively investigated several switch patterns to enhance the sparse switch pattern. We call these switch patterns *enhanced sparse switch patterns*, as shown in Fig. 3.6. To simplify the description, we use a bus width of 4, but the enhanced sparse switch patterns can be easily derived for other bus widths. In this figure, a set of black dots represents the sparse switch pattern. We call the set of switch patterns shown in Fig. 3.6 (a) formed by the white dots, indicating the added switches, *shift patterns*. In the set, there are 3 shift patterns, indexed from 1 to 3. The added switch patterns shown in Fig. 3.6 (b) are called *vertical control* patterns and the ones in Fig. 3.6 (c) are called *control* patterns. Each set of vertical control patterns or control patterns contains 4 switch patterns indexed from 1 to 4. The sparse switch patterns with the shift patterns are called the *shift enhanced sparse* switch patterns. Similarly, we also define the *vertical control enhanced sparse* switch patterns and the *control enhanced sparse* switch patterns.

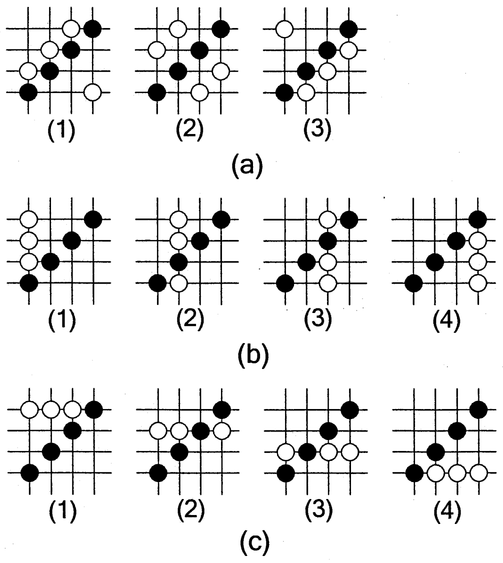


Fig. 3.6 The Enhanced Sparse Switch Patterns Investigated in the Research

Note that we do not add extra switches to every sparse switch pattern on multi-bit tracks. Instead, we introduce a new architectural parameter F_{c_enh}, which is the fraction of routing buses that have enhanced switch patterns, so that the added density on the multi-bit tracks can be controlled. It is crucial to be able to balance the routing flexibility offered by the extra switches against the additional area those switches require. In Section 4.2, we investigate just where this balancing point lies.

To generate the enhanced sparse switch patterns, we first generate the sparse switch pattern as introduced in section 3.1. Then to the sparse switch pattern we add extra switches. We did not incorporate the process of generating the enhanced sparse switch patterns in the multi-bit aware place and route tool. Instead, we built an external program which can output the description of the enhanced sparse switch patterns to the architecture file. We then modified the multi-bit aware place and route tool to accept a description for any arbitrary switch pattern in the architecture file.

Fig. 3.7 shows the algorithm which adds the control patterns shown in Fig. 3.6 (c) to the sparse switch pattern to generate the control enhanced sparse switch patterns in the external program. As shown, we only show the algorithm for bus width 4, but the algorithm can be easily adapted to any bus width.

```
NumEnhancedBus=floor(NumMultiBitTracks × Fc_Enh / BusWidth)
ControlPatternIndex= 1
FOR each side of a multi-bit logic block
    SET the current routing bus to be the NumEnhancedBus from the last bus in the
    multi-bit tracks
    FOR each routing bus in the enhanced routing buses
        FOR each input bus on the current side of a multi-bit logic block
            IF there is a sparse switch pattern on the current intersection between the
            routing bus and the input bus
                Put a control pattern at ControlPatternIndex
                Increase ControlPatternIndex by 1
                IF ControlPatternIndex>4
                    ControlPosition=1
                ENDIF
            ENDIF
        ENDFOR
    ENDFOR
ENDFOR
```

Fig. 3.7. The Algorithm Adding the Control Patterns to the Sparse Switch Pattern

In Fig. 3.7, *NumEnhancedBus* represents the number of routing buses that have enhanced switch patterns. The number of routing buses that have enhanced switch patterns is a fraction of the total number of multi-bit tracks and the fraction is represented by F_{c_enh}. *BusWidth* is the bus width of the multi-bit tracks. *ControlPatternIndex* is the index of a control pattern ranging from 1 to 4. The control patterns are added to the sparse switch patterns side by side with a multi-bit logic block. For each side of a multi-bit logic block, control patterns are added along the enhanced routing tracks to the intersections of the input buses and the routing buses where there is the spare switch pattern. Every time a control pattern is added to the sparse switch patterns, *ControlPatternIndex* increments by one until the variable equals 4 and is then reset to 1. We can use the similar basic algorithm to generate the other two enhanced sparse switch patterns. Instead of putting control patterns on a sparse switch pattern, we put a shift pattern or a vertical pattern.

Note that we only put the added switches in the multi-bit input connection blocks. The switches in the input connection blocks are constructed with multiplexers, so adding a switch to an input connection block means adding an input pin to a multiplexer in the connection block. Adding a switch to an input connection block consumes less active area than adding a switch, which is a pass transistor, to the output connection block.

The algorithm outputs the description of the control enhanced sparse switch patterns to the architecture file. Fig. 3.8 shows an example of the output of the algorithm. First, the parameter following a keyword *coarseTracksInUserMatrix* declares the number of multi-bit tracks in the switch patterns: 4 in this example. The keyword *fineTracksInUserMatrix* declares the number of single-bit tracks in the switch pattern: it is 0 because the algorithm doesn't generate switch patterns for the single-bit tracks. The following lines describe the matrix itself. The matrix for input pins of a multi-bit logic block is described first and is followed by the matrix for output pins. For both the input and output pins, the matrix is shown side-by-side along the edge of the multi-bit logic block. For example, *inputUserMatrix top* means the matrix following the keyword is for the input pins on the top side of the multi-bit logic block. In the matrix, *c* indicates a switch in the sparse switch pattern, *1* represents an added switch and *0* means no switch. The set of switches denoted by *c* in a bus must form the sparse switch pattern or else these switches will be treated as added switches.

The control enhanced sparse switch patterns described in Fig. 3.8 is shown in Fig. 3.9. In this figure, a black dot represents a switch in the sparse switch pattern and a white dot denotes an added switch. A thin short line is a logic block input pin and a thick short line is a logic block output pin. The long lines represent multi-bit routing tracks. In the switch patterns shown, the bus width is 4. There are a total of 40 input and 16 output pins in the switch pattern shown, distributed on the 4 sides of a multi-bit logic block. Among the 40 input pins, 8 input pins are located on both of the top and the right side and 12 are located on both of the bottom and the left side of the multi-bit logic block. The 16 output pins spread evenly on the 4 sides of the logic block. From left to right or from bottom to top, every 4 pins can be grouped together to form a bus.

```
coarseTracksInUserMatrix 4
fineTracksInUserMatrix 0
inputUserMatrix top:         0 0 0 0 c 1 1 1 n
                             0 0 0 0 0 c 0 0 n
                             0 0 0 0 0 0 c 0 n
                             0 0 0 0 0 0 0 c n
inputUserMatrix bottom:      0 0 0 0 c 0 0 0 c 0 0 0 n
                             0 0 0 0 1 c 1 1 0 c 0 0 n
                             0 0 0 0 0 0 c 0 1 1 c 1 n
                             0 0 0 0 0 0 0 c 0 0 0 c n
inputUserMatrix left:        0 0 0 0 c 0 0 0 c 1 1 1 n
                             0 0 0 0 0 c 0 0 0 c 0 0 n
                             0 0 0 0 0 0 c 0 0 0 c 0 n
                             0 0 0 0 1 1 1 c 0 0 0 c n
inputUserMatrix right:       c 0 0 0 c 0 0 0 n
                             1 c 1 1 0 c 0 0 n
                             0 0 c 0 1 1 c 1 n
                             0 0 0 c 0 0 0 c n
outputUserMatrix top:        c 0 0 0 n
                             0 c 0 0 n
                             0 0 c 0 n
                             0 0 0 c n
outputUserMatrix bottom:     c 0 0 0 n
                             0 c 0 0 n
                             0 0 c 0 n
                             0 0 0 c n
outputUserMatrix left:       0 0 0 0 n
                             0 0 0 0 n
                             0 0 0 0 n
                             0 0 0 0 n
outputUserMatrix right:      c 0 0 0 n
                             0 c 0 0 n
                             0 0 c 0 n
                             0 0 0 c n
```

Fig. 3.8 The Control Enhanced Sparse Switch Patterns Described in an Architecture File

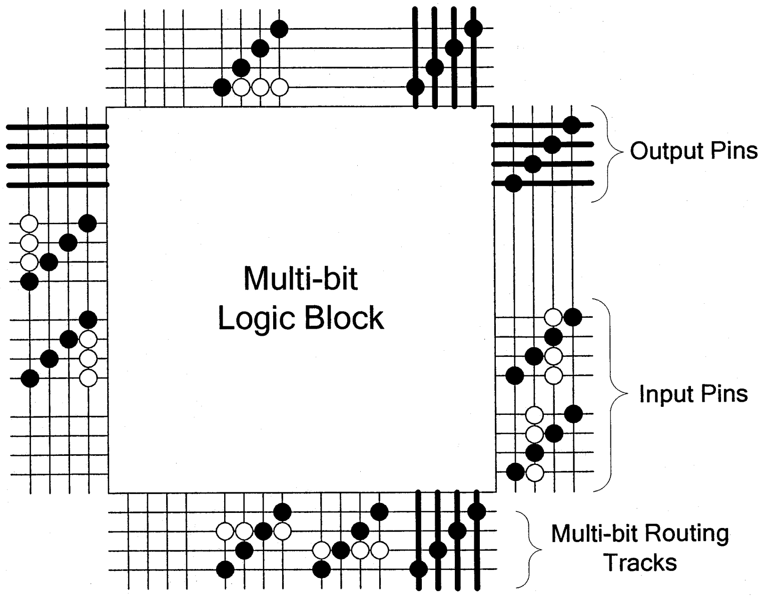


Fig. 3.9 An enhanced sparse switch pattern

We call the switch patterns generated by an external program and described in the architecture file the *external switch patterns*. After the multi-bit aware place and route tool reads the external switch patterns, the tool compares the number of multi-bit tracks in these switch patterns with that required by the tool. If the tool requires less multi-bit tracks than those in the external switch patterns, the extra multi-bit tracks at the end of these switch patterns will be ignored. If the tool needs more multi-bit tracks than those in the external switch patterns, the tool will generate switch patterns on multi-bit tracks to fill in the blank. The same process is applied to the single-bit tracks as well.

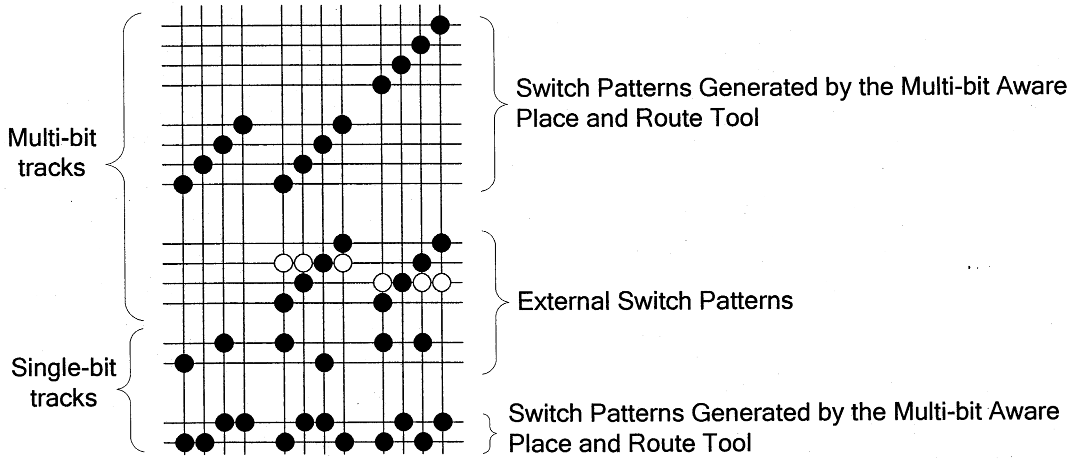
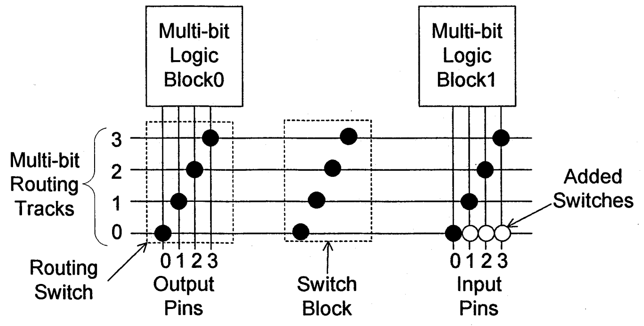
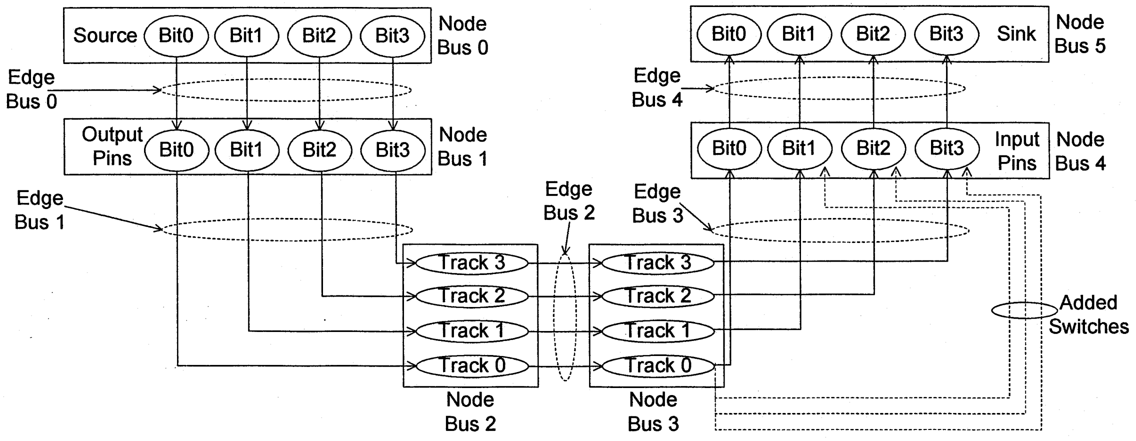

