Ph.D. Thesis Proposal: Routing Architecture and Place and Route Tools for DP-FPGA

Andy G. Ye

June 2, 2000

1 Introduction

Field Programmable Gate Arrays (FPGAs) consist of field programmable logic cells connected by field programmable routing resources. Logic cells typically consist of SRAM based look-up tables (LUTs), but they also can be based on other technologies like multiplexers. In this study, we will focus on LUT based FPGAs. FPGA routing resources typically are wire segments connected by SRAM controlled pass transistors. A typical structure of an FPGA is shown in Figure 1.

A majority of current commercial FPGAs are general purpose in nature. Their logic cells and routing resources are largely optimized towards implementing random logic circuits. However, as the logic capacity of FPGAs increases, they are increasingly used to implement data-path circuits which process multiple-bit data. To efficiently process multiple-bit data, data-path circuits are often designed as multiple similarly structured bit slices. FPGA architectures can be designed to take advantage of this similarity in order to increase the logic density and speed of data-path applications. DP-FPGA [9] is such an FPGA architecture.

This research will further develop the DP-FPGA architecture first proposed by Don Cherepacha and David Lewis in 1996. DP-FPGA is a novel FPGA architecture optimized for data-path applications. It uses a novel technique called *programming bit sharing* to increase the logic density of data-path oriented designs. In their study, Cherepacha and Lewis proposed several logic block architectures. Based on their logic block designs, routing

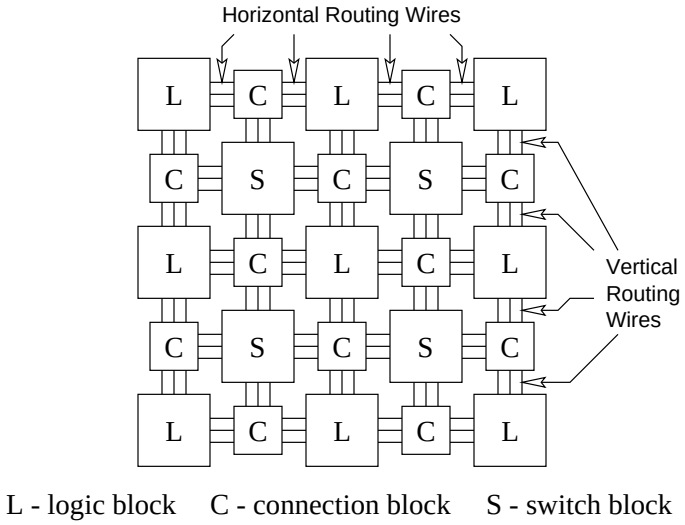


Figure 1: A Simple FPGA Architecture

area for DP-FPGA was estimated. The authors found that DP-FPGA can achieve over twice the logic density of conventional FPGA architectures for data-path oriented applications.

The study of Cherepacha and Lewis mainly concentrated on logic block design. The routing architecture for DP-FPGA remains an open question. Consequently, little work has been done on the place and route tools for DP-FPGA. Our proposed research will concentrate on the routing architecture and the place and route tools for DP-FPGA. In modern FPGAs, logic cells typically consume less than 10% of total chip area. The rest is consumed by routing resources. Routing also is the major cause of circuit delay. Due to the importance of routing in both logic density and speed of FPGAs, this proposed research will play an essential role in the development of DP-FPGA.

2 Background

Several background topics are reviewed in this section. Section one reviews the previous DP-FPGA research by Cherepacha and Lewis. Section two reviews several general purpose FPGA routing architectures. Section three reviews the FPGA CAD flow.

2.1 DP-FPGA

DP-FPGA, as proposed by Cherepacha and Lewis, consists of three major blocks, which are shown in Figure 2. The memory block consists of banks of SRAM. It can be configured to emulate memory systems of different width and depth. The control block is structurally similar to general purpose FPGAs. It is designed to emulate control circuits which are typically random logics and can be efficiently emulated by general purpose FPGAs. The only difference between the architecture of this block and a general purpose FPGA is the lack of arithmetic support. Since a majority of arithmetic computation is contained in the data-path portion of a circuit, there is no need for the control block to contain structures like carry chains. The detailed architectures of the above two blocks and their interconnection with the data-path block was not investigated in detail in the Cherepacha and Lewis study.

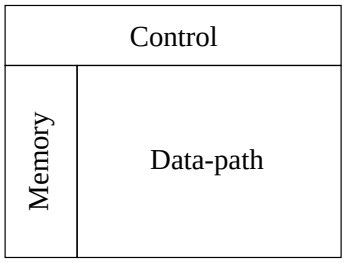
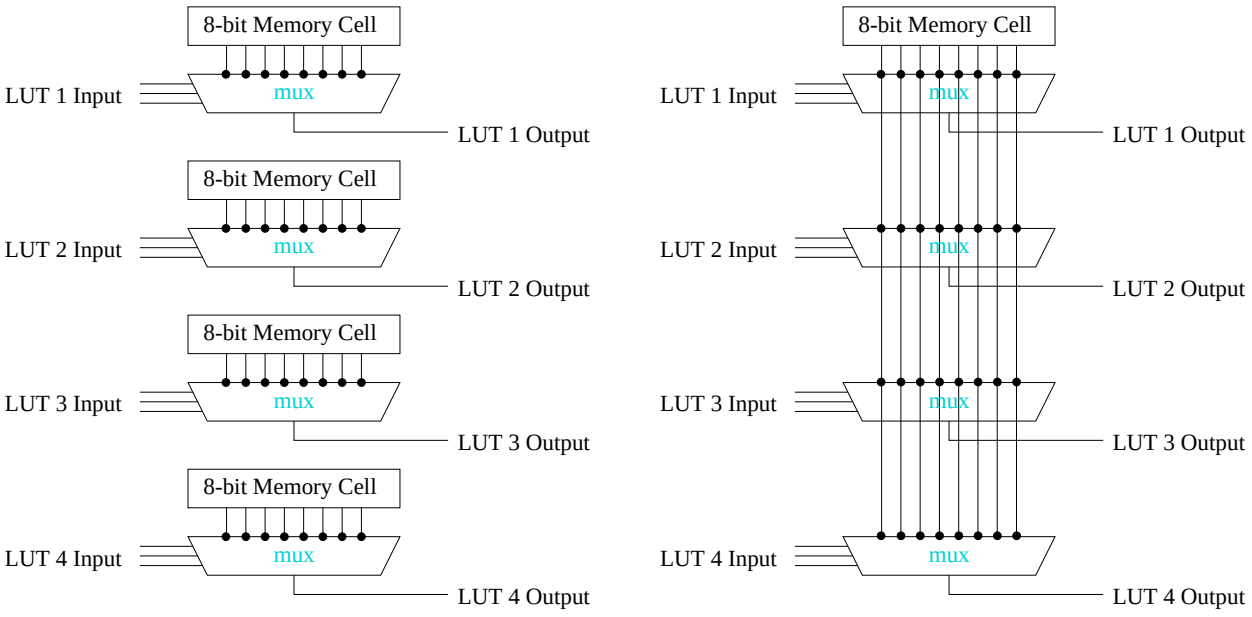


Figure 2: DP-FPGA Architecture Overview

The third block is the data-path block. Like general purpose FPGAs, it also consists of LUTs and programmable routing structures. Its architecture, however, is optimized for circuits composed of multiple copies of similar, or even identical, bit slices. The optimization technique employed is called *programming bit sharing* by Cherepacha and Lewis. The idea is simple but effective. It is illustrated in Figure 3b for applying to a logic block and Figure 4 for applying to a connection block.

A look-up table of a regular, single context, FPGA consists of a bank of SRAM cells and a multiplexer. The depth of the SRAM cell is one and it should be as wide as the data input of the multiplexer. The output bits of the SRAM are directly connected to the data input pins of the multiplexer. The inputs to the LUT control the select signals of the multiplexer. Based on the LUT inputs, an SRAM output is selected by the multiplexer and sent to the



(a) 4 Regular LUTs

(b) 4 Programming Bit Sharing LUTs

Figure 3: Programming Bit Sharing in DP-FPGA Logic Blocks

LUT output. Four regular LUTs are shown in Figure 3a.

