On pin-to-wire routing in FPGAs

by

Niyati Shah

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science and Engineering
Graduate Department of Electrical & Computer Engineering
University of Toronto

Abstract

On pin-to-wire routing in FPGAs

Niyati Shah

Master of Applied Science and Engineering

Graduate Department of Electrical & Computer Engineering

University of Toronto

2012

While FPGA interconnect networks were originally designed to connect logic block output pins to input pins, FPGA users and architects sometimes become motivated to create connections between pins and specific wires in the interconnect. These *pin-to-wire* connections are motivated by both a desire to employ routing-by-abutment, in modular, pre-laid out systems, and to make direct use of resources in the fabric itself. The goal of this work is to measure the difficulty of forming such pin-to-wire connections. We show that compared to a flat placement of the complete system, the routed wirelength and critical path delay increase by 6% and 15% respectively, and the router effort increases 3.5 times. We show that while pin-to-wire connections impose increased stress on the router, they can be used under some circumstances. We also measure the impact of increasing routing architecture flexibility on these results, and propose a low-cost enhancement to improve pin-to-wire routing.

Acknowledgements

I would like to thank my supervisor Jonathan Rose, for his continual guidance, valuable support and helpful insights, that have enabled me to develop not only as a researcher and an engineer, but also as a person.

I would also like to thank Huimin (Hannah) Bian from Altera for some good suggestions in this work and Jason Luu for his great support and advice.

I would like to give special thanks to University of Toronto for financially supporting my research by granting me University of Toronto fellowships.

Further, I wish to thank all my colleagues in the Computer Group and in LP392, for making my Masters so much more enjoyable.

Last but not the least, I would like to thank my parents, for their unwavering support and encouragement throughout my Masters, and my friend Harpuneet for his continual understanding, support, encouragement and humour. I could not have done this without them.

Contents

List of Tables

List of Figures

Chapter 1

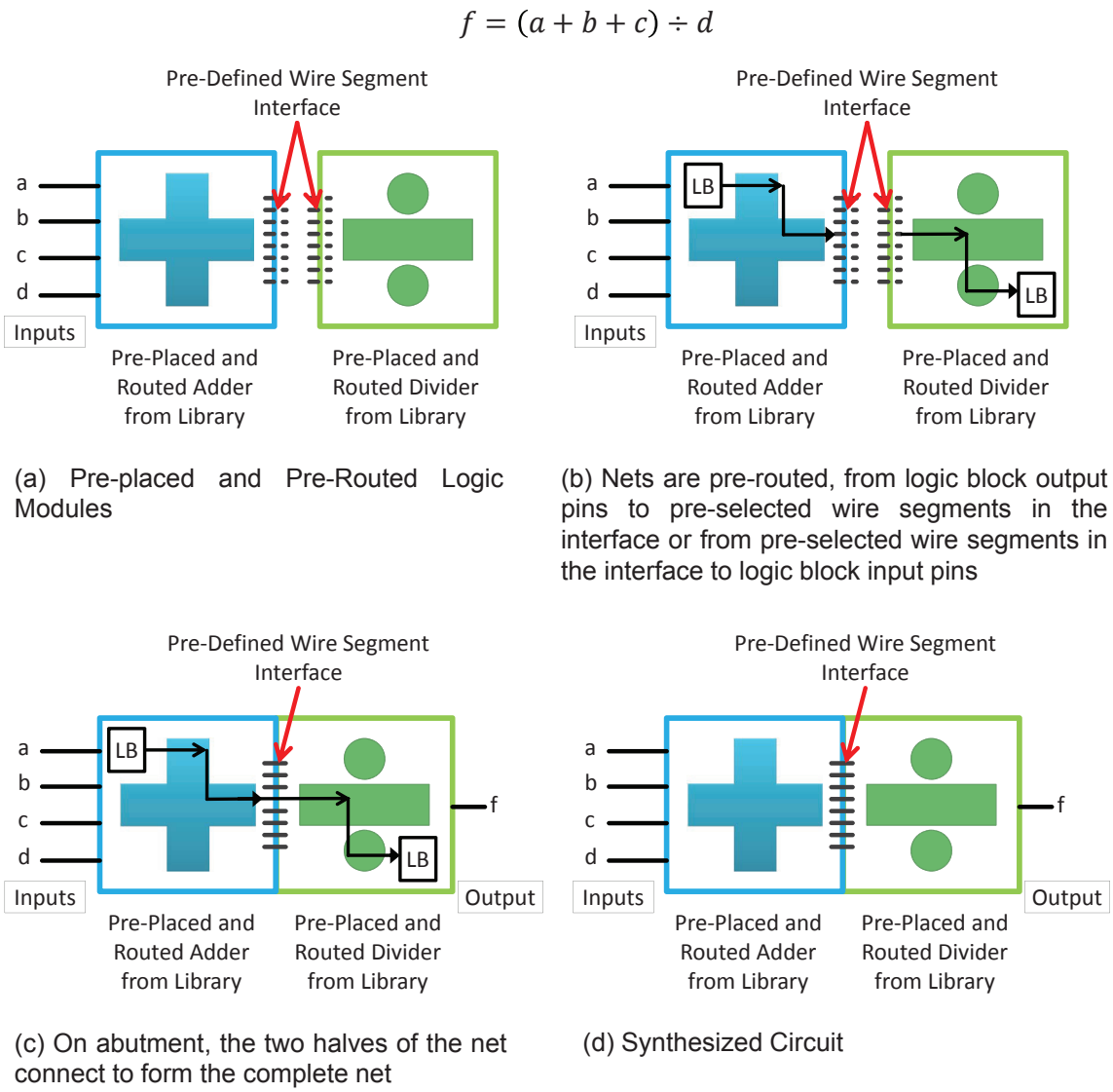
Introduction

Over the past decade, Field-Programmable Gate Arrays (FPGAs) have evolved dramatically in logic density, size and complexity [3]. The largest FPGAs such as the Virtex 7 [4] consist of as many as 1.5 million Logic Elements. In addition, contemporary FPGA architectures consist of a large number of logic blocks that provide specific functionality including DSPs, memories, multiplexers and other special-purpose computational blocks [5][6][7][8]. Thanks to these attributes, FPGAs now boast of enormous flexibility and high compute power, in addition to offering lower time-to-market, lower non-recurring engineering costs and high versatility. The high compute power and flexibility of FPGAs have made them viable alternatives to ASICs and DSPs in some markets [9]. Hence, it is but natural that they have found their way into several industries, including communications, aerospace and military and defence.

The high logic density and complexity of FPGAs also allows larger and more complex circuits to be synthesized onto them. That in turn translates into even larger computing requirements for the CAD tools running the design compilation and place and route. These tools are run on CPUs, which have not kept pace with the increasing FPGA design complexity [3]. As a result, FPGA CAD compile times threaten to increase to unacceptable times. For moderate to large designs, compile times today lie anywhere

from a few hours to a few days. If left unaddressed, these increasing compile times would make FPGAs impractical for use by designers. As Placement and Routing form a major portion of the CAD compile time, eliminating or reducing those would potentially help reduce compile times.

One way to eliminate a significant portion of the place and route time (and hence, reduce compile time) is to employ pre-placed and pre-routed logic modules that connect together by the abutment of wire segments at their interface. We refer to this technique



(a) Pre-placed and Pre-Routed Logic Modules

(b) Nets are pre-routed, from logic block output pins to pre-selected wire segments in the interface or from pre-selected wire segments in the interface to logic block input pins

(c) On abutment, the two halves of the net connect to form the complete net

(d) Synthesized Circuit

Figure 1.1: An illustration of Routing-by-Abutment

as *routing-by-abutment*. This is somewhat similar to what is done for power grids using ASIC standard cells [10] or for full custom VLSI design using hand designed datapath cells [11]. Figure 1.1 illustrates an example in which the function f is synthesized using routing-by-abutment. One advantage of this technique is that the abutting logic modules can be created independently provided the specific wire segments that form the abutting connections are known in advance. Moreover, the idea can be extended to connect several logic modules that use the same wire abutment pattern. Clearly, as demonstrated in Figure 1.1, for this idea to work, nets crossing between the modules would have to be routed from logic block output pins to a specific wire segments or from a specific wire segments to logic block input pins. We will refer to such connections as *pin-to-wire* routing. Then, when the abutting modules would be placed next to each other, the two halves of the nets would connect to form a complete net, thereby eliminating the need for inter-modular routing. This elimination of inter-modular routing along with the coarse placement of logic modules could help eliminate a significant portion of the compile time. Clearly, for this technique to work, effective pin-to-wire routing is essential.

However, due to lack of interconnect flexibility, this requirement for effective pin-to-wire routing conflicts with the basic purpose of the FPGA interconnection network, which is pin-to-pin routing. Good architects of those networks [5],[6],[7],[8] choose a quantity of routing and the total routing flexibility to satisfy the interconnection demand of broad classes of application circuits, but only for pin-to-pin connections. This suggests that there may be an insufficient quantity and flexibility of routing if connections are made between pins and wire segments because there will be fewer stages of the interconnection network available for such connections. For example, for pin-to-wire routing, the final connection block and intra-cluster routing multiplexers will be missing, giving far fewer potential paths from start to finish. Similarly, for wire-to-pin routing, the output connection block's flexibility is missing. Thus, it is clear that pin-to-wire routing will be more difficult, for each such connection, than pin-to-pin routing. However, most FP-

GAs are somewhat over-architected in the routing to ensure that circuits with widely varying routing demand will achieve routability. We hypothesize that some of this extra routability could make up for some of the loss of routability in pin-to-wire routing, and hence, we seek to understand and measure if this is true. The goal of this work is to experimentally measure the impact of pin-to-wire routing on routing wirelength, critical path speed and router effort. We will do this using a two step approach. In the first step, we run experiments that enable us to measure the various instances of pin-to-wire routing on a fairly classical architecture. As a second step, we will vary the classical architecture by adding or reducing flexibility and observe how it influences the results of the experiments from the first step. In doing so we seek to understand how much pin-to-wire routing could be used in an FPGA, and perhaps enable its above mentioned application.

Studying pin-to-wire routing is also beneficial for motivations other than routing-by-abutment. One such motivation arises in the use of partial reconfiguration of FPGA modules [12]. Here, specific areas of the FPGA are designated to contain different functional modules at different times, yet these must connect to neighbouring modules using the same wire segments. This connectivity problem has been solved by using overlapping 'proxy' LUTs [13] at the cost of some significant amount of intervening logic. If the wire segments themselves can be used to achieve abutment routing, it may reduce this cost.

Another motivation for pin-to-wire routing arises whenever an architect considers making use of the circuits elements within the routing architecture itself: a classic observation in this realm is that although it is inefficient to implement multiplexers in an FPGA using LUTs, it is ironic that the FPGA itself is full of multiplexers as the key building blocks of the routing fabric. In order to make use of those multiplexers (after architecturally modifying access to their selection inputs) there has to be an ability to connect to specific data inputs on the routing multiplexers from output pins of logic blocks.

By quantifying and analyzing the impact of pin-to-wire routing (the goal of this work), we seek to enable some of its applications and provide a guide for FPGA architects and designers.

1.1 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents background on FPGA architecture terminology and tool flow. It also reviews prior related work in the area. In Chapter 3 we describe our experimental methodology for measuring various instances of pin-to-wire routing, and also give the results and understanding generated. In Chapter 4, we measure the impact of routing architecture changes on pin-to-wire routing. In Chapter 5, we conclude.

Chapter 2

Background

The objectives of this chapter are to review the basics of FPGA architecture, including logic block architecture and routing architecture; to provide an overview of the CAD flow needed to synthesize a circuit onto an FPGA; and to describe relevant prior work relating to pin-to-wire routing including abutment-oriented routing, partial reconfiguration and the use of circuit elements within the routing fabric as logic elements.

2.1 FPGA Architecture

An FPGA architecture consists of a set of logic blocks (LBs) that implement the logic required by a circuit to be synthesized, Input/Output (I/O) blocks that communicate with the off-chip world, and an interconnection network of programmable routing that connects them.

This work targets a class of FPGA architectures known as *island-style* FPGAs. As depicted in Figure 2.1, in this architecture style, logic blocks surrounded by programmable interconnect are laid out in a 2-D array with I/O blocks at the periphery. The basic logic block architecture and routing architecture are reviewed in subsections 2.1.1 and 2.1.2 respectively.

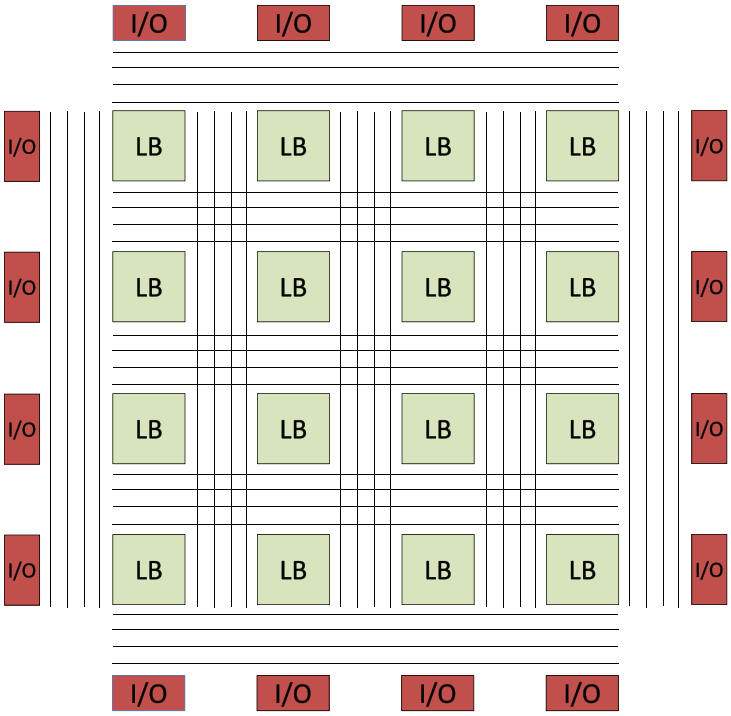


Figure 2.1: An island-style FPGA architecture

2.1.1 Logic Block Architecture

This work employs cluster-based logic blocks that have a two-level hierarchy; the first level consists of a *Basic Logic Element* (BLE). In the academic literature, a BLE consists of a K-input lookup table (LUT) and a register that are connected together as shown in Figure 2.2a. The two-input multiplexer allows the BLE output to be either registered or unregistered. The second level of hierarchy comprises of a collection of N BLEs. The resulting structure of the cluster-based logic block is referred to as a *logic cluster* and is illustrated in Figure 2.2b. A cluster-based logic block has I inputs and N outputs, where $I \leq K{\cdot}N$. For the scope of this work, we set I according to equation 2.1, which was suggested by Ahmed et al. [1] for 98% BLE utilization.

$$I = \frac{K}{2}(N+1) \tag{2.1}$$

The complete cluster-based logic block also contains local routing to interconnect the BLEs. Note that the logic cluster depicted in Figure 2.2b is *fully connected*, as all of the

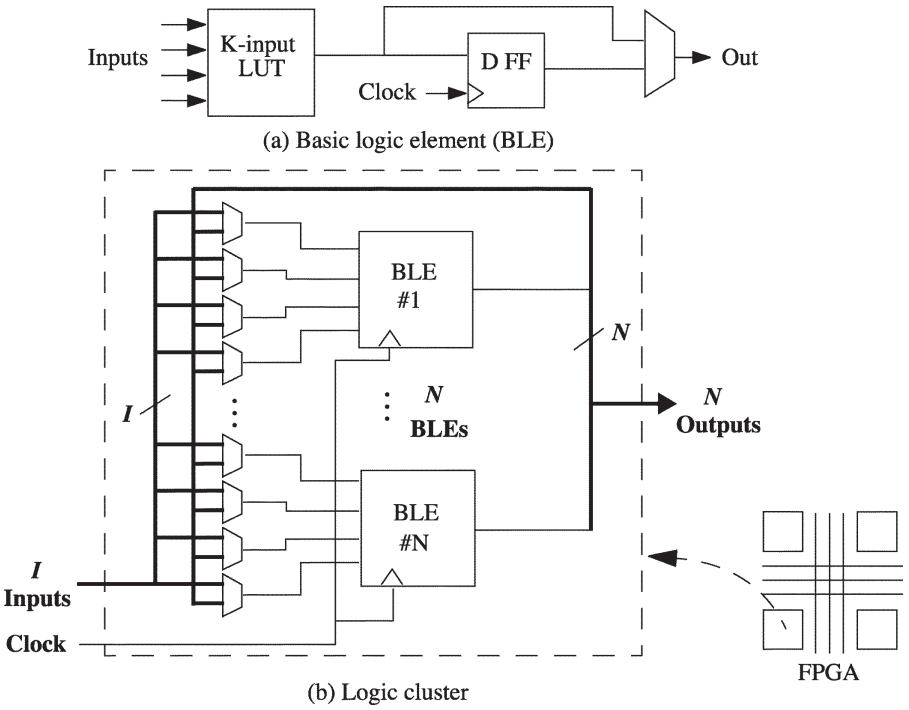


Figure 2.2: Structure of (a) basic logic element (BLE) (b) logic cluster. [1]

$K{\cdot}N$ BLE inputs can connect to all of the I cluster inputs and the N cluster outputs. Albeit, the logic blocks in commercial FPGAs are much more complex, but they still employ a similar hierarchical cluster-based approach [5][6][7].

Each individual logic block implements a small portion of the logic required by the circuit to be synthesized. Then, for the circuit's logic functionality to be fully realized, these blocks need to be connected to each other and to the I/O blocks. This connectivity is provided by the routing architecture which is described in the following subsection.

2.1.2 Routing Architecture

In this section we will review the parameters used to describe a basic routing architecture and also the notations used to denote those parameters.

As illustrated in Figure 2.3, in an island style FPGA, logic blocks are surrounded on all four sides by routing channels of pre-fabricated wiring segments [2]. The length of

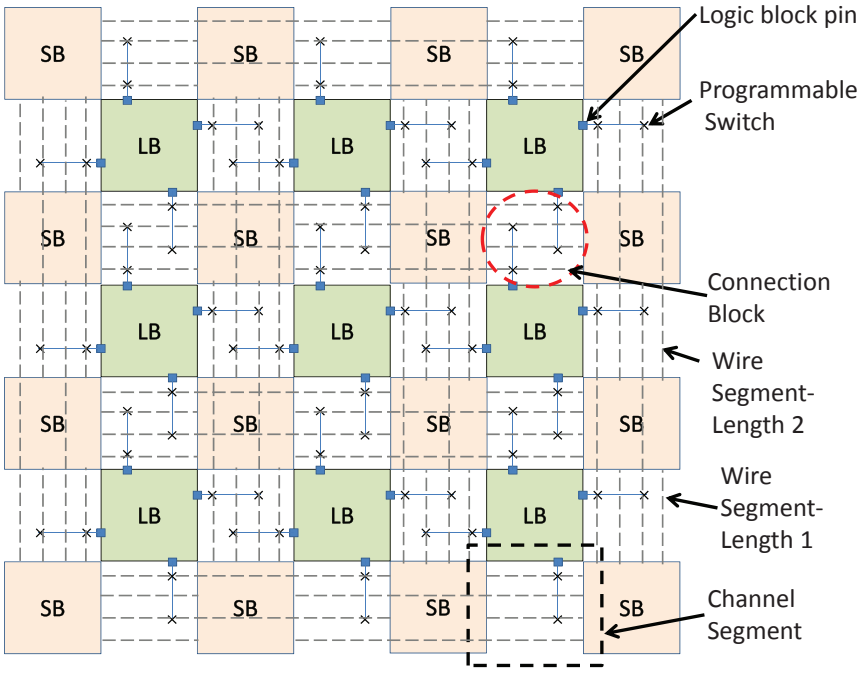


Figure 2.3: FPGA Architecture and Related Terminology.

a routing channel that spans a single logic block is known as a *channel segment*. The *channel width*, denoted by W, is given by the number of wires or tracks in a routing channel.

The input and output pins on a logic block connect to some or all of the tracks in the neighbouring channel segment via a *connection block*, which is a collection of programmable switches. The fraction of tracks in a neighbouring channel segment to which an input pin can connect is called *input connection block flexibility* or Fc_{in}. Likewise, the fraction of tracks in a neighbouring channel segment that a output pin can drive is known as *output connection block flexibility* or Fc_{out}.

At the intersection of each horizontal and vertical channel, a *switch block* is located. A *switch block* is a collection of programmable switches that allows some of the wire segments incident on to it to connect to other wire segments [2]. The number of wire segments that can be driven by a wire segment incident at a switch block is known as *switch block flexibility* or Fs.

The *length* (L) of a wire segment is defined by the number of logic blocks it spans. When L > 1, the starting points of the wire segments in a routing channel are generally staggered to enhance routability, as each logic block can then reach another logic block multiple units away by using just a single wire segment. It is assumed that all wire segments (irrespective of their length) can connect to other wire segments and logic block pins at their end points. For wire segments with L > 1, however, another architectural parameter known as *internal population* becomes relevant. If a wire segment can drive other wire segments from its interior, it is said to be *internally switch block populated*. Likewise, if a wire segment can connect to logic block pins from its interior, it is said to be *internally connection block populated*.

For the scope of this work, we will assume that the routing architecture employs the *single-driver* approach [14] in which a wire segment can only be driven from one end, as this is now the dominant commercial approach. Under this scenario, the output connection block multiplexer and the switch block multiplexer driving a wire segment are one and the same.

2.2 FPGA CAD Flow

The size and complexity of modern FPGAs make it impractical to manually configure a circuit onto them, since doing so would require correctly configuring millions of programmable switches. Undoubtedly synthesizing even a single circuit manually would take an unacceptably long time. Instead, FPGA users describe the circuit using Hardware Description Languages (HDL) such as Verilog or VHDL. The CAD tools then take this circuit description and perform many optimizations, ultimately turning it into a *bitstream configuration* file that can be used to configure the FPGA. The bitstream configuration sets the state of each programmable switch in the FPGA such that the desired circuit is realized.

Figure 2.4 shows a typical FPGA CAD flow. The first step in the flow is known as

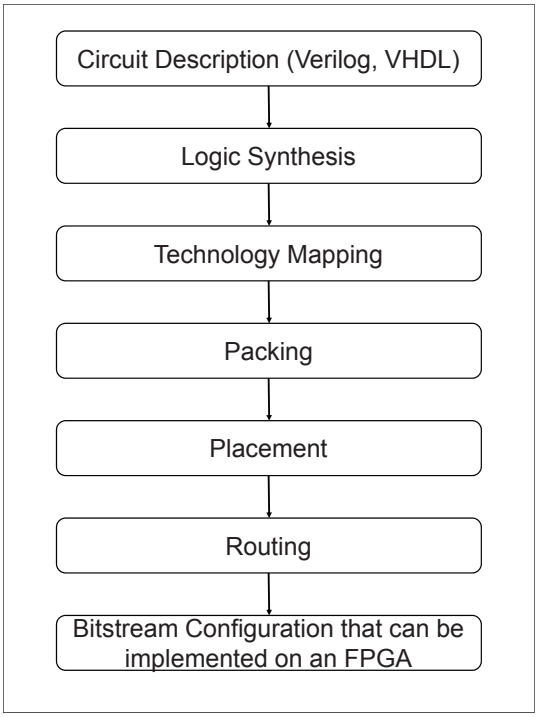


Figure 2.4: FPGA CAD flow

Logic Synthesis, and it performs two major tasks: first, it converts the circuit description in HDL into a netlist of Boolean logic gates and other primitive structures present on the physical FPGA (also known as *elaboration*). Second, it performs technology independent logic optimizations on this netlist of basic logic gates to simplify the logic and remove any redundancy. The next step in the flow, *Technology Mapping*, maps this optimized netlist of basic logic gates to primitives specific to the FPGA (LUTs and flipflops). The end result is a technology mapped netlist. The third step in the process, *Packing*, is essential when the logic blocks present in the FPGA contain more than a single primitive such as a LUT or BLE. During the *Packing* step, the technology mapped netlist obtained in the previous step is mapped onto the complex cluster-based logic blocks present on the FPGA. The optimization criteria for this step include packing connected LUTs together so as to minimize the circuit delay and the wiring between logic blocks; and to utilize each logic block to its maximum capacity so as to reduce the number of logic blocks required

to map the netlist [2]. Subsequently, the *Placement* step determines the location of the logic blocks on the FPGA that would map each of the logic blocks required by the circuit. Placing connected logic blocks together so as to reduce the amount of wiring required to connect them and minimizing circuit delay are common optimization criteria at this step.