Multi-bit
tracks
Single-bit
tracks
Switch Patterns Generated by the Multi-bit Aware
Place and Route Tool
External Switch Patterns
Switch Patterns Generated by the Multi-bit Aware
Place and Route Tool

Fig. 3.10 Merged Switch Patterns

When there are switch patterns generated by both an external program and the multi-bit aware place and route tool, these switch patterns are merged in the way illustrated in Fig. 3.10. The external switch patterns are inserted into the switch patterns generated by the multi-bit aware place and route tool, between the multi-bit tracks and the single-bit tracks, so that multi-bit tracks can stay together and so do the single-bit tracks as shown in Fig. 3.10.



(a) Routing Resource in the Enhanced parse Architecture



(b) Routing Resource Graph

Fig. 3.11 Routing Resource Graph for the Enhanced Sparse Architecture

To model the enhanced sparse switch patterns in the multi-bit aware place and route tool, we modified the rr-graph in the tool. Fig. 3.11 (a) shows an enhanced sparse switch pattern. The corresponding rr-graph is illustrated in Fig. 3.11 (b). In Fig. 3.11 (a), the white dots denote added switches that connect multi-bit tracks to logic block input pins. The edges representing these switches in the rr-graph are denoted by dashed lines as shown in Fig. 3.11 (b). Since the added switches do not connect any buses, the edges representing these switches do not belong to any edge bus. The existing multi-bit aware place and route tool, however, assigns an edge bus index and a bit position to every edge that connects to multi-bit tracks. We modified the tool so that the modified tool only assigns edge bus indices and bit positions to the edges representing switches in the sparse switch pattern.

3.3. Design Space Analysis

The major change in the sparse and enhanced sparse architectures introduced in this chapter compared to earlier architectures is the design of the connection block. In this section, we compare the number of switch patterns that are possible in the connection block designs for the sparse and enhanced sparse architectures with that for the conventional and configuration memory sharing routing architectures. Note that two switch patterns can be different in either: 1) the arrangement of the switches (*switch arrangement*), or 2) whether configuration memory sharing is used for these switches. In this section, we use the number of possible switch patterns in a connection block to quantify the design space available to a particular routing architecture.

To simplify the following description, we use three parameters to characterize a connection block: W for the number of routing tracks, P for the number of logic block I/O pins, and M for the bus width of multi-bit tracks. We also use an example 2-bit wide connection block containing 4 logic block I/O pins and 6 routing tracks (W=6, P=4, M=2).

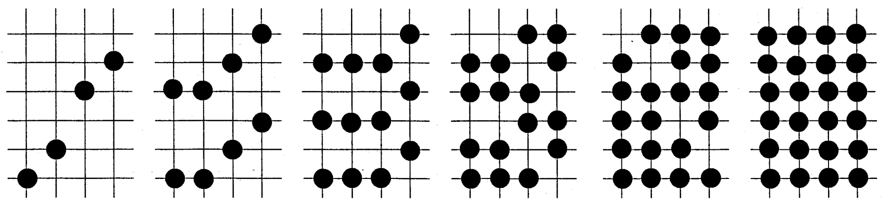


Fig. 3.12 Switch Patterns Generated By VPR

For the conventional architectures, the number of connection block switch patterns VPR can generate only depends on the number of routing tracks in the connection block. In a connection block containing 6 tracks, VPR can generate 6 switch arrangements, as shown in Fig. 3.12. Since the conventional architecture does not employ configuration memory sharing, the number of switch patterns is the same as the number of switch arrangements. To generalize this result: in a connection block containing W tracks, VPR can generate W switch patterns.

The number of all the possible switch arrangements for a connection block can be calculated by the formula: $\left(\sum_{i=1}^{W} \binom{W}{i} \right)^{P}$. The example connection block in Fig. 3.12 has almost 16 million possible switch arrangements. The number of switch patterns in the conventional architecture is therefore only a tiny fraction of all possible switch patterns for a connection block.

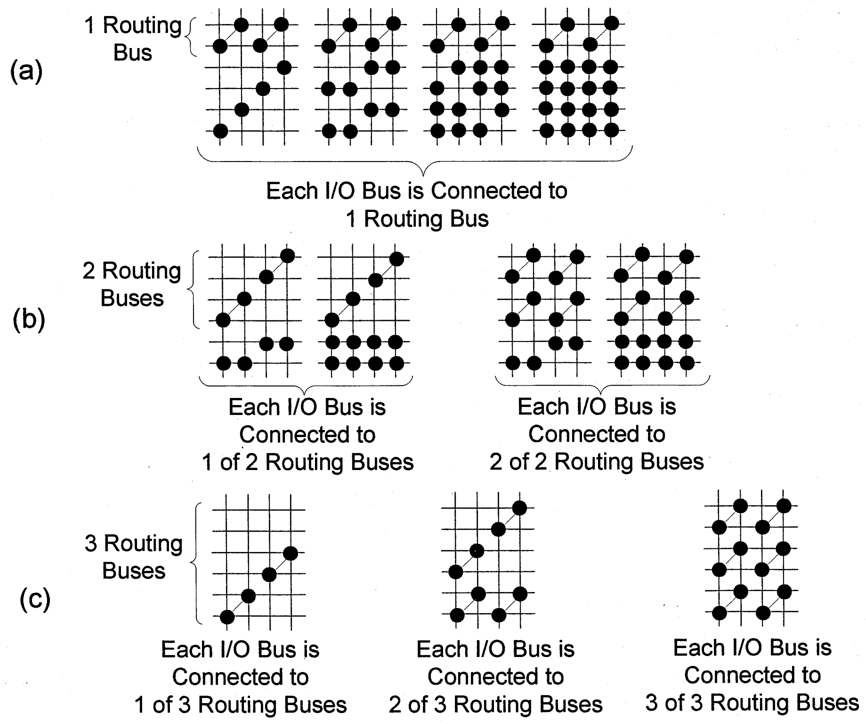

(a)
1 Routing Bus
Each I/O Bus is Connected to 1 Routing Bus
(b)
2 Routing Buses
Each I/O Bus is Connected to 1 of 2 Routing Buses
Each I/O Bus is Connected to 2 of 2 Routing Buses
(c)
3 Routing Buses
Each I/O Bus is Connected to 1 of 3 Routing Buses
Each I/O Bus is Connected to 2 of 3 Routing Buses
Each I/O Bus is Connected to 3 of 3 Routing Buses

Fig. 3.13 Switch Patterns in The Configuration Memory Sharing Routing Architecture

The configuration memory sharing routing architecture makes a step towards the wide open design space. The number of connection block switch patterns that can be created in the configuration memory sharing routing architecture depends on the number of tracks in the connection block and the bus width of the routing architecture. As shown in Fig. 3.13, when there are 6 routing tracks in a 2 bit wide connection block, 11 switch patterns can be generated. Fig. 3.13 (a) shows the switch patterns for the connection block with only 1 routing bus. In the case, there is only 1 switch pattern for the multi-bit connection block and 4 switch patterns for the single-bit connection block. When there are two routing buses in the connection block as shown in Fig. 3.13 (b), two switch patterns can be generated for the multi-bit connection block, and for each of the two switch patterns, two switch patterns can be created for the single-bit connection block. When there are 3 routing buses in the connection block, 3 switch patterns can be generated for the multi-bit connection block and there is no single-bit connection block, as shown in Fig. 3.13 (c). For any M bit wide connection block with W tracks, there are $\sum_{i=1}^{(W/M)} [i \times \max(1, W - i \times M)]$ switch patterns that can be generated in the configuration memory sharing routing architecture.

The connection block switches in the sparse architecture are arranged in the same way as those in the configuration memory sharing routing architecture, but they are controlled differently. The number of connection block switch patterns in the sparse architecture, therefore, is the same as for configuration memory sharing routing architecture. The switch patterns are, however, distinct; the sparse architecture switch patterns are not the same as the configuration memory sharing routing architecture.

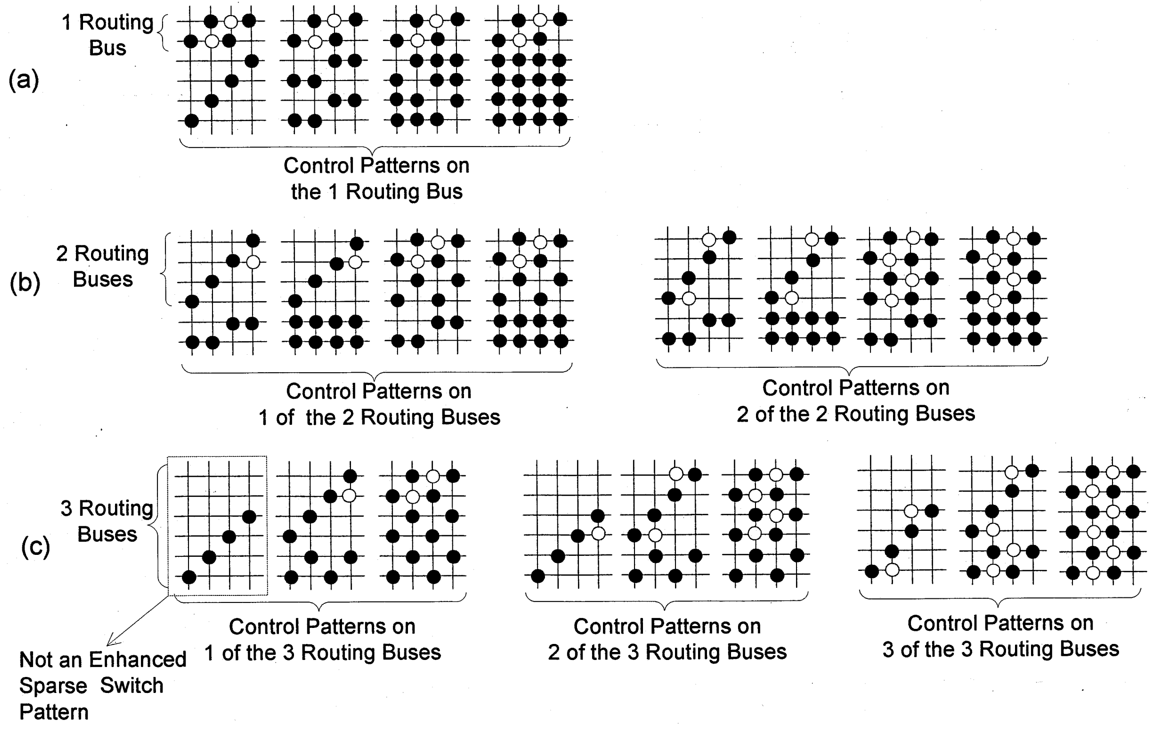

1 Routing Bus
(a)
Control Patterns on the 1 Routing Bus
2 Routing Buses
(b)
Control Patterns on 1 of the 2 Routing Buses
Control Patterns on 2 of the 2 Routing Buses
3 Routing Buses
(c)
Not an Enhanced Sparse Switch Pattern
Control Patterns on 1 of the 3 Routing Buses
Control Patterns on 2 of the 3 Routing Buses
Control Patterns on 3 of the 3 Routing Buses

Fig. 3.14 Switch Patterns in the Control Enhanced Sparse architecture

The number of connection block switch patterns that can be produced in the enhanced sparse architecture depends on the number of routing tracks, the number of logic block I/O pins and the bus width of the connection block. Fig. 3.14 shows the possible switch patterns for the enhanced sparse architecture. The black dots in the switch patterns in Fig.