The SRAM cells of DP-FPGA, however, are connected to the data input pins of several multiplexers. For example, four multiplexers are controlled by a single bank of SRAM cells in Figure 3b. This structure, in effect, creates several identically programmed look-up tables. Since data-path circuits often contain identical bit slices, this scheme increases the logic density of FPGAs.

The programming bit sharing idea also can be applied to connection blocks. The connection block structure proposed by Cherepacha and Lewis is illustrated in Figure 4. In this architecture, the output bit of each programming bit sharing LUT is connected to a unique routing channel. All connection switch sets connecting each LUT output to its routing channel are identically controlled by one bank of SRAM. Again, this architecture takes advantage of the identical bit slices in data-path circuits to save silicon area.

Cherepacha and Lewis also designed a 4-bit carry look ahead carry chain for the logic cell and a shift block for routing. The logic block architecture was extensively studied. The detailed routing architecture was not investigated.

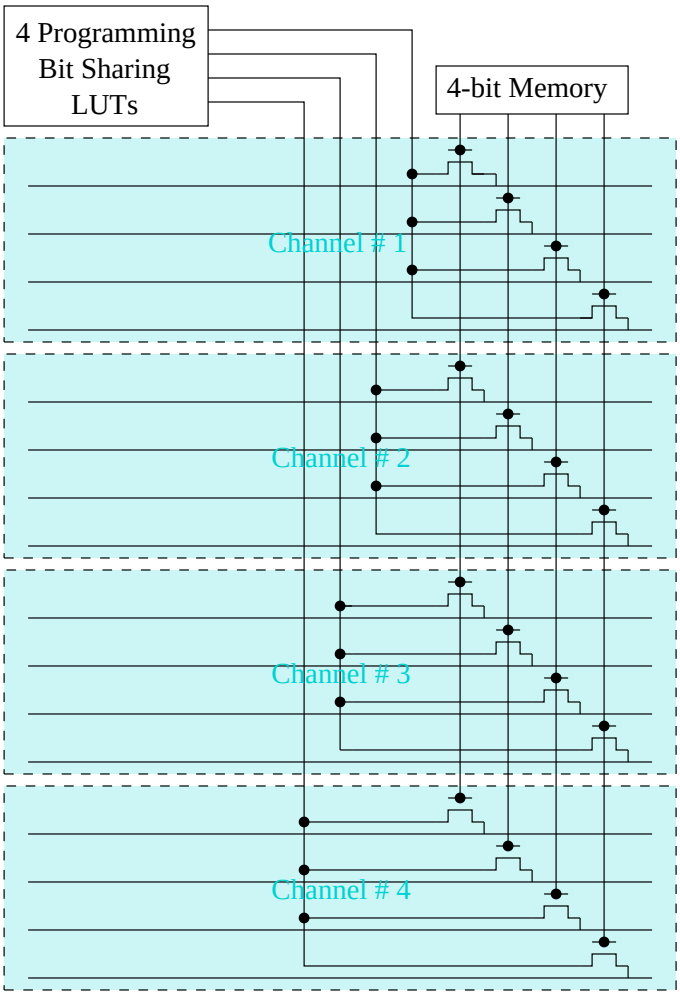


Figure 4: Programming Bit Sharing in DP-FPGA Connection Blocks

2.2 FPGA Routing Architectures

This sub-section reviews several FPGA routing architectures. The generic structure of an FPGA is shown in Figure 1. The logic block, L, typically consists of several LUTs. The simplest FPGA logic blocks contain only one LUT. Typically 3, 4, or 5-input LUTs are used. The connection block, C, typically consists of SRAM controlled pass transistors or tri-state buffers. These pass transistors and buffers act as programmable switches providing connectivity from logic block pins to routing wires. The switch block, S, also consists of similar programmable switches. These switches provide connectivity between vertical and horizontal routing wires.

Various techniques have been developed over the years to improve the density and performance of FPGAs. Many are effective and widely used in commercial FPGAs. Three of these techniques are reviewed in Section 2.2.1 to Section 2.2.3.

2.2.1 Long Wire Segments

In the simplest FPGAs, routing wires are wire segments that only span the width or height of one logic block. These segments are connected together using pass transistors in switch blocks to create signal paths.

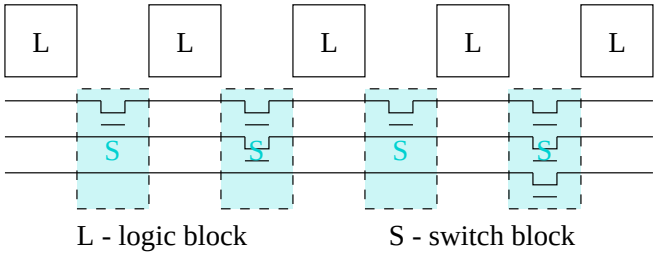


Figure 5: Routing Architecture Utilizing Long Wires

Modern FPGAs often use longer wire segments to reduce the excessive delay caused by pass transistors in long signal paths. A typical routing architecture containing long wire segments is shown in Figure 5. For clarity, only horizontal routing resources are shown. In the figure, three types of wire segments are used — wires that span one logic block, wires that span two logic blocks and wires that span four logic blocks. Long signal paths can utilize these long wires to reduce the total pass transistor count in the paths. Using long wire segments also can potentially reduce the total number of programmable switches, therefore increase the logic density of FPGAs.

2.2.2 Buffers

Buffers can be used to further reduce the delay of long signal paths [5]. In a buffered routing architecture, some of the pass transistors in switch blocks are replaced by tri-state buffers. Since tri-state buffers are active devices, they can reduce the propagation delay of signals over a long routing path. A buffered architecture is illustrated in Figure 6. Only horizontal

routing resources are shown. Since tri-state buffers are larger than pass transistors, using buffers in routing can decrease overall logic density of FPGAs.

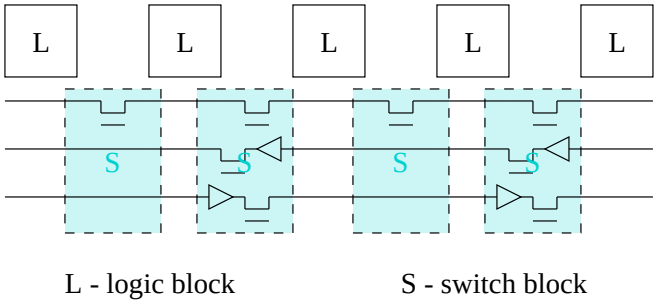


Figure 6: Routing Architecture Utilizing Buffers

2.2.3 Pipelined Routing

The use of registers in FPGAs was first investigated in [19]. This research found that FPGAs with a register to LUT ratio of 1:1 usually have good performance and logic density. A majority of current commercial FPGAs have near 1:1 register to LUT ratios.

Pipelined routing architectures, on the other hand, have a much higher register to LUT ratio. CAD tools use these extra registers to create pipelines in routing. A representative pipelined routing architecture is shown in Figure 7. Only horizontal routing resources are shown. Re-timing is usually required to ensure correct operation of circuits after routing.

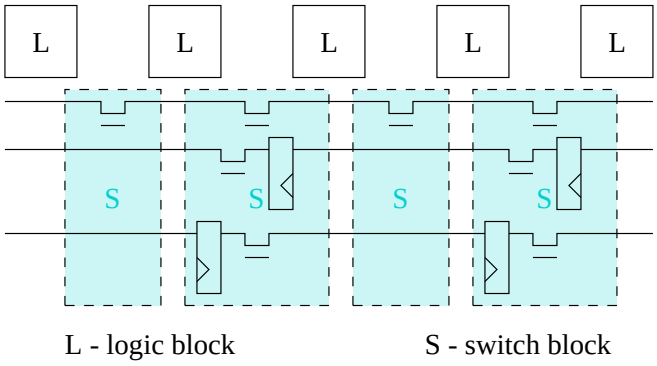


Figure 7: Pipelined Routing Architecture

Pipelined routing architectures can potentially decrease the clock cycle time of circuit

implementations at the expense of greater area. To benefit from a pipelined routing architecture, applications must have a large amount of parallelism either in forms of parallel operations or parallel tasks that can be executed in a C-slow fashion [22].