Once the Placement step is complete and the location of each logic block in the circuit has been determined, the final step in the process, known as *Routing*, connects these placed logic blocks together by suitably configuring the programmable interconnect. FPGA routers generally represent the routing architecture using a directed graph. Wire segments and logic block input and output pins become nodes in this graph, while the connectivity between them is represented by the graph edges. Once this directed graph has been created, then the problem of routing logic blocks reduces to finding a path in the graph between the logic block pins that need to be connected. The fixed and limited number of routing resources in an FPGA dictate that this path be as short as possible. Moreover, this path must be unique for each net, since sharing of resources between nets would result in shorts. Accordingly, most routers include some sort of a congestion avoidance algorithm to resolve routing resource contention amongst nets. Additionally, some routers, known as *timing driven* routers, also aim to optimize the circuit's delay by ensuring that nets on or near the critical path are routed using short paths and fast routing resources [2].

The exploration of pin-to-wire routing in this work requires modifications to the routing step of the CAD flow, and the router employed for this work is the timing driven router of VPR 5.0.2 [15]. The following subsection summarizes the details of this router and its routing algorithm.

2.2.1 The VPR Router

VPR uses a routing resource graph [16] to represent the details of the FPGA's routing architecture [2]. Wire segments and logic block pins (input or output) become nodes in this routing resource graph. The programmable switches become directed edges; a unidirectional switch such as a buffer is represented using a single directed edge whereas a bidirectional switch such as a pass transistor is represented using a pair of directed edges between the appropriate pair of nodes [2].

Further, to model *logically equivalent* pins, such as those commonly found in FPGA LUTs and/or logic blocks, the routing resource graph in VPR contains two additional node types: *source* nodes and *sink* nodes. Logically equivalent pins are those that can be exchanged for one another without changing the functionality of the logic, like the input pins of an OR gate. Logically equivalent pins provide additional flexibility to the router. For example, since all the pins on a LUT are logically equivalent, the router can

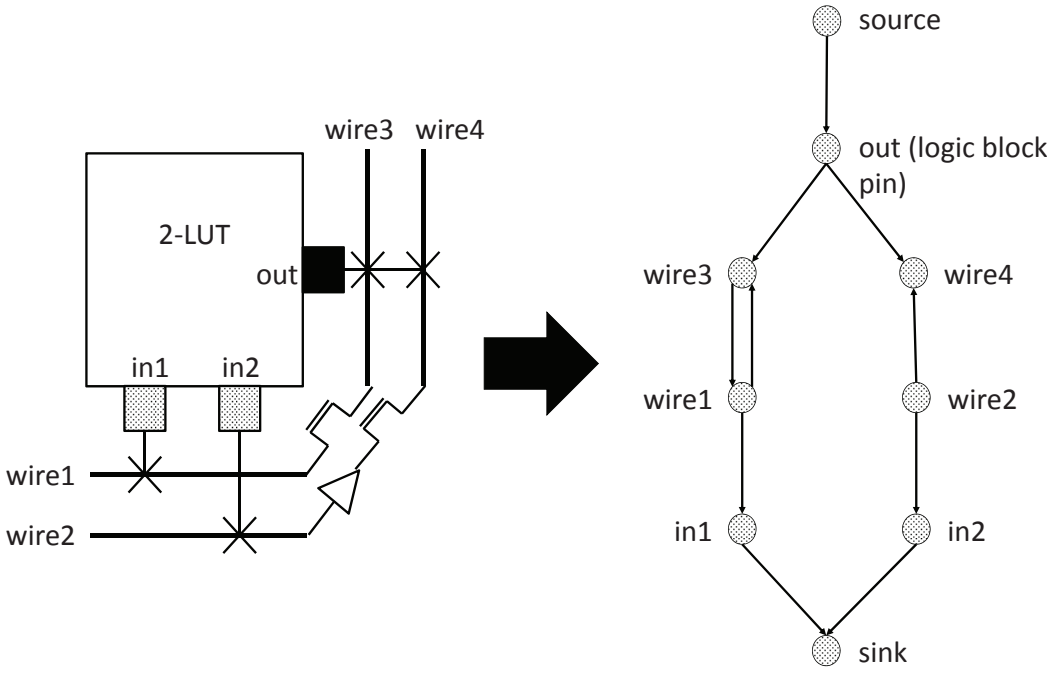


Figure 2.5: A routing resource graph corresponding to a portion of an FPGA whose logic block contains a single 2-input, 1-output LUT [2]

complete a given connection using any LUT input pin, and the LUT configuration can be altered to compensate for any re-ordering of connections [2]. The routing resource graph contains a single source node for each logically-equivalent set of output pins, and a single sink node for each logically-equivalent set of input pins [2]. A source node connects to each of its corresponding output pin nodes via directed edges. Likewise, each input pin node connects to its corresponding sink node via a directed edge. All nets begin at source nodes and terminate at sink nodes. Figure 2.5 illustrates an example routing resource graph.

The explicit connectivity information contained in the routing resource graph is not enough to perform timing driven routing and timing analysis. Hence, each routing resource graph node has additional information associated with it, including its type (wire, output pin, source etc.), location in the FPGA array, capacitance and metal resistance [2]. Likewise, each graph edge is annotated with its "switch type", so that information regarding the switch, such as its intrinsic delay, equivalent resistance etc. can be easily retrieved [2].

Once the routing resource graph has been created, the router can route nets on it by connecting the required logic block input and output pins. To route the circuit's netlist, VPR's timing driven router uses the Pathfinder [16] negotiated congestion-delay algorithm. In this algorithm, the amount of importance each connection allocates to reducing delay versus avoiding congestion is determined by how timing critical it is. Timing critical connections pay more importance to reducing delay and less to avoiding congestion. As a consequence, connections that are timing critical are routed using fast (low-delay) paths even if they are congested, while connections that are non-timing critical take slower uncongested paths. Pathfinder itself is based on Nair's [17] iterative approach. In the first iteration, nets are routed for minimum delay, permitting resource overuse. Then, the penalty of using overused resources is incremented and each net is ripped-up and re-routed. If resource overuse exists, the penalty of using overused

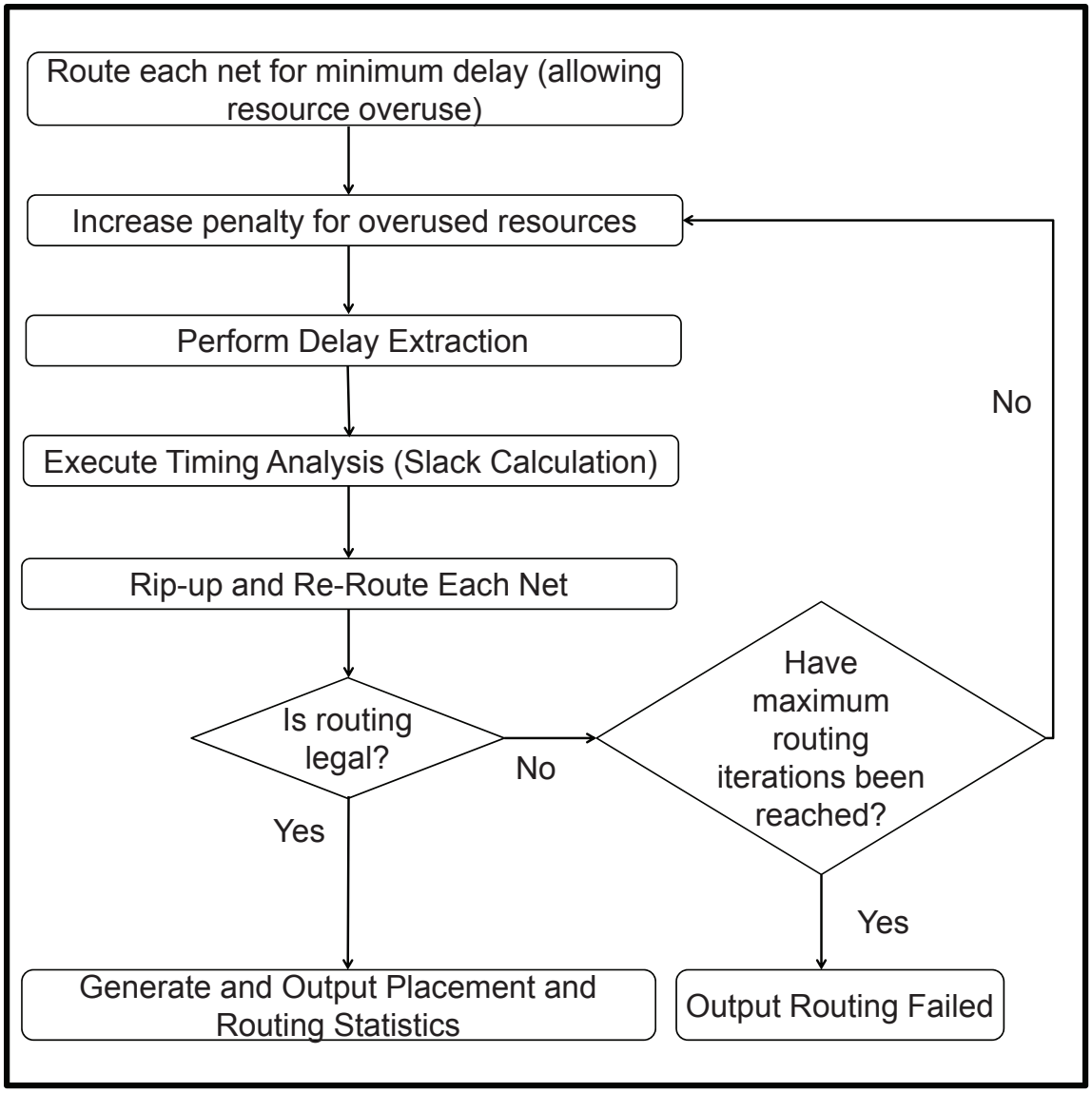


Figure 2.6: Flow of VPR's timing driven routing algorithm

resources is again incremented and another routing iteration is performed. This process of increasing penalties for overused resources and ripping-up and re-routing signals continues until a feasible routing solution is achieved or the maximum routing iterations are reached (at which point, routing is declared infeasible for the given circuit). The penalty of using overused resources gradually increases with the number of iterations. The result is that timing critical connections, which pay less attention to congestion (or resource overuse), tend to keep overused routing resources that are on their minimum delay paths whereas

non-timing critical connections, which pay more attention to congestion, are forced to move away from their minimum delay path routing resources and use other (potentially slower) routing resources, thereby resolving congestion.

In order to determine which connections are timing critical, a full timing analysis is performed at the end of each routing iteration. In VPR, the *criticality* of a source-sink connection for a net with source i and some sink j is given by the following equation:

$$Crit(i,j) = 1 - \frac{slack(i,j)}{\mathrm{D}_{max}} \text{ [2]} \tag{2.2}$$

where D_{max} represents the delay of the circuit's critical path and *slack(i,j)* represents the amount of delay that can be added to this connection before it becomes critical. The value of Crit(i,j) can lie between 0 and 1, and the higher the value, the more critical the connection. During each iteration, the delay extractor in VPR incrementally computes the Elmore delay of each net as it is being routed, using a tree describing the net's routing. At the end of an iteration, the timing analyzer uses these delays in its slack calculations. The timing analyzer can compute both the critical path of a circuit and the slack of every source-sink connection to be routed [2]. It does so using a levelized *timing graph*, which is a graph describing the circuit timing relationships. Once a routing iteration has been completed, the net delays computed by the delay extractor are back-annotated into the timing graph. Next, two breadth-first traversals of the timing graph are performed to determine the critical path and the slack of each source-sink connection [2]. In the subsequent routing iteration, the slack information is used to determine the criticality of each source-sink connection. This criticality information is then employed to determine whether a given connection focuses more on improving delay or on avoiding congestion. Figure 2.6 illustrates the flow of VPR's timing driven routing algorithm.

The timing driven router in VPR uses an A* or directed search [18] through the routing resource graph to connect the required net terminals. This A* router is based on the basic maze routing algorithm by Lee [19]. But unlike most basic maze routers that perform a breadth first search through the interconnect to connect net terminals, the

directed search router starts at the net source and expands the wavefront in the direction of the net sinks, resulting in a narrower wavefront. This ensures that the router does not spend a large chunk of its time exploring wire segments in the wrong direction. The router needs a sorted priority ordered queue to maintain a list of which nodes are to be pursued while routing. This priority ordered queue is implemented in VPR using a heap data structure. We use the heap push and pop operations performed by the router on this queue to measure the router effort. The push count measures the number of nodes that are placed on the heap as potential candidates for expansion, whereas the pop count measures the number of nodes that are actually explored to arrive at the solution.

2.3 Related Work

In Chapter 1, we introduced three major motivations for pin-to-wire routing: first, a desire to employ routing-by-abutment, as commonly done in custom VLSI, to build modular, pre-laid out systems (and thereby reduce compile time for FPGAs). Second, partial reconfiguration of FPGAs often requires that circuits in the FPGA connect by abutment. Third, pin-to-wire routing is required to make use of resources that reside within the routing fabric itself, such as the plentiful multiplexers in the fabric, or even the configuration bits themselves. In the following subsections, we will review prior work in each of these areas.

2.3.1 Routing-by-Abutment

As mentioned in the previous chapter, routing-by-abutment involves having pre-placed and pre-routed logic modules that connect together by the abutment of the wire segments at their interface.

An approach similar to this was adopted by Athanas et al. [12] to deal with the issue of connecting between reconfigurable 'sandbox' regions in an FPGA. In this approach,

reconfigurable modules are encased in wrapper structures before placement and routing. The wrapper structure provides anchor points for the module's ports, which exist at pre-defined locations so that compilation of different modules will overlap and therefore connect. At run-time, module-level placement is performed with emphasis on reducing the interconnect between neighbouring modules, and attempts to make use of routing-by-abutment. If the placement is unable to achieve abutment of all the modules that need to connect, then routing is performed using a greedy approach and is restricted to the channel segment between adjacent modules. The routing architecture is abstracted to make the routing solution fast. Both the use of anchor points and abstracted routing implicitly sacrifice area and routed wire length to achieve the routing by abutment, but the paper does not quantify how much. In this work, we are interested in measuring that cost.

2.3.2 Partial Reconfiguration

When using the partial reconfiguration feature of an FPGA, each reconfigurable region needs to be placed and routed independently, yet have connections between those regions. Researchers have previously looked at several methods of making those connections.

The partial reconfiguration design flow provided by Xilinx [13] solves the boundary crossing connection problem in a similar way, by using a fixed placement for a one-input LUT that must be the same for every dynamic logic module that is placed in a reconfigurable region. The routing to that LUT from the non-reconfigurable or static region is determined during the configuration of the static region and is kept untouched during the configuration of the partially reconfigurable region. However, routing to that LUT within the reconfigurable region can vary based on the logic and the routing of the dynamic module being implemented in that reconfigurable region. This LUT, one of which is required for every signal that crosses the boundary, is called 'proxy' logic. A clear disadvantage of this method is the logic overhead involved in implementing such

LUTs and also, the delay resulting from passing through the LUT (which is 0.4ns on a Virtex II FPGA [20]).

Koch et al. [20] solve the problem of connection between regions by pre-assigning the routes that flow between regions, eliminating the need for proxy logic. In an example implementation of a reconfigurable instruction set for a processor, they use this approach to connect 252 wires between a partially reconfigurable and a static region. Seventy-two of these wires have a fanout of 5, and so the authors note that each of these wires would require 5 proxy LUTs to cross the boundary; while the remaining wires would only require a single proxy LUT to cross the boundary. In this example, Koch's approach results in a saving of 540 LUTs, which would have been required had the proxy LUT approach been employed. Koch also points out that his method does not incur the extra delay for passing through those LUTs. As a result, both the area occupied by the proxy logic LUTs and the delay incurred for passing through them get eliminated. The authors do not, however, indicate if there is any difficulty (in terms of area, wirelength, delay and router effort) in the routing required around these pre-assigned wires, which is the purpose of this research.

2.3.3 Employing elements within the routing fabric

In this subsection we will review two prior works that make use of the circuit elements within the routing fabric: the first makes use of intra-cluster routing multiplexers, while the second uses the switch block configuration bits to implement wide shallow user memories.

Moctar et al. [21] seek to make use of the intra-cluster routing multiplexers to implement shifters for floating point mantissa alignment. This requires signals to be routed to the multiplexer's inputs in a pre-specified order which means that there is pin-to-wire routing. The authors measure the impact of this difficulty by measuring the increase in minimum routing channel width. They show that the minimum routing channel width

increases proportionally with the number of shifters in the netlist. They do not observe an increase in critical path delay, but it is also not clear how much stress their example architectures are under. Also, the benchmarks used in their work are I/O limited with relatively sparse logic block utilization, which would result in a relatively easier routing problem.

Oldridge et al. [22] use the SRAM configuration bits within a switch block to implement wide shallow user memories. A given configuration bit in the switch block can only be accessed using a single incoming wire and a single outgoing wire. This implies that in order to use a given configuration bit as user memory, one has to drive its corresponding incoming wire from a logic block output pin and drive a logic block input pin from the corresponding outgoing wire. Clearly pin-to-wire routing occurs in this situation. The authors recognize the difficulty and lack of flexibility associated with pin-to-wire routing and take certain steps at both the placement and routing level to deal with it. During the mapping stage of the CAD flow, the user circuit is mapped into switch block sized memories. However, these switch block memories are very demanding on the routing, and may in cases when the memory size equals the channel width, cause a lot of congestion in their surrounding channel segments. To prevent congestion hot spots, switch block memories are placed a certain distance apart from each other. They are also placed away from the FPGAs physical edges, as edge switch blocks tend to have fewer channel segments around them and hence can cause higher congestion. The authors tested these placement algorithm enhancements with the regular VPR timing driven router and a self-designed benchmark circuit that consisted of a 2x2 crossbar with data buffers and an input data width of 120 bits. In order to average out the dependence of the placer on the initial random seed, they iterated every place and route attempt 10 times with different random seeds. It was observed that with the normal router the benchmark circuit failed to route 5 times out of 10, even at high channel widths such as 150 or 160. To improve routability, the authors enhanced the router such that it considered all the inputs of a given switch

block memory to be logically equivalent, and which user memory was stored in which configuration bit was determined during routing. To ensure correctness, a switch block memory's outputs were routed once all its inputs had been correctly routed. Using the enhanced version of the router, it was observed that the routability of the test circuit improved considerably, increasing to as high as 9 times out of 10 for channel widths of 150 and 160. In this work, the authors faced the issue of pin-to-wire routing and dealt with it by making placement and routing algorithm modifications; however, the experiments were performed on a bi-directional routing architecture and the results were highly context specific and based only on a single benchmark circuit.

2.4 Summary

This chapter has presented relevant prior work on FPGA architecture, routing algorithms, and pin-to-wire routing methods. As discussed, prior work does not attempt to measure the various difficulties of pin-to-wire routing, which we will describe next.

Chapter 3

Experiment Design and Results

This chapter describes our design methodology, followed by our experimental setups, their corresponding results and the understanding generated from those results. It also outlines the VPR [15] code changes required to enable these experiments.

3.1 Design Methodology

Our purpose is to measure the impact of pin-to-wire routing on area, speed and router effort. We will do this in the context of a hypothetical (but fairly standard) FPGA architecture and open-source tools. The set of architecture parameters for the FPGA used in these experiments are as follows: The logic architecture is a homogeneous array of logic clusters that contain four 4-input LUTs with 10 input pins per cluster. The routing architecture employs the single-driver approach [14] with staggered length 4 segments, a Wilton switch block [23] with $F_s = 3$ and a connection block with $Fc_{in} = 0.15$ and $Fc_{out} = 0.25$. Note that all length 4 wire segments are internally switch block populated [2]. The resulting switch block multiplexers range in size from 9-10 data inputs. The delays are based on the iFar repository delays for a 45nm CMOS process [24] [25]. In the following, we will also make use of the classical 20 largest MCNC benchmarks [26]. Although these benchmarks are relatively small, we expect that because there are many

wires and switches, we will gain useful insight into the pin-to-wire question. Nevertheless, the effect of fixed routing structures like carry chains would remain unknown, and be a part of the future work.

Table 3.1 lists the set of circuits used in the experiments, and their characteristics -

Table 3.1: Net and BLE Statistics

Circuit	# BLE's	Total Nets	Single-Sided Nets		Double Sided Nets
alu4	1522	1019	369	370	280
apex2	1878	1408	588	517	303
apex4	1262	959	331	332	296
bigkey	1707	1036	375	454	207
clma	8383	6133	2725	2663	745
des	1591	1499	416	939	144
diffeq	1497	1179	476	489	214
dsip	1370	919	379	371	169
elliptic	3604	2449	1066	887	496
ex1010	4598	3410	1534	1420	456
ex5p	1064	859	262	273	324
frisc	3556	2279	907	873	499
misex3	1397	996	406	339	251
pdc	4575	3178	1266	1174	738
s298	1931	1011	413	397	201
s38417	6406	5045	2304	2342	399
s38584.1	6447	4704	2187	2001	516
seq	1750	1286	492	452	342
spla	3690	2539	1046	902	591
tseng	1047	827	348	318	161

the number of logic elements, the number of nets and several other attributes described in the discussion below. To gather these statistics, the benchmark circuits were technology-mapped using SIS [27], packed using T-V pack [28] and placed and routed using VPR 5.0.2 [15]. We will now present measurements of pin-to-wire routing in a number of different experimental contexts. Most of these contexts are meant to be proxies for the wiring-by-abutment motivations presented in the Introduction, with varying degrees of difficulty.

3.2 Experiment 1: Easy Abutment-Oriented Routing or *Basic* Pin-to-Wire Routing

The first experiment is as follows: we begin by taking a benchmark circuit and run packing using T-VPack [28], followed by placement and routing using VPR 5.0.2 [15], which employs the timing-driven router described in [2]. This flow is used to determine the minimum number of tracks per channel required to route; we then set the track count to be 30% higher than this minimum (as is common [15]) for all experiments with this circuit. However, we will explore the effect of this assumption later in the chapter. The placement and routing step is then re-run at this track count, and we measure the critical path delay, wirelength and router effort. These measurements form the *base* measurements for that circuit. Table 3.2 gives the geometric mean (over the entire benchmark suite), for each metric, for the base routing.

Table 3.2: Average Base Measurements acquired on the Base Routing

Routing Problem	Wirelength (x 10^3 LBs)	Critical Path Delay (x 10^{-9} seconds)	Heap Push Count (x 10^6)	Heap Pop Count (x 10^6)
Base	397	4.68	21.6	2.26

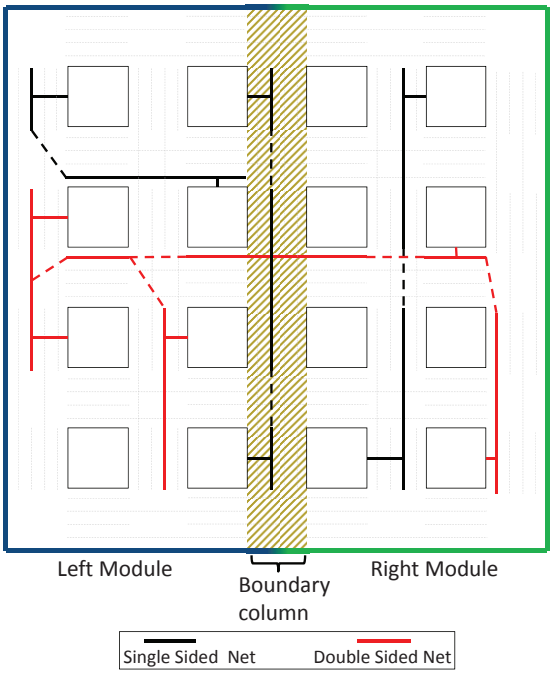


Figure 3.1: Partitioned Layout

Next, to serve as a proxy for routing by abutment, we divide the circuit in half, by drawing a vertical line down the middle of the circuit (creating a 'middle' or 'boundary' column of the layout) as illustrated in Figure 3.1. This partition gives rise to two modules which we will call *Left* and *Right*. We will route these, in a sense, by abutment. The placement of the Left and Right modules will remain intact from this original placement. In our experiments, we will deal with two kinds of nets from this placement: those whose sources and sinks reside wholly within the Left or Right modules (*single-sided* nets) and those that cross from one side to the other (*double-sided* nets). We will use the double-sided nets to model the inter-module nets in the routing-by-abutment approach. For each circuit, Table 3.1 gives the number of nets that are on the left side, right side, and the number that are double-sided.