3.14 (a) are arranged in the same way as those in Fig. 3.13 (a), but there are control patterns (shown with white dots) on the multi-bit tracks in Fig. 3.14 (a). In Fig. 3.14 (b), the switch patterns are divided into two groups according to the number of routing buses that have control patterns. The black dots in each group are arranged in the same way as those shown in Fig. 3.13 (b). Similarly, the switch patterns in Fig. 3.14 (c) are divided into 3 groups. The black dot arrangements in each group are the same as those shown in Fig. 3.13 (c). The first switch pattern in the first group, however, is not an enhanced sparse switch pattern and should not be counted with the other patterns in this row of the figure. In this extra switch pattern, the control pattern can not be added to the top routing bus because this bus does not have any switches from the sparse switch pattern. This situation can only occur when the number of routing buses is larger than the number of logic block I/O buses. The amount by which the number of routing buses exceeds the number of logic block I/O buses determines how many switch patterns should be subtracted from the total number of possible switch patterns. As a result, we can generate 20 enhanced spare switch patterns in total for the connection block shown in Fig. 3.14. We can generalize for any M bit wide connection block that contains W routing tracks and P logic block I/O pins in the enhanced sparse architecture to compute the number of switch patterns: $\sum_{i=1}^{(W/M)} \left[i \times i \times \max(1, W - i \times M) - \sum_{j=1}^{\max(i-P/M,0)} j \right]$.

Note that the control enhanced sparse architecture is only one of the 3 enhanced sparse architectures we investigate in this research. When the bus width of the enhanced sparse architecture is more than 2, we can have all 3 types of enhanced sparse switch patterns. Consequently, the number of the enhanced sparse switch patterns is 3 times of that of the control enhanced sparse switch patterns.

Table 3.1 shows the number of switch patterns that can be generated for 4 bit wide connection blocks with 8 logic block I/O pins and 8, 12, or 16 routing tracks. When there are 8 routing tracks in the connection block, 8 switch patterns can be generated in the conventional architecture, 6 in the configuration memory sharing and the sparse architecture respectively, and 24 in the enhanced sparse architectures. The sparse and the enhanced sparse architecture introduced in this research together more than double the design space that has been previously explored by the conventional and the configuration memory sharing routing architectures. This design space expansion with the two new routing architectures is even more obvious when there are 12 or 16 routing tracks in the switch pattern.

Table 3.1 Connection Block Design Space

W	M	P	Conventional	Configuration Memory Sharing	Sparse	Enhanced Sparse
8	4	8	8	6	6	24
12	4	8	12	19	19	96
16	4	8	16	44	44	276

In the remainder of the thesis, we compare these two new routing architectures to the earlier routing architectures in more detail, and empirically report on their area efficiency.

CHAPTER 4

TRADEOFFS BETWEEN ROUTING SWITCH DENSITY AND ROUTING FLEXIBILITY

In this chapter, we compare the routing switch density in conventional and configuration memory sharing routing architectures to that of the new routing architectures introduced in this thesis: sparse and enhanced sparse architectures. From the extra routing switch density in the sparse and the enhanced sparse over the configuration memory sharing routing architecture, we then derive the extra routing flexibility needed to compensate for the density increase in the two new routing architectures. The evaluation covers a wide range of F_c values for all of these routing architectures. Section 4.1 describes the circuit assumptions for the evaluation presented in Section 4.2.

4.1. Circuit Assumptions

Our evaluation is based on the following circuit level assumptions. First, as in [4], we assume each SRAM cell is constructed out of six minimum width transistors. We also assume that two types of buffers are used in our architectures – a 4x buffer with four times the minimum driving strength and a 5x buffer with five times the minimum driving strength. Both buffers use a two-stage design with a stage ratio of four and five respectively.

The 5x buffers are used in constructing the two types of routing switches found in the switch blocks – a full switch and a half switch. Each track segment is assumed to have a segment length of 2. Bus width in our evaluation is 4. The general circuit design of switch blocks is described in section 2.1.3. The switch block design for the configuration memory sharing routing architecture is illustrated in section 2.3.1 and that for the sparse and the enhanced sparse architectures is in section 3.1

For the conventional, the configuration memory sharing, and the sparse architectures, we assume, as shown in Fig. 4.1, each logic block input pin is connected to its neighboring routing channel through a 4x buffer followed by a multiplexer. The multiplexer is constructed out of $2I_m - 2$ pass transistors of minimum width and is controlled by $\lceil \log_2(I_m) \rceil$ SRAM cells, where I_m is the number of multiplexer data inputs. The multiplexer is then connected to routing tracks through a shared 4x buffer.

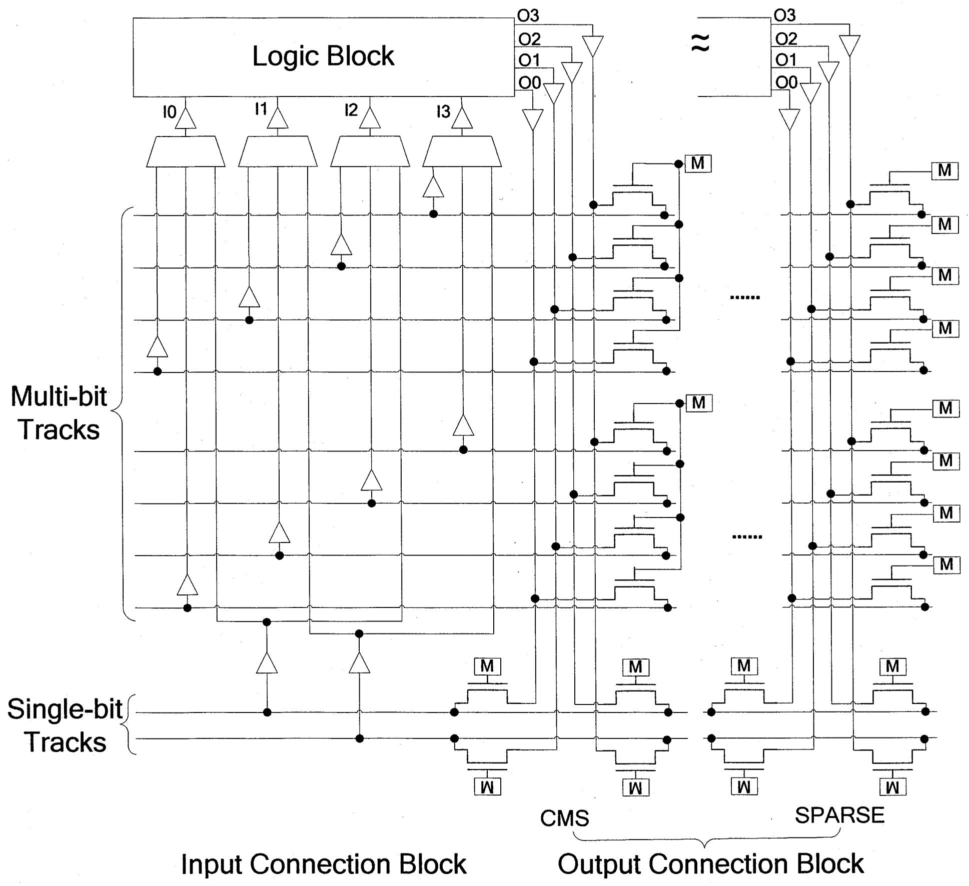


Fig. 4.1 Input and Output Connection Blocks in the Configuration Memory Sharing and Sparse Architectures

As shown in Fig. 4.1, each output pin is connected to a 5x buffer. The buffer is then connected to a set of routing tracks with each track connected to the output of the buffer through a 5x pass transistor. For the multi-bit routing tracks in the configuration memory sharing architecture, every four pass transistors share a single bit of configuration memory. For single-bit tracks in the configuration memory sharing architecture and all tracks in the conventional and the sparse architectures, the pass transistors are independently controlled.

For the enhanced sparse architectures, as shown in Fig. 4.2, we use the same circuit assumption as that in the sparse architectures except that in the input connection block, there are several extra connections indicated by thick lines in the figure.

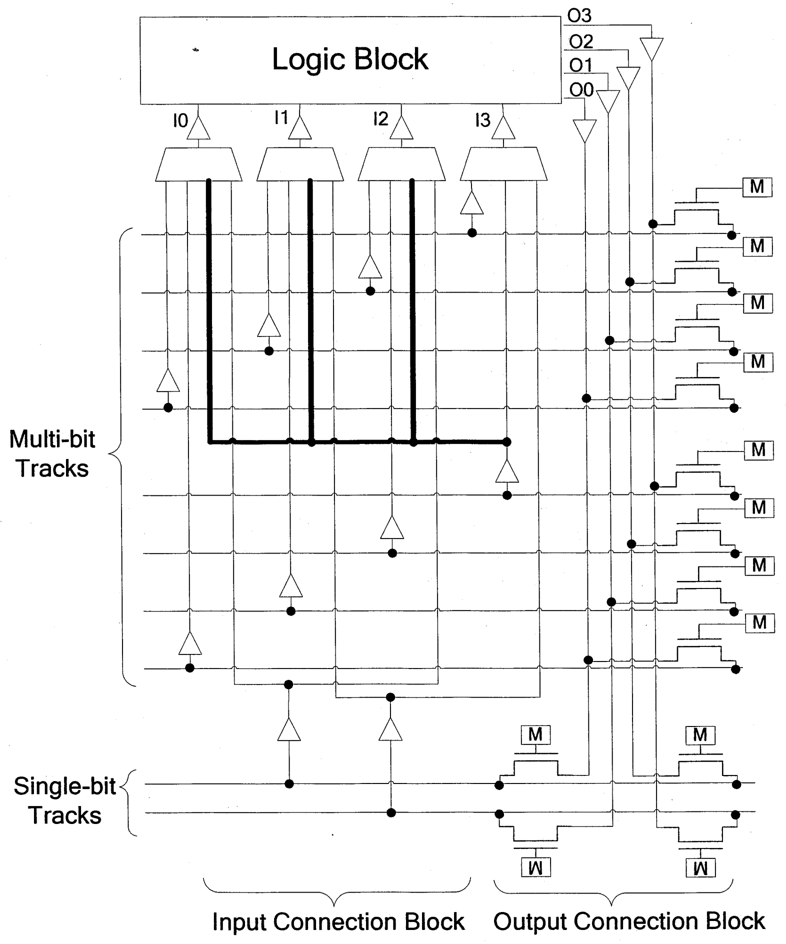


Fig. 4.2 Input And Output Connection Block In the Enhanced Sparse Architecture

Using these basic circuit components, we can evaluate the active routing area in each of the four routing architectures. Note that, in this work, the active area is measured using the equivalent minimum width transistor area as defined in [4] and is calculated using the following equation:

$$Area = \sum_{All\_Trans} \left(0.5 + \frac{Drive\_Strength\_of\_the\_Current\_Trans}{2 \times Drive\_Strength\_of\_Min\_Width\_Trans} \right)$$

4.2. Routing Switch Density Comparison

The routing switch density for a routing architecture is measured by the active routing area per *tile* for the routing architecture. A tile contains a logic block, the horizontal channel at the bottom and the vertical channel on the left of the logic block. In this demonstration, we assume that the configuration memory sharing, the sparse and the enhanced sparse architectures contain only multi-bit routing tracks and the routing channels in all the routing architectures contain 40 routing tracks.

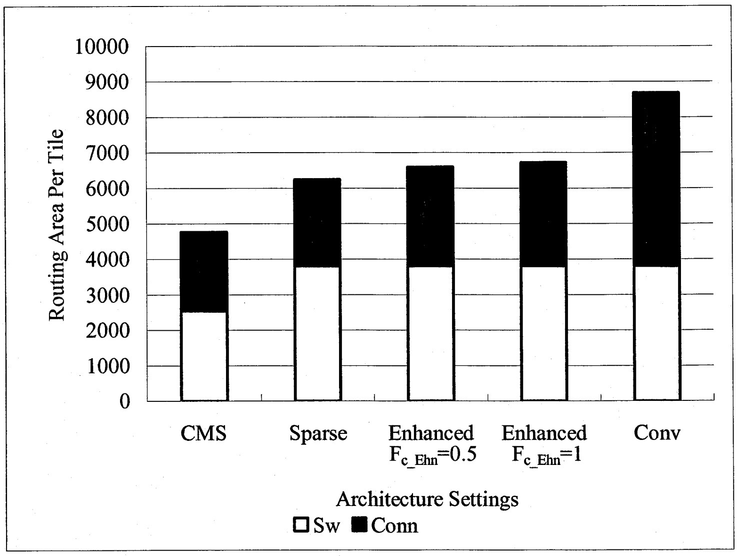


Fig. 4.3 Routing Area Per Tile in Minimum Width Trans When F_{c_in}/F_{c_out}=0.4/0.25

Fig. 4.3 plots the routing area per *tile* for the configuration memory sharing (CMS), the sparse, the enhanced sparse (Enhanced) and the conventional (Conv) routing architectures when F_{c_in}/F_{c_out}=0.4/0.25. F_{c_in} represents the F_c value for logic block input pins and F_{c_out} represents the F_c value for logic block output pins. As shown, the configuration memory sharing, sparse and enhanced sparse architectures of different F_{c_enh} values all consume significantly less area per tile than the conventional architectures.

Fig. 4.3 also breaks down the tile area into switch block (Sw) and connection block (Conn) area where switch block area is the total area consumed by the switch block and the connection block area is the total area consumed by the connections between the logic block input/output pins and the routing tracks. As shown, the sparse and the enhanced sparse architectures consume the same amount of switch block area and less connection block area than the conventional architectures. The configuration memory sharing architectures, on the other hand, consume both less switch block area and less connection block area per tile.