A pipelined routing architecture is proposed in [22]. The goal of the research is to minimize cycle time. A hierarchical routing architecture is used. Registers were inserted into routing wire segments to achieve the target frequency of 250 MHz.

2.3 FPGA CAD Flow

Circuits to be implemented on FPGAs are typically described in high-level hardware description languages like VHDL and Verilog. Translating high-level descriptions into FPGA programming bit-streams usually involves four steps: high-level synthesis, technology mapping, placement, and routing as shown in Figure 8.

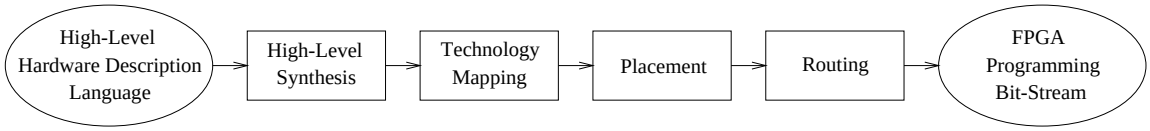


Figure 8: CAD flow for FPGA

In high-level synthesis, high-level hardware descriptions are translated into a net-list of logic equations. The logic equations are then translated into a net-list of logic blocks in technology mapping. The physical location of each logic block is then determined in the placement phase. Logic blocks are then connected in the routing phase of the CAD flow.

3 Research Focus

In this section, we will discuss our primary research focus — routing architectures for data-path applications. New routing architectures will be designed to take advantage of data-path regularities. We will present our initial designs, and discuss possible CAD algorithms. The methodologies of evaluating the effectiveness of these designs will also be discussed. These discussions will serve as the basic framework for the thesis.

Our secondary research focus will include LUT and I/O architectures for data-path applications. They will have relative minor impact on the overall FPGA performance. We have not extensively studied these issues yet and will not discuss them in detail in the proposal.

Our DP-FPGA architecture is designed to be highly efficient at transporting and distributing multiple-bit data buses. The routing architecture design does not dictate the use of either programming bit sharing LUTs (Section 2.1) or regular LUTs. The effectiveness of both in combination with our routing architecture will be investigated as a part of the thesis.

A routing architecture that is highly efficient at routing multiple-bit data buses potentially can be inefficient at routing random logic circuits. We design our architecture to accommodate random logics and reduce the inefficiency as much as possible.

As research progresses, circuits that do not route well on our architecture will be discovered. If they contribute significantly to the implementation cost of data-path applications, the DP-FPGA architecture described below will be modified to accommodate these new circuits.

Initial designs on three fundamental routing structures — local routing architectures in logic clusters, connection box architectures, and switch box architectures — will be presented in Section 3.1, 3.2, and 3.3, respectively. We will perform an initial analysis of these architectures and compare them to the traditional architectures. Transistor count will be used as an approximate measure of silicon area. We will also discuss the potential impact of these new designs on the utilization of other FPGA resources. The methodology that we have selected to evaluate the architecture alternatives will be discussed in Section 3.4. Finally, in Section 3.5, we will discuss CAD algorithms for packing, placing, and routing.

3.1 Coarse Grain Local Routing Architecture

Modern FPGA logic blocks typically are logic clusters containing several look-up tables [1, 5]. LUTs are connected to the FPGA resources outside their own cluster through a set of cluster level I/O pins. The local routing architectures inside logic clusters generally are quite different from the global routing architectures outside the clusters. Local routing architectures need to provide enough connectivity so that look-up tables can be highly utilized during

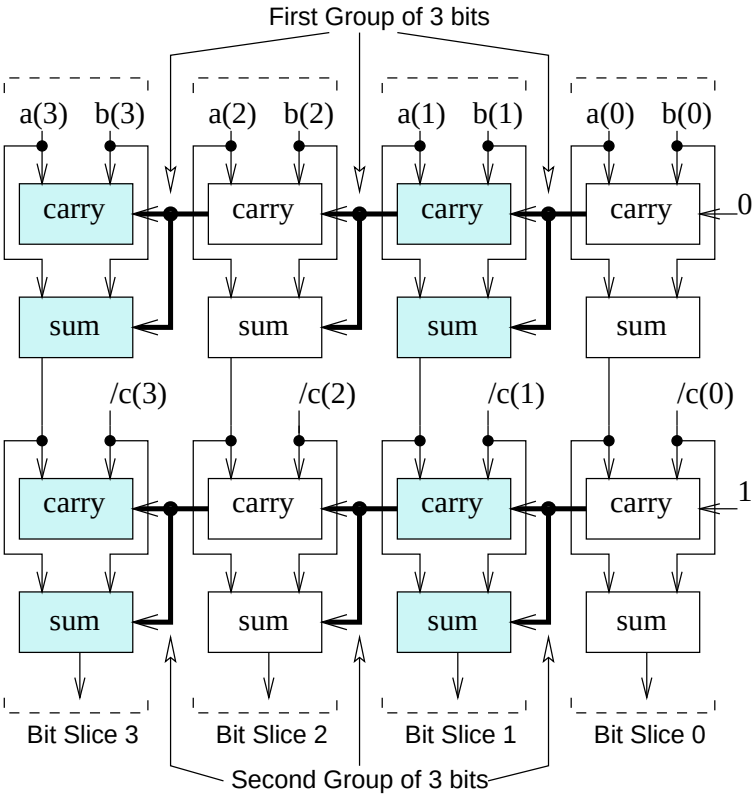


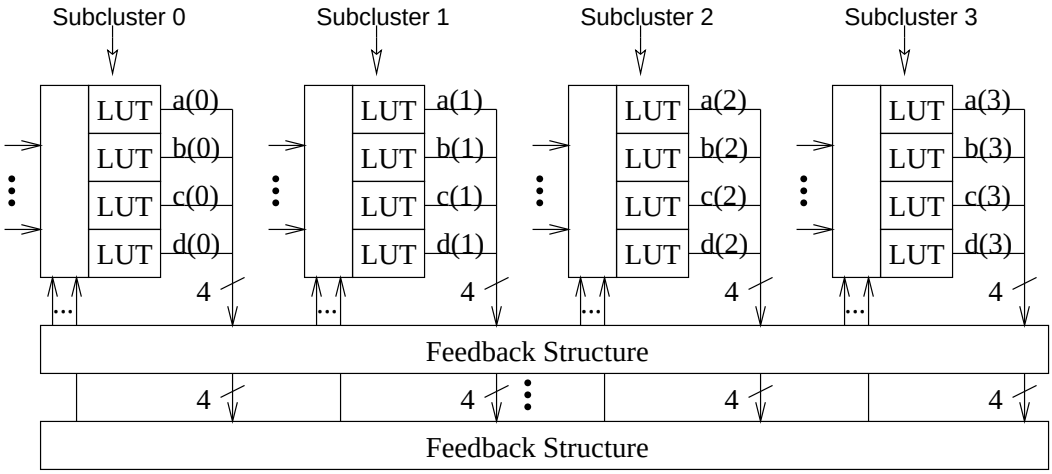
Figure 9: A Four Bit Wide Data-path Circuit

packing. Full connectivity seems to be a good choice for clusters containing a small number of LUTs. It is widely used in the industry [1, 23] and academic studies. Such an architecture can connect each look-up table input to any available cluster inputs or look-up table outputs. The optimal cluster size for full connectivity is around 4 to 10 LUTs [5]. Fully connected clusters of larger sizes are less efficient.