In the first experiment, we re-run the routing of this same placement, but with the following restrictions:

1. All single-sided nets are restricted to be routed on their respective sides, as would

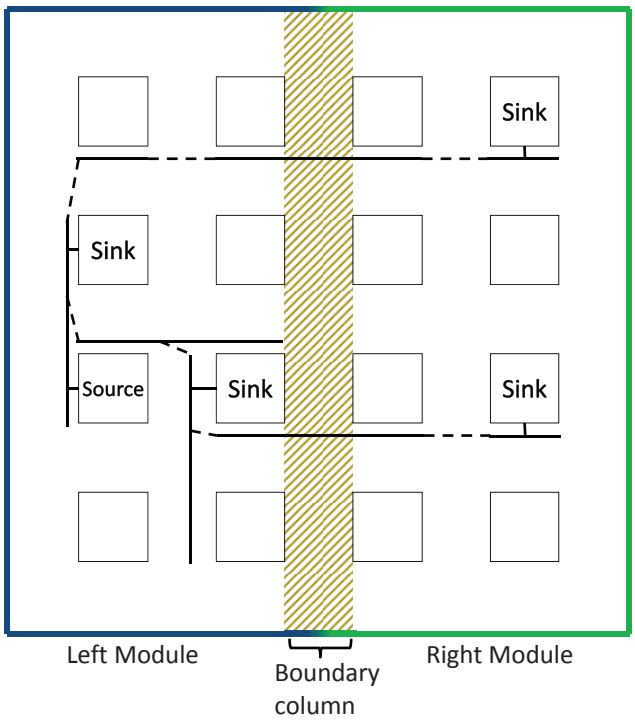


Figure 3.2: A Multiple Crossing Double-Sided Net in the Original Netlist

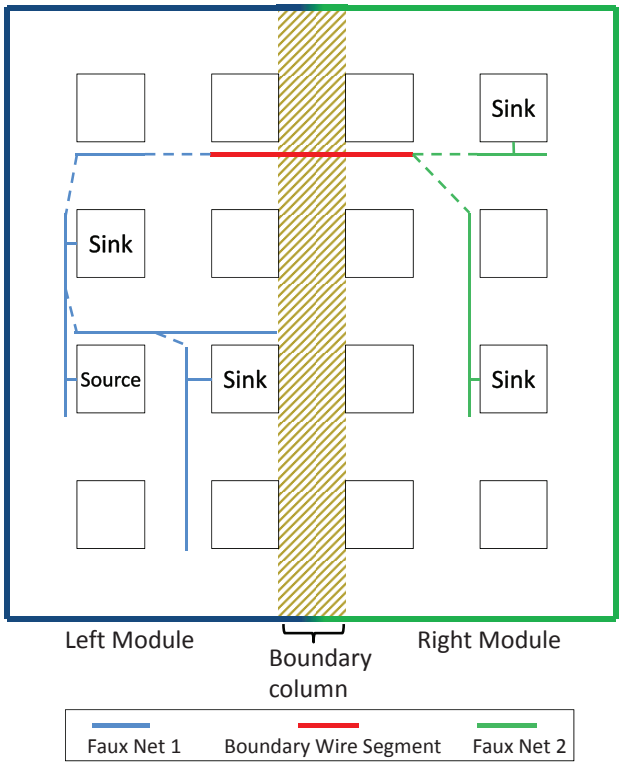


Figure 3.3: Faux-Nets built from Net shown in Figure 3.2, now with just one crossing point

be true in modules routed by abutment. (In the base routing, these nets could have travelled across the boundary and back).

2. Every double-sided net will be split into two nets, called *faux* nets, one on the Left and one on the Right. In addition, one of the wire segments in the base routing of the double-sided net that crosses the middle column will be chosen as the *boundary* wire segment for that net, as illustrated in Figure 3.2. If the source of the net is on the left side of the boundary, then the left side faux net consists of the source, all the left side sinks, and the boundary segment acting as a net sink. The right side faux net consists of the boundary wire segment (acting as a *source*) and the remaining right side sinks. (If the source is on the right side, then these constructions reverse appropriately). The constructed faux nets for Figure 3.2 are illustrated in Figure 3.3. Note that even though the base routing of the original net may have crossed the middle more than once, in the new routing scenario it will only cross once, as illustrated between Figures 3.2 and 3.3. This experiment design decision, made to be similar to what would happen in routing-by-abutment, has specific side-effects discussed in the results below.

We will also consider two different orderings of the routing process: the first is called *Contiguous Net Ordering* (CO), in which nets are routed in the same order as the base routing, and where the two faux nets that make up each double-sided net are routed one after the other. We are interested in this base case, as we would generally expect the results of this approach to be the most similar to the original base routing. The second net ordering more directly reflects what would happen in routing-by-abutment, and is called *Partition-Wise Ordering* (PWO): here all the nets in the netlist are rearranged such that the Left nets are routed before the Right nets. This includes the faux nets, and so all Left faux nets are routed together with left-sided single nets.

In all cases, the routing algorithm used is timing-driven. For the base case, the original timing-driven VPR algorithm is used, and the circuit is run through timing analysis as

usual to determine the critical path. However, to enable pin-to-wire routing, i.e. the case where the double-sided nets are split into two faux nets, changes are made to the routing resource graph and the timing driven router. These changes are described in the following subsection.

3.2.1 Algorithmic Modifications to the Routing Resource Graph and Timing Driven Router

To enable pin-to-wire routing, modifications to the VPR code are introduced in three stages: before the routing step (*pre-processing* changes), at the routing step (*routing algorithm changes*) and finally, after the routing step (*post-processing* changes). The following paragraphs describe each of these in detail.

Pre-processing changes are changes that are performed before the routing step, to prepare the netlist and the routing resource graph for pin-to-wire routing. The first step in the pre-processing stage is to acquire the required wire segments, since they are essential for pin-to-wire routing. To select these wire segments, code is added to VPR. The code parses the base routing to acquire the desired set of boundary wire segments. After the boundary wire segments have been procured, pin-to-wire routing is introduced in the netlist. Introduction of pin-to-wire routing in the netlist is done in two steps. First, we modify the routing resource graph so that the wire segments can act as both sources and sinks. Second, each net in the netlist for which a boundary wire segment has been selected is split into faux nets and the net's sinks are distributed between these faux nets. In addition, sources and sinks corresponding to the desired boundary wire segment are added as the faux nets' terminals. For one faux net, the source associated with the boundary wire segment is added as a terminal, while for the other faux net the sink associated with the boundary wire segment is added as a terminal. Additional code is added to ensure that the net ordering can be changed between CO and PWO. Once the new netlist has been generated, one last change is performed to prepare for

the routing step: the bounding box generation is modified such that the faux nets can calculate their bounding boxes using the boundary wire segments as terminals. Further, for the experiments that simulate a routing-by-abutment scenario, the bounding boxes of the nets are modified so that they are restricted to their designated modules. At this stage the pre-processing of the netlist and the routing resource graph is complete.

The next phase of changes are the *routing algorithm changes*. These changes describe the modifications made to the timing driven router so that it can correctly handle pin-to-wire routing. Changes are made in two major areas: inside the cost functions, and before the timing analyzer. First, we discuss the changes made to the cost functions. In our experiments, we allowed the two faux nets corresponding to a single net to share the boundary wire segment without triggering a resource overuse. To enable this sharing, the cost functions were modified such that when the two faux nets corresponding to a single real net reused the boundary wire segment, it would not be considered a resource overuse; but if faux nets or even any other normal net attempted to share this boundary wire segment, it would count as a resource overuse. Next, we discuss the changes made prior to timing analysis. In actual routing-by-abutment, each logic module would be placed and routed separately and the timing constraints for the pin-to-wire or wire-to-pin nets would be derived using some estimation technique. However, instead of doing that, to keep things simple, we perform timing analysis on the complete nets, like would have been in pin-to-pin routing. The timing analyzer (as described in the background subsection 2.2.1) is left unchanged, it only performs timing analysis on pin-to-pin connections. As described in that subsection, the timing analyzer uses the delays of each source-sink connection to compute the net slacks. But the delay calculations, which are performed as the net is being routed, are performed on the nets of the new netlist. Hence, additions had to be made to the code to calculate the net delays of the original nets using the net delays of the new nets. Net delays of the two faux nets corresponding to a single original net were combined to produce the net delay of the original net. Care is

taken so that the delay for passing through the boundary wire segment is only accounted for once. These complete net delays were then used for timing analysis by the timing analyzer, which returned the slacks for each source to sink connection in the original netlist. These net slacks were then used to allocate slacks to the source-sink connections of the new netlist. The slacks for the pin-to-pin connections were copied as such from the original net slacks. However, the slacks assigned to the faux nets need to be set carefully. The slack assigned to the source-to-boundary connection is set to be the worst case (smallest) slack for all of the downstream boundary-to-sink connections on the other faux net. This is to ensure that the most critical source-sink connections on the other side are fed by a net that is treated as equally critical, optimizing the speed in the most appropriate way. Another minor change in this phase involved adding counters to the code to measure heap push and pop counts.

Once the routing step is complete, the *post-processing* changes derive the routing of the original netlist by processing the routing of the new netlist. The routing of the faux nets corresponding to a single original net are merged back, while ensuring that the merged net routing maintains a tree structure. The merged routing is then subjected to checking, and placement and routing statistics are derived from it. The combination of all the above changes enable the pin-to-wire experiments discussed in this chapter.

3.2.2 Experiment 1 Results

Before discussing the results, it is important to note that the selection of the boundary wire segment turned out to be quite important - making poor choices for the set of boundary segments could result in a deterministically unroutable circuit. (We also take this as evidence that pin-to-wire routing can cause immediate problems). We found that this unroutable situation occurred primarily in three ways: The first set of issues occurred when there were several boundary segments in close proximity - if there are too many that are too close together, they simply block each other from routing success. Also,

Table 3.3: Percentage Increase in Average Metrics for the Easy Abutment Routing Experiment

Net Order	Wirelength	Critical Path Delay	Heap Push Count	Heap Pop Count
CO	-4%	5%	13%	104%
PWO	-4%	5%	11%	99%

if an input or an output pin of a logic block is close to a set of unassociated boundary segments (boundary segments assigned to nets that neither terminate nor originate at this pin), then that pin might simply be blocked. So it is important to make sure that the boundary segments are not too close together. The second issue occurred if the boundary segment of an unidirectional wire simply pointed in the wrong direction, (left to right, say) when the net needed to go the other way. The third issue occurred when both the faux nets wanted to expand using the boundary segment and the boundary segment had insufficient exit points, resulting in resource contention between the nets and hence, routing failure.

For each of the cases (base, modified CO, modified PWO), we measure the total wirelength required after routing, the critical path delay, and the *router effort*. As mentioned in Subsection 2.2.1, the router effort, at a fixed channel width, is measured as the number of heap push and pop operations performed by the VPR router. The heap in VPR contains the priority-ordered list of wiring segments to pursue while routing. The push count measures the number of nodes that are placed on the heap as potential candidates for expansion, whereas the pop count measures the number of nodes that are actually explored to arrive at the solution.

Finally, we note that two of the circuits, *bigkey* and *diffeq*, failed to route in this experiment (both for the CO and PWO net orderings), likely due to the reduced flexibility of the wire sources and sinks.

Table 3.3 gives the experimental results: the first column indicates which net ordering was used, while the subsequent columns give the percentage increase in geometric mean (across all benchmarks, relative to the base case), for total routed wirelength, critical path delay, heap push count, and pop count, respectively. Please note that for these, and any following computations, metric measurements relating to circuits that failed to route were not included in the average. Interestingly, for the contiguous net order case (CO), the total wirelength is actually lower than the base case. While we were surprised by this, there are two explanations: first that the router is given a good starting point (one of the crossing points in the base case) and so can use its effort to find better routes. The second is that, for the double-sided nets, multiple crossings are forbidden in the approach described above, and so the total net length may end up shorter, but sacrificing a better connection for the more critical path. The latter hypothesis is borne out by the increase in critical path delay. The most telling result is that the heap pop count has increased by a factor of two. In other words, the router effort has significantly increased. We conclude that the smaller number of choices available to the pin-to-wire connections in the faux nets has increased the effort needed to succeed in routing. It is interesting, also, to note that the routing order of the nets has little effect on the results, likely due to the inherently iterative nature of the Pathfinder routing algorithm [29].

3.3 Experiment 2: Harder Abutment Routing or *Perturbed* Pin-to-Wire Routing

In experiment 1, the boundary nodes chosen were set to be the ones that the were chosen in the original routing, base case. In experiment 2, we make one simple change to the above. Here we explicitly choose a different segment than the one chosen in the original routing, but in roughly the same row/column location of the original boundary segment. The code implements this by starting at the original boundary wire segment and then alternately testing wire segments lying above or below this wire segment (in the same

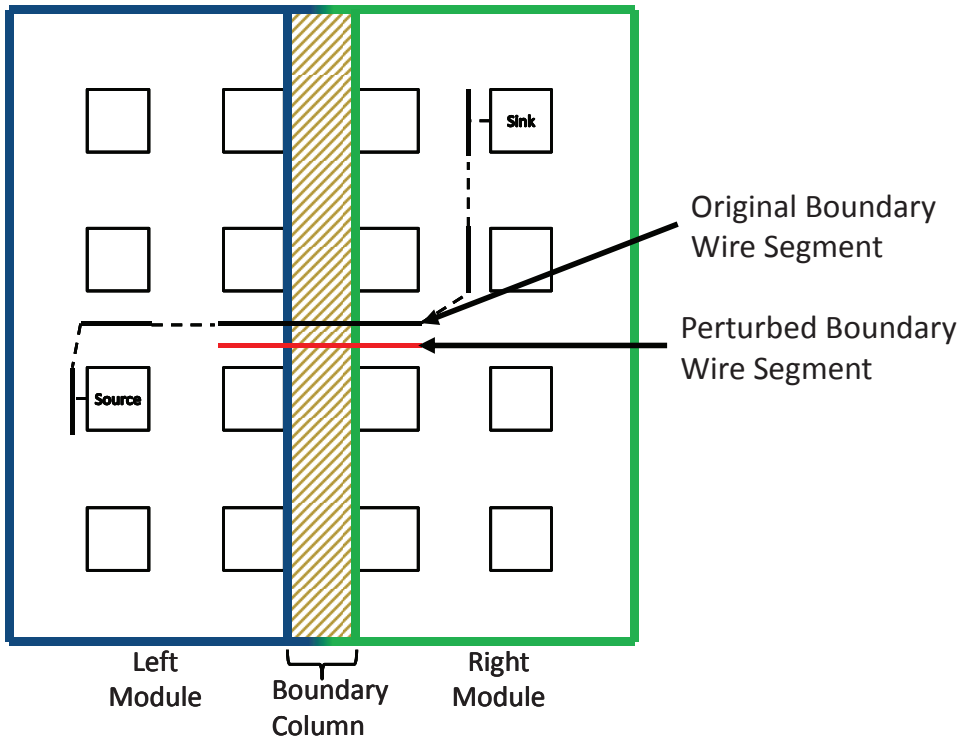


Figure 3.4: Selecting a Perturbed Boundary Wire Segment for a Double Sided Net

channel segment), to check if they meet all the constraints as outlined in Subsection 3.2.2. The first wire segment either above or below the given wire segment that satisfies these constraints is selected as the new boundary wire segment. For vertical wire segments, the code searches similarly in the left and right directions. As illustrated in Figure 3.4, by doing so we are explicitly selecting a *different* segment than was known to have already worked. We expect that this is more like the typical pin-to-wire routing situation faced in abutment-style routing, where knowledge of a previous full route is not known. Table 3.4 gives the same summary results of this case, again compared to the base case, as in Table 3.3. It is interesting to see that the results are dramatically different: the average wirelength *increases* by about 6% (rather than decreasing) and the critical path delay increases by about 16%, a significantly worse change. Furthermore, the router is now working almost three and a half times harder in terms of heap pop counts, and 66% harder on heap pushes. Clearly, there is some pain and suffering dealing with these boundary routes (which represent on average 19% of all of the nets in Table 3.1). However, it is also

Table 3.4: Percentage Increase in Average Metrics for the Harder Abutment-Oriented Routing Experiment

Net Order	Wirelength	Critical Path Delay	Heap Push Count	Heap Pop Count
CO	6%	16%	67%	264%
PWO	6%	16%	66%	255%

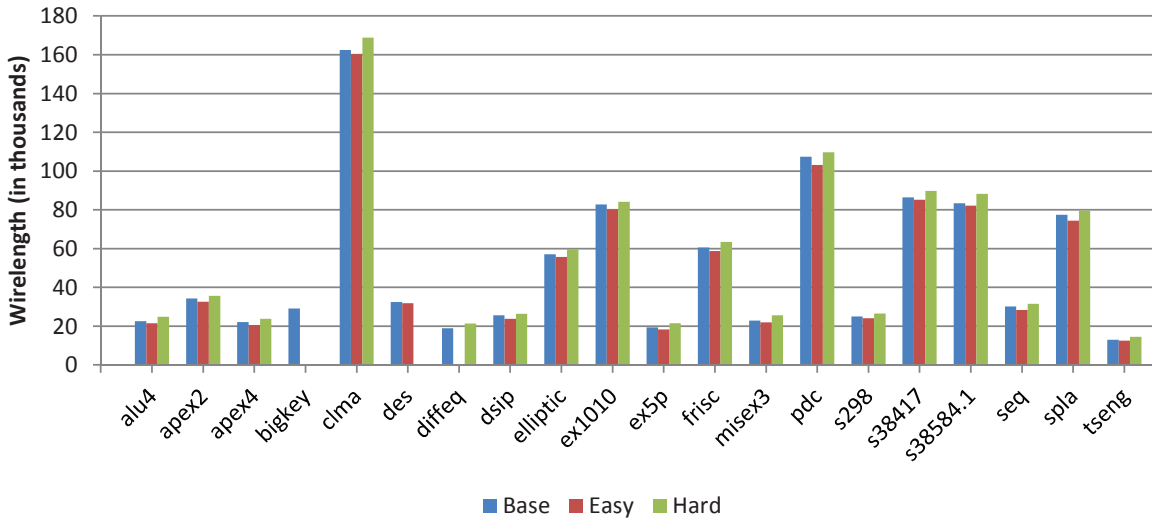


Figure 3.5: Wirelength for Expts 1 and 2

interesting to note that these nets did mostly successfully route (with two exceptions, of the circuits *bigkey* and *des*, both of which had a faux net for which no connecting path existed), and as long as the boundary wire nodes are not too congested among themselves, pin-to-wire routing can succeed at some cost.

Figures 3.5, 3.6, 3.7 and 3.8, give the circuit-by-circuit results for experiments 1 and 2 in wirelength, critical path delay, heap push count and heap pop count (we give results for partition-wise ordering of nets only, since they were similar to the contiguous ordering case).

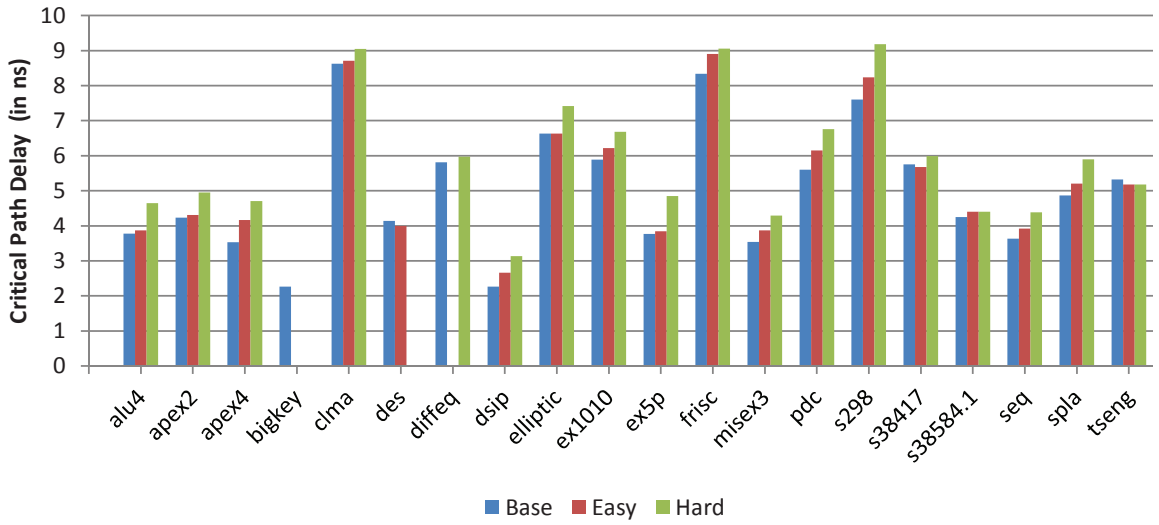


Figure 3.6: Critical Path Delay for Expts 1 and 2

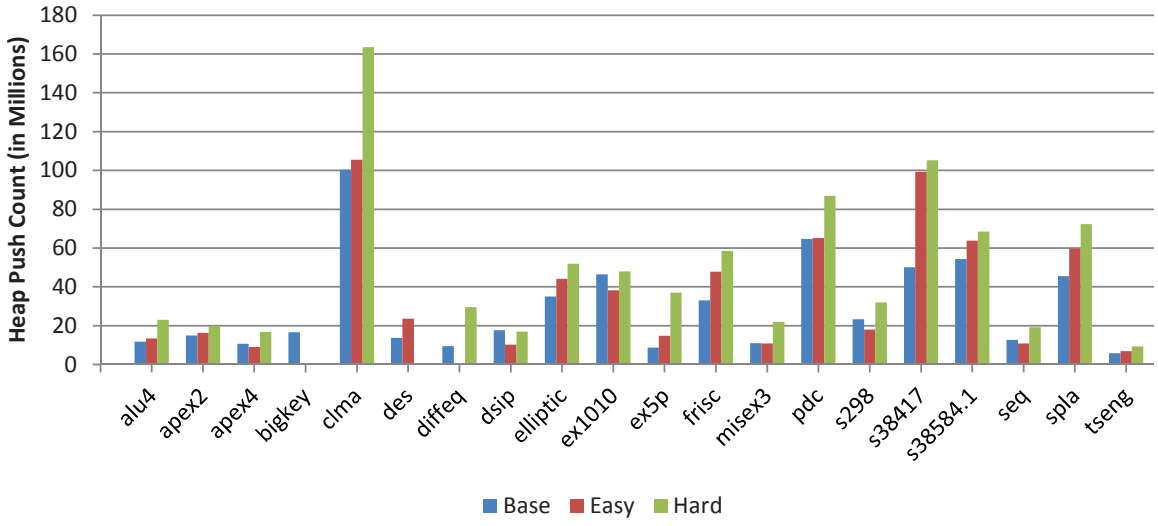


Figure 3.7: Heap Push Count for Expts 1 and 2

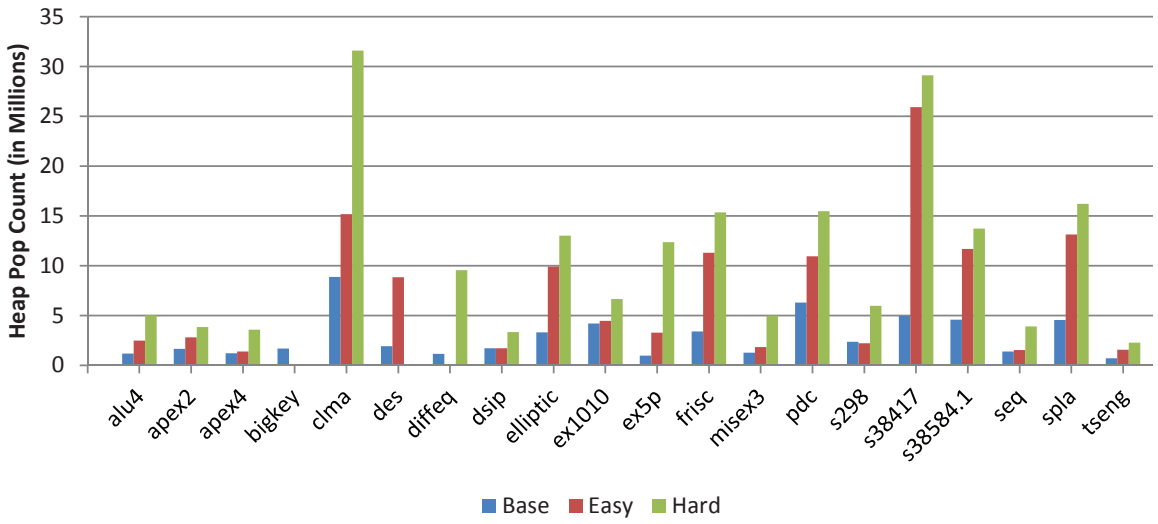


Figure 3.8: Heap Pop Count for Expts 1 and 2

3.4 Experiment 3: Dispersed Pin-to-Wire Routing

The previous experiments focused on modelling the routing-by-abutment motivation presented first in the Introduction. Nevertheless, pin-to-wire routing is also motivated by other applications attempting to use the plentiful multiplexers present in the FPGA's routing fabric [21] or to implement wide shallow memories using switch block configuration bits [22]. Both these applications will benefit from knowing how much pin-to-wire routing can be introduced in a netlist before the area, delay and routability of the routed netlist are significantly impacted. We suspect that introducing a small amount of pin-to-wire routing in a netlist may not be significantly detrimental to the final routing, but that it will become more difficult as the amount is increased.