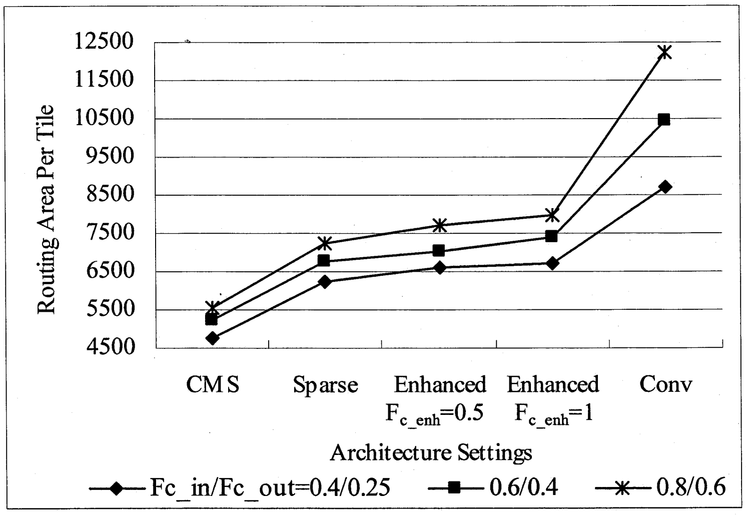


Fig. 4.4 Routing Area per Tile in Minimum Width Transistors

We vary the F_c values from F_{c_in}/F_{c_out}=0.4/0.25 to 0.8/0.6 in the example above and the area per tile for all the routing architectures is plotted in Fig. 4.4. It is important to note that with F_c set to 0.8 for input pins and 0.6 for output pins the configuration memory sharing, the sparse and the enhanced sparse architectures still consume significantly less area than the conventional architecture with F_c values of 0.4 and 0.25.

Table 4.1 Single-Bit and Multi-bit Track Reduction

	F_c Input	F_c Output	Area Per Multi-Bit Track	40 Multi-Bit Track Area	Increase from CMS 0.4/0.25	Multi-Bit Track Reduction	Single-Bit Track Reduction
CMS	0.4	0.25	59.5	2380.6	0.0	0.0	0.0
	0.6	0.4	65.4	2616.6	236.0	3.6	2.2
	0.8	0.6	69.2	2768.6	388.0	5.6	3.6
Sparse	0.4	0.25	78.0	3118.6	738.0	9.5	6.8
	0.6	0.4	84.8	3390.6	1010.0	11.9	9.3
	0.8	0.6	90.4	3614.6	1234.0	13.7	11.4
Enhanced F_{c_Ehn}=0.5	0.4	0.25	82.5	3298.6	918.0	11.1	8.5
	0.6	0.4	88.1	3525.6	1145.0	13.0	10.5
	0.8	0.6	96.4	3854.6	1474.0	15.3	13.6

The above results show that, for the same F_c values, the area per multi-bit track is much less than the area per single-bit track. As the F_c values increase, the area per track increases as well. For example, with configuration memory sharing, when F_c values are set to 0.4 and 0.25 for the input and output pins (as in [22]**Error! Reference source not found.**[22]), the area of a multi-bit track is 59.5 per tile. In this architecture, forty multi-bit tracks consume 2380.6 equivalent minimum width transistors in a tile. In the sparse architecture with the same F_c values, the area per track increase to 78. Forty tracks now consume 3118.6 minimum width transistors. This increase in 738 minimum width transistors can be compensated by a reduction of 9.5 multi-bit or 6.8 single-bit tracks through the increase in the routing flexibility of multi-bit track (assuming the F_c values are set to 0.4 and 0.25 for input and output pins respectively for the single-bit tracks). The same calculation can be carried out for all configuration memory sharing, sparse and enhanced sparse architectures shown in Fig. 4.4 and are summarized in Table 4.1.

From Table 4.1, one can see that the sparse and the enhanced sparse architectures require significant reduction of routing tracks to compensate for the extra area consumption on multi-bit tracks. To evaluate the area efficiency of these two routing architectures, we perform a set of experiments in this research and we report the results in Chapter 6, after the routing tools for this set of experiments are selected in the next chapter.

CHAPTER 5

ROUTING TOOL SELECTION

Two existing routing tools that we select for our architectural evaluation are: 1) the *conventional routing tool* in VPR; 2) the *multi-bit aware routing tool* in the multi-bit aware place and route tool. The conventional routing tool is designed for the conventional architecture and routes signals one bit at a time. The multi-bit aware routing tool, on the other hand, is designed for the configuration memory sharing architecture and identifies multi-bit signals, which in turn are routed in groups by the tool. Similar to the configuration memory sharing architecture, the sparse and enhanced sparse architectures have multi-bit routing tracks. The multi-bit tracks in the sparse and enhanced sparse architectures, however, are independently controlled by configuration memory cells as in the conventional architecture. As such, we can use either the conventional routing tool or the multi-bit aware routing tool for the sparse and enhanced sparse architectures. In this chapter, we consider each of these alternatives.

Table 5.1 Routing Area for the Sparse Architecture

Benchmarks	Minimum Routing Area in Minimum Width Transistor		Conventional Over multi-bit Aware
	Multi-bit Aware	Conventional	
code_seq_dp	2.64E+05	2.56E+05	-3.0%
Dcu_dpath	7.86E+05	7.22E+05	-8.2%
ex_dpath	2.88E+06	2.53E+06	-12.3%
exponent_dp	5.09E+05	4.42E+05	-13.1%
Icu_dpath	3.34E+06	2.94E+06	-11.8%
imdr_dpath	1.19E+06	1.06E+06	-10.7%
Incmod	7.23E+05	6.53E+05	-9.6%
mantissa_dp	1.09E+06	9.74E+05	-10.8%
multmod_dp	1.44E+06	1.38E+06	-4.3%
pipe_dpath	2.63E+05	2.53E+05	-3.7%
prils_dp	2.56E+05	2.39E+05	-6.7%
rsadd_dp	2.02E+05	1.96E+05	-2.9%
smu_dpath	3.96E+05	3.78E+05	-4.6%
ucode_dat	1.07E+06	9.57E+05	-11.0%
ucode_reg	5.30E+04	5.24E+04	-1.1%
Average	**9.65E+05**	**8.69E+05**	**-9.9%**

To find out which routing tool should be employed in our evaluation, we used both tools to route 15 data-path circuits from the Pico-Java processor [30] on the sparse routing architecture. We set the F_c values on both multi-bit and single-bit tracks to 0.4 for logic block input pins and 0.25 for logic block output pins, as in the research in [27]. Fixing F_c values, we then varied the number of multi-bit tracks from 4 to 64 tracks. For a given number of multi-bit tracks, we search for the minimum number of single-bit tracks that are required to implement each benchmark circuit. The minimum implementation routing area for each benchmark circuit, measured in minimum width transistors, is shown in Table 5.1. As shown, the average routing area obtained by the conventional routing tool is unexpectedly smaller (9.9%) than the multi-bit aware routing tool's result.

We find this difference to be surprising because we started this research with two basic assumptions: 1) the multi-bit aware and the conventional routing tools have similar area efficiency when routing either single- or multi-bit signals one bit at a time, and 2) routing multi-bit signals in groups as in the multi-bit aware routing tool is more area efficient than breaking these signals into single-bit signals as in the conventional routing tool. If these two assumptions were correct, we should see better routing results for the sparse routing architecture from the multi-bit aware routing tool than the conventional tool. The actual result, however, is the opposite, which motivated us to examine our assumptions more carefully. The remainder of this chapter describes our efforts to better understand the results presented in Table 5.1.

Table 5.2 Routing Area for the Conventional Architecture

Benchmarks	Routing Area In Minimum Width Transistors		Conventional Over Multi-bit Aware
	Multi-bit Aware	Conventional	
code_seq_dp	2.663E+05	2.570E+05	-3.5%
dcu_dpath	8.811E+05	8.159E+05	-7.4%
ex_dpath	3.460E+06	2.999E+06	-13.3%
exponent_dp	5.397E+05	4.531E+05	-16.1%
icu_dpath	4.072E+06	3.325E+06	-18.3%
imdr_dpath	1.287E+06	1.161E+06	-9.7%
Incmod	7.274E+05	6.863E+05	-5.7%
mantissa_dp	1.164E+06	1.013E+06	-13.0%
multmod_dp	1.466E+06	1.383E+06	-5.6%
pipe_dpath	2.712E+05	2.653E+05	-2.2%
prils_dp	2.535E+05	2.432E+05	-4.1%
rsadd_dp	2.087E+05	1.914E+05	-8.3%
smu_dpath	4.165E+05	3.891E+05	-6.6%
ucode_dat	1.164E+06	1.066E+06	-8.5%
ucode_reg	5.646E+04	5.865E+04	3.9%
Average	**1.082E+06**	**9.538E+05**	-11.9%

We begin with our first assumption that the two routing tools have similar efficiency when routing signals one bit at a time. Rather than studying the sparse architecture which has a mixture of single and multi-bit routing tracks, we decided to simplify our analysis by applying the two routing tools to the conventional architecture where signals can only be routed one bit at a time. As shown in Table 5.2, similar to the result obtained for the sparse routing architecture, the average routing area obtained by the conventional routing tool again is 11.9% smaller than that obtained by the multi-bit aware routing tool.

To understand why the multi-bit aware routing tool is less area efficient than the conventional routing tool for the conventional architecture, we looked more deeply into the tools. Both the conventional and multi-bit aware routing tools use many routing iterations to successfully route a circuit [31]. During each routing iteration, previously routed nets are *ripped up* and rerouted based on the congestion cost obtained in the previous routing iteration. The routing process stops when a feasible routing solution is found or the number of iterations has reached a limit. Fig. 5.1 and Fig. 5.2 show a routing iteration in the conventional and multi-bit aware routing tools, respectively.

```
Sort all nets by decreasing fan-out of the nets
FOR each net
    IF the net is routed
        Rip up the net
    ENDIF
    Route the net on either multi-bit or single-bit tracks
ENDFOR
```

Fig. 5.1 A Routing Iteration in the Conventional Routing Tool

As shown in Fig. 5.1, the conventional routing tool sorts all the multi-bit and single-bit signals by decreasing fan-out and then routes signals bit by bit on either the multi-bit routing tracks or the single-bit tracks, whichever has the lowest routing cost. The routing tool removes the routing for one net (rip up a net) at a time and re-route the net immediately.

In Fig. 5.2, the multi-bit aware routing tool routes multi-bit signals first, followed by single-bit signals sorted by the decreasing fan-out of the nets. When routing multi-bit signals, the multi-bit aware routing tool first rips up all the nets in a multi-bit signal and tries to route the multi-bit signal on the multi-bit tracks as a coherent group. If the attempt fails, the nets in the multi-bit signal will be routed as a sequence of individual signals on either the multi-bit or single-bit tracks.

```
FOR each multi-bit signal
    FOR each bit in the multi-bit signal
        IF the net at the current bit position in the multi-bit signal is routed
            Rip up the net
        ENDIF
    ENDFOR
    Route the multi-bit signal on multi-bit tracks
    IF the multi-bit signal can not be routed on multi-bit tracks
        FOR each bit in the multi-bit signal
            Route the net at the current bit position on either multi- or single-bit tracks
        ENDFOR
    ENDIF
ENDFOR

Sort all the single-bit nets by decreasing fan-out of the nets
FOR each single-bit net
    IF the net is routed
        Rip up the net
    ENDIF
    Route the net on either multi-bit or single-bit tracks
ENDFOR
```

Fig. 5.2 A Routing Iteration in the Multi-bit Aware Routing Tool

On a conventional routing architecture, which does not contain any multi-bit tracks, there are two differences between the multi-bit aware and the conventional routing tools: 1) the order in which nets are considered for routing, and 2) the way in which nets are ripped up before re-routing. In the multi-bit aware routing tool, multi-bit signals are routed before single-bit signals, whereas the signals are routed in order of decreasing fan-out in the conventional routing tool. The multi-bit aware routing tool also rips up all the nets in a multi-bit signal at the same time and these signals are then routed individually on single-bit routing tracks. The conventional routing tool routes all signals as single-bit signals, so it only rips up and routes one net at a time.

To find out which difference between the two routing tools contributes most to the results shown in Table 5.2, we performed two experiments, each of which targets one of the two differences. In the first experiment, we changed the routing sequence in the conventional routing tool to be the same as the one in the multi-bit aware routing tool, while in the second experiment, we modified the way signals are ripped up in the multi-bit aware routing tool to be the same as in the conventional routing tool. If these are the only two differences between the two routing tools, the routing results in the first and the second experiment should be identical. Also, the area result will tell us how much of the 11.9% difference can be attributed to each of differences between the two tools.

```
FOR each multi-bit signal
    FOR each bit in the multi-bit signal
        IF the net at the current bit position in the multi-bit signal is routed
            Rip up the net
        ENDIF
        Route the net on either multi-bit or single-bit tracks
    ENDFOR
ENDFOR

Sort all the single-bit nets by decreasing fan-out of the nets
FOR each single-bit net
    IF the net is routed
        Rip up the net
    ENDIF
        Route the net on either multi-bit or single-bit tracks
ENDFOR
```

Fig. 5.3 Experiment 1: Change the Routing Sequence in the Conventional Routing Tool

In the first experiment, we changed the routing sequence in the conventional routing tool so that the conventional routing tool routes multi-bit signals before single-bit signals as in the multi-bit aware routing tool. The process is shown in Fig. 5.3.