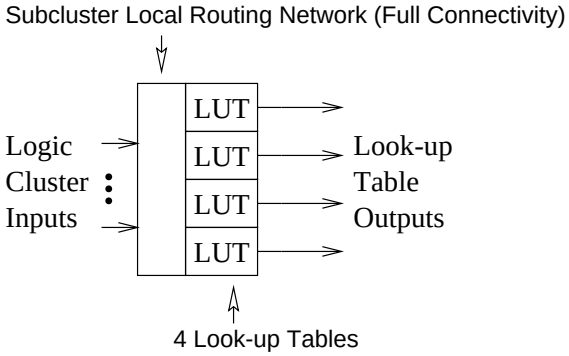
Our study will investigate if data-path circuits can benefit from simpler local routing structures and larger cluster sizes. Data-path circuits usually contain a higher degree of regularity than random logic circuits. They are typically structured out of multiple copies of a bit-slice. Multiple-bit carry or cascade signals usually are used to communicate information across bit-slices. Figure 9 shows a simple 4-bit wide circuit implementing the function $a+b-c$. Here two groups of 3-bit signals are used to connect four bit-slices. For a local routing architecture optimized for data-path applications, full connectivity might be beneficial within structures emulating bit-slices; but it will be an overkill for routing across these structures. We believe that the inter-bit-slice connectivity can be much simplified since data transfer across bit-slices takes place in terms of groups of multiple-bit data. Programming bit sharing can also be used to reduce the programming SRAM bits controlling the inter-bit-slice routing switches.

Based on the above observation, we have proposed a new local routing architecture. The coarse grain local routing architecture divides a logic cluster into several subclusters. Within each subcluster, LUTs are fully connected. These subclusters are designed for implementing portions of bit-slices. Ideally, all subclusters in the same cluster should be used to implement similar circuitries from distinct bit-slices. The inter-subcluster routing architecture is optimized towards this assumption. We will use an example to highlight the details of the architecture.

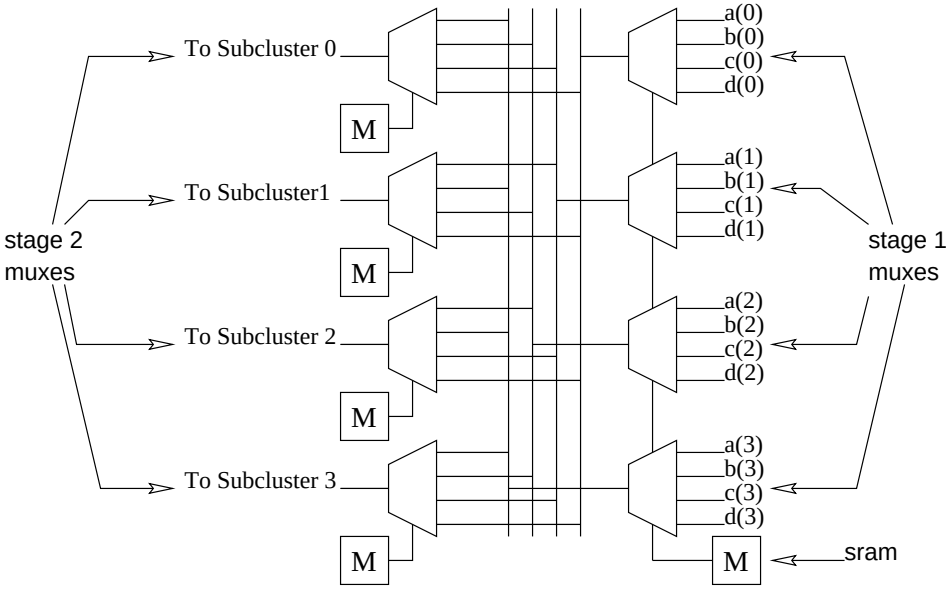
A 16 LUT version of the architecture is illustrated in Figure 10(a). This cluster is divided into four subclusters. Each contains four fully connected look-up tables. The subclusters are designed to implement portions of bit-slices in 4 LUT chunks. Within a cluster, each subcluster ideally should be used to implement identical sections of neighboring bit-slices. The outputs of the subclusters are feed back to the subcluster inputs through the feedback structure shown in Figure 10(c). Inside the feedback structure, subcluster outputs are



(a) Logic Cluster Structure



(b) Detailed Subcluster Structure



(c) Feedback Structure

Figure 10: A 16 LUT Coarse Grain Local Routing Architecture

Full Connectivity (4 input LUTs)			Coarse Grain Local Routing (4 input LUTs, $\frac{4\ subcluster}{cluster}$)						
$\frac{LUTs}{cluster}$	$\frac{inputs}{cluster}$	$\frac{tran.}{cluster}$	$\frac{LUTs}{cluster}$	$\frac{inputs}{cluster}$	$\frac{inputs}{subcluster}$	$\frac{LUTs}{subcluster}$	$\frac{feedback\ pins}{subcluster}$	$\frac{trans.}{cluster}$	$\frac{col.\ 3}{col.\ 9}$
8	16	2752	8	16	4	2	2	1424	52%
16	32	9088	16	32	8	4	4	4480	49%

Table 1: Coarse Grain Local Routing vs. Traditional Local Routing

grouped into 4-bit wide buses. Each contains a unique output bit from every subcluster. The feedback structure consists of two stages. First, one of the 4-bit data bus is selected through a network of four multiplexers. Then another four multiplexers, each selecting one bit from the chosen bus, provide one bit feedback to their corresponding subclusters. This feedback structure is duplicated as many times as the number of feedback pins on each subcluster. Note that first stage feedback multiplexers can utilize programming bit sharing in their control SRAM.

We made a comparison between the traditional architecture and our new architecture. Transistor count is used as a metric. The results are summarized in Table 1. Our new architecture requires 48% and 51% less transistors for 8 LUT clusters and 16 LUT clusters respectively. We also have manually implemented a 4×2 multiplier and an 8-bit adder on the 16 LUT logic cluster. Our new architecture provides sufficient routing resources for both circuits. These suggest that the coarse grain architecture might be better than the regular fine grain architecture. A more comprehensive study will be conducted as a part of the thesis. Real benchmarks will be used to investigate the impact of the new local routing architecture on the LUT utilization of look-up tables.

3.2 Coarse Grain Switch Box Architecture

The task of transporting multi-bit buses from one location to another occurs frequently in data-path applications. Our coarse grain switch boxes are designed to be more efficient in data bus transportation.

In a regular routing architecture, each routing track is controlled by an independent

set of SRAM bits. It transports an n-bit bus from one location to another by identically configuring n routing tracks. Our routing network reduces the number of programming bits required to transport a bus by sharing programming SRAM among the routing tracks.

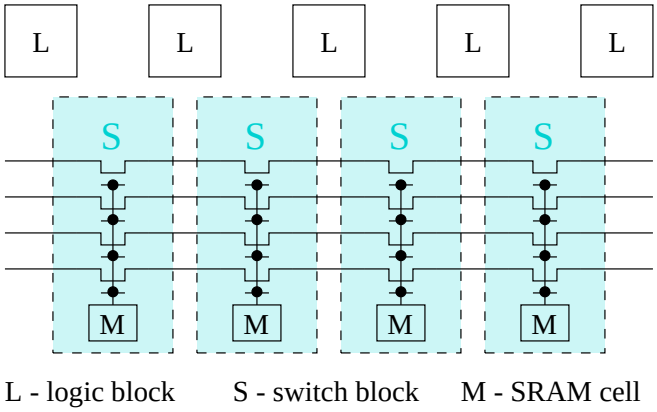


Figure 11: Coarse Grain Switch Boxes

Figure 11 illustrates a coarse grain switch box. For clarity, only one horizontal routing channel is shown. There are four tracks in the channel. All four share a single set of programming bits. Every SRAM bit controls four pass transistors — one from each track. The degree of programming bit sharing is four. These four tracks behave as four identically configured regular routing tracks. Without programming bit sharing, each track consumes 36 transistors per switch box. With programming bit sharing, each track only consumes 15. Like all the other coarse grain architectures, this architecture saves transistors by providing less flexibility, which will lead to lower utilization of routing resources. We will study the overall effect of the architecture in the thesis.