The next experiment allows the amount of pin-to-wire routing being introduced in a netlist to vary. Once the base measurements have been acquired as described in subsection 3.2, the routing step is re-run on the same placement but with these modifications:

1. A fraction of nets from the netlist are selected, which sets the amount of pin-to-wire routing in a netlist.

2. For each selected net, a wire segment lying near the centre of the net's bounding box in the base routing of this net is chosen as the *target wire segment* for that net. (This is similar to the *boundary* wire in experiments 1 and 2, except that it is not constrained to be in the middle channel). This is illustrated in Figure 3.9.

3. The selected net will then be split and replaced by two faux nets. The sinks of the net that lie on the same side of the target wire segment as the source of the net form the sinks of the first faux net, with the target wire acting as a sink. The remaining sinks form the part of the second faux net, for which the target wire segment acts as a source. The constructed faux nets for the net in Figure 3.9 are illustrated in Figure 3.10.

4. The nets are routed in the same order as in the base routing.

The experiment was run with four cases, where the number of nets selected ranged through 20%, 30%, 50% and 100%.

Through this experiment we measured how an increasing fraction of selected nets, each of which contains the pin-to-wire and wire-to-pin connections described above, would affect the same metrics used in experiments 1 and 2. Figure 3.11 plots the percentage increase in geometric mean (across all benchmarks, relative to the base case) for wirelength and critical path delay against the fraction of all nets that are split (as described above). Here we can see behaviour similar to Experiment 1, which shows the wirelength decreasing (from 1% to 7%), and the critical path delay increasing from 4% to 16%. The forced split again serves to prevent extra wire from yielding a reduced critical path. Figure 3.12 plots the percentage increase in heap push and pop counts as the percentage of split nets (and hence, pin-to-wire routing) increases. Here we also see a significant increase in the router effort, in the worst case, the heap push count is doubled while the heap pop count is sextupled. Clearly, the router is having to work much harder to make these pin-to-wire connections.

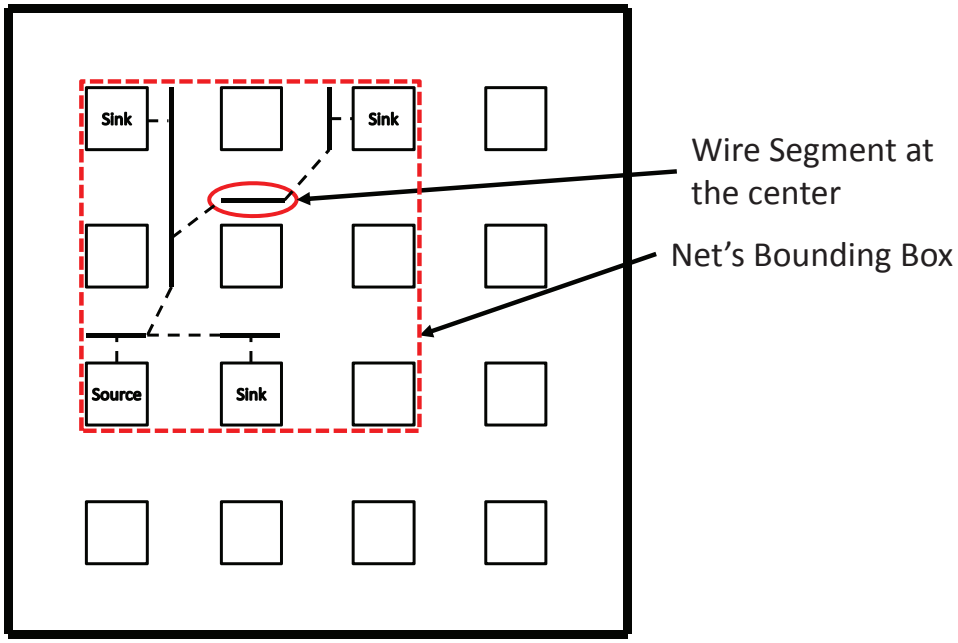


Figure 3.9: Selecting a target wire segment from the base routing of a net for the Dispersed Experiment

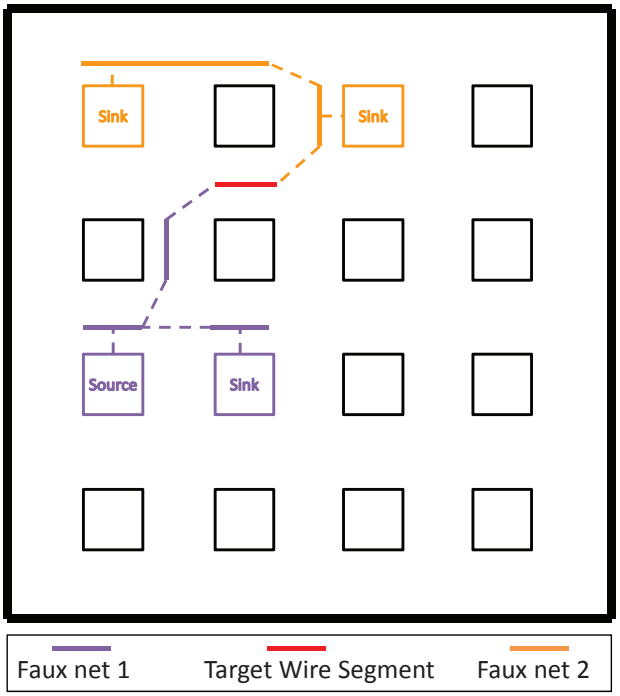


Figure 3.10: Faux-Nets built from the net shown in Figure 3.9

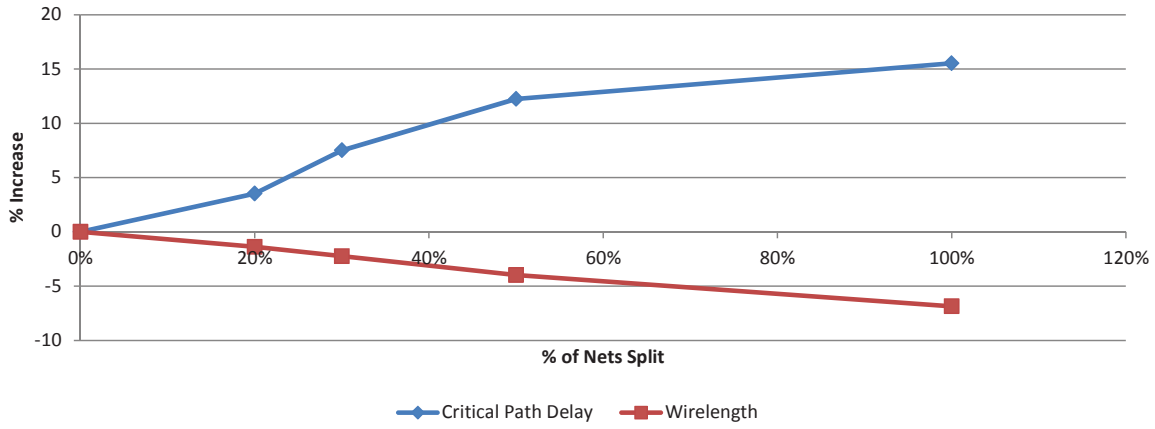


Figure 3.11: Percentage increase in geometric mean of Dispersed routing over Base routing as a function of amount of pin-to-wire routing, for the wirelength and critical path delay metrics

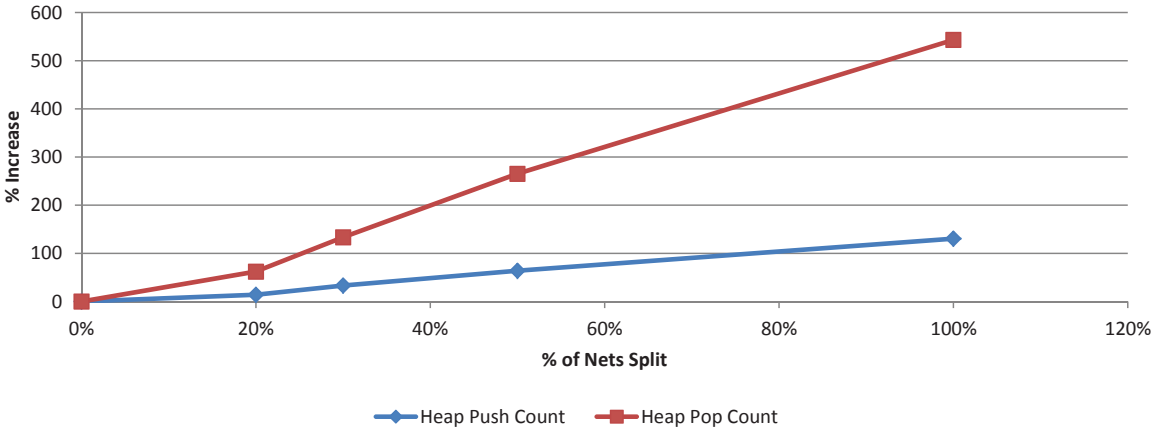


Figure 3.12: Percentage increase in geometric mean of Dispersed routing over Base routing as a function of amount of pin-to-wire routing, for the heap push and pop count metrics

It is interesting to note that only one circuit suffered failed to route across all of experiment 3 - the circuit *des* when 100% of the nets were selected (the reason was that there were too many targets in close proximity, which is a problem as described above). Also, we calculated the average fraction of all of the used segments that were being 'fixed' as target segments in each experiment. When the percentage of nets split ranges from 20%, 30%, 50% to 100%, the fraction of target wire segments that make up all of the routed segments ranges from 2.2%, 4.3%, 9.0% to 18%.

3.5 Experiment 4: Harder Dispersed (or *Dispersed Perturbed*) Pin-to-Wire Routing

In experiment 3, the target wire segment was set to be one of the wire segments in the net's original routing chosen in the base case, lying near the centre of the net's bounding box. In this experiment, we explicitly select a target wire other than the original target wire but in roughly the same row/column location. This is similar to the increase in difficulty faced in experiment 2, and more like the expected pin-to-wire situation as described there.

Figure 3.13 plots the percentage increase in geometric mean (across all benchmarks, relative to the base case) for wirelength and critical path delay against the fraction of all nets that are split (as described above). The results are quite different from experiment 3 – the wirelength increases from 1% to 15%, and the critical path delay increases from 8% to 33%. Figure 3.14 plots the percentage increase in heap push and pop counts as the percentage of split nets (and hence, pin-to-wire routing) increases. There is also a significant increase in the router effort, in the worst case, the heap push count is quadrupled while the heap pop count increases by a factor of 14. Compared to experiment 3, eight circuits failed to route when 100% of the nets were selected, while three circuits failed to route when 50% nets were selected. No routing failures were observed when 20%

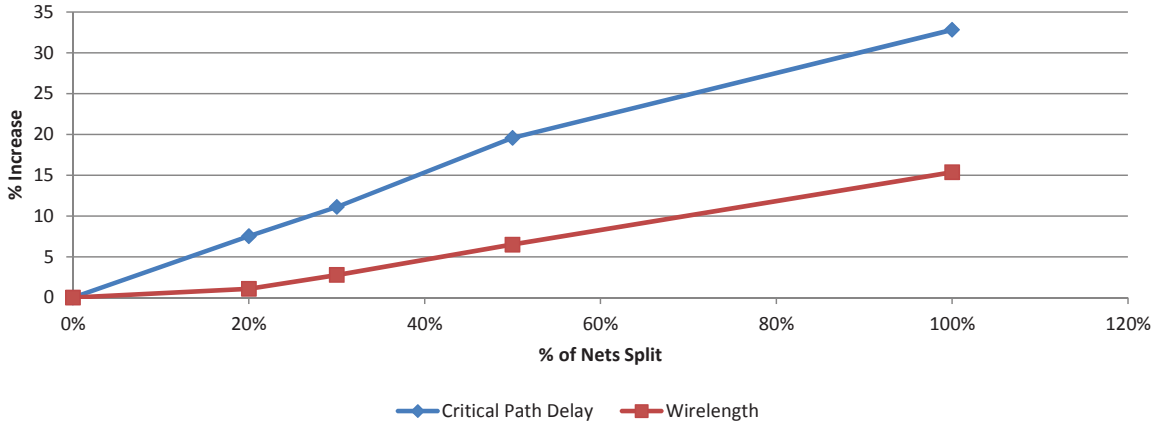


Figure 3.13: Percentage increase in geometric mean of Dispersed Perturbed routing over Base routing as a function of amount of pin-to-wire routing, for the wirelength and critical path delay metrics

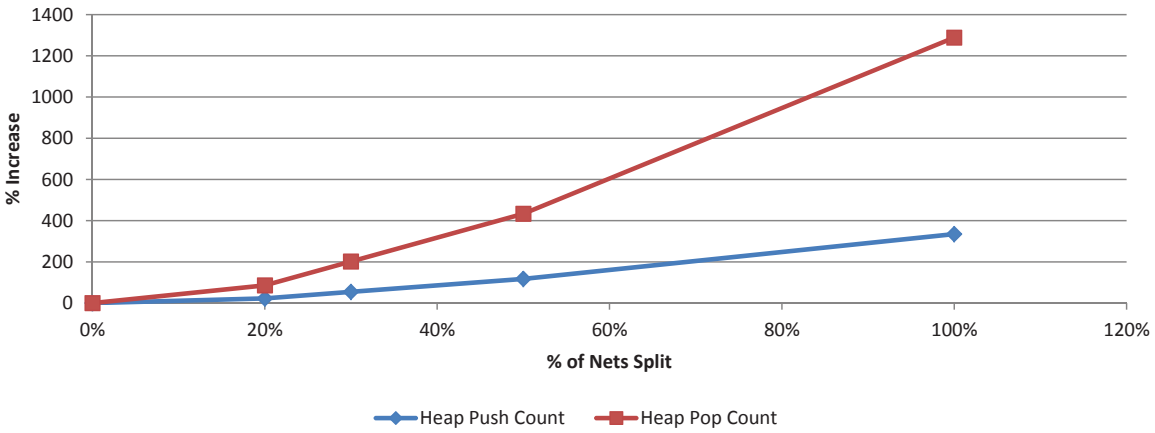


Figure 3.14: Percentage increase in geometric mean of Dispersed Perturbed routing over Base routing as a function of amount of pin-to-wire routing, for the heap push and pop count metrics

or 30% nets were selected. These failures in routing were a result of congestion, either near the target wire segments or around the logic block input pins. This data indicated that the router has to work extremely hard to make these pin-to-wire connections.

3.6 Experiment 5: Effects of changing flexibility by varying channel width

In all the experiments that we have performed so far the channel width used for each circuit is 30% higher than its minimum channel width. This extra channel width is allocated in accordance with common experimental practice; however, we realize that this extra channel width puts all our benchmark circuits under relatively low stress when subjected with pin-to-wire routing. As we mentioned previously, FPGAs are architected such that circuits with even a very high demand for routing can route successfully. This raises the question: what happens when circuits with high demands for routability, i.e. circuits that are under high stress are subjected to pin-to-wire routing. Can such circuits afford the cost of pin-to-wire? It also makes us curious to know what happens when circuits under very low stress are subjected to pin-to-wire routing. To answer these questions, we will use two of our most realistic previous experiments: Perturbed and Dispersed Perturbed. For the Perturbed Experiment, we will only consider the PWO net ordering, as results of the Perturbed Experiment in Section 3.3 indicate that the net ordering has a negligible impact on the results. Similarly, for the Dispersed Perturbed Experiment we will only consider the case in which 100% of the nets are selected for splitting, i.e. when the circuits are subjected to the highest amount of pin-to-wire routing. We will repeat these experiments while varying the channel width from zero to 50% higher than minimum and observe the results. As the track count increases, we expect the routability of the circuits to improve and the gap between pin-to-pin and pin-to-wire routing to reduce.

In the following two subsections, we will study the impact of increasing track count on Perturbed and Dispersed Perturbed pin-to-wire routing.

3.6.1 Effects on Perturbed Pin-to-Wire Routing

As mentioned above, we repeat the Perturbed pin-to-wire routing experiment while gradually increasing the track count from minimum to 50% higher than minimum. At the minimum track count, circuits are under very high stress while at 50% higher than minimum track count circuits are under relatively low stress. Figure 3.15 plots the number of *unroutes* as a function of increasing track count for the Perturbed pin-to-wire routing experiment. Henceforth, we will use the term *unroutes* to refer to circuits that failed to route. As can be seen from the graph, at minimum track count, when circuits are under maximum stress, only 2 of the 20 benchmark circuits succeed in routing. But slightly increasing the track count by 10% enables 9 of the 20 benchmarks to succeed routing. The plot shows what we expected, that routability continues to improve as channel width increases. The increase in unroutes from 0 to 1 when increasing the extra tracks from 40% to 50%, is due to the dependence of pin-to-wire routing on the target wire segments. An unfortunate selection has resulted in the single unroute, but one can conclude that increasing the track count certainly improves routability.

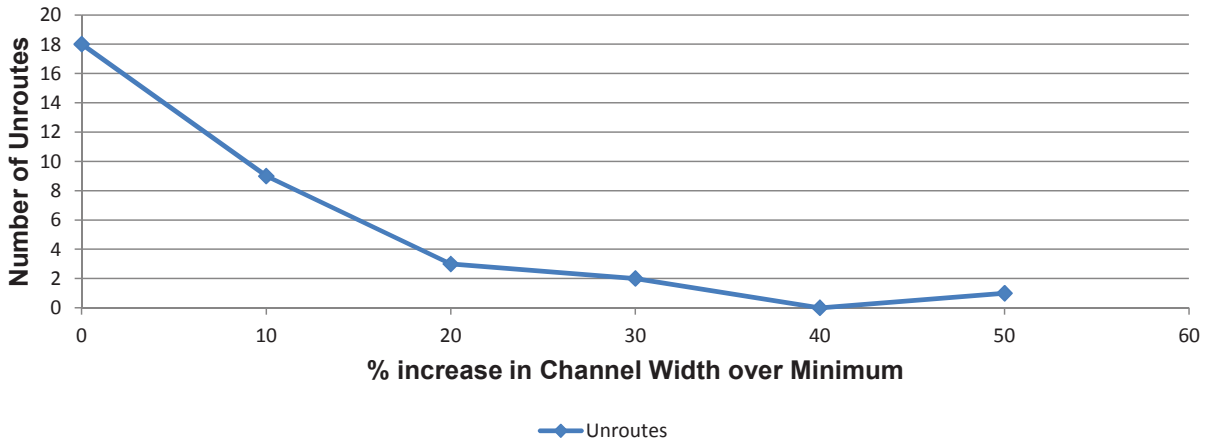


Figure 3.15: Number of Unroutes (out of 20) as a function of increasing track count for Perturbed pin-to-wire routing

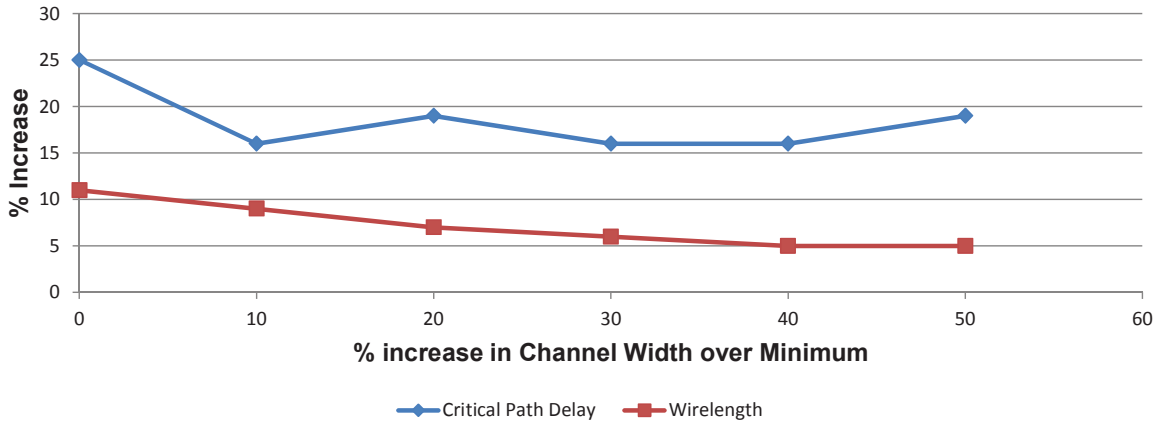


Figure 3.16: Percentage Increase in Geometric Mean of Perturbed routing over pin-to-pin routing, as a function of increasing channel width, for the wirelength and critical path delay metrics

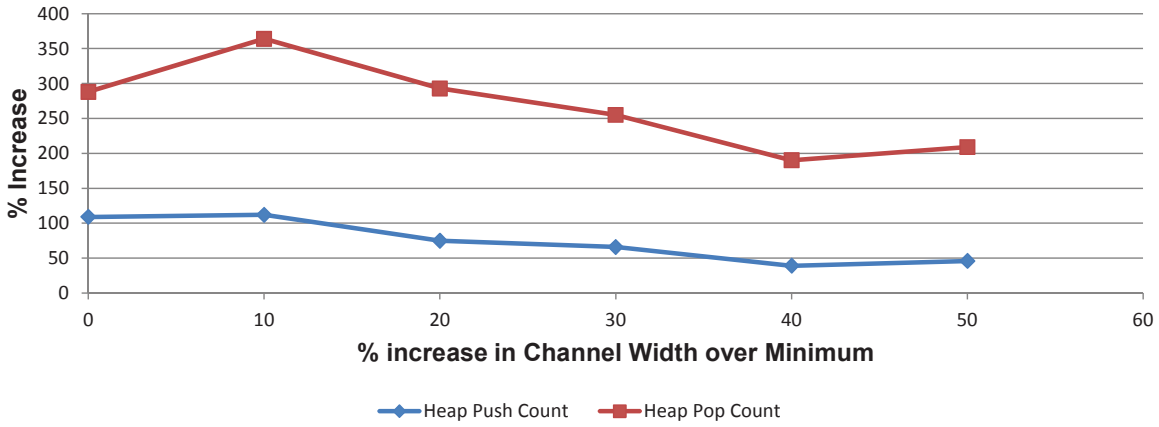


Figure 3.17: Percentage Increase in Geometric Mean of Perturbed routing over pin-to-pin routing, as a function of increasing channel width, for the heap push and pop count metrics

Figure 3.16 plots the percentage increase in geometric mean of Perturbed routing over pin-to-pin routing as a function of increasing track count, for the critical path delay and wirelength metrics. Figure 3.17 gives the same summary of results, but for heap push and pop count metrics. For each data point, both the Perturbed pin-to-wire routing and the pin-to-pin routing have been performed on the same track count. From Figure 3.16 one can observe that the degradation in critical path delay oscillates between 16% and 19% degradation. The first data point only represents two benchmark circuits and so it is not reliable. The curve indicates that increasing track count has no noticeable impact on delay. On the other hand, increasing track count does reduce the wirelength degradation by 5%. Hence, on increasing the track count, the performance gap between pin-to-pin and pin-to-wire routing reduces. Finally, Figure 3.17 illustrates that the degradation in both the heap push and pop count reduces with increasing track count. Again the first data point can be ignored for the aforementioned reason. The slight increase in both the heap push and pop count as the extra track count is increased from 40% to 50% can be attributed to two probable causes: first, an unfortunate selection of the target wire segments, and second, noise resulting from VPR's pattern graph generator.