The second experiment modified the way signals are ripped up in the multi-bit aware routing tool. As shown in Fig. 5.4, the routing tool first rips up all the nets in a multi-bit signal and at the same time, the routing tool temporarily stores the routing for the nets at non-zero bit positions. After the attempt to route the multi-bit signal on multi-bit tracks fails, which is always the case on the conventional architecture, the routing tool recovers the routings for the nets at non-zero bit positions in the multi-bit signal, and then rips up and routes the nets in the multi-bit signal one by one. For multi-bit aware architecture, if the multi-bit signal is successfully routed on the multi-bit tracks, the routing process frees the temporarily stored routing for the nets at non-zero bit positions and continues to route the next signal. After finishing routing all the multi-bit signals, the routing tool will route single-bit signals sorted in the decreasing fan-out order. With the modification, the multi-bit aware routing tool rips up nets for both multi- and single-bit signals the same way as the conventional routing tool on the conventional architecture.

```
FOR each multi-bit signal
    IF the multi-bit signal is routed
        FOR each bit in the multi-bit signal
            IF the current bit position is 0
                Rip up the net at the current bit position in the multi-bit signal
            ELSE
                Temporarily store the routing for the net at the current bit position
                Rip up the current net at the current bit position
            ENDIF
        ENDFOR
    ENDIF
    Route the multi-bit signal on multi-bit tracks
    IF the multi-bit signal can not be routed on multi-bit tracks
        IF there are temporarily stored routing for the  nets at non-zero bit positions
           in the multi-bit signal
            Recover the routing for these nets
        ENDIF
        FOR each bit in the multi-bit signal
            IF the net at the current position is not ripped up
                Rip up the net
            ENDIF
            Route the net at the current bit position on either multi- or single-bit tracks
        ENDFOR
    ELSE
        Free temporarily stored routing for the nets at non-zero bit positions
    ENDIF
ENDFOR

Sort all the single-bit nets by decreasing fan-out of the nets
FOR each single-bit net
    IF the net is routed
        Rip up the net
    ENDIF
    Route the net on either multi-bit or single-bit tracks
ENDFOR
```

Fig. 5.4 Experiment 2: Change the Way of Ripping Up Nets in the Multi-bit Aware Routing Tool

Table 5.3 Comparison of Routing Area for the Conventional Architecture

Benchmarks	Routing Area In Min Width Transistors			
	Conventional	Experiment 1	Experiment 2	Multi-bit Aware
code_seq_dp	2.570E+05	2.663E+05	2.663E+05	2.663E+05
dcu_dpath	8.159E+05	8.159E+05	8.159E+05	8.811E+05
ex_dpath	2.999E+06	2.999E+06	2.999E+06	3.460E+06
exponent_dp	4.531E+05	4.650E+05	4.650E+05	5.397E+05
icu_dpath	3.325E+06	3.377E+06	3.377E+06	4.072E+06
imdr_dpath	1.161E+06	1.148E+06	1.148E+06	1.287E+06
Incmod	6.863E+05	6.978E+05	6.978E+05	7.274E+05
mantissa_dp	1.013E+06	1.013E+06	1.013E+06	1.164E+06
multmod_dp	1.383E+06	1.367E+06	1.367E+06	1.466E+06
pipe_dpath	2.653E+05	2.541E+05	2.541E+05	2.712E+05
prils_dp	2.432E+05	2.482E+05	2.482E+05	2.535E+05
rsadd_dp	1.914E+05	1.958E+05	1.958E+05	2.087E+05
smu_dpath	3.891E+05	3.748E+05	3.748E+05	4.165E+05
ucode_dat	1.066E+06	1.066E+06	1.066E+06	1.164E+06
ucode_reg	5.865E+04	5.865E+04	5.865E+04	5.646E+04
Average	**9.538E+05**	**9.565E+05**	**9.565E+05**	**1.082E+06**
	→ 2.7%		11.6 % ←	

The results are shown in Table 5.3, where the routing results for the conventional and the multi-bit aware routing tools are in column 2 and column 5 as reference. Columns 3 and 4 show the routing results for the two experiments. Changing the routing sequence in the conventional routing tool to be the same as in the multi-bit aware routing tool (experiment 1) yields a slight area increase of 2.7%. Modifying the way the multi-bit aware routing tool rips up nets to match how nets are ripped up in the conventional routing tool (experiment 2) caused a significant 11.6% drop in average routing area from the original multi-bit aware routing tool result. Therefore, the different ways the two routing tools rip up nets accounts for most of the difference in the routing results obtained from the single-bit and multi-bit aware routing tools on the conventional architecture. Because the two routing results for experiment 1 and experiment 2 are identical, we have also verified that only these two differences exist between the conventional and the multi-bit aware routing tools.

These two experiments, however, are not able to fully account for the area differences shown in Table 5.1 for the sparse architecture. In particular, the algorithm shown in Fig. 5.4 can only rip up or recover complete nets at non-zero bit positions in a multi-bit signal. In multi-bit aware routing architectures, a multi-bit signal with multiple fan-outs can be partially routed on multi-bit tracks after being completely ripped up. In the case, the multi-bit aware routing tool needs to recover only the unrouted parts of the nets at non-zero bit positions in the multi-bit signal, and then partially rip up each bit before re-routing it. Changing the way in which the nets are ripped up for multi-bit aware routing architectures requires, therefore, a much more extensive modification than what was needed for the conventional architecture. Since our primary research focus is on routing architectures rather than routing tool design, we chose not to divert our studies; we leave to future work the required modifications to the multi-bit aware routing tool for multi-bit aware routing architectures. Until these modifications are made, however, the multi-bit aware routing tool will have lower area efficiency than the conventional routing tool when routing signals on multi-bit aware routing architectures without configuration memory sharing.

Investigating the second assumption we described at the beginning of this chapter – that routing multi-bit signals in groups as in multi-bit aware routing tools is more area efficient than routing multi-bit signals bit by bit as in the conventional routing tool – also depends upon these extensive modifications to the multi-bit aware routing tools. We must therefore defer an in depth examination of this assumption to future work.

The lower efficiency of multi-bit aware routing tool compared to the conventional routing tool when routing signals bit by bit, however, means we must be careful to avoid including differences in the routing tool when comparing the sparse and enhanced sparse routing architectures to other architectures. Consequently, for the remainder of the thesis, when we compare the sparse and enhanced sparse routing architectures to the configuration memory sharing architecture, we will use the multi-bit aware routing tool. When we compare to the conventional architectures, we will use the conventional routing tool. In this way, we keep the comparison strictly fair and avoid inaccuracies due to differences between the multi-bit aware and conventional routing tools. The results of these comparisons are in Chapter 6.

CHAPTER 6

EXPERIMENTAL RESULTS

To understand the trade-off between the routing flexibility and the routing switch density, we employ 15 data-path circuits from the Pico-Java processor [30]. These circuits are implemented on both the conventional and the multi-bit aware routing architectures.

Through these implementations, we attempt to address the following four questions: 1. What is the effect of configuration memory sharing on the overall area efficiency of FPGAs? 2. What is the most efficient method of increasing the area efficiency of the sparse architecture – increasing the flexibility of the input connection blocks through control and shift enhancements or reducing the implementation area of the output and switch blocks through configuration memory sharing? 3. How does the area efficiency of the multi-bit aware routing architectures compare to the area efficiency of the conventional architecture? 4. How does the performance of the multi-bit aware routing architectures compare to the that of the conventional one?

Table 6.1 Notations for Architectural Parameters

Classification		Parameter	Description
Parameters for Conventional Architecture		F_{c_in}	Fraction of routing tracks a logic block input pin connects to
		F_{c_out}	Fraction of routing tracks a logic block Output pin connects to
		W	Number of routing tracks per channel
Parameters for Multi-bit aware routing architecture	Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	Fraction of single-bit tracks a logic block input pin connects to
		$F_{c\_out\_single}$	Fraction of single-bit tracks a logic block Output pin connects to
		W_{single}	Number of single-bit tracks per channel
	Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	Fraction of routing buses a logic block input bus connects to
		$F_{c\_out\_multi}$	Fraction of routing buses a logic block Output bus connects to
		W_{multi}	Number of multi-bit tracks per channel
		F_{c_enh}	Fraction of routing buses that have the enhanced sparse switch patterns
Common Parameters		L	The length of a routing track segment

In this chapter, we characterize the variation in the architectures we studied by several architectural parameters, whose notations are described in Table 6.1.

6.1. Effect of Configuration Memory Sharing on Area Efficiency

Table 6.2 Parameter Settings for the Sparse and the Configuration Memory Sharing Routing Architectures

Classification	Parameter	Minimum Value	Maximum Value
Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	0.4	N.A.
	$F_{c\_out\_single}$	0.25	N.A.
	W_{single}	Best	N.A.
Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	0.4	1.0
	$F_{c\_out\_multi}$	0.25	0.8
	W_{multi}	4	64
Common Parameters	L	2	N.A.
	Routing Tool	Multi-Bit Aware [29]	N.A.

To investigate the effect of configuration memory sharing on the area efficiency of FPGAs, we measure the amount of routing area that is required to implement the benchmark circuits for both the configuration memory sharing and the sparse architectures. The parameter settings used in this investigation are shown in Table 6.2. As shown, the F_c values ($F_{c\_in\_single}$ and $F_{c\_out\_single}$) are kept constant for single-bit tracks in both architectures. These values were found to be the most efficient for the configuration memory sharing architecture in [22][22]. We assume that they are equally efficient for the sparse architecture. Similarly, the routing track segment length was set to two and the minimum area implementation of the multiplexer is used.

We varied F_c values ($F_{c\_in\_multi}$ and $F_{c\_out\_multi}$) for the multi-bit tracks for both the configuration memory sharing and the sparse architectures. For each set of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$ values, we varied the number of multi-bit tracks from 4 to 64 tracks. For a given number of multi-bit tracks, we search for the minimum number of single-bit tracks that are required to implement each circuit.

Fig. 6.1 and Fig. 6.2 show the average number of routing track segments that are required to implement a circuit for the configuration memory sharing and the sparse architectures, respectively. There are two curves in each figure – one shows the average number of single-bit track segments that are required to implement a circuit while the other shows the average number of multi-bit track segments. As shown in Fig. 6.1, for the configuration memory sharing architecture, the utilization of the multi-bit tracks increases with the increasing values of $Fc_{\_in\_multi}$ and $Fc_{\_out\_multi}$. In particular, when $Fc_{\_in\_multi}$ is set to 0.4 and $Fc_{\_out\_multi}$ is set to 0.25, 2663 multi-bit track segments are required to implement a circuit. When $Fc_{\_in\_multi}$ is increased to 1.0 and $Fc_{\_out\_multi}$ is increased to 0.8, only 1985 multi-bit track segments are required. Note that this reduction is largely due to the more efficient use of the multi-bit tracks by the multi-bit signals since the number of single-bit track segments stays largely the same over all values of $Fc_{\_in\_multi}$ and $Fc_{\_out\_multi}$.

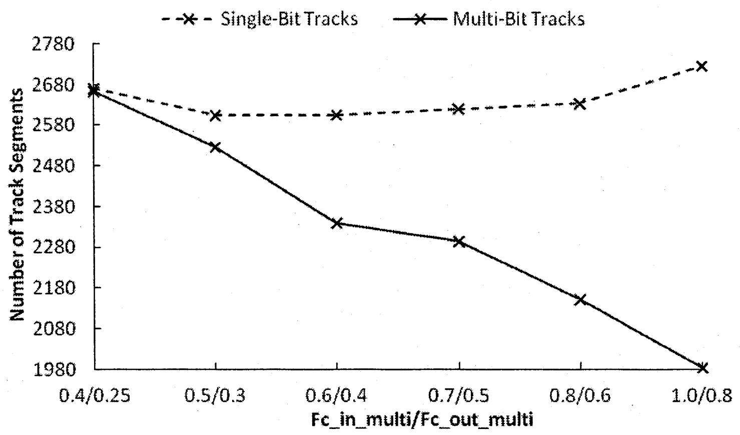


Fig. 6.1 Number of Single-Bit and Multi-Bit Routing Tracks in the Configuration Memory Sharing Routing Architecture

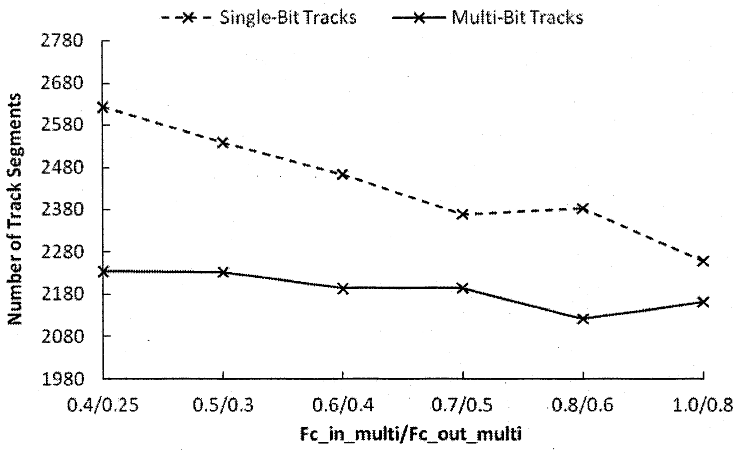


Fig. 6.2 Number of Single-Bit and Multi-Bit Routing Tracks in the Sparse architecture

Fig. 6.2 shows that when configuration memory sharing is removed from the multi-bit tracks, there is a substantial reduction in the number of multi-bit track segments for small values of $Fc_{\_in\_multi}$ and $Fc_{\_out\_multi}$. This reduction is due to the increase in multi-bit track flexibility. As the values of $Fc_{\_in\_multi}$ and $Fc_{\_out\_multi}$ increase, the utilization of the multi-bit tracks increases as well. This increase, however, is due to the more efficient use of multi-bit tracks by multi-bit signals as well as single-bit signals. Consequently, the total number of single-bit track segments decreases with increasing values of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$.