Our coarse grain routing architecture will contain coarse grain switch boxes of different granularities. We can characterize each routing channel in such a network by a bit vector (n_1, n_2, ..., n_m) where n_i is the number of routing tracks with programming sharing of degree i. Through experiments, we will search for a good mix of routing tracks with different degrees of programming bit sharing.

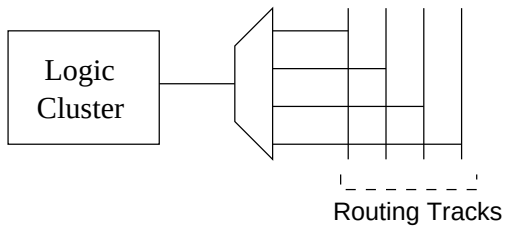


Figure 12: A Regular Input Connection Box Multiplexer

3.3 Coarse Grain Connection Box Architecture

The role of a regular connection box is to connect the I/O pins of logic clusters to routing tracks. Regular connection boxes typically are characterized by a single parameter F_c, which describes the number of tracks that a pin is able to connect to. Regular input connection boxes are typically constructed out of multiplexers [5]. Figure 12 shows a multiplexer connecting four routing tracks to a logic cluster input pin. For this connection, F_c is equal to four.

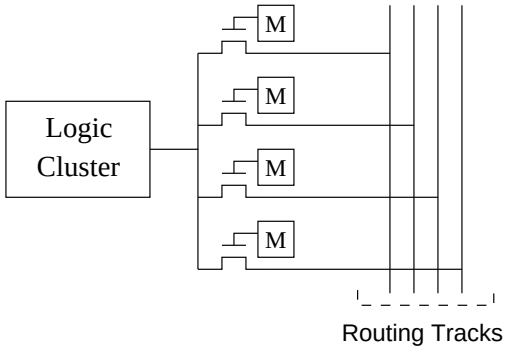


Figure 13: A Pin in a Regular Output Connection Box

Regular output connection boxes are typically constructed out of tri-state buffers [5]. Figure 13 shows a typical connection of a logic cluster output pin to four routing tracks. Again for this connection, F_c is equal to four.

In our coarse grain connection boxes, logic cluster I/O pins are grouped into buses. Pins in each group are connected to the routing tracks through a unified connection structure. The number of tracks that a pin can connect to is affected by the I/O configuration of other

pins in the group. Here, we use F_c to describe the maximum number of routing tracks that a logic cluster pin is able to connect to.

This section is divided into two subsections. In Section 3.3.1, we will discuss the architecture for our input connection boxes; and in Section 3.3.2, we will discuss the architecture for our output connection boxes.

3.3.1 Input Connection Boxes

Figure 14 and Figure 15 illustrate an $F_c = 16$ version of our connection box design. The connection box for one logic cluster is shown in the figures. There are four subclusters in the cluster. Each subcluster has four input pins. The structure is explained in detail below.

As described in the previous section, our global routing network groups signals into n-bit wide buses. The signals in each bus are transported together by the routing network. The role of the coarse grain connection boxes is to efficiently select and distribute these bus signals from the tracks to the subclusters in a logic cluster.

Our connection boxes select signals from the routing channels in groups of n bits, where n is the granularity of the global routing network. This selection scheme can be implemented using programming bit sharing muxes similar to the ones labeled as "bus selecting muxes" shown in Figure 15.

We use the same circuit shown in Figure 9 to illustrate the considerations that lead to our distribution network design. The circuit is redrawn and re-labeled in Figure 16. It is a 4-bit wide data-path circuit implementing the function $a + b - c$. There are three 4-bit wide inputs to this circuit — a, b, and c. Input bits $a(3)$, $b(3)$, and $c(3)$ are fed exclusively to the bit-slice #3. Input bits $a(2)$, $b(2)$, and $c(2)$ are fed exclusively to the bit-slice #2 and so on. This is generally characteristic of ripple carry type of circuits, where one bit from each input is fed into each bit-slice. Since the subclusters are designed to implement bit-slices, our coarse grain connection boxes have the ability to distribute one bit from each selected bus to a subcluster. This is achieved using muxes similar to the ones labeled as "bit selecting muxes" in Figure 15.

We also added one more level of muxes to our connection boxes. They are labeled as "mode-selecting muxes" in Figure 14. These muxes are motivated by the circuit illustrated in