From the results, one can conclude that as the track count increases, the performance gap between Perturbed pin-to-wire routing and pin-to-pin routing reduces. In the next subsection, we will discuss how increasing the track count in the above manner influences Dispersed Perturbed pin-to-wire routing.

3.6.2 Effects on Dispersed Perturbed Pin-to-Wire Routing

In Dispersed Perturbed pin-to-wire routing, circuits face significantly more pin-to-wire routing than in the 'perturbed' case. Hence, it is interesting to observe how circuits under varying levels of stress respond to such a high level of pin-to-wire routing. Figure 3.18 plots the number of unroutes as a function of increasing track count for the Dispersed Perturbed pin-to-wire routing experiment. The graph shows that at low track counts,

when circuits are under high stress levels, all benchmarks fail to route. As the track count increases, and the circuit stress levels reduce, routability improves, with only 3 benchmarks failing to route at 50% higher than minimum channel width. We also plot the percentage increase in geometric mean of Dispersed Perturbed routing over pin-to-pin routing as a function of increasing track count. Figure 3.19 and Figure 3.20 summarize the results. Again, for each data point, both the Dispersed Perturbed pin-to-wire routing and the normal pin-to-pin routing have been performed on the same track count. Figure 3.19 illustrates that as the track count increases, the performance degradation in critical path delay reduces. The performance degradation reduces by 13% as the extra available track count increases from 20% to 50%. But the wirelength plot in Figure 3.19 indicates that increasing the track count does not yield a noticeable decrease in wirelength degradation. Like the delay, the router effort as illustrated in Figure 3.20 reduces with increase in track count. One can hence conclude, that circuits under low stress definitely respond better in the face of pin-to-wire routing, while circuits under extremely high stress maybe incapable of handling high amount of pin-to-wire routing.

From observing the routability trend depicted in Figure 3.18, it can be concluded that if one were to attempt to completely remove the unroutes resulting from pin-to-wire

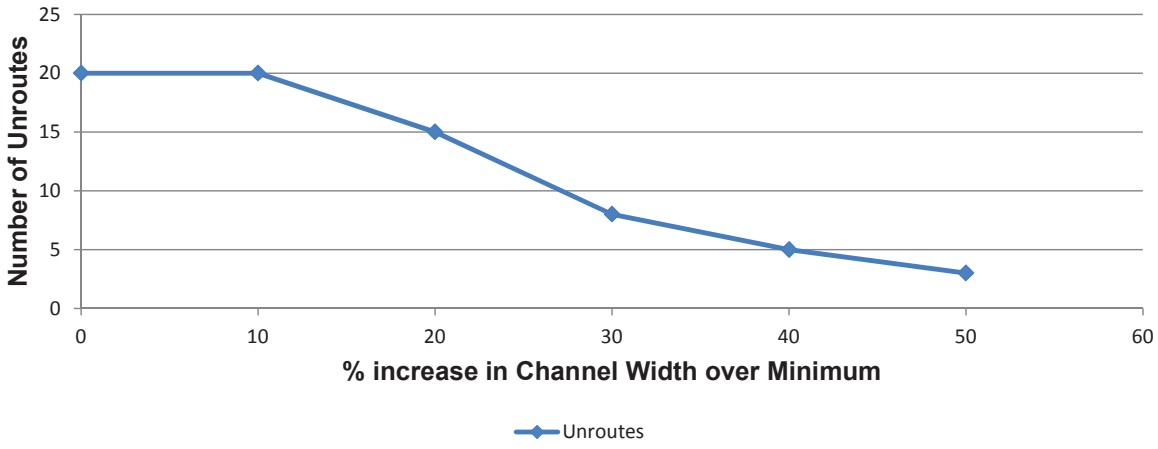


Figure 3.18: Number of Unroutes (out of 20) as a function of increasing track count for Dispersed Perturbed pin-to-wire routing

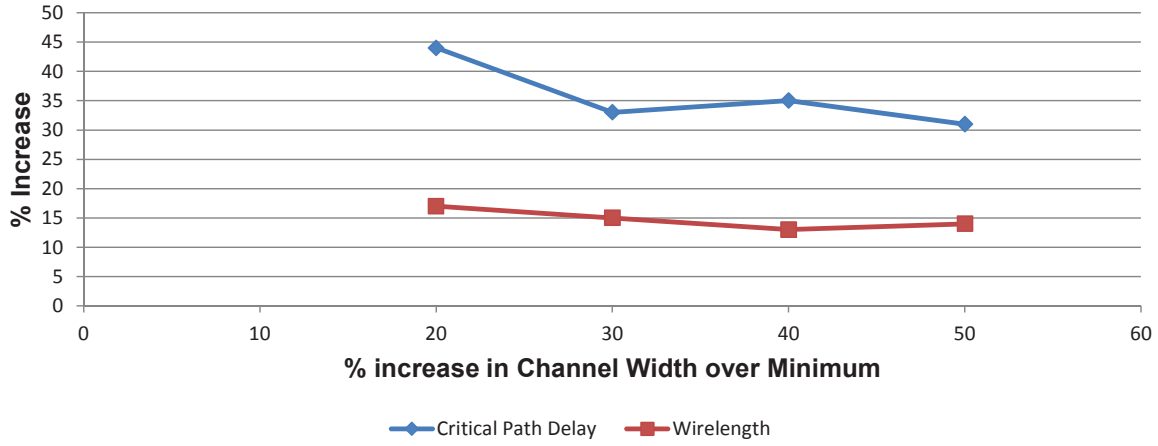


Figure 3.19: Percentage Increase in Geometric Mean of Dispersed Perturbed routing over pin-to-pin routing, as a function of increasing channel width, for the wirelength and critical path delay metrics

routing by simply increasing the track count, 50% or higher track count than minimum would have to be employed. However, Figure 3.18 also depicts that the unroutes approach zero even though 100% of the nets in the circuit are split to create pin-to-wire routing This suggests that with lower amounts of pin-to-wire routing, zero unroutes can be achieved even at track counts lower than 50% higher than minimum. Figure 3.15, which plots the unroutes for the Perturbed case in which circuits face significantly lower amount of pin-

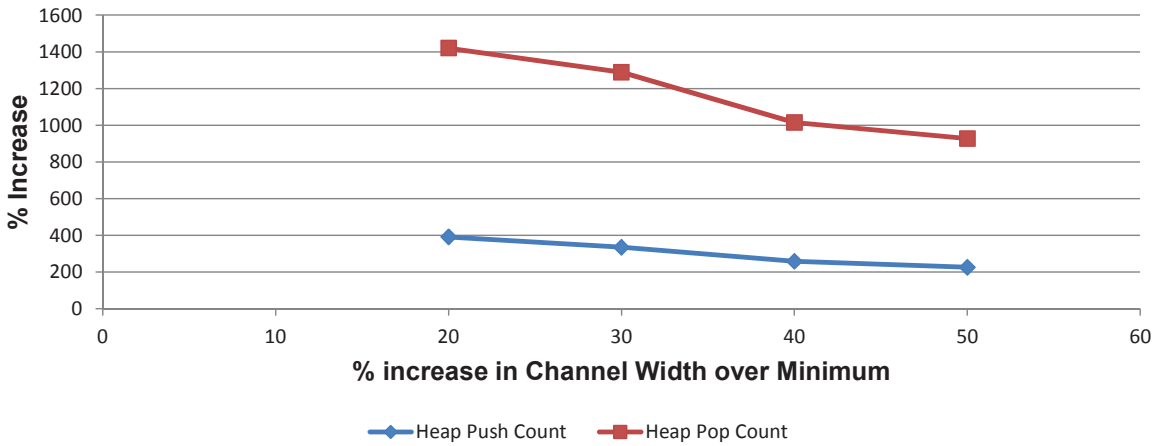


Figure 3.20: Percentage Increase in Geometric Mean of Dispersed Perturbed routing over pin-to-pin routing, as a function of increasing channel width, for the heap push and pop count metrics

to-wire routing than in Dispersed Perturbed case, illustrates this point, as zero unroutes are achieved at 40% higher track count than minimum.

3.7 Summary

In this chapter, we observed the impact of pin-to-wire routing on the performance of the router and a specific routing architecture. We performed two sets of experiments: first, those that simulated a routing-by-abutment scenario, and second, those that varied the amount of pin-to-wire routing in a netlist. The former indicated that pin-to-wire routing resulted in a significant increase in wirelength, delay and router effort, and an occasional loss of routability. The latter showed that as the amount of pin-to-wire routing present in a netlist increases, the performance degradation in wirelength, delay and router effort also increases and the routability decreases. We also noticed that reducing circuit stress levels by increasing track count better enables them to combat pin-to-wire routing. However, it is clear that the routing architecture will have a significant impact on the ability of circuits to handle pin-to-wire routing. In the next chapter, we will study this impact.

Chapter 4

Impact of Routing Architecture on Pin-to-Wire Routing

In the previous chapter, we measured the difficulty faced by a router and a specific routing architecture in the face of pin-to-wire routing. We did so by performing four experiments, which were called: Basic, Perturbed, Dispersed and Dispersed Perturbed. The first two simulated a routing-by-abutment scenario while the last two measured the effect of the *amount* of pin-to-wire routing on wirelength, delay and routability. However, it is clear that the routing architecture itself will have a significant impact on pin-to-wire routing, and so we measure that in this chapter. Recall our hypothesis that pin-to-wire routing is inherently more difficult compared to pin-to-pin routing due to the reduced flexibility available in connecting to or from a wire. This suggests that if there were more flexibility in the architecture, then the challenges faced by pin-to-wire routing would be reduced. The aim of this chapter is to test this hypothesis by performing a series of experiments that add flexibility to the architecture used in Chapter 3. In doing so we seek to answer the following two questions:

1. How does pin-to-wire routing compare to pin-to-pin routing on more flexible architectures?

2. Is there a good choice for a low cost architectural modification that better enables pin-to-wire routing?

The experiments in Chapter 3 were performed on a classical architecture with the following logic and routing architecture:

1. 4 x 4-input LUTs per cluster
2. 10 input pins per cluster
3. $\text{Fc}_{in} = 0.15$ and $\text{Fc}_{out} = 0.25$
4. Wilton Switch Block topology
5. Fs = 3
6. Length 4 segments with a staggered distribution and a single driver approach
7. Channel width 30% higher than minimum channel width

In this chapter we will enhance this architecture, which we will henceforth refer to as the *Standard Architecture*, by individually modifying each of the following routing architecture parameters: Fs (Section 4.2), Fc_{in} (Section 4.3) and Fc_{out} (Section 4.4). For each we observe how these enhancements impact the results of pin-to-wire routing for the two of our most realistic previous experiments: Perturbed and Dispersed Perturbed. For the Perturbed experiment, we will only consider the PWO net ordering, as experiments in Chapter 3 indicate that the net ordering has a negligible impact on the results. Similarly, for the Dispersed Perturbed experiment, we will only consider the case in which 100% of the nets are selected for splitting, i.e. when the circuits are subjected to the highest amount of pin-to-wire routing. We conclude by presenting an approach for selecting a cost effective enhancement, that can help architects determine which architecture would yield the most benefit at the lowest cost.

The next section describes the design methodology we employ for the subsequent experiments.

4.1 Design Methodology

As explained above, through the experiments we perform in this chapter we seek to answer two questions: first, how does pin-to-wire routing compare to pin-to-pin routing on more flexible architectures; and second, can we modify the Standard Architecture in some low cost manner such that pin-to-wire routing is better enabled. Before we begin to answer these questions, we will define some terms to make the discussion more clear. The first term we define is called *Enhanced Architecture*, and it describes any architecture that has been enhanced in flexibility over the Standard Architecture by increasing either the Fs, Fc_{in} or Fc_{out} parameters, or a combination thereof. Recall that in Chapter 3, we defined the term *Base* routing to refer to an instance of pin-to-pin routing when performed on the Standard Architecture. Now we define *Enhanced Base* to mean an instance of pin-to-pin routing when performed on an Enhanced Architecture. The next term we define is *Enhanced Perturbed*, which refers to an instance of Perturbed pin-to-wire routing when performed on an Enhanced Architecture. Similarly, we also define the term *Enhanced Dispersed Perturbed*, to refer to an instance of Dispersed Perturbed pin-to-wire routing when performed on an Enhanced Architecture. Also, at times, we will use the term *Enhanced Pin-to-Wire* to collectively refer to *Enhanced Perturbed* and *Enhanced Dispersed Perturbed* routing.

Now, in order to answer our first question, we need to compare pin-to-wire routing when performed on an architecture with increased flexibility (over the Standard Architecture), to pin-to-pin routing, when performed on the exact same flexible architecture. Accordingly, we need to compare Enhanced Pin-to-Wire to Enhanced Base as a function of increasing architecture flexibility. Our second question can be answered by understanding how the impact of pin-to-wire routing changes in comparison to Base routing, as the architecture flexibility is increased. Accordingly, we need to compare Enhanced Pin-to-Wire to Base as a function of increasing architecture flexibility.

Consequently, as illustrated in Table 4.1, for each architectural parameter Fs, Fc_{in}

and Fc_{out}, we will do the following experiments:

1. Establish the baseline effect of increasing the desired architectural parameter on pin-to-pin routing by comparing Enhanced Base to Base as a function of increasing architecture flexibility

2. Explore how the gap between Perturbed pin-to-wire and pin-to-pin routing changes with increase in architecture flexibility by comparing Enhanced Perturbed to Enhanced Base as a function of increasing architecture flexibility

3. Observe whether Perturbed pin-to-wire routing gets better enabled on increasing architecture flexibility by comparing Enhanced Perturbed to Base as a function of increasing architecture flexibility

4. Repeat experiment 2, but for the Dispersed Perturbed pin-to-wire case

5. Repeat experiment 3, but for the Dispersed Perturbed pin-to-wire case

The reader is requested to refer to Table 4.1 as and when required to help navigate through this chapter.

4.1.1 Caveat

A note on the accuracy of metrics. For all our subsequent experiments, the delay model is inaccurately optimistic for the following three reasons: first, we have not modelled the impact of increasing multiplexer sizes (as a result of increasing Fs, Fc_{in} or Fc_{out}), on the multiplexer delays. Second, we have not modelled the fact that increasing routing architecture flexibility increases the tile size, and hence, the length of the wires. Longer wires would result in higher delays. Third, we do not account for the parasitic capacitances resulting from the input connection block multiplexers and/or routing switches.

In the next section, we observe the effects of increasing the switch block flexibility parameter Fs.

Table 4.1: Experiments performed for each architectural parameter, and their corresponding Section numbers

#	Experiment	A		vs.	B		Architecture Parameter and Corresponding Section #		
		Architecture	Routing Problem		Architecture	Routing Problem	Fs	Fc_{in}	Fc_{out}
1	Enhanced Base vs. Base	Enhanced	Pin-to-Pin		Standard	Pin-to-Pin	4.2.2	4.3.1	4.4
2	Enhanced Perturbed vs. Enhanced Base	Enhanced	Pin-to-Wire		Enhanced	Pin-to-Pin	4.2.3	4.3.2	4.4
3	Enhanced Perturbed vs. Base	Enhanced	Pin-to-Wire		Standard	Pin-to-Pin	4.2.4	4.3.3	4.4
4	Enhanced Dispersed Perturbed vs. Enhanced Base	Enhanced	Pin-to-Wire		Enhanced	Pin-to-Pin	4.2.5	4.3.4	4.4
5	Enhanced Dispersed Perturbed vs. Base	Enhanced	Pin-to-Wire		Standard	Pin-to-Pin	4.2.6	4.3.5	4.4

4.2 Effect of the Switch Block Flexibility Parameter (Fs)

Recall that the Switch Block Flexibility Parameter (Fs) determines the number of neighbouring wire segments that can be driven by a given wire segment. Increasing Fs should result in better pin-to-wire results. In this section, we first describe the code modifications required to support fine-grained Fs increases, and then give experimental results.

4.2.1 Routing Resource Graph Generation Modifications

The original VPR 5.0.2 [15] code only allows Fs increments in multiples of 3. However, incrementing Fs in steps of 3 adds a significant amount of flexibility to the architecture at every step, preventing us from gaining insights on how pin-to-wire routing responds to a more incremental increase in flexibility. To observe the impact of increasing flexibility on a more minute scale, we will increase Fs in steps of 1. In order to enable this, the routing graph generation algorithm had to be modified, which we now describe. Recall the discussion of the routing resource graph generation in VPR described in Subsection 2.2.1.

Consider the switch block illustrated in Figure 4.1, in which we define two kinds of wires incident on the switch block - an *ending wire*, which is a wire that terminates at the switch block; and a *passing* wire, which is a wire that passes through the switch block. Passing wires have typically only been used to drive wire segments orthogonal to their direction (although they could drive parallel wires), whereas ending wires can drive wire segments on both their orthogonal and opposing sides. Figure 4.2 illustrates these connections. The key issue in enabling units-of-1 increases in switch block flexibility is to choose which of the 3 sides an ending or passing wire will connect to. For values of Fs that are multiples of 3, the connection pattern is left the same as that in the original code (each wire segment drives Fs/3 wire segments on each side that it can drive). For values of Fs that are not multiples of 3, we have either one (Fs modulo 3 = 1) or two (Fs

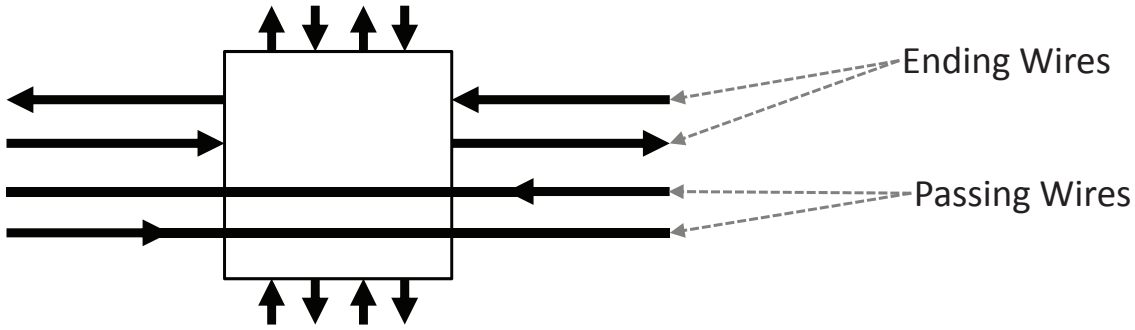


Figure 4.1: Passing and Ending Wires at a Switch Block

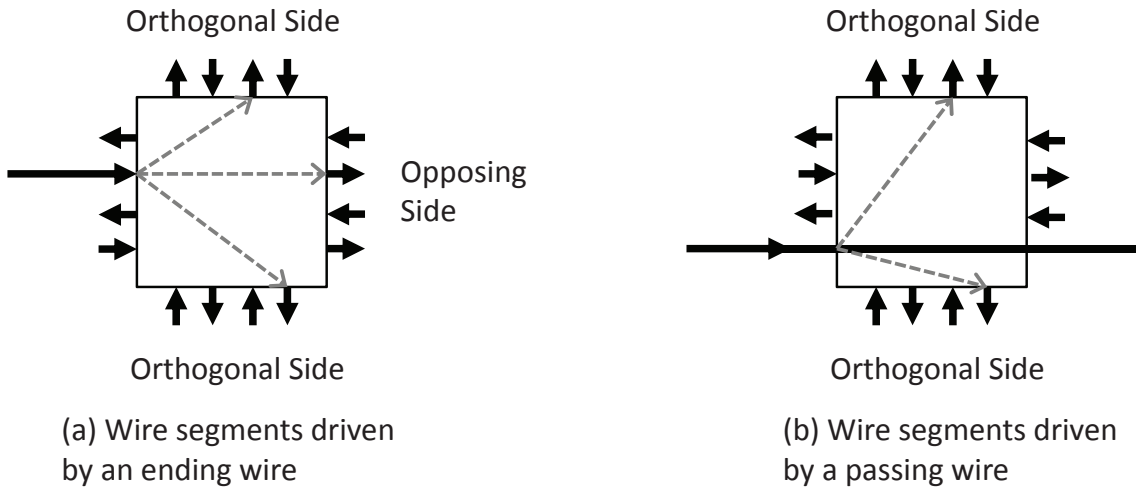


Figure 4.2: Connections made by Passing and Ending Wires within a Switch Block

modulo 3 = 2) extra connections available. For such values of Fs, we allocate the extra connections as follows:

1. The ending wires use the extra connections available (either one or two), to connect to their opposing sides

2. If one extra connection is available, the passing wire segments on a side use it to alternately connect to either their top or bottom orthogonal side. If two extra connections are available, each passing wire segment makes one additional connection to its top orthogonal side and one to its bottom orthogonal side (Vertically oriented passing wires use the extra connections similarly to connect to their left and right orthogonal sides)

The ending wires always favor the opposing side since it was empirically observed that providing extra connections on the opposing side led to better improvements in wirelength and critical path delay. Which specific wire segments a given wire segment can drive on any particular switch block side are determined using a switch block multiplexer size balancing technique, adopted as is from the original VPR 5.0.2 algorithm.

4.2.2 Impact of Increasing Fs on Pin-to-Pin Routing

Let us begin by observing how conventional pin-to-pin routing responds to an increase in Fs. This is related to prior work done by Rose et al. in [30] and Betz et al. in [2]. Intuitively, we would expect both wirelength and critical path delay to reduce with increasing Fs. This is because increasing Fs adds flexibility and choices which can be used to find more direct paths between the source and the sink. We would also expect the heap push count to go up since at every wire segment we are now faced with more choices, and hence, more potential nodes to explore. On the other hand, we can expect the heap pop count to reduce, since more flexibility results in more options, making it easier to find a solution and hence, reducing the number of nodes that are actually explored. Figure 4.3 depicts the percentage increase in geometric mean (over the entire benchmark suite), of pin-to-pin routing performed on the Enhanced Architecture (Enhanced Base routing), over pin-to-pin routing performed on the Standard Architecture (Base routing), as a function of increasing Fs, for the critical path delay and wirelength metrics. Similarly, Figure 4.4 gives the percentage increase in geometric mean of the heap push and pop count metrics. As can be seen from both these figures, all of our expectations are borne out, pin-to-pin routing behaves as expected.