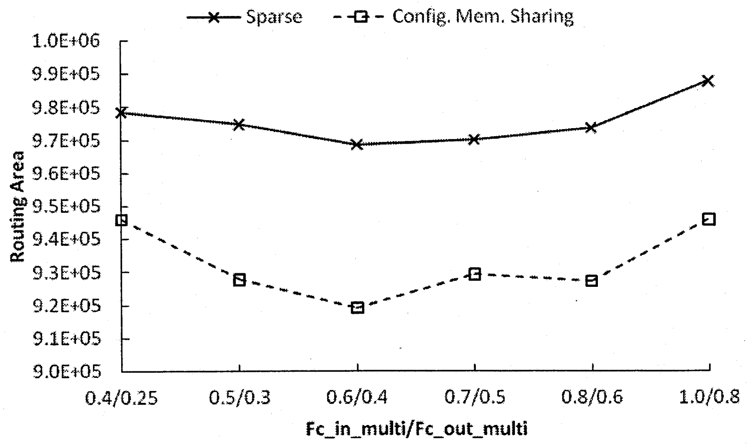


Fig. 6.3 Routing Area for Sparse and Configuration Memory Sharing Routing Architectures

Fig. 6.3 shows the effect of multi-bit track utilization on routing area. In the figure, area is the minimum average routing area across 15 benchmarks. The curve above is for the sparse architecture while the curve below is for the configuration memory sharing routing architecture. As shown, the best sparse architecture consumes 5% more routing area than the best configuration memory sharing architecture.

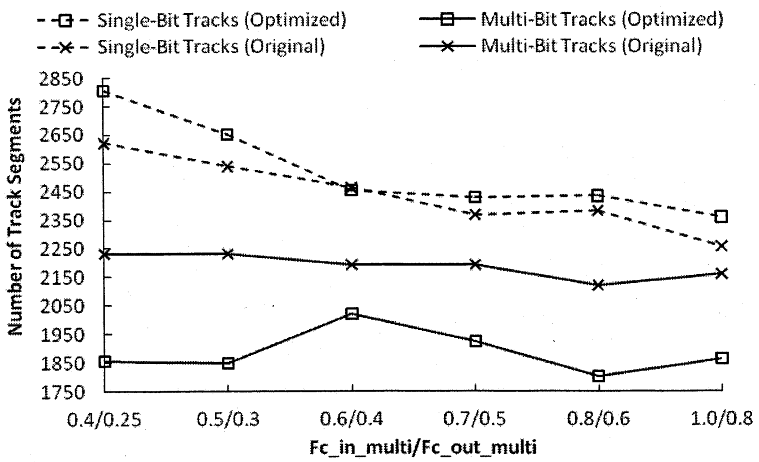


Fig. 6.4 Multi-Bit Track Reduction Due to Routing Algorithm Optimization

In Fig. 6.3, the same multi-bit aware routing algorithm [29] is used for both the configuration memory sharing and the sparse architectures. The algorithm, however, is specialized for configuration memory sharing. Since it is very expensive to route a single bit signal on a set of configuration memory sharing routing resources, this algorithm encourages multi-bit signals to use multi-bit tracks as much as possible. It heavily penalizes the act of breaking a multi-bit signal into individual bits and routing some of the bits on single-bit tracks and the remaining bits on multi-bit tracks [27][29]. Without configuration memory sharing, however, the penalty still forces the single-bit and multi-bit signals to stay on their respective routing resources even when one type of resource is much more congested than the other.

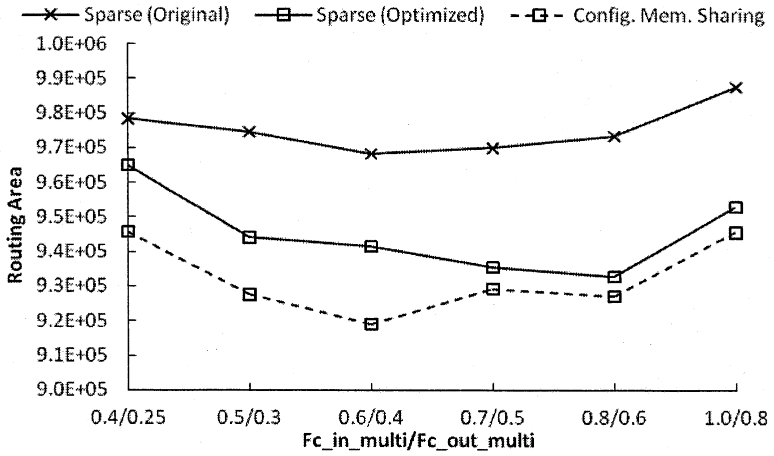


Fig. 6.5 Reduction in Routing Area for the Sparse Architecture Due to Routing Algorithm Optimization

We modified the routing algorithm for the sparse architecture by removing the penalty. With the penalty removed, more multi-bit signals can be routed on single-bit tracks. This results in a slight increase in the number of single-bit track segments and a significant reduction in the number of multi-bit track segments as shown in Fig. 6.4. Consequently, the routing area of the sparse architecture is further reduced as shown in Fig. 6.5.

To determine the best proportion of multi-bit routing tracks for the configuration memory sharing and the sparse architectures, we repeat the above experiment by fixing $F_c s$ ($F_{c\_in\_multi}$ and $F_{c\_out\_multi}$) while varying the proportion of multi-bit tracks. For the configuration memory sharing architecture, we set $F_{c\_in\_multi}$ to 0.6 and $F_{c\_out\_multi}$ to 0.4 because these values are found the most area efficient in Fig. 6.5. For the sparse architecture, however, we set $F_c s$ to the second most area efficient values ($F_{c\_in\_multi} = 0.7$

and $F_{c\_out\_multi}$ = 0.5). When we set $F_{c\_in\_multi}$ = 0.8 and $F_{c\_out\_multi}$ = 0.6 in the sparse architecture, the switches on multi-bit tracks are dense enough that even changing the number of multi-bit tracks by a single bus of 4 tracks can significantly impact the number of single-bit tracks required to the point that the percentage of multi-bit tracks can change by more than 10%. For this reason, we were not able to collect data points in certain percentage ranges for several of the benchmarks in our suite. For example, we were not able to generate any area results for the code_reg circuit with between 30%-40% or 50%-60% multi-bit tracks. For the results in this section, therefore, we chose to set $F_c s$ to the values that are slightly less (0.3%) area efficient but allowed us to collect a more complete set of data points for all the 10 percentile ranges of multi-bit tracks we examined.

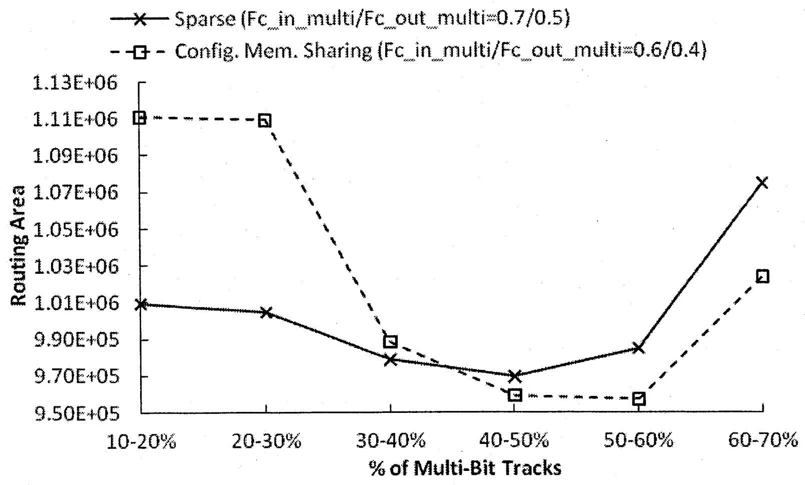


Fig. 6.6 Routing Area vs. % of Multi-Bit Tracks

The routing area results as a function of the percentage of multi-bit tracks for the configuration memory sharing and sparse architectures are shown in Fig. 6.6. As shown, the best number of multi-bit tracks as a percentage of the total number of tracks is between 50 to 60% for the configuration memory sharing architecture and 40 to 50% for the sparse architecture. The best sparse architecture consumes 1.3% more routing area than the configuration memory sharing routing architecture. Fig. 6.7 shows the average number of track segments as a function of the percentage of multi-bit tracks. As shown the most area efficient sparse architecture uses 16% fewer track segments than the most area efficient configuration memory sharing routing architecture.

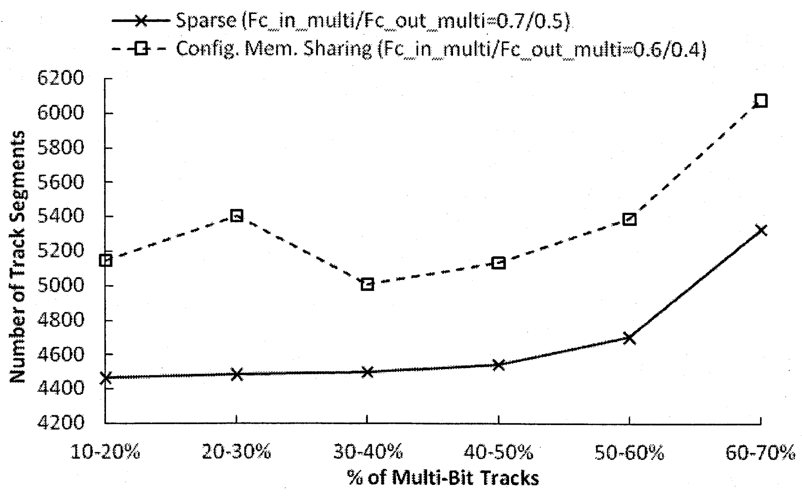


Fig. 6.7 Track Segments vs. % of Multi-Bit Tracks

6.2. Shift and Control Enhanced Sparse Architectures Vs. Configuration Memory Sharing Architecture

Table 6.3 Parameter Settings for the Enhanced Sparse and the Configuration Memory Sharing Routing Architectures

Classification	Parameter	Minimum Value	Maximum Value
Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	0.4	N.A.
	$F_{c\_out\_single}$	0.25	N.A.
	W_{single}	best	N.A.
Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	0.6 (Configuration Memory Sharing)/ 0.7 (Enhanced Sparse)	N.A.
	$F_{c\_out\_multi}$	0.4 (Configuration Memory Sharing)/ 0.5 (Enhanced Sparse)	N.A.
	W_{multi}	4	64
	F_{c_enh}	0	1.0
Common Parameters	L	2	N.A.
	Routing Tool	Multi-Bit Aware [29] (Optimized)	N.A.

In this section, we evaluate the effect of the shift and control enhanced sparse architectures on the area efficiency of FPGAs. The parameter settings used in this investigation are shown in Table 6.3. In particular, the same parameter values from Section 6.1 are used for the single-bit tracks. For multi-bit tracks, the $F_{c\_in\_multi}$ and

$Fc_{\_out\_multi}$ values are set to 0.6 and 0.4 for the configuration memory sharing architecture and 0.7 and 0.5 for the enhanced sparse architecture, as in Section 6.1. For each set of $Fc_{\_in\_multi}$ and $Fc_{\_out\_multi}$ values, we vary the $Fc_{\_enh}$ value from 0 to 1.0. For each value of $Fc_{\_enh}$, we vary the number of multi-bit tracks from 4 to 64 to find the minimum number of single-bit tracks that is required to route a circuit.