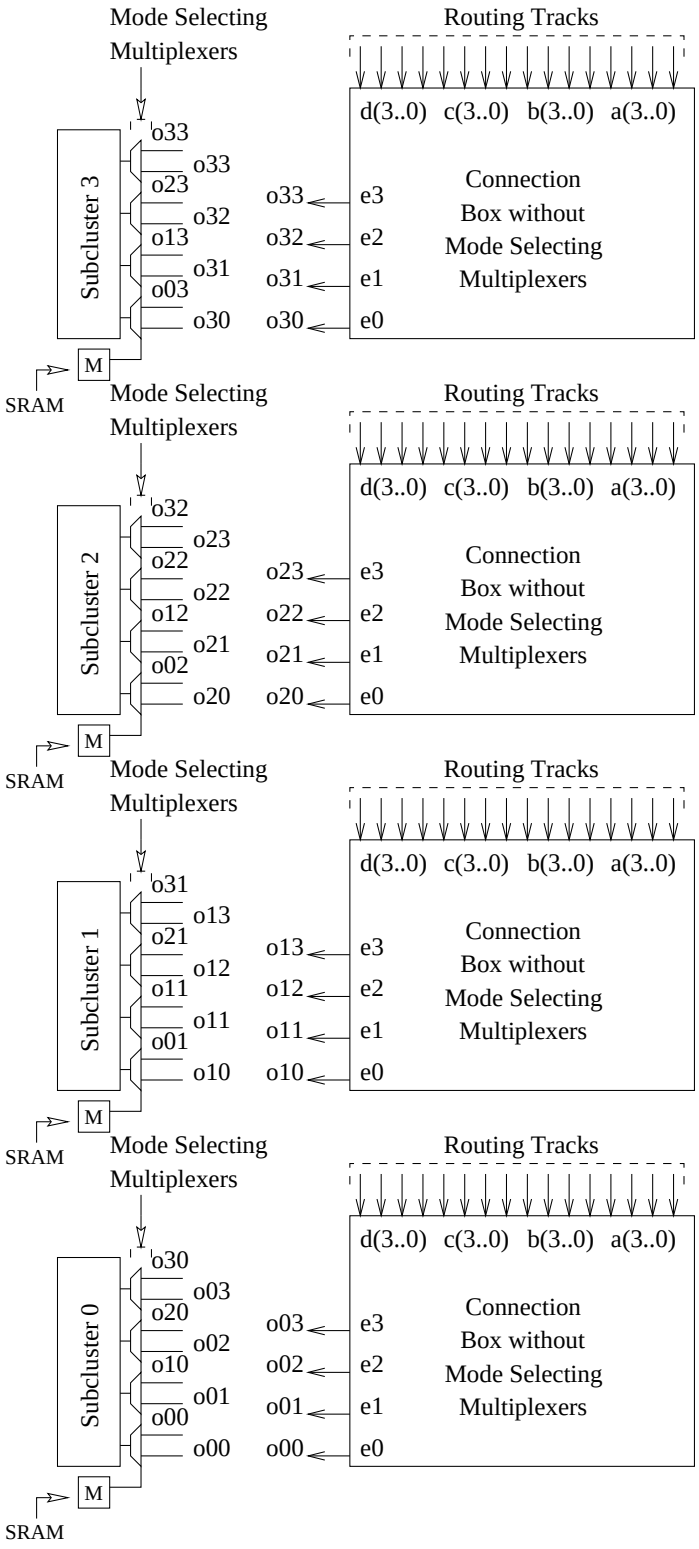


Figure 14: A Coarse Grain Input Connection Box

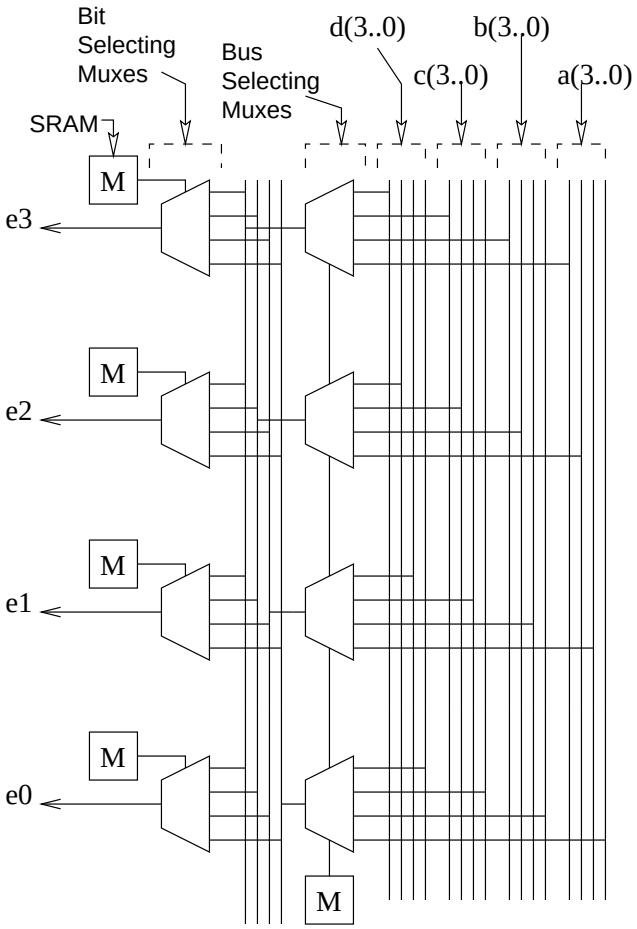


Figure 15: The Connection Box without Mode Selection Multiplexers Block

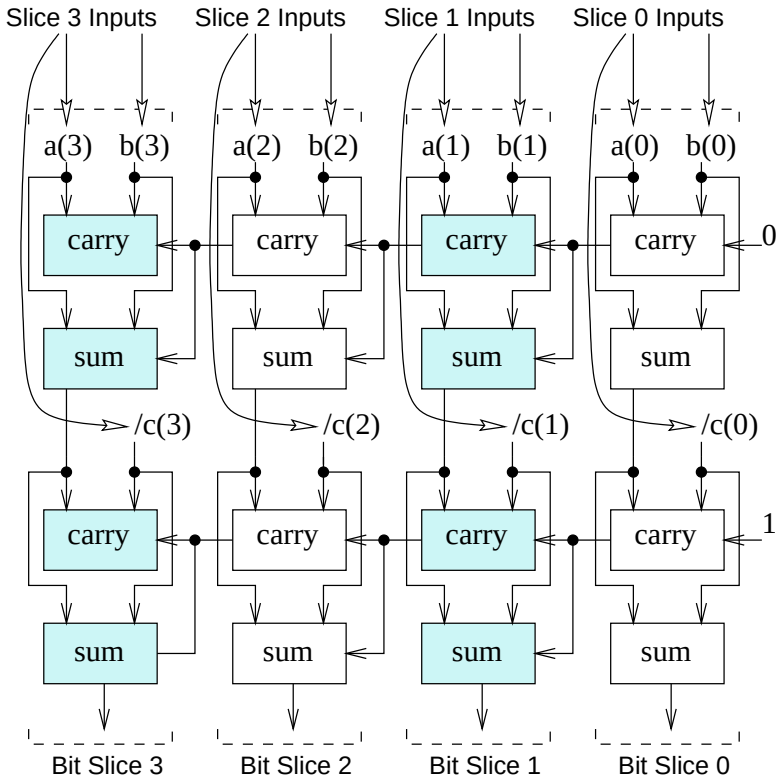
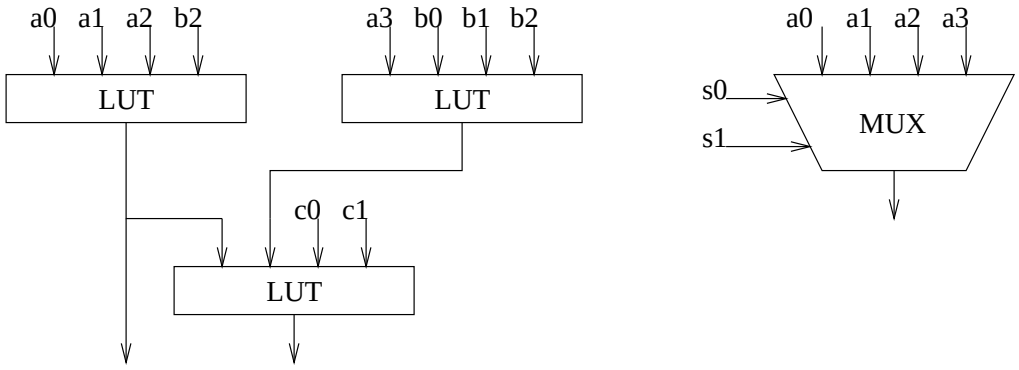


Figure 16: A Four Bit Wide Data-path Circuit



(a) A Random Logic Circuit

(b) A 4:1 Multiplexer

Figure 17: A Finite State Machine State Generation Circuit

Figure 17a. This is a random logic circuit that takes in three buses, (a_3,a_2,a_1,a_0), (b_3,b_2,b_1,b_0), and (c_1,c_0), as inputs. It produces a two-bit output. This can be the state generation logic of a finite state machine (FSM), where the input buses represent FSM inputs and current state and the output bits represent bits of the next state. This motivates us to create two modes of operation for our connection boxes. When the control input of the "mode-selecting muxes" is set to zero, the signals are routed from the routing tracks to the subclusters as described in the previous paragraphs. When the control input of the "mode-selecting muxes" is configured to one, entire buses, selected by the "bus selecting muxes", are routed to the subclusters. In this mode, each subcluster can be used to implement an independent random logic circuit with several n-bit wide bus inputs (one 4-bit wide bus input in Figure 14). This mode of operation is also similar to the I/O characteristics of some data-path components like the multiplexer shown in Figure 17b. Here, the mux takes in two buses as inputs and produces a single bit output.

For $F_c = 32$, the coarse grain architecture uses 46 transistors per input pin, while the traditional architecture requires 102 transistors per input pin. For $F_c = 64$, the coarse grain architecture uses 64 transistors per input pin, while the traditional architecture requires 174 transistors.

The coarse grain connection boxes are less flexible than the traditional connection boxes since tracks are selected in chunks of n-bit wide buses. This might lead to lower utilization of both look-up tables and routing tracks. We will investigate these issues fully as a part of the thesis through experimentation.

3.3.2 Output Connection Boxes

Our output connection boxes are modified versions of the one proposed in the Cherepacha and Lewis study. The same types of circuits that motivate our input connection box design guide the output connection box design. Figure 18 shows a four subcluster logic cluster and its output connection box. Each subcluster has a 4-bit output. The routing channels route signals in groups of four bits. The output connection box has two modes of operation which are configured through the "mode selecting muxes".