Figure 4.3 shows that the critical path delay initially reduces with increase in flexibility, with diminishing returns after Fs=4. This behaviour can be explained as follows: the timing driven router in VPR 5.0.2, routes the critical path using the fastest resources and shortest paths, and any increase in flexibility serves to make this path shorter and faster. At some point; however, the path cannot be made any shorter due to the place-

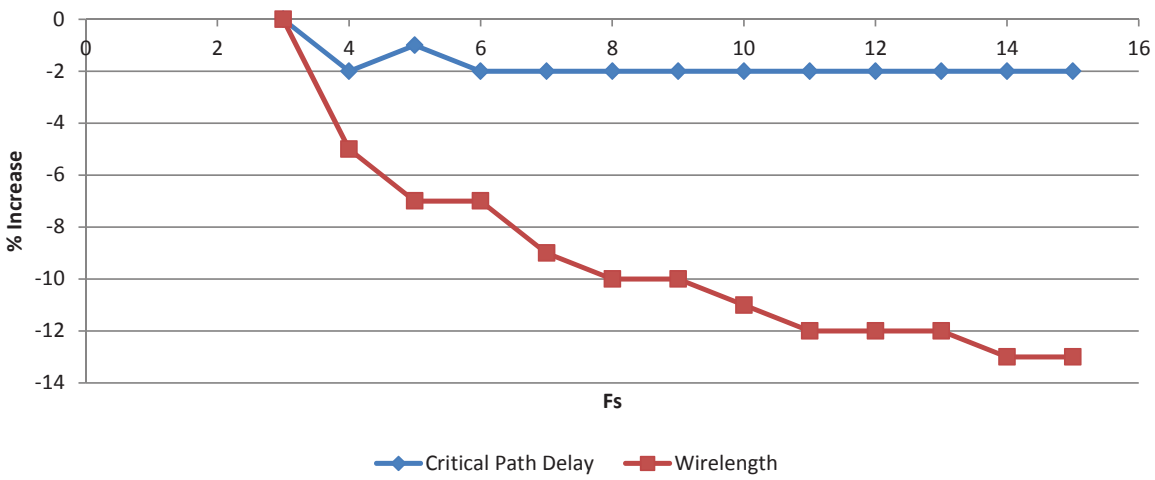


Figure 4.3: Percentage Increase in Geometric Mean of Enhanced Base over Base as a function of increasing Fs, for critical path delay and wirelength metrics

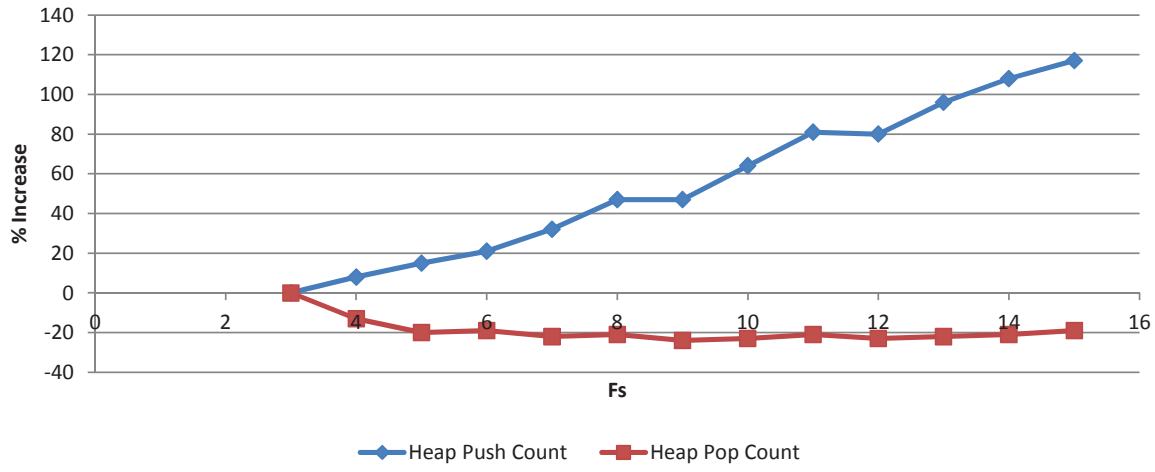


Figure 4.4: Percentage Increase in Geometric Mean of Enhanced Base over Base as a function of increasing Fs, for heap push and pop count metrics

ment of the net's terminals and/or other flexibility constraints (such as Fc_{in} and Fc_{out}). Accordingly, the critical path delay initially improves but then flattens out. One may note that the data indicates an increase in delay as Fs increases from 4 to 5, this increase may most likely be noise resulting from VPR's pattern graph generator. Recent work by colleagues has illustrated that the pattern generator has some anomalies.

From Figure 4.3 one can also observe that unlike critical path delay, the routed wirelength continues to reduce as flexibility increases. This is because wirelength here is the total wirelength of all nets, critical as well as non-critical, and increasing flexibility would certainly help reduce the path lengths of those less critical nets that had been routed using slower longer paths. This improvement in wirelength does eventually flatten out, when paths can no longer be made any shorter, dictated by the underlying placement. The graph illustrates this, as an increase in Fs from 3 to 4 results in a 6% improvement in wirelength, while an increase in Fs from 13 to 14 (or even 15) only results in a 1% improvement in wirelength.

Figure 4.4 illustrates that the heap push count increases proportionally with an increase in Fs (as expected), while the heap pop count initially reduces but then flattens out. This can be attributed to the fact that initially when extra flexibility is introduced in the architecture, solutions become easier to find, and hence, heap pop count reduces. However, adding additional flexibility beyond Fs=6 does not help, since some basic number of nodes have to be explored to connect the sources to the sinks, owing to their placement and/or other flexibility constraints (Fc_{in} and Fc_{out}).

Now that we have explored the baseline effect of enhancing the architecture, we will next study the impact of increasing Fs on pin-to-wire routing, the focus of this research. As mentioned before, we expect that adding flexibility to an FPGA architecture would help mitigate the problems associated with pin-to-wire routing. To test this hypothesis, we study the impact of increasing Fs on both Perturbed (subsections 4.2.3 and 4.2.4) and Dispersed Perturbed (subsections 4.2.5 and 4.2.6) pin-to-wire routing.

4.2.3 Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Enhanced Architecture

In this subsection, we explore whether enhancing the architecture by increasing Fs reduces the gap between Perturbed pin-to-wire routing and pin-to-pin routing. Figure 4.5 depicts the percentage increase in geometric mean (over the entire benchmark suite), of Enhanced Perturbed over Enhanced Base as a function of increasing Fs, for the wirelength and critical path delay metrics. Figure 4.6 illustrates the percentage increase in geometric mean for the heap push count and pop count metrics. In Figure 4.5 we can see that as Fs increases, the performance gap between pin-to-wire and pin-to-pin routing reduces, for both critical path delay and wirelength. The performance degradation in critical path delay reduces by 5%, but it does not become negligible. The flattening of the critical path delay curve indicates that adding additional flexibility can not help to mitigate the difficulty faced in performing pin-to-wire routing. The small increase in delay as Fs increases from 5 to 6, may be noise resulting from VPR's pattern graph generator. The degradation in routed wirelength also reduces by 5%, making the degradation in wirelength almost negligible.

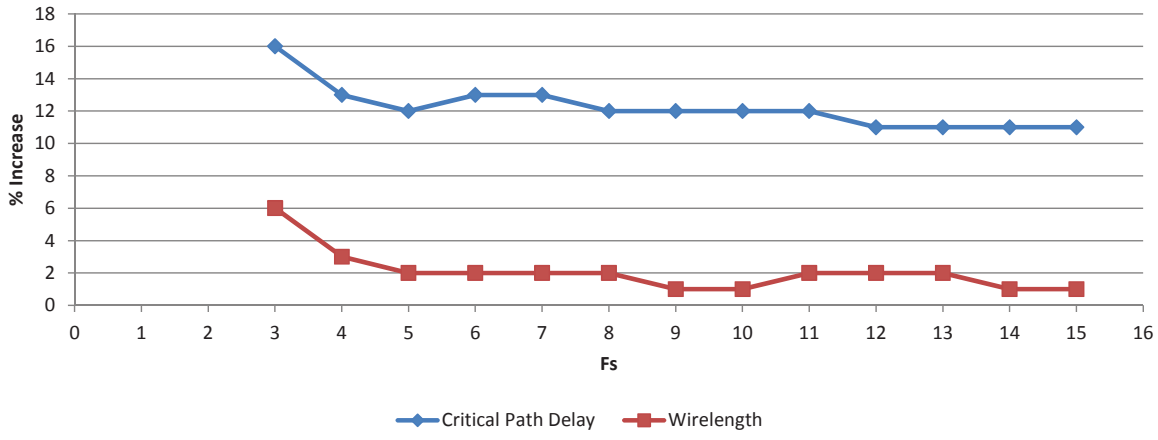


Figure 4.5: Percentage Increase in Geometric Mean of Enhanced Perturbed over Enhanced Base as a function of increasing Fs, for critical path delay and wirelength metrics

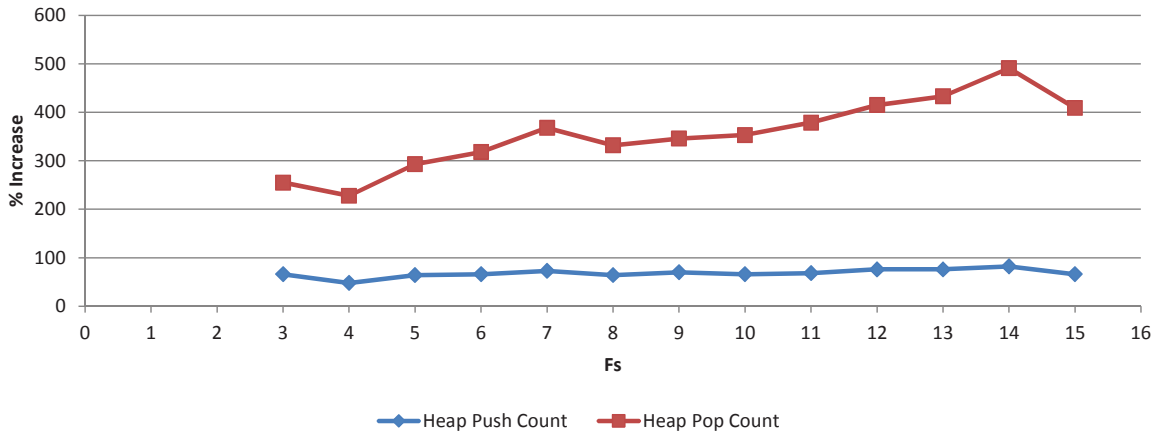


Figure 4.6: Percentage Increase in Geometric Mean of Enhanced Perturbed over Enhanced Base as a function of increasing Fs, for heap push and pop count metrics

As indicated in Figure 4.6, a small increase in Fs reduces the gap between pin-to-pin and pin-to-wire routing for the heap push and pop count metrics. However, it is interesting to note that increasing Fs any further does not reduce the extra router effort resulting from pin-to-wire routing; rather, increases it. As Fs increases, the degradation in heap push count stays relatively the same as compared to pin-to-pin routing, while the degradation in heap pop count continues to worsen as compared to pin-to-pin routing. The relatively constant increase in heap push count, even at high flexibility, is another indication of the difficulty of pin-to-wire routing.

The increase in heap pop count can be explained as such: increasing Fs increases the nodes to be explored and hence, the heap push count. More nodes on the heap creates more paths to be explored, which is helpful if a solution could be obtained quickly. In the case of pin-to-pin routing, the input connection block flexibility allows the input pin to connect to a fraction of tracks in the neighbouring channel (for example, if W = 100 and Fc_{in} = 0.15, then each input pin could connect to 15 tracks in each channel), and in addition, since all input pins are logically equivalent, the router has many choices of which wire segments to use to connect to the logic block input pins. Consequently, in pin-to-pin routing, additional paths could potentially help in obtaining a solution quickly.

However, in pin-to-wire routing we have exactly one wire segment that needs to be driven. Adding potential paths for exploration may result in paths being found such that they lead close to the target wire segment but not actually to it. For example, a potential path may lead to a wire segment in an adjacent track. In such cases, the router must back track, looking for a new path. In pin-to-wire routing the final solutions are limited, and yet, on increasing Fs, a large number of promising but unsuccessful paths are created. The router is forced to explore these paths and hence, as Fs increases, the degradation in heap pop count increases significantly. Clearly, the difficulty of pin-to-wire routing is evident.

To summarize, a large increase in Fs reduces the gap between Perturbed pin-to-wire routing and pin-to-pin routing for both critical path delay and wirelength metrics, but significantly worsens the gap in router effort.

4.2.4 Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Standard Architecture

We are interested in knowing if there exists a low cost architectural modification that can better enable pin-to-wire routing. Accordingly, in this section, we will explore whether enhancing the architecture by increasing Fs better enables Perturbed pin-to-wire routing as compared to pin-to-pin routing on the Standard Architecture. To study this we plot the percentage increase in geometric mean (over the entire benchmark suite), of Enhanced Perturbed over Base as a function of increasing Fs, for all the metrics. Figure 4.7 and Figure 4.8 give the summary of the results. Recall from Section 3.3 that Perturbed pin-to-wire routing suffered 2 unroutes on the Standard Architecture. A minor increase in Fs, from 3 to 4, causes these unroutes to disappear. In addition, as illustrated in Figure 4.7, this minor increase in flexibility greatly improves both critical path delay and wirelength. The critical path delay and wirelength curves both flatten out for reasons explained in subsection 4.2.2.

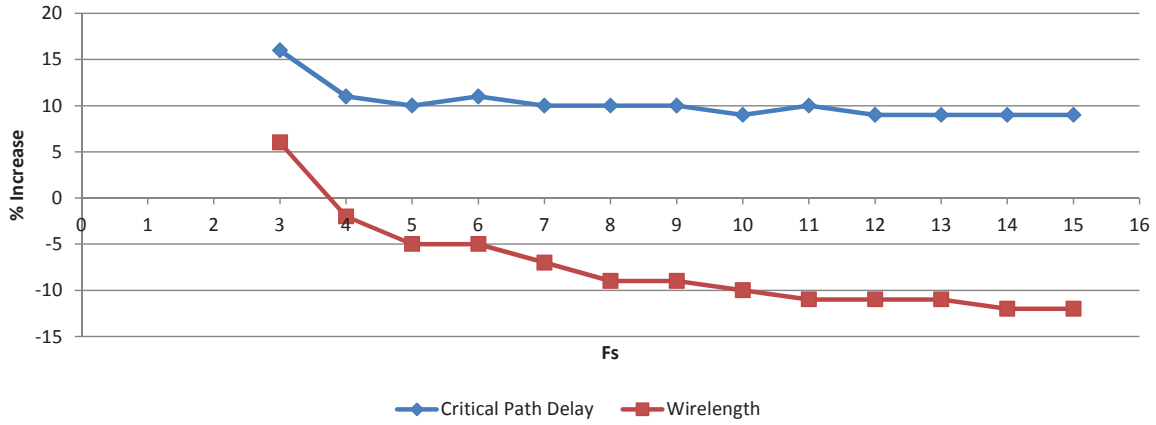


Figure 4.7: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Fs, for critical path delay and wirelength metrics

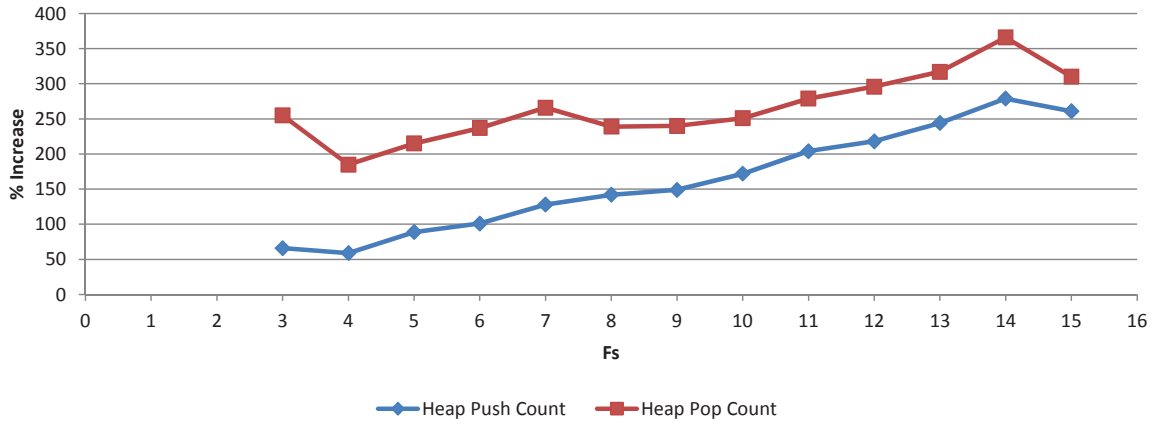


Figure 4.8: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Fs, for heap push and pop count metrics

Similarly, as illustrated in Figure 4.8, a minor increase in flexibility also improves both heap push and pop counts, and this may be attributed to the fact that solutions are now easier to find. The reduction in pop count may be due to the fact that a small increase in flexibility significantly improves connectivity, while at the same time not creating way too many false potential paths. Figure 4.8 also demonstrates that beyond Fs=4, both heap push and pop count increase with increasing Fs. The increase in heap push count is for reasons explained in subsection 4.2.2 while the increase in heap pop count can be attributed to the reasons explained in the previous subsection. The occasional decrease

in push and pop count with increasing Fs may be a result of noise caused by VPR's pattern graph generator.

In the following subsections, we will observe the impact of increasing Fs on Dispersed Perturbed pin-to-wire routing.

4.2.5 Dispersed Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Enhanced Architecture

In this subsection, we observe how the gap between Dispersed Perturbed pin-to-wire routing and pin-to-pin routing responds to an increase in flexibility. This experiment is different from the one described in section 4.2.3, as unlike in that experiment, circuits are now exposed to a much higher degree of pin-to-wire routing. Further, the routing-by-abutment scenario is absent, i.e. nets are no longer restricted to designated FPGA modules. Figure 4.9 depicts the percentage increase in geometric mean (over the entire benchmark suite), of Enhanced Dispersed Perturbed over Enhanced Base as a function of increasing Fs, for the wirelength and critical path delay metrics. Similarly, Figure 4.10 provides the percentage increase in geometric mean for the heap push count and pop count metrics.

All metrics exhibit trends similar to those obtained when Enhanced Perturbed pin-to-wire routing was compared to Enhanced Base routing, and for the same reasons. However, as compared to that experiment, the decrease in the performance degradation of wirelength and critical path delay with increasing flexibility are much more substantial. It is interesting to note that increasing flexibility not only reduces the gap between pin-to-pin and pin-to-wire routing for the wirelength metric, it actually makes pin-to-wire wirelength better than that of pin-to-pin. This suggests that just like we did in the Perturbed case, we have inadvertently sacrificed delay for wirelength.

The increase in the performance degradation of heap pop count with increasing Fs

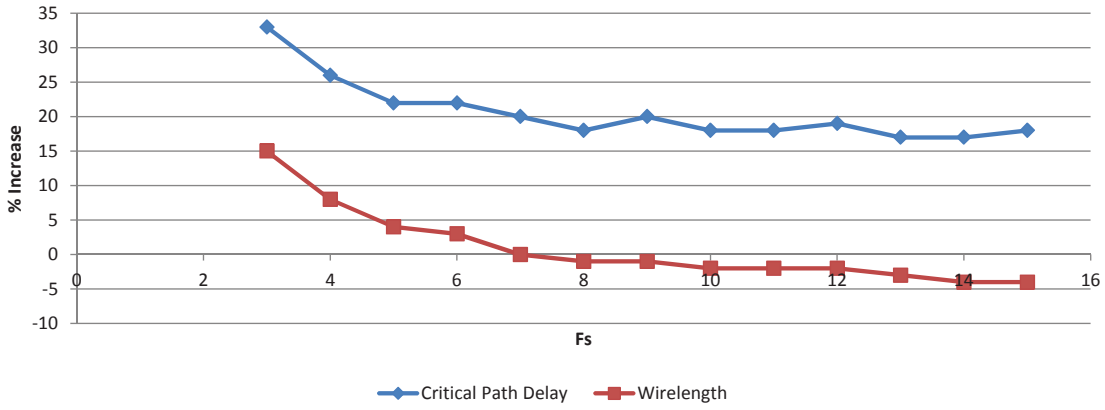


Figure 4.9: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Enhanced Base as a function of increasing Fs, for critical path delay and wirelength metrics

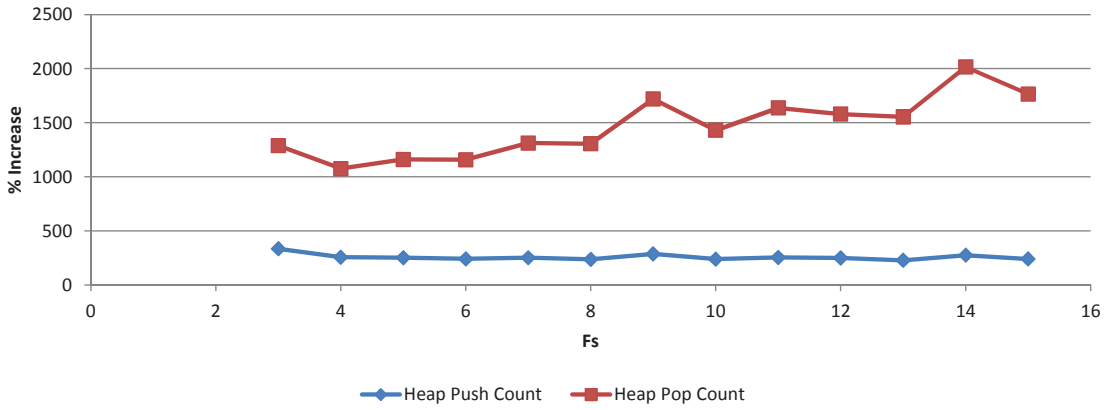


Figure 4.10: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Enhanced Base as a function of increasing Fs, for heap push and pop count metrics

is also much more substantial. Again, as in the case of Perturbed pin-to-wire routing, we can observe that enhancing the architecture by increasing Fs does indeed reduce the gap between pin-to-wire and pin-to-pin routing for the wirelength and critical path delay metrics. For the router effort; however, the gap reduces provided Fs is not increased beyond 6. In the next subsection, we will study whether enhancing the architecture by increasing Fs better enables Dispersed Perturbed pin-to-wire routing.

4.2.6 Dispersed Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Standard Architecture

As in subsection 4.2.4, we plot the percentage increase in geometric mean of Enhanced Dispersed Perturbed over Base as a function of increasing Fs, for all metrics. Figure 4.11 and Figure 4.12 give the summary of the results. As illustrated in Figure 4.13, we also plot the variation in the number of circuits that failed to route as a function of increasing Fs. As can be seen from Figure 4.13, routability improves significantly with increase in Fs, reducing from 8 at Fs =3 to 0 at Fs=6. This in itself indicates that a small increase in flexibility can certainly better enable Dispersed Perturbed pin-to-wire routing. The plots also indicate that all the metrics have trends similar to those obtained in the case of Perturbed pin-to-wire routing (subsection 4.2.4), again for the same reasons.