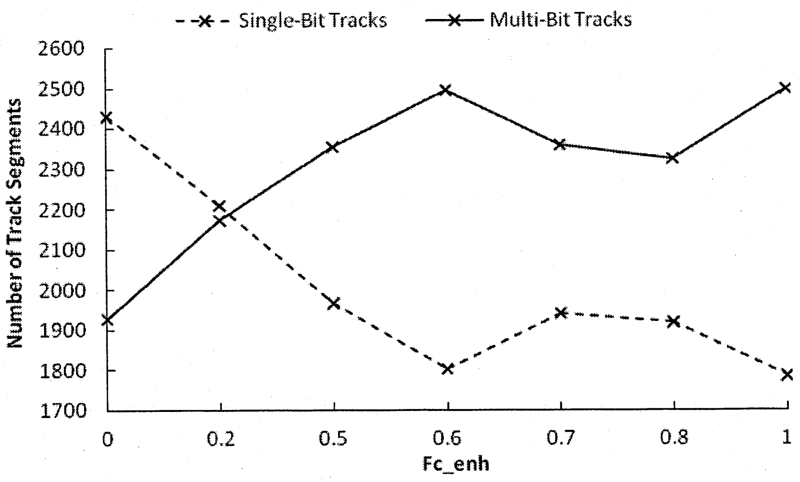


Fig. 6.8 Number of Single-Bit and Multi-Bit Routing Tracks in the Control-Enhanced Sparse Architecture

For the control enhanced sparse architecture, Fig. 6.8 shows the effect of $Fc_{\_enh}$ on the average number of routing track segments that is required support a circuit. As shown, as the value of $Fc_{\_enh}$ increases from 0 to 0.6, more and more single-bit signals are routed on the multi-bit tracks due to higher routing flexibility on multi-bit tracks. Consequently the number of multi-bit tracks increases and the number of single-bit tracks decreases. As $Fc_{\_enh}$ increases beyond 0.6, the number of multi-bit and single-bit tracks stabilizes as most of the signals that can be efficiently routed through the multi-bit tracks have already been moved onto these tracks.

Fig. 6.9 shows the effect of control and shift enhanced sparse architecture on routing area. As shown, both the control and shift enhanced sparse architectures are more efficient than the configuration memory sharing architecture over all percentage values of multi-bit tracks. In particular, the best control enhanced sparse architecture (at 60%-70% multi-bit tracks) is 2.6% smaller than the best configuration memory sharing architecture (at 50%-60% multi-bit tracks) and 1.4% smaller than the best shift enhanced sparse architecture (at 50%-60% multi-bit tracks). Fig. 6.10 shows the best control enhanced sparse architecture requires 18% fewer routing tracks than the best configuration memory sharing architecture.

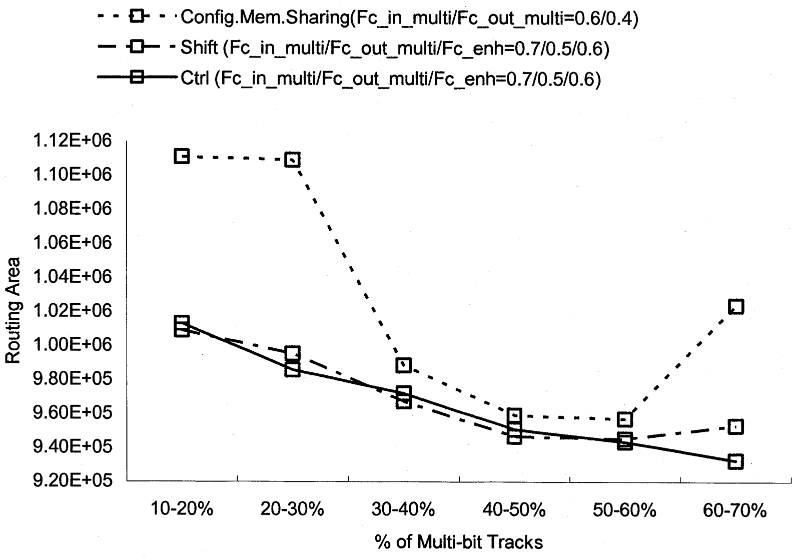


Fig. 6.9 Routing Area vs. % of Multi-Bit Tracks

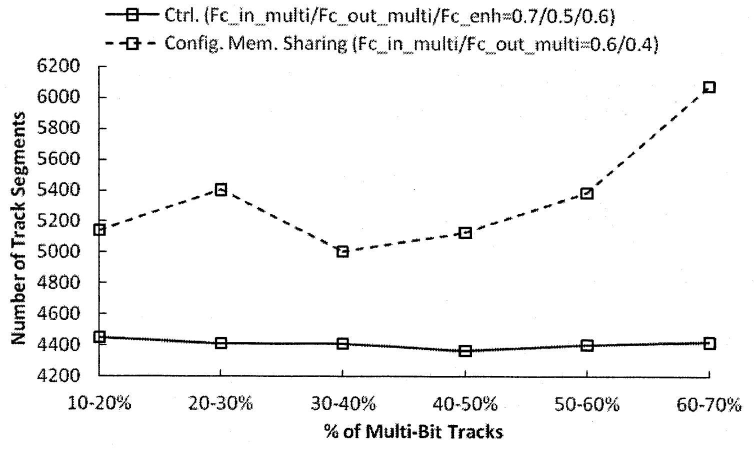


Fig. 6.10 Track Segments vs. % of Multi-Bit Tracks

Table 6.4 summarizes the best implementation area for each benchmark circuit for the configuration memory sharing and the control enhanced sparse architectures. These circuits are divided into four groups according to the percentage of multi-bit signals that they contain. As shown, for circuits with less than 60% multi-bit signals, the enhanced sparse architecture are, on average, more area efficient than the configuration memory sharing architecture. In particular, the enhanced sparse architecture consumes up to

18.9% less routing area than the configuration memory sharing architecture. For circuits with more than 60% multi-bit signals, the configuration memory sharing architecture is more area efficient. Compared to the configuration memory sharing architecture, the enhanced sparse architecture at worst consumes 9.8% more routing area for the ucode_reg benchmark circuit. This benchmark circuit, however, is very small in area among all the benchmark circuits, which means it has little influence on the average routing area. The enhanced sparse architecture, on average, only consumes 1.9% more routing area than the configuration memory sharing architecture for circuits that contain over 60% multi-bit signals.

Table 6.4 Routing Area Vs. % of Multi-Bit Signals Per Circuit

Percentage Range	Benchmark (% of Multi-Bit Signals)	Routing Area (1E+5) Config. Mem. Sharing	Routing Area (1E+5) Enhanced Sparse ($F_{c\ enh}$=0.6)	Area Increase Over Config.Mem.Sharing
30-40%	multmod_dp (32%)	18.4	14.9	-18.9%
	Average	**18.4**	**14.9**	**-18.9%**
40-50%	prils_dp (42%)	2.87	2.64	-7.8%
	code_seq_dp (46%)	2.81	2.71	-3.5%
	exponent_dp (47%)	5.19	4.91	-5.4%
	Incmod (48%)	7.86	6.86	-12.7%
	Average	**4.68**	**4.28**	**-8.5%**
50-60%	smu_dpath (51%)	4.29	3.89	-9.3%
	imdr_dpath (53%)	11.4	11.6	1.6%
	pipe_dpath (56%)	2.72	2.63	-3.1%
	mantissa_dp (56%)	10.4	10.6	2.5%
	Average	**7.19**	**7.18**	**-0.2%**
Over 60%	icu_dpath (61%)	32.2	31.1	-3.5%
	ucode_dat (61%)	9.36	9.86	5.3%
	rsadd_dp (61%)	2.02	1.95	-3.2%
	ex_dpath (61%)	26.2	28.0	6.8%
	dcu_dpath (65%)	7.33	7.64	4.3%
	ucode_reg (74%)	0.58	0.64	9.8%
	Average	**12.9**	**13.2**	**1.9%**

6.3. Multi-Bit Aware Vs. Conventional architectures

Having compared the area efficiency of the multi-bit aware routing architectures, this section compares two of the most area efficient multi-bit aware routing architectures – the sparse and the control enhanced sparse architectures – to the area efficiency of the conventional architecture. Table 6.5 shows the parameter settings used in this investigation. In particular, the conventional routing tool [31][32] is used since none of the architectures employs configuration memory sharing and our results show the conventional routing tool outperforms the multi-bit aware tool [29] for these architectures.

Table 6.5 Parameter Settings for the Conventional, Sparse, and Control-Enhanced Sparse Architectures

Classification	Parameter	Minimum Value	Maximum Value
Parameters for Conventional Architecture	F_{c_in}	0.3	0.8
	F_{c_out}	0.25	N.A.
	W	Best	N.A.
Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	0.4	N.A.
	$F_{c\_out\_single}$	0.25	N.A.
	W_{single}	Best	N.A.
Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	0.4	1.0
	$F_{c\_out\_multi}$	0.25	0.8
	W_{multi}	4	64
	F_{c_enh}	0	1.0
Common Parameters	L	2	N.A.
	Routing Tool	Conventional [31][32]	N.A.

For the conventional architecture, the F_{c_out} value is set to 0.25. It is shown to be the most area efficient in previous studies [33]. The F_{c_in} value is varied from 0.3 to 0.8 to measure the minimum routing area that is required to implement the benchmarks. Fig. 6.11 shows that 0.4 is the most area efficient F_{c_in} value for the conventional architecture.

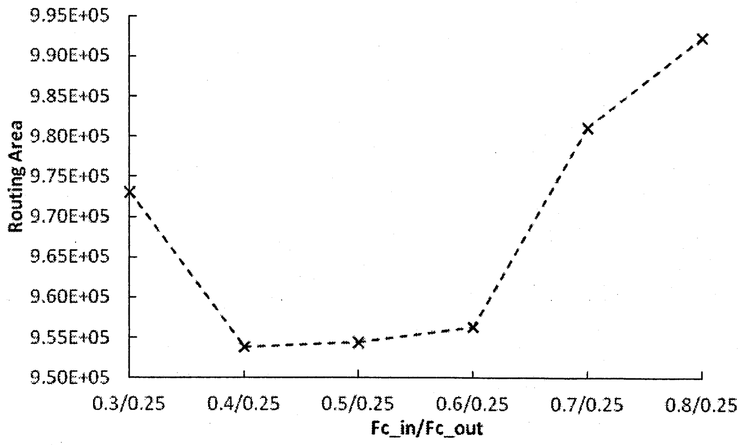


Fig. 6.11 Most Area Efficient F_{c_in} Values for the Conventional architecture

For the sparse and enhanced sparse architectures, we vary $F_{c\_in\_multi}$, $F_{c\_out\_multi}$, W_{multi}, and F_{c_enh} as in Section 6.1 and Section 6.2. We found that for the conventional routing tool, the most area efficient parameter values are $F_{c\_in\_multi} = 0.6$ and $F_{c\_out\_multi} = 0.4$ for the sparse architecture and $F_{c\_in\_multi} = 0.6$, $F_{c\_out\_multi} = 0.4$, and $F_{c\_enh} = 0.8$ for the control enhanced sparse architecture.

Fig. 6.12 shows the routing area as a function of the percentage of multi-bit tracks for the three architectures. As shown, the control enhanced sparse architecture is the most area efficient. At 70%-80% multi-bit tracks, it consumes 10% less area than the conventional architecture. The figure also shows that the sparse architecture also consumes less area than the conventional architecture. At 50%-60% multi-bit tracks, the sparse architecture is 5.8% smaller than the conventional architecture.

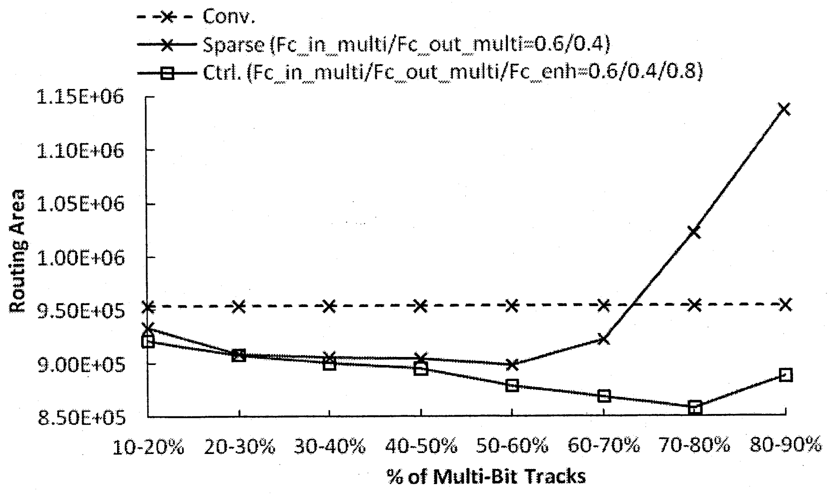


Fig. 6.12 Routing Area Vs. % of Multi-Bit Tracks

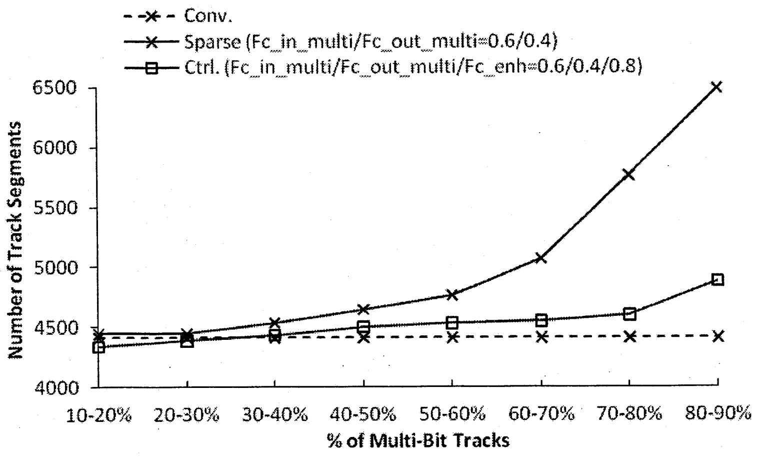


Fig. 6.13 Track Segments Vs. % of Multi-Bit Tracks

Finally, Fig. 6.13 shows the number of track segments per circuit as a function of the percentage of multi-bit tracks. As shown, both the sparse and the enhanced sparse architectures consume more track segments than the conventional architecture – with the best sparse architecture employing 8% more track segments and the best control enhanced sparse architecture employing 4% more track segments.