For the ripple carry type of circuits like the one shown in Figure 16, each bit-slice usually

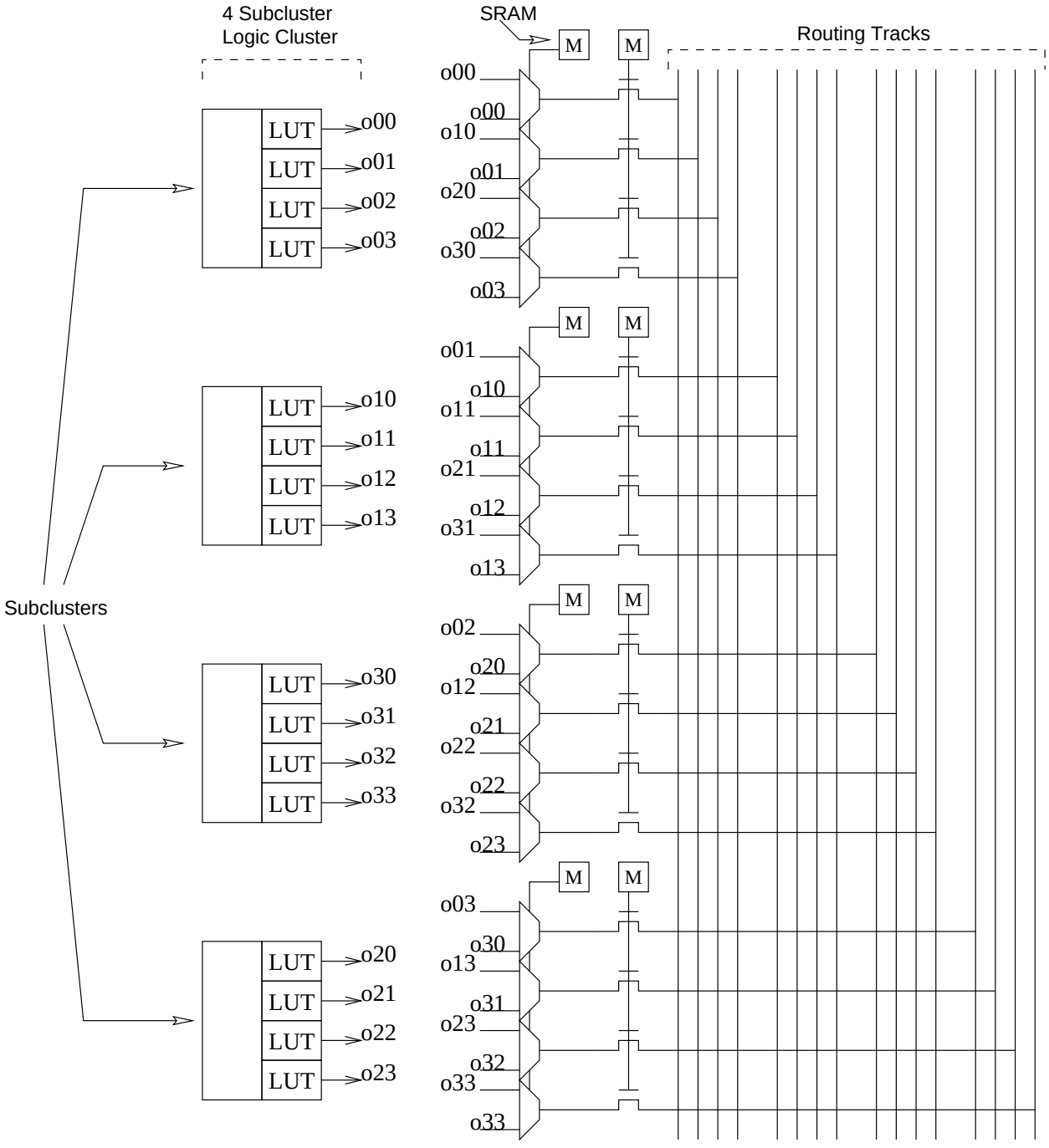


Figure 18: A Coarse Grain Output Connection Box

produces the same number of output bits. These bits typically can be grouped into logical buses with every bus containing one bit of each bit-slice output. For example in Figure 16, each bit-slice produces a 1-bit output. These bits form a single 4-bit data bus. It is very likely that all bits of this bus will be routed to the same destination. In the first mode of operation, our output connection boxes are configured for this type of output. Here, one bit from each subcluster output is grouped together to form a bus. This bus is connected to a group of programming bit sharing routing tracks through programming bit sharing pass transistors.

When implementing random logic circuits in multiple subclusters, each subcluster might produce multiple-bit outputs that need to be routed to the next subcluster for processing. It is beneficial to route these intermediate signals together. In the second mode of operation, our output connection box is configured for random logic type circuits like the one shown in Figure 17a. Outputs from the same subclusters are routed to a group of programming bit sharing routing tracks.

Comparing to a conventional output connection box with the same F_c value, the coarse grain connection box, shown in Figure 18, uses only 88% transistors. The transistor saving is due to programming bit sharing.

3.4 Architecture Evaluation Methodology

The architecture discussed above mainly targets data-path applications. As the result, it might perform poorly for fine grain random logic circuits. For example, the coarse grain connection boxes, discussed in Section 3.3, couple the control signals of several cluster level I/O pins together. Once the I/O signals of one subcluster are configured, the I/O pin configuration for all other subclusters becomes limited. The coarse grain switch boxes have the same shortcomings. A switch box contains groups of tracks. Each group is controlled by a single set of programming SRAM. If a group of tracks is used to route a narrower bus, the unused tracks in the group will likely be wasted. Through experiments, we will determine if dedicated fine grain routing resources, in addition to the coarse grain routing resources, will be beneficial to the majority of data-path applications.

We will use an evaluation methodology similar to the one used in [5]. The evaluation of

the goodness of an architecture will be based on experiments. We will use a detailed transistor area model for area evaluation and the Elmore delay model for delay investigation. For a detailed discussion on the methodology, please refer to [5].

A collection of benchmarks will be used. Possible benchmarks include MCNC benchmarks, CPU designs, and graphic accelerator designs. VPR [5] will be used to place and route the same benchmarks using traditional architectures as a comparison. For the new coarse grain architectures, several CAD tools will be build. These tools will include synthesizer, packer, placer, and router.

The synthesizer will translate a high-level hardware description language into look-up tables. It will perform synthesis, technology independent optimization, and technology mapping. The packing tool will closely cooperate with the synthesizer to pack look-up tables into logic clusters. The placer and router will be modified versions of the VPR tools. The detailed CAD flow will be discussed in the next section.

3.5 CAD Flow for Coarse Grain FPGA

This section is divided into two subsections. In Section 3.5.1, we will discuss the synthesizer and the packing tool. In Section 3.5.2, we will discuss the placer and the router.

3.5.1 Synthesizing and Packing

For highest quality, we will handcraft many essential circuits into macros. The packing, routing, and placement process of these macros will be entirely done by hand. Several macros that we plan to implement are adders, subtracters, and multipliers. We will also include in our CAD tool a semi-automatic synthesizing and packing feature. The methodology that we plan to use is discussed below.

We will use compiler hints to indicate the start and end of autonomous wide I/O random logic or ripple carry circuits. For example, in VHDL we might have some comments like:

```
-- START COARSE GRAIN
-- END COARSE GRAIN
```

to annotate the start and end of a circuit, respectively. Our synthesizer will isolate the circuit. The technology independent optimization and technology mapping will be invoked on each of these isolated circuit units. Circuits that will benefit from these hints include FSM state generation logic, FSM output generation logic, wide decoders, and ripple carry circuits.

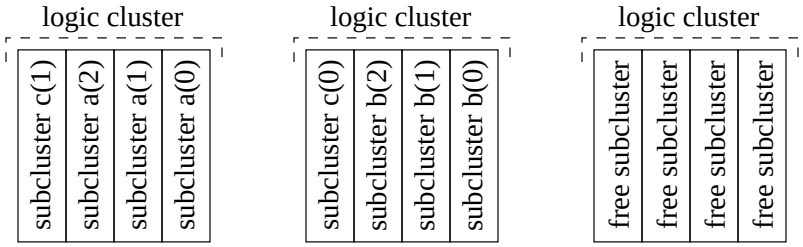
The packing tool will be invoked on each individual circuit unit to pack it into logic clusters. A plausible packing method will be similar to the bin packing problem, where each subcluster represents a bin. A logic cluster represents a group of related bins. We will first allocate enough logic clusters so that there are just enough LUTs for implementing the circuit in question. Simulated annealing will be used to find good cluster configurations based on the weighted sum of the following three cost factors:

- A cost factor based on if the cluster configurations are legal configurations. That is if there are enough logic cluster I/O pins and local routing resources to accommodate the LUTs packed into each cluster. Initially, illegal configurations will be tolerated. As the iteration progresses, the packing tool will become less and less likely to allow illegal cluster configurations. The final cluster configurations must be legal.