To summarize, we have observed that enhancing the architecture by adding a small amount of Fs, much better enables pin-to-wire routing and also reduces the gap between pin-to-pin and pin-to-wire routing for all the metrics. However, increasing Fs beyond 5 or 6, significantly increases the heap pop count, while not providing significant gains in the

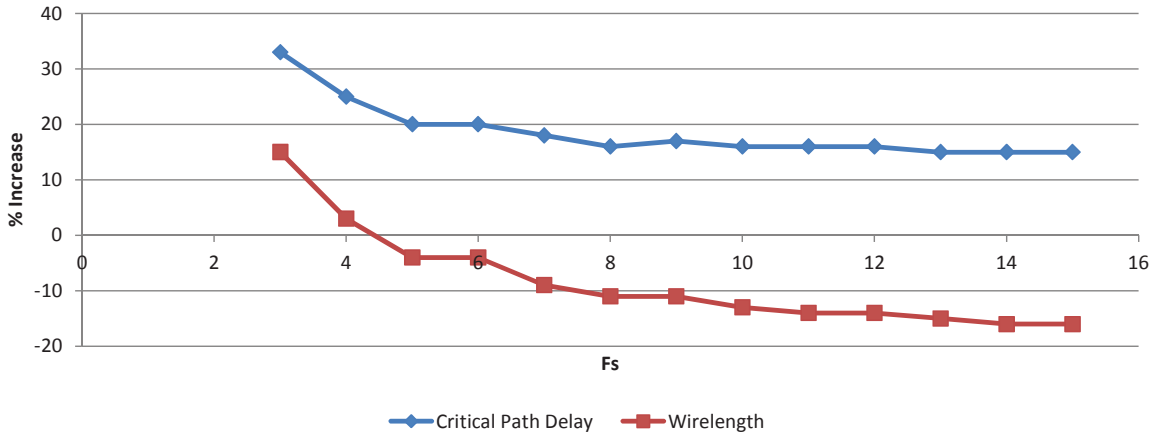


Figure 4.11: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Fs, for critical path delay and wirelength metrics

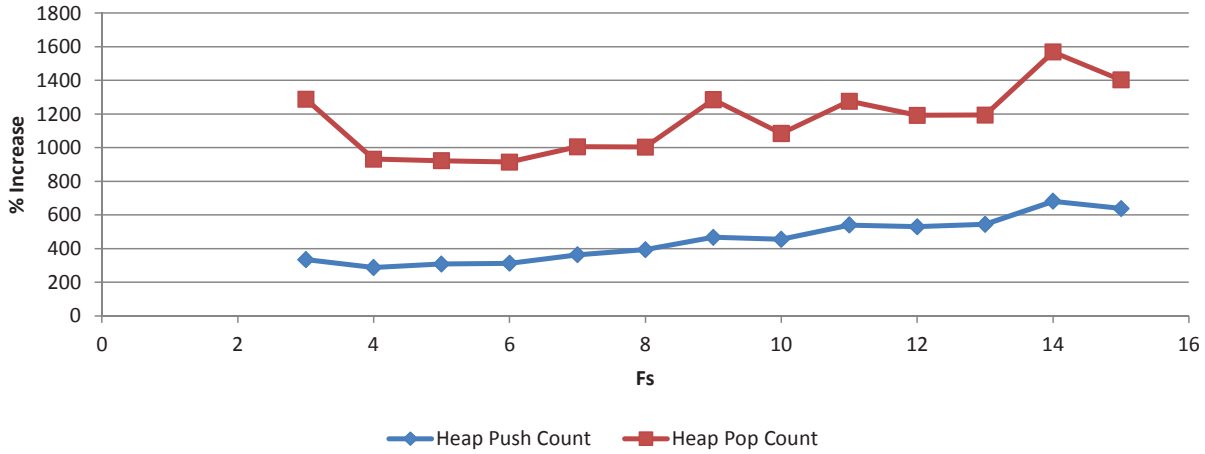


Figure 4.12: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Fs, for heap push and pop count metrics

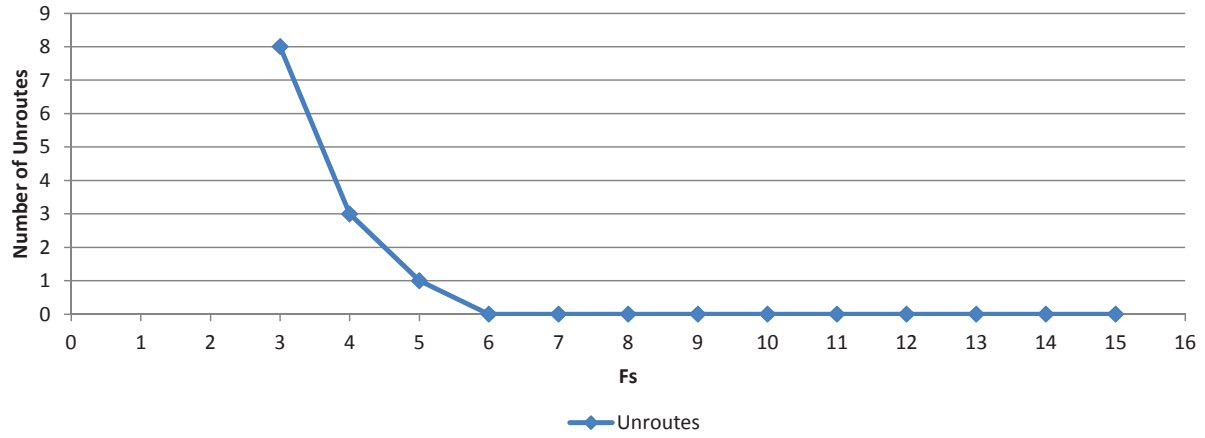


Figure 4.13: Number of Unroutes as a function of increasing Fs for Dispersed Perturbed Pin-to-Wire Routing

other metrics. This indicates that keeping Fs between 4-6 (inclusive), yields maximum benefits.

In the next section, we will observe the impact of increasing Fc_{in} on pin-to-wire routing.

4.3 Effect of the Input Connection Block Flexibility Parameter (Fc_{in})

The input connection block flexibility parameter or Fc_{in} determines the fraction of tracks in the neighbouring channel that an input pin can connect to. We enhance the routing architecture by slowly increasing Fc_{in} from 0.15 (the Fc_{in} value in the Standard Architecture) to 0.50 and observe how these enhancements influence the results of both pin-to-pin and pin-to-wire routing. The input connection block topology is generated by VPR 5.0.2 and has not been modified for this work.

4.3.1 Impact of Increasing Fc_{in} on Pin-to-Pin Routing

Just like we did for Fs, we will begin by observing the impact of increasing Fc_{in} on conventional pin-to-pin routing. Again, this is related to prior work done by Rose et al. in [30] and Betz et al. in [2]. Intuitively, when we increase Fc_{in}, we expect critical path delay, wirelength and router effort to decrease. The reason is as follows: when we increase Fc_{in}, the number of ways to enter a logic block input pin increase, and since a logic block has a number of logically equivalent input pins (10 in our case), the number of ways of entering a logic block increase significantly. This increase enables shorter connections for nets, resulting in a decrease in both critical path delay and wirelength. The increase in connectivity and the ease of entering a logic block also make it easy and quick for the router to find a path connecting the net terminals, reducing both the heap pop and push count. The router does not have to explore a lot of nodes to reach the sinks, which reduces the heap pop count and since, fewer nodes are explored, fewer nodes are added to the heap (only neighbouring nodes of the explored nodes are added to the heap), and hence, the heap push count also reduces.

To measure the impact of increasing Fc_{in} on pin-to-pin routing, we compare Enhanced Base to Base as a function of increasing Fc_{in}. Figure 4.14 plots the percentage increase

in geometric mean (over the entire benchmark suite), of Enhanced Base over Base as a function of increasing Fc_{in}, for the critical path delay and wirelength metrics. Similarly, Figure 4.15 provides the percentage increase in geometric mean for the heap push and pop count metrics. As can be seen from Figure 4.14 and Figure 4.15, all our expectations are realized, all the metrics behave as expected.

Figure 4.14 depicts that the critical path delay reduces with increase in Fc_{in} with diminishing returns after Fc_{in}=0.30. The explanation for this is the same as that given in Subsection 4.2.2 i.e. adding Fc_{in} flexibility beyond a certain point does not improve critical path delay owing to the net terminals' placement and/or other flexibility constraints (such as switch block flexibility). Figure 4.14 also demonstrates that the impact on wirelength is much more pronounced, and the wirelength continues to improve with increasing Fc_{in} with diminishing returns after Fc_{in}=0.45. Again, as explained in Subsection 4.2.2, the impact on wirelength is much more pronounced since wirelength gives the total of wirelength of all nets in the circuit, both critical and non-critical and any increase in flexibility certainly helps shorten those nets that had been routed using longer, slower paths. At some point however, the added flexibility can no longer help owing to the placement of the nets' terminals and other flexibility constraints (such as switch block flexibility). The graph illustrates this point, as the wirelength curve flattens out at Fc_{in}=0.45. Figure 4.15 illustrates that both the heap push and pop count continue to reduce with increasing Fc_{in}, with diminishing returns after Fc_{in}=0.40. The curves exactly track each other, since as the flexibility increases, the router finds solutions quickly, less nodes are explored to arrive at the solution (reduction in heap pop count), and accordingly, fewer nodes are pushed on the heap (reduction in heap push count). For reasons explained above, after a certain point, the added flexibility does not improve the router effort, which is demonstrated by the flattening of the push and pop count curves.

Now that we have explored the baseline effect of enhancing the architecture, we will next study the impact of increasing Fc_{in} on pin-to-wire routing. In order to test

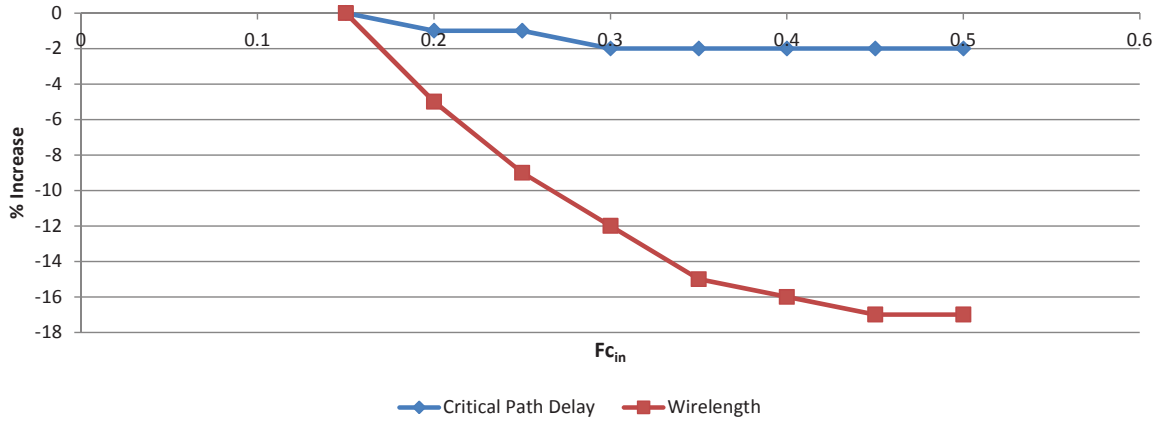


Figure 4.14: Percentage Increase in Geometric Mean of Enhanced Base over Base as a function of increasing Fc_{in}, for wirelength and critical path delay metrics

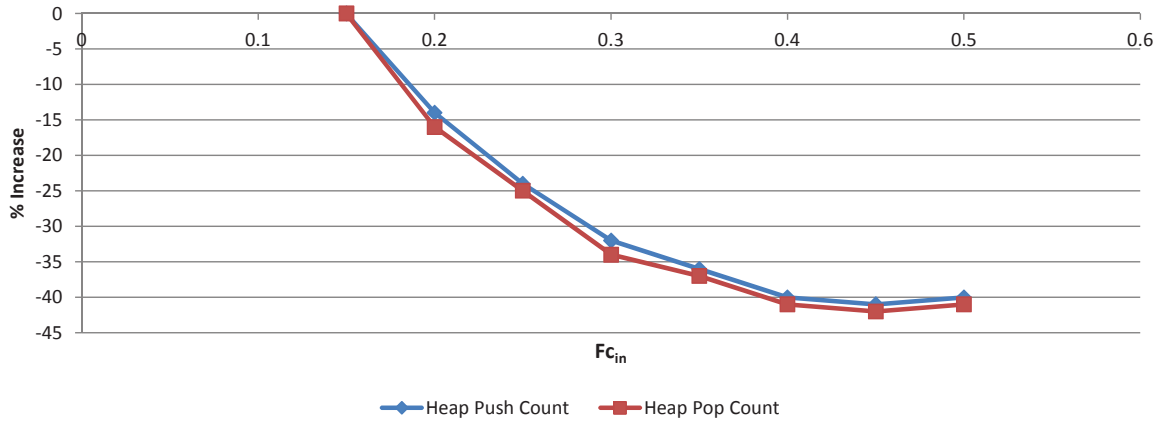


Figure 4.15: Percentage Increase in Geometric Mean of Enhanced Base over Base as a function of increasing Fc_{in}, for heap push and pop count metrics

our hypothesis that adding flexibility to an FPGA architecture would help mitigate the problems associated with pin-to-wire routing, we study the impact of increasing Fc_{in} on both Perturbed pin-to-wire routing (subsections 4.3.2 and 4.3.3) and Dispersed Perturbed pin-to-wire routing (subsections 4.3.4 and 4.3.5).

4.3.2 Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Enhanced Architecture

In this subsection, we explore whether enhancing the architecture by increasing Fc_{in} reduces the gap between Perturbed pin-to-wire routing and pin-to-pin routing. Figure 4.16 depicts the percentage increase in geometric mean (over the entire benchmark suite), of Enhanced Perturbed over Enhanced Base as a function of increasing Fc_{in}, for the wirelength and critical path delay metrics. Similarly, Figure 4.17 provides the percentage increase in geometric mean for the heap push count and pop count metrics. As can be observed from Figure 4.16, as the Fc_{in} increases, i.e. as the architecture flexibility increases, the gap between Perturbed pin-to-wire routing and pin-to-pin routing reduces. However, the net reduction in the degradation of critical path delay and wirelength is

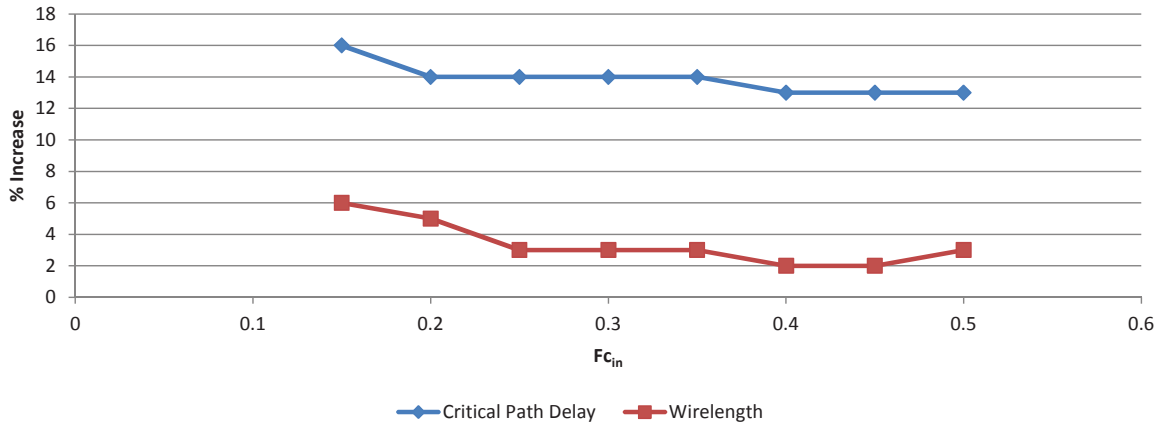


Figure 4.16: Percentage Increase in Geometric Mean of Enhanced Perturbed over Enhanced Base as a function of increasing Fc_{in}, for critical path delay and wirelength metrics

only 3% and 4% respectively. One can infer that although increasing Fc_{in} reduces the gap between pin-to-wire and pin-to-pin routing, the reduction is not very significant for the delay and wirelength metrics. Moreover, both the delay and wirelength curves indicate diminishing returns after an Fc_{in}=0.40, implying that adding additional flexibility does not help mitigate the difficulty faced in pin-to-wire routing. The slight increase in wirelength when Fc_{in} increases from 0.45 to 0.50 can be attributed to noise.

The impact of increasing Fc_{in} on router effort is rather interesting as is depicted in Figure 4.17. The performance degradation in both heap push and pop count initially

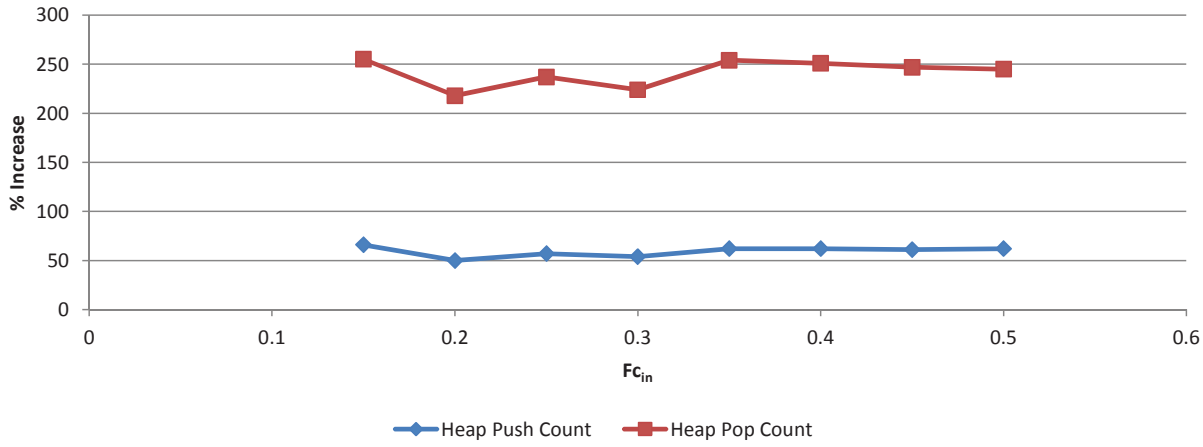


Figure 4.17: Percentage Increase in Geometric Mean of Enhanced Perturbed over Enhanced Base as a function of increasing Fc_{in}, for heap push and pop count metrics

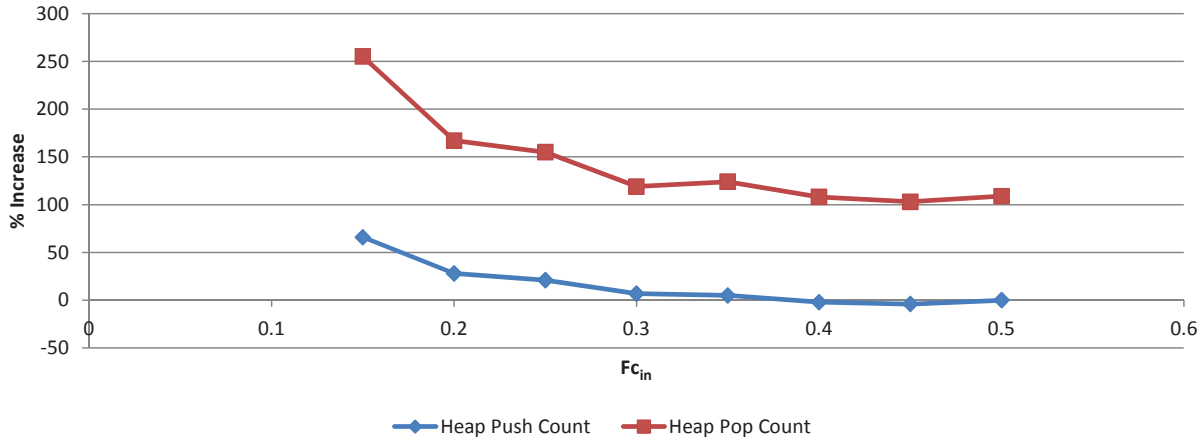


Figure 4.18: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Fc_{in}, for heap push and pop count metrics

reduces with increase in flexibility, then increases and becomes constant. To understand these trends, one must keep in mind that we are looking at the percentage increase between pin-to-wire and pin-to-pin routing, which means that the graph is influenced by the rates at which Perturbed pin-to-wire and pin-to-pin change in response Fc_{in}.

Recall from Figure 4.15 in the last subsection that for pin-to-pin routing, both the heap push and pop count reduce slowly but steadily with increase in Fc_{in} until an Fc_{in} of 0.4, beyond which we get diminishing returns. Next, let us look at how Perturbed pin-to-wire routing behaves in comparison to itself at increasing Fc_{in}. For this we observe Figure 4.18, which plots the percentage increase in geometric mean of Perturbed pin-to-wire routing on Enhanced Architecture over Base routing, as a function of increasing Fc_{in}, for the heap push and pop count metrics. This plot shows that initially with a small increase in flexibility both the heap push and pop count improve significantly, gradually slowing down and giving diminishing returns after an Fc_{in} of 0.30. In other words, for pin-to-pin routing, the rate of improvement in router effort in response to increasing Fc_{in} is slow but steady with diminishing returns after Fc_{in}=0.40; while for the Perturbed pin-to-wire routing, the rate of improvement in router effort in response to increasing Fc_{in} is initially very high, but reduces quickly with diminishing returns after Fc_{in}=0.30.

The net result can be seen in the plots of Figure 4.17, when Fc_{in} changes from 0.15 to 0.20, since the improvement rate of pin-to-wire routing is higher than that of pin-to-pin, we see a net reduction in degradation. When Fc_{in} increases from 0.20 to 0.25, the rate of improvement in pin-to-wire routing is slower than that of pin-to-pin and hence, we see an increase in degradation. However, from Fc_{in} of 0.25 to 0.30, the rate of improvement in Perturbed pin-to-wire routing outperforms the rate of improvement in pin-to-pin routing, and hence, we observe a decrease in the gap between Perturbed pin-to-wire and pin-to-pin. After Fc_{in} of 0.30, Perturbed pin-to-wire routing shows diminishing returns, while pin-to-pin continues to improve until Fc_{in} of 0.40, when it shows diminishing returns as well. Consequently, in the heap push and pop count plots of Figure 4.17, we observe the

gap between Perturbed pin-to-wire and pin-to-pin increase from Fc_{in} of 0.30 to 0.40 and then stay constant.

To conclude, one can infer that a small increase in Fc_{in} reduces the gap between Perturbed pin-to-wire and pin-to-pin for all metrics. However, for the router effort, adding flexibility beyond a certain point causes the gap to increase and then plateau. This is because pin-to-pin routing continues to perform better than pin-to-wire routing at increased flexbilities. Figure 4.16 and Figure 4.17 indicate that for maximum benefits, Fc_{in} should be kept in a range of 0.20-0.30.

Having studied whether enhancing the architecture by increasing Fc_{in} reduces the gap between Perturbed pin-to-wire routing and pin-to-pin routing, we will now explore whether enhancing the architecture by increasing Fc_{in} better enables Perturbed pin-to-wire routing.

4.3.3 Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Standard Architecture

As mentioned in Subsection 4.1, we would like to discover a low cost architectural modification that can help us better enable Perturbed pin-to-wire routing. Thus, in this subsection we test whether increasing architecture flexibility, by increasing Fc_{in}, better enables Perturbed pin-to-wire routing. To study this we plot the percentage increase in geometric mean (over the entire benchmark suite), of Enhanced Perturbed over Base as a function of increasing Fc_{in}, for all the metrics. Figure 4.19 and Figure 4.18 give the summary of the results. Figure 4.19 indicates that both critical path delay and wirelength improve with increasing flexibility, but yield diminishing returns on increasing Fc_{in} beyond 0.45. These diminishing returns are due to reasons explained in Subsection 4.3.1. The improvement in both delay and wirelength can be attributed to the fact that as flexibility increases, the number of ways of getting into a logic block increase, and especially for wire-to-pin connections, solutions can be found which are shorter and

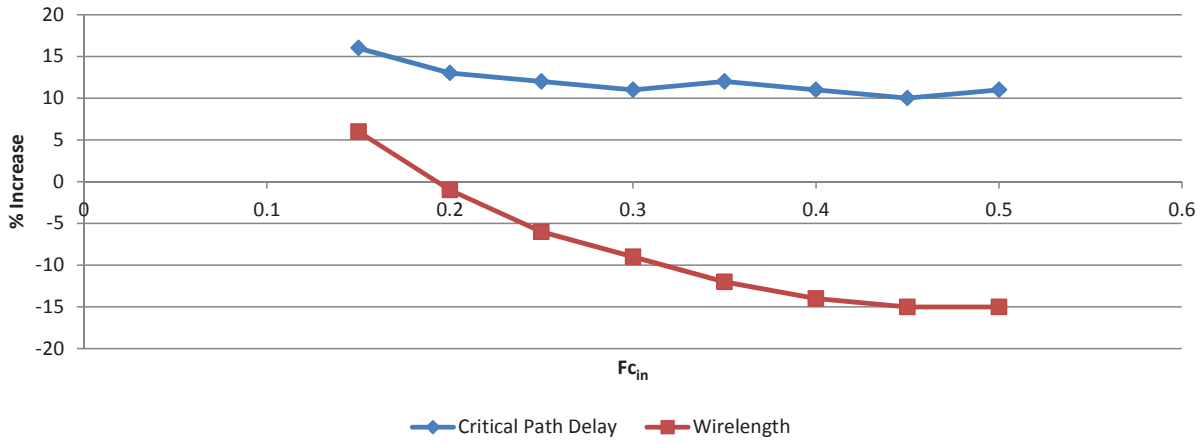


Figure 4.19: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Fc_{in}, for critical path delay and wirelength metrics

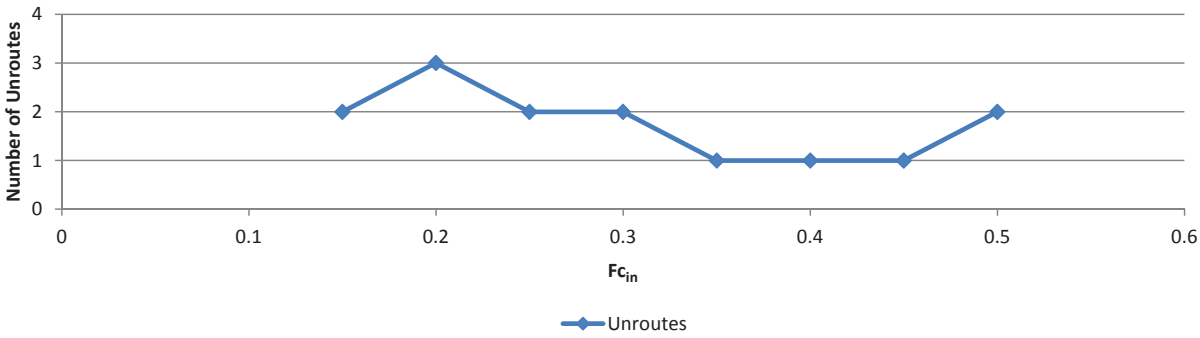


Figure 4.20: Number of Unroutes as a function of increasing Fc_{in} for Perturbed Pin-to-Wire Routing

faster. The wire-to-pin connections no longer have to hunt around for finding ways of entering the logic blocks, they can use more direct paths.

Similarly, Figure 4.18 indicates that with increasing flexibility both heap push and pop count reduce. As explained previously in Subsection 4.3.1, this is because increasing Fc_{in} significantly increases the number of ways in which a logic block can be entered, making solutions quick and easy to find. Since solutions are found more quickly, less nodes are explored (heap pop count reduces), and consequently, less nodes are pushed on the heap (heap pop count reduces). However, the plot also indicates that increasing Fc_{in} beyond 0.30 yields diminishing returns and these diminishing returns are due to reasons explained in Subsection 4.3.1. One can conclude that increasing Fc_{in} certainly better

enables Perturbed pin-to-wire routing.

Finally, Figure 4.20 plots how many circuits fail to route (out of the 20 total benchmark circuits), as a function of increasing Fc_{in}, when subjected with Perturbed pin-to-wire routing. As can be seen adding flexibility by increasing Fc_{in} does not significantly improve routability for Perturbed pin-to-wire routing. This is because as mentioned in Chapter 3, Section 3.3, unroutes in Perturbed pin-to-wire routing exist because there is no connecting path between the net's source and the target wire segment. Since increasing Fc_{in} does not increase the ways to get out of a logic block, or the ways to enter a wire segment, it does not improve routability for Perturbed pin-to-wire routing. The occasional increase in unroutes are due to the fact that the unroutes depend on the target wires being selected, and as the architecture varies with increasing flexibility, the target wire selection varies slightly, and sometimes one selection is more unfortunate than others.

In the following subsections, we will study the impact of increasing Fc_{in} on Dispersed Perturbed pin-to-wire routing.

4.3.4 Dispersed Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Enhanced Architecture

In this subsection, we observe how the gap between Dispersed Perturbed pin-to-wire routing and pin-to-pin routing responds to an increase in Fc_{in}. As previously mentioned, this experiment is different from the one described in section 4.3.2, as unlike in that experiment, circuits are now exposed to a much higher degree of pin-to-wire routing. Also, the routing-by-abutment scenario is absent, i.e. nets are no longer restricted to designated FPGA modules. Figure 4.21 depicts the percentage increase in geometric mean (over the entire benchmark suite), of Enhanced Dispersed Perturbed over Enhanced

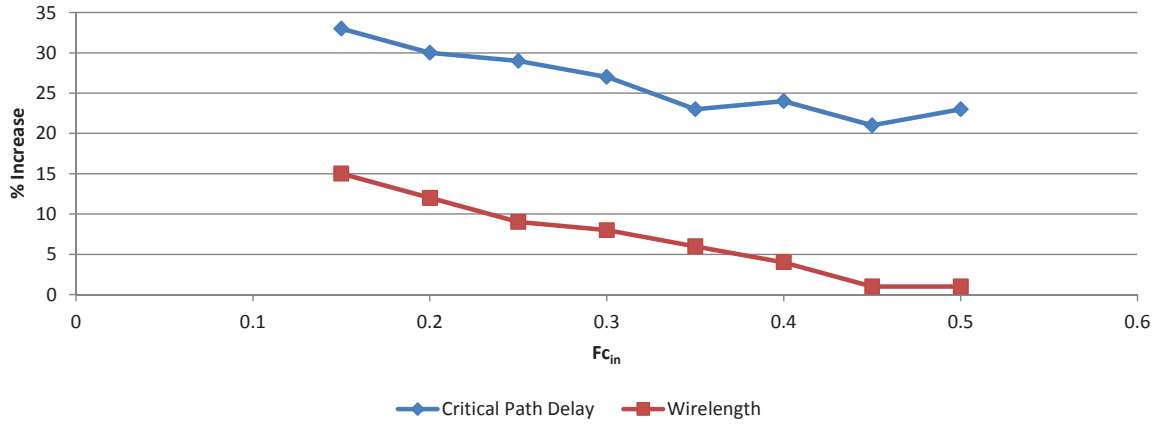


Figure 4.21: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Enhanced Base as a function of increasing Fc_{in}, for critical path delay and wirelength metrics

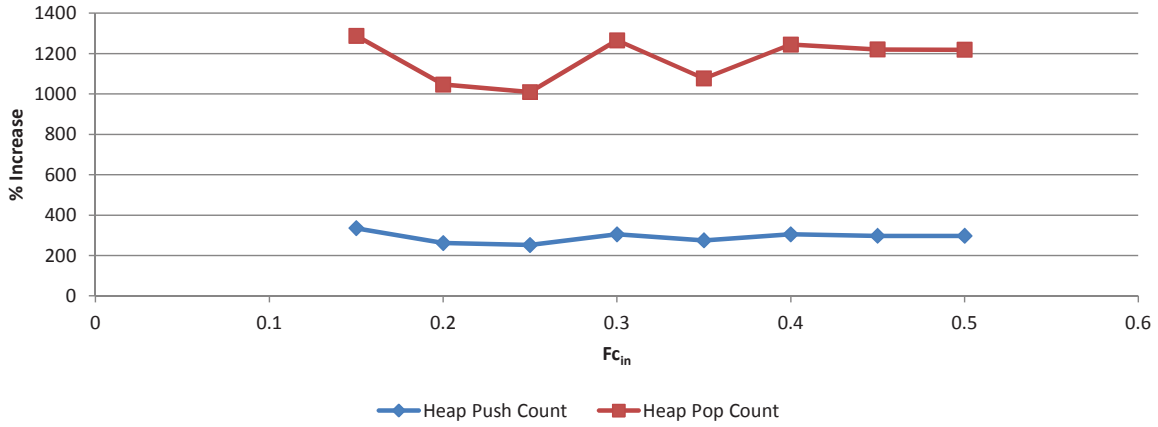


Figure 4.22: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Enhanced Base as a function of increasing Fc_{in}, for heap push and pop count metrics

Base, as a function of increasing Fc_{in}, for the wirelength and critical path delay metrics.

Similarly, Figure 4.22 provides the percentage increase in geometric mean for the heap push count and pop count metrics. All metrics exhibit trends similar to those obtained when Enhanced Perturbed was compared to Enhanced Base, and for the same reasons. However, as compared to the Perturbed case, the decrease in the performance degradation of wirelength and critical path delay with increasing flexibility are much more substantial. It is interesting to note that increasing flexibility not only reduces the gap between pin-to-

pin and pin-to-wire routing for the wirelength metric, it makes it negligible. As indicated in Figure 4.22, both heap push and pop count exhibit trends similar to those exhibited while performing Perturbed pin-to-wire routing, i.e. their performance degradation in comparison to pin-to-pin initially reduces, and then increases and plateaus. Again, as in the case of Perturbed pin-to-wire routing, we can observe that enhancing the architecture by increasing Fc_{in} does indeed reduce the gap between pin-to-wire and pin-to-pin routing for the critical path delay and wirelength metrics. For the router effort; however, the gap reduces only as long as Fc_{in} is not increased beyond 0.35. A slight increase in flexibility reduces the gap, while adding too much flexibility again worsens the gap. In the next subsection, we will study whether enhancing the architecture by increasing Fc_{in} better enables Dispersed Perturbed pin-to-wire routing.

4.3.5 Dispersed Perturbed Pin-to-Wire Routing on Enhanced Architecture vs. Pin-to-Pin Routing on Standard Architecture

As in subsection 4.3.3, we plot the percentage increase in geometric mean of Enhanced Dispersed Perturbed over Base, as a function of increasing Fc_{in}, for all metrics. Figure 4.23 and Figure 4.24 give the summary of the results. The plots indicate that all the metrics have trends similar to those obtained in case of Perturbed pin-to-wire routing (subsection 4.3.3), again for the same reasons.

As illustrated in Figure 4.25, we also plot the variation in the number of circuits that failed to route (out of 20 total circuits), as a function of increasing Fc_{in}. As can be seen from Figure 4.25, routability improves significantly with increase in Fc_{in}. The unroutes reduce from 8 at Fc_{in}=0.15 to 3 at Fc_{in}=0.40. The unroutes do not completely disappear for as indicated in Chapter 3, Section 3.5, some of the unroutes in the Dispersed Perturbed experiment were due to congestion around logic block input pins, while others were due

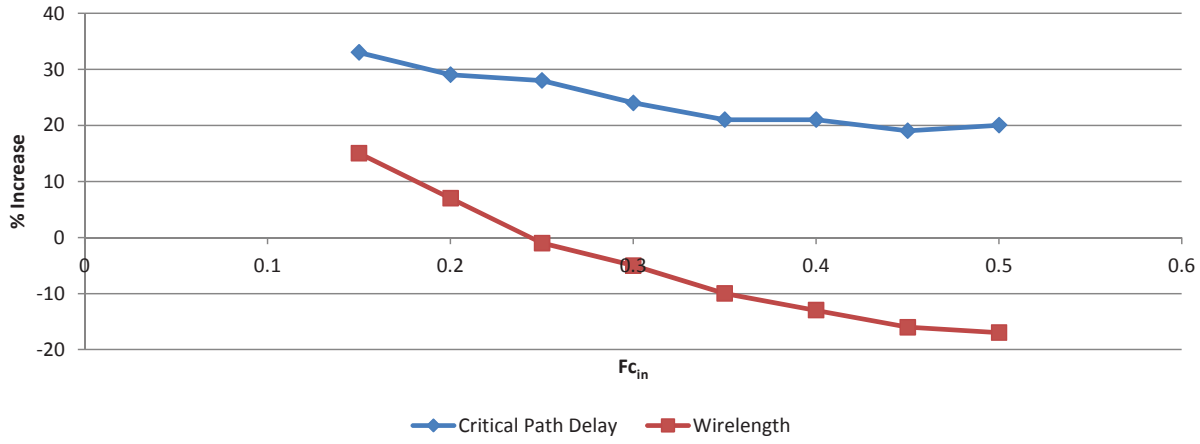


Figure 4.23: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Fc_{in}, for critical path delay and wirelength metrics

to congestion in the wire segments. Increasing Fc_{in} can only help with the former and hence, can only eliminate some of the unroutes, but not all. Also, as mentioned previously in Subsection 4.3.3, the occasional increase in unroutes can be attributed to the fact that the unroutes depend on the target wires being selected, and as the architecture varies with increasing flexibility, the target wire selection varies slightly, and sometimes one selection is more unfortunate than others. The decrease in unroutes is a clear indication that increasing Fc_{in} better enables Dispersed Perturbed pin-to-wire routing.

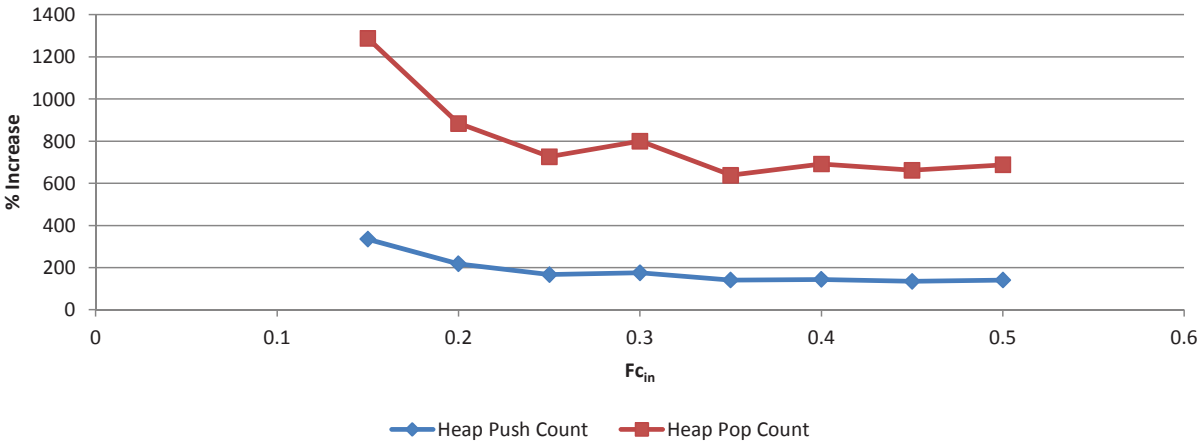


Figure 4.24: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Fc_{in}, for heap push and pop count metrics

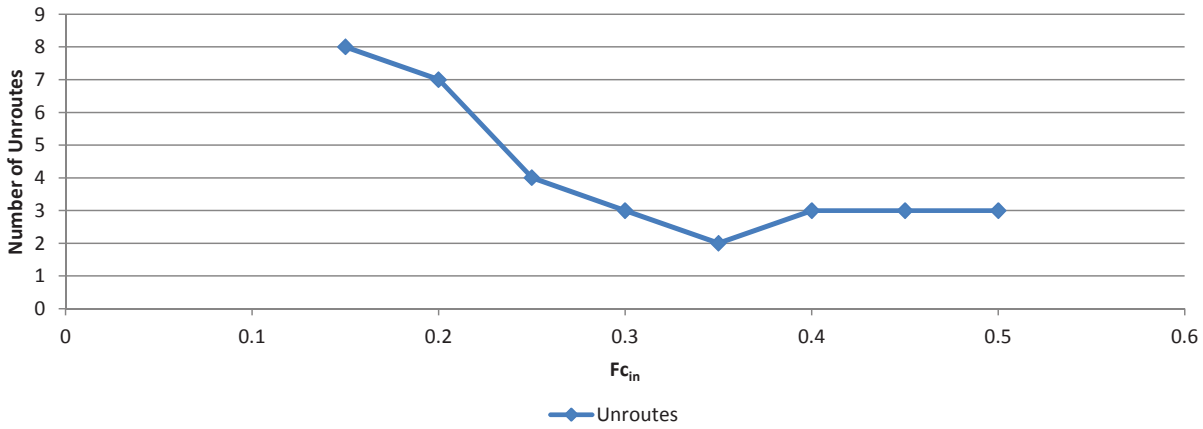


Figure 4.25: Number of Unroutes as a function of increasing Fc_{in} for Dispersed Perturbed Pin-to-Wire Routing

To summarize, we have observed that enhancing the architecture by adding a small amount of Fc_{in} much better enables pin-to-wire routing and also reduces the gap between pin-to-pin and pin-to-wire routing, for all the metrics. However, increasing Fc_{in} beyond 0.35 eliminates the reduction achieved in the gap between pin-to-pin and pin-to-wire routing for the heap push and pop count metrics. This indicates that keeping Fc_{in} between 0.20-0.35 (inclusive) yields maximum benefits.

Next, we will observe the impact of increasing Fc_{out} on pin-to-wire routing.

4.4 Effect of the Output Connection Block Flexibility Parameter (Fc_{out})

The output connection block flexibility parameter or Fc_{out} determines the fraction of tracks in the neighbouring channel that an output pin can drive. We enhance the routing architecture by slowly increasing Fc_{out} from 0.25 (the Fc_{out} value in the Standard Architecture) to 0.50 and observe how these enhancements influence the results of both pin-to-pin and pin-to-wire routing. The output connection block topology is generated by VPR 5.0.2 and has not been modified for this work. Our experiments revealed that

increasing Fc_{out} resulted in negligible improvements in both pin-to-pin and pin-to-wire routing. Analysis revealed that the reason behind that was rather simple. Due to staggering and our use of length 4 wire segments, only about one-fourth of the tracks in a channel can be driven in any neighbouring channel segment. Since an Fc_{out} of 0.25 allows the output pin to drive one-fourth of the tracks in a neighbouring channel, at an Fc_{out} of 0.25, the output pin is driving almost all the available wires in its neighbouring channel segment. Hence, increasing Fc_{out} any more has almost no effect since even though the higher Fc_{out} allows the output pin to drive more wires, more wires that can be driven are not available. As a result, increasing Fc_{out} yields negligible benefits for both pin-to-pin and pin-to-wire routing.

4.5 Determining a low cost architectural modification

In this section, we will present an approach for selecting a cost effective enhancement to the routing architecture, that can help architects select which architecture would best enable pin-to-wire routing at the lowest cost. We will also demonstrate an example use of our approach to pick an enhancement that eliminates unroutes and minimizes the degradation in router effort, while also reducing the performance degradation in both wirelength and critical path delay.

Our first step in this direction would be to express the cost of each flexibility parameter (either Fs or Fc_{in}, we maintain Fc_{out} at its value in the Standard Architecture as it had a negligible impact on pin-to-wire routing), in terms of a common denominator. Second, we use this metric to determine the costs of all our flexibility combinations. Next, we enhance the routing architecture by simultaneously increasing a combination of these parameters, and observe how that influences the performance degradation in pin-to-wire routing when compared to Base. As mentioned above, our aim is to find a parameter

combination, that better enables pin-to-wire routing, while not significantly increasing the cost of the routing architecture. Hence, in the third step, we plot the percentage increase in geometric mean, of Enhanced Pin-to-Wire routing over Base routing, as a function of increasing cost, for all metrics. We then use these plots to demonstrate how to choose a low cost architecture that better enables pin-to-wire routing.

To execute our first step, we will begin by estimating the cost of Fs and Fc_{in} in terms of the number of multiplexer data inputs they generate. While this is a very rough approximation of cost, it would enable us to get an idea of the cost of increasing different architecture flexibility parameters. Since the multiplexer transistors are typically small in size, it is reasonable to simply count the number of inputs as a proxy for area, as each input leads to a roughly proportional number of transistors. Also, for the remainder of this section, we will use the terms multiplexer inputs and multiplexer legs interchangeably.

Recall the details of FPGA's logic block and routing architectures from Subsections 2.1.1 and 2.1.2 respectively. We will use that knowledge to establish the cost of the Fc_{in} parameter in terms of the number of multiplexer inputs it generates. Let us suppose that $Fc_{in} = \alpha$, where α lies between 0 and 1. This means that any logic block input pin can connect to $\alpha \cdot W$ tracks in its neighbouring channel segment. Consequently, the input connection block multiplexer for any input pin in the routing architecture would have $\alpha \cdot W$ multiplexer inputs. Since there are 10 input pins per logic block, the number of input connection block multiplexer inputs per logic block would be given by $10 \cdot \alpha \cdot W$. The number of logic blocks in a routing architecture depend on its grid size, and are given by $x \cdot y$, where x and y give the grid size in the x and y dimensions respectively. Accordingly, the total multiplexer input count (for the entire routing architecture), as a result of Fc_{in} can be given by the equation 4.1, which is as follows:

$$\textit{Total Multiplexer inputs generated by the } Fc_{in} \textit{ parameter} = 10 \cdot \alpha \cdot W \cdot x \cdot y \qquad (4.1)$$

Next, we establish the cost of the Fs parameter in terms of the number of multiplexer inputs it generates. Again, it is important to keep in mind the routing architecture as de-

scribed in Subsection 2.1.2 and the switch block description provided in Subsection 4.2.1. As a result of the single driver approach, out of the W tracks per channel, only $\frac{W}{2}$, are incident upon any given switch block side. And due to staggering of length 4 wires, only a quarter of these incident wires, are ending wires for a given switch block side. Accordingly, at any switch block side, there are $\frac{W}{8}$ ending wires, and $\frac{3W}{8}$ passing wires. Each ending wire can drive a total of Fs other wire segments on the remaining switch block sides, while each passing wire can drive only a total of $\frac{2Fs}{3} + Fs\%3$ wire segments on its orthogonal sides. Thereby, the total wire segments any given switch block side can drive are given by: $\frac{W}{8} \cdot Fs + \frac{3W}{8} \cdot (\frac{2Fs}{3} + Fs\%3)$. The number of wire segments that the wires incident at a switch block can drive actually represent the number of multiplexer inputs they can generate. As a result one can say that for any switch block side, $\frac{W}{8} \cdot Fs + \frac{3W}{8} \cdot (\frac{2Fs}{3} + Fs\%3)$ multiplexer inputs are generated. Since there are 4 sides in a switch block, the number of multiplexer inputs generated per switch block then are: $\frac{1}{2}(W \cdot Fs + 3W \cdot (\frac{2Fs}{3} + Fs\%3))$. The number of switch blocks in the routing architecture are given by $(x+1) \cdot (y+1)$; and so,

$$\textit{Total Multiplexer inputs generated by the Fs parameter} = \frac{1}{2}(W \cdot Fs + 3W \cdot (\frac{2Fs}{3} + Fs\%3)) \cdot (x+1) \cdot (y+1) \tag{4.2}$$

Now that we have established a common denominator in which both our metrics can be expressed, we move to our second step, i.e. we compute the cost of our potential Fs and Fc_{in} combinations in terms of this metric. Table 4.2 gives the values of W, x and y for each benchmark circuit, based on the Standard Architecture. Using these statistics and equations 4.1 and 4.2, we computed the average cost in multiplexer legs (over the entire benchmark suite), for each Fs and Fc_{in} combination we plan to employ in our subsequent experiments. Table 4.3 gives the results of these computations. We refer to the average multiplexer leg count obtained on our Standard Architecture as the *Standard*

Count. We also calculate the percentage increase in the average multiplexer leg count of an Enhanced Architecture, over that of the Standard Architecture, for each Fs and Fc_{in} combination. These computations are illustrated in Table 4.4.

Table 4.2: Circuit Stats Employed in Multiplexer Leg Computations

Benchmark	minW+30%	Grid Size	
		x	y
alu4	48	20	20
apex2	62	23	23
apex4	60	19	19
bigkey	32	36	36
clma	68	47	47
des	34	42	42
diffeq	44	20	20
dsip	36	36	36
elliptic	60	31	31
ex1010	62	35	35
ex5p	62	17	17
frisc	62	30	30
misex3	52	19	19
pdc	78	35	35
s298	44	23	23
s38417	50	41	41
s38584.1	44	41	41
seq	60	22	22
spla	74	31	31
tseng	44	17	17

Table 4.3: Average multiplexer leg count as a function of Fs and Fc_{in}

Average Multiplexer Leg Count (x 10^5)	Fc_{in}			
Fs	0.15	0.2	0.3	0.4
3	2.56	2.76	3.16	3.56
4	3.42	3.62	4.03	4.43
5	4.29	4.49	4.89	5.3
6	4.51	4.71	5.11	5.51
7	5.37	5.58	5.98	6.38
8	6.24	6.44	6.85	7.25
9	6.46	6.66	7.06	7.46
12	8.41	8.61	9.01	9.42
15	10.4	10.6	11	11.4

Table 4.4: Percentage Increase in Average Multiplexer Leg Count over Standard Count as a function increasing Fs and Fc_{in}

% Increase	Fc_{in}			
Fs	0.15	0.2	0.3	0.4
3	0	8	24	39
4	34	42	58	73
5	68	76	92	107
6	76	84	100	116
7	110	118	134	150
8	144	152	168	184
9	153	161	176	192
12	229	237	253	268
15	305	313	329	345

Next, we are going to observe the impact of increasing both Fs and Fc_{in} on pin-to-wire routing, for both Perturbed and Dispersed Perturbed cases. We plot the percentage increase in geometric mean of Enhanced Perturbed over Base, as a function of percentage increase in average multiplexer leg count (over Standard Count), for all metrics. Figures 4.30, 4.29, 4.28 and 4.27 give the summary of results. Figure 4.26 plots the number of circuits that failed to route (out of twenty total benchmark circuits), as a function of increasing multiplexer leg count. Observe that these would be the same graphs we would obtain, if we were to plot the percentage increase in geometric mean of Enhanced Perturbed over Base as a function of increasing Fs and Fc_{in}. In other words, each data point on these graphs corresponds to an architecture with a particular Fs and Fc_{in} value. Table 4.4 can be used to look up the Fs and Fc_{in} combination corresponding to a given percentage increase in multiplexer leg count cost.

Using these plots, we will now show how our multiplexer leg count approach can be used to pick low cost enhancements that yield maximum benefits. We will pick a low-cost architecture that eliminates unroutes, and minimizes the degradation in critical path delay and wirelength (i.e. gives maximum improvement in performance). Here, we focus first on improving critical path delay and wirelength as opposed to router effort, because we wish to respect one of our original motivations i.e. routing-by-abutment. In routing-by-abutment inter-modular routing happens offline, and hence, performance is more of a priority then reducing inter-modular compile time. Neverthless, we do demonstrate how our approach can be applied to the heap push and pop count metrics, for readers that are interested.

The plot in Figure 4.26 indicates that as the percentage increase in