Table 6.6 summarizes the best implementation area for each benchmark circuit for the conventional, sparse and control enhanced sparse architectures. These circuits are divided into four groups according to the percentage of multi-bit signals in these circuits. As shown, for circuits with more than 50% multi-bit signals, the sparse and the control enhanced sparse architectures are, on average, significantly more area efficient than the conventional architecture. In particular, the sparse and the control enhanced sparse architectures consume 13.1% and 14.1% less routing area, respectively, than the conventional architecture when averaging across benchmark circuits that contain over 60% multi-bit signals. For the circuit with less than 40% multi-bit signals, the conventional architecture is significantly more area efficient (28.5%) than the sparse architecture but only slightly more area efficient (3.8%) than the enhanced sparse architecture.

Table 6.6 Routing Area Vs. % of Multi-Bit Signals Per Circuit

Percentage Range	Benchmark (% of Multi-bit Signals)	Routing Area (1E+5)			Area Increase over Conventional	
		Conventional	Sparse	Enhanced Sparse	Sparse	Enhanced Sparse
30-40%	multmod_dp (32%)	13.8	17.8	14.4	28.5%	3.8%
	Average	**13.8**	**17.8**	**14.4**	**28.5%**	**3.8%**
40-50%	prils_dp (42%)	2.43	2.58	2.42	6.2%	-0.4%
	Code_seq_dp (46%)	2.57	2.84	2.70	10.5%	5.2%
	exponent_dp (47%)	4.53	4.64	4.39	2.3%	-3.2%
	Incmod (48%)	6.86	6.35	6.40	-7.4%	-6.7%
	Average	**4.10**	**4.10**	**3.98**	**0.1%**	**-2.9%**
50-60%	Smu_dpath (51%)	3.89	4.00	3.97	2.7%	2.0%
	Imdr_dpath (53%)	11.6	10.7	10.2	-7.6%	-12.4%
	Pipe_dpath (56%)	2.65	2.57	2.44	-2.8%	-8.0%
	mantissa_dp (56%)	10.1	9.84	9.17	-2.8%	-9.5%
	Average	**7.07**	**6.79**	**6.44**	**-4.0%**	**-9.0%**
Over 60%	Icu_dpath (61%)	33.3	28.3	28.1	-15.0%	-15.5%
	ucode_dat (61%)	10.7	8.92	9.05	-16.3%	-15.1%
	rsadd_dp (61%)	1.91	1.92	1.77	0.2%	-7.7%
	Ex_dpath (61%)	30.0	26.8	26.2	-10.8%	-12.5%
	Dcu_dpath (65%)	8.16	7.12	6.92	-12.8%	-15.2%
	ucode_reg (74%)	0.59	0.52	0.56	-11.9%	-4.6%
	Average	**14.1**	**12.3**	**12.1**	**-13.1%**	**-14.1%**

6.4. Performance Comparison.

Our research focus is on FPGA routing area efficiency, but it is still important to characterize how performance is affected by the differences in the conventional, configuration memory sharing, sparse and enhanced sparse routing architectures. We want to know if the area efficiency of the sparse and enhanced sparse architectures is gained at the cost of performance degradation. In this section, the performance of an architecture is measured by the geometric mean of the routing delays in the 15 benchmark circuits routed on the architecture.

Table 6.7 Routing Delays of 15 Benchmarks Routed on the 4 Best Architectures

Benchmarks	Routing Delays (seconds)			
	Conventional	Config.Mem. Sharing	Sparse	Enhanced
code_seq_dp	9.413E-09	8.009E-09	8.711E-09	9.413E-09
dcu_dpath	3.797E-09	3.797E-09	3.797E-09	3.797E-09
ex_dpath	2.767E-08	2.767E-08	2.556E-08	2.626E-08
exponent_dp	1.558E-08	1.558E-08	1.558E-08	1.558E-08
icu_dpath	2.275E-08	2.275E-08	2.275E-08	2.275E-08
imdr_dpath	2.907E-08	2.907E-08	2.907E-08	2.907E-08
incmod	2.837E-08	2.767E-08	2.767E-08	2.837E-08
mantissa_dp	4.348E-09	4.348E-09	4.348E-09	4.348E-09
multmod_dp	2.345E-08	2.135E-08	2.345E-08	1.347E-08
pipe_dpath	1.067E-08	1.067E-08	1.067E-08	1.067E-08
prils_dp	1.152E-08	1.082E-08	1.152E-08	1.012E-08
rsadd_dp	2.626E-08	2.556E-08	2.626E-08	2.556E-08
smu_dpath	2.556E-08	2.556E-08	2.556E-08	2.556E-08
ucode_dat	3.797E-09	3.797E-09	3.095E-09	2.944E-09
ucode_reg	2.242E-09	2.242E-09	2.242E-09	2.242E-09
Geometric Mean	**1.224E-08**	**1.194E-08**	**1.193E-08**	**1.144E-08**

Table 6.7 shows the routing delays of the 15 benchmarks routed on the most area efficient conventional, configuration memory sharing, sparse and enhanced sparse architectures. The geometric means of the routing delays across the 15 benchmarks for the 4 architectures are listed at the bottom of the table. The geometric means of the routing delays show that the most area efficient sparse and enhanced sparse architectures have slightly better performance than the most area efficient conventional and configuration memory sharing architectures.

CHAPTER 7

CONCLUSION

7.1. Thesis Summary

When averaging across all the benchmarks, we found that the sparse and the enhanced sparse architectures are as area efficient as the configuration memory sharing architecture. In particular, the sparse architecture only consumes 1.3% more routing area than the configuration memory sharing architecture while the enhanced sparse architecture consumes 2.6% less. We also found that the sparse architecture is 5.8% more area efficient than the conventional architecture while the enhanced sparse architecture is 10% more area efficient. Furthermore, the greater routing flexibility allows the sparse and the enhanced sparse architectures to use 16-18% fewer routing tracks than the configuration memory sharing architecture. Even with significantly sparser routing switches, the sparse and enhanced sparse architectures use only 4-8% more routing tracks than the conventional architecture. Moreover, our experimental results show that the performance of the sparse and the enhanced sparse architectures are as good as that of the conventional and the configuration memory sharing architectures.

We discovered another important result when we grouped the individual benchmark area results into buckets where the percentage of multi-bit signals in benchmark circuits falls within a 10% range. We found that the enhanced sparse architecture is more area efficient (up to 18.9% better) than the configuration memory sharing architecture when the percentage of multi-bit signals is low (30%-40%). But even when the percentage of multi-bit signals is over 60%, where we would expect the configuration memory sharing routing architecture to have its best result, the enhanced sparse architecture is, on average, only 1.9% worse. Compared to the conventional architecture, the enhanced sparse architecture gives the highest area improvement (14.1%) for circuits with over 60% multi-bit signals, but is only 3.8% worse than the conventional architecture for circuits with between 30%-40% multi-bit signals. These results suggest that the enhanced sparse architecture is, on average, more area efficient across a broad range of circuits than either the conventional or configuration memory sharing routing architectures, which each target circuits with few or many multi-bit signals, respectively, but are much less efficient on the opposite class of circuits.

7.2. Suggestions For Future Work

Following the work in the thesis, several projects can be taken up:

1. Improve the multi-bit aware routing tool and reexamine the configuration memory sharing architecture

The experimental results in this thesis show that the conventional, sparse and enhanced sparse architectures are more area efficient with the conventional routing tool than the multi-bit aware routing tool. As described in Chapter 5, the multi-bit aware routing tool can be improved for the multi-bit aware routing architectures and a direction to improve the tool is introduced in the same chapter.

This thesis also shows that routing results for the configuration memory sharing architecture are worse than those for the conventional architecture. The efficiency difference between these two architectures suggested by the routing results, however, can be from the efficiency difference between the conventional and the multi-bit aware routing tools. For this reason, we think it is not fair to conclude that the configuration memory sharing architecture is less efficient than the conventional architecture in this thesis. We suggest to reexamine the configuration memory sharing architecture after the multi-bit aware routing tool is improved.

2. Search for more efficient enhanced sparse architectures

The enhanced sparse architectures we investigated show very good area efficiency for data-path oriented applications. However, the design space we explored in this research is only a small portion of the total design space for the enhanced sparse architectures. For future work, we suggest to search for: 1) new patterns to add on the sparse switch pattern; 2) new algorithm to spread added patterns onto routing buses. We deem our work in this thesis as a step stone to research for new switch patterns in FPGA routing architectures. From this work, we see a great opportunity to enhance the area efficiency of FPGAs through routing architecture research.

REFERENCES

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

[2] Stratix IV Device Family Overview, July 2008, http://www.altera.com/literature/ lit-stratix-iv.jsp.

[3] Virtex-5 Family Overview, Sep., 2008, http://www.xilinx.com/support/documentation/ virtex-5.htm.

[4] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs,* Kluwer Academic Publishers, February 1999. ISBN 0-7923-8460-1.

[5] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff and J. Rose, "The Stratix II Logic and Routing Architecture," *ACM/Sigda International Symposium on Field Programmable Gate Arrays*, 2005, pp. 14–20.

[6] R. Tessier, V. Betz, D. Neto, A. Egier and T. Gopalsamy, "Power-efficient RAM Mapping Algorithms for FPGA Embedded Memory Blocks," *IEEE Trans. on Computer-Aided Design of Circuits and Systems,* Feb. 2007, pp. 278–290.

[7] J. H. Anderson and F. N. Najm, "Low-power programmable routing circuitry for FPGAs," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* , 2004, pp. 602-609.

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

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

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

[11] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1998, pp. 389–397.

[12] R. Bittner, M. Musgrove, and P. Athanas, "Colt: An Experiment in Wormhole RunTime Reconfiguration," *High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic. SPIE*, November 1996, pp. 187–194.

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

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

[15] C. Ebeling, D. Cronquist, P. Franklin, and C. Fisher, "RaPiD – A configurable computing architecture for compute-intensive applications," *International Workshop on Field Programmable Logic and Applications*, 1997, pp. 126–135.

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

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

[18] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," *Proceedings of the IEEE Symposium of Field-Programmable Custom Computing Machines*, April 1997, pp. 24–33.

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

[20] D. Cherepacha and D. Lewis, "DP-FPGA: An FPGA Architecture Optimized for Datapaths," *VLSI Design*, 1996, pp. 329–343.

[21] D. Cherepacha, "A Field Programmable Gate Array Architecture Optimized for Datapaths," M.A.Sc. Thesis, University of Toronto, 1994.

[22] A. Ye and J. Rose, "Using Bus-Based Connections to Improve Field-Programmable Gate Array Density for Implementing Datapath Circuits," *IEEE Transations on Very Large Scale Integration(VLSI),* Vol. 14, No. 5, May 2006, pp. 462–473.

[23] G. G. Lemieux and S. D. Brown, "A detailed router for allocating wire segments in field programmable gate arrays," *Proceedings of the ACM Physical Design Workshop*, April 1993, pp. 215–226.

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

[25] Steven J. E. Wilton, "Architecture and Algorithms for Field Programmable Gate Arrays with Embedded Memory," Phd thesis, University of Toronto, 1997.

[26] Herman Schmit, Vikas Chandra. "FPGA Switch Block Layout and Evaluation," *IEEE/ACM International Symposium on Field Programmable Gate Arrays*, February 2002, pp. 11–18.

[27] A. Ye, "Field-Programmable Gate Array Architectures and Algorithms Optimized for Implementing Datapath Circuits," Ph.D. Thesis, University of Toronto, Nov. 2004

[28] V. Betz, *VPR and T-VPack User's Manual (Version 4.30)*, March 27, 2007

[29] A. Ye and J. Rose, "Measuring and Utilising the Correlation Between Signal Connectivity and Signal Positioning for FPGAs Containing Multi-Bit Building Blocks," *IEECDT,* Vol. 153, No. 3, May 2006, pp. 146–156.

[30] Pico-Java Processor Design Documentation, Sun Microsystems, 1999

[31] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *FPL,* 1997, pp. 213–222.

[32] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and Routing Tools for the Triptych FPGA," *TVLSI*, Vol. 3, No. 4, December 1995, pp. 473–482.

[33] V. Betz and J. Rose, "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density," *FPGA*, 1999, pp. 59–68.

(4)
BL-78-22