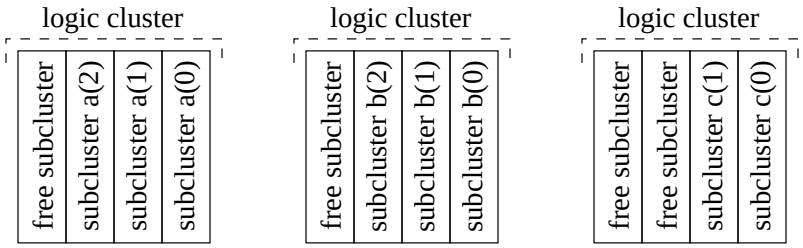
- A cost factor based on the utilization of global routing resources by cluster I/Os. This factor should be inversely proportional to the global routing resource utilization.

- During the simulated annealing, the total number of available LUTs will be gradually increased in order to lower the previous two cost factors. The number of LUTs used introduces an additional cost.

An important question arises regarding to the relative weights of these three cost factors. How important is LUT utilization vs. global routing resource utilization? Higher LUT utilization means LUTs are used more efficiently, but it might cause difficulties for cluster I/O signals to be grouped into buses, therefore lowering the utilization of global routing resources. Logic clusters with lower LUT utilization allow more flexible LUT placement. This might allow a higher degree of global routing resource utilization. The weights in the cost function should reflect this trade-off. They will be determined experimentally.



(a) Packing Configuration #1



(b) Packing Configuration #2

Figure 19: An Packing Example

One example is illustrated in Figure 19. In the figure, each logic cluster has four subclusters. Here, we have three groups of subclusters — group a, b, and c — to be packed into logic clusters. Group a and b consist of three subclusters each. Group c consists of two subclusters. Each subcluster has one bit output. Let us assume that the outputs of the subclusters from each group can be grouped into a bus and it is beneficial to route signals in each bus together through the global routing network. If the subclusters are packed into the configuration shown in Figure 19(a), only two logic clusters are used. However, more global routing resources will be consumed since the output of the c blocks can not be readily grouped into a bus. The configuration shown in Figure 19(b), on the other hand, uses three logic clusters, but the output of subcluster group c can be readily grouped into a bus and potentially will use less global routing resources.

For circuits that are not annotated by the compiler hints, we will use a greedy algorithm for packing. First we will pack a single subcluster in a logic cluster. This will limit most I/O configuration choices on the remaining subclusters in the cluster. Given the limitation, these remaining subclusters will be packed. Low utilization of LUTs and routing resources might result. If the architecture contains fine grain routing resources, regular routing algorithms

will be used to pack fine grain random logics.

3.5.2 Placing and Routing

The original VPR placing and routing tools can be used for coarse grain FPGA architectures. The cost functions of both, however, have to be adjusted to reflect the fact that multiple-bit data buses cost significantly less per bit than single-bit data in coarse grain FPGA.

4 Conclusions

In this proposal, we have presented a framework for designing and evaluating new DP-FPGA routing architectures. We have reviewed the previous DP-FPGA study, several general FPGA routing architectures, and presented our research focus. We will systematically explore the design space through experiments. As the research progresses, the framework presented above will also evolve and adapt to new questions that arise.

References

[1] Altera Corporation. *Altera Data Book*, Altera Corporation, 1998.

[2] Michael J. Alexander and Gabriel Robins. "New Performance-Driven FPGA Routing Algorithms," *32nd ACM/IEEE Design Automation Conference*, 1995.

[3] Vaughn Betz and Jonathan Rose. "Effect of the Prefabricated Routing Track Distribution on FPGA Area-Efficiency," *IEEE Transactions of VLSI*, Vol. 6, No. 3, page 445-456, September 1998.

[4] Vaughn Betz and Jonathan Rose. "Directional Bias and Non-Uniformity in FPGA Global routing Architectures," *ICCAD 1996*, page 652-659, November 1996.

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

[6] Stephen Brown and Jonathan Rose. "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Design and Test of Computers*, Vol. 13, No. 2, page 42-57, 1996.

[7] Stephen Brown, Jonathan Rose, and Zvonko Vranesic. "A Detailed Router for Field-Programmable Gate Arrays," *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, Vol. 11, No. 5, page 620-628, May 1992.

[8] Stephen D. Brown, Jonathan Rose and Zvonko G. Vranesic. "A Stochastic Model to Predict the Routability of Field-Programmable Gate Arrays," *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, Vol 12, No. 12, page 1827-1838, December, 1993.

[9] Don Cherepacha and David Lewis. "DP-FPGA: An FPGA Architecture Optimized for Datapaths," *VLSI Design*, Vol. 4, No. 4, page 329-343, 1996.

[10] Paul Chow, Soon Ong Seo, Jonathan Rose, Kevin Chung, Gerard Paez-Monzon and Immanuel Rahardja. "The Design of an SRAM-Based Field-Programmable Gate Array," To Appear in *IEEE Transactions on VLSI*.

[11] Kevin Chung, Satwant Singh, Jonathan Rose, and Paul Chow. "Using hierarchical logic blocks to improve the speed of field-programmable gate arrays," *FPGAs*, Will Moore and Wayne Luk, Eds., chapter 3.3, page 103-113. Abingdon EE&CS Books, England, 1991.

[12] D. Jones and D. M. Lewis. "A Time-Multiplexed FPGA Architecture for Logic Emulation," *IEEE 1995 Custom Integrated Circuits Conference*, Session 24.2.1, page 495-498, 1995.

[13] Muhammad Khellah, Stephen Brown and Zvonko Vranesic. "Modeling Routing Delays in SRAM-Based FPGAs," *Proc. CCVLSI-93*, page 6B.13-6B.18, November 1993.

[14] Mohammed A. S. Khalid and Jonathan Rose. "The Effect of Fixed I/O Pin Positioning on the Routability and Speed of FPGAs," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, Vol. 11, No. 5, page 620-628, May 1992.

[15] Yen-Tai Lai and Ping-Tsung Wang. "Hierarchical Interconnection Structures for Field Programmable Gate Arrays," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, Vol. 5, No. 2, page 186-196, June 1997.

[16] Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, and Brad Hutchings. "A Reconfigurable Arithmetic Array for Multimedia Applications," *FPGA 99: International Symposium on Field Programmable Gate Arrays*, page 135-143, February 1999.

[17] Anmol Mathur and C. L. Liu. "Compression-Relaxation: A New Approach to Timing-Driven Placement for Regular Architectures," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 6, page 597-599, June 1997.

[18] Larry McMurchie and Carl Ebeling. "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs".

[19] Jonathan Rose, Robert Francis, David Lewis, and Paul Chow. Architecture of Field-Programmable Gate Arrays: "The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 4, page 1217-1225, October 1990.

[20] Jonathan Rose and Stephen Brown. "The Effect of Switch Box Flexibility on Routability of Field Programmable Gate Arrays," *IEEE 1990 Custom Integrated Circuits Conference*, page 27.5.1-27.5.4, 1990.

[21] Carl Sechen and Alberto Sangiovanni-Vincentelli. "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, Vol. sc-20, No. 2, page 510-522, April 1985.

[22] William Tsu, Kip Macy, Atul Joshi, Randy Huang, Norman Walker, Tony Tung, Omid Rowhani, Varghese George, John Wawrzynek, and Andre Dehon. "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array," *FPGA 99: International Symposium on Field Programmable Gate Arrays*, page 125-134, February 1999.

[23] Xilinx Corporation. *Xilinx Data Book*, Xilinx Corporation, 1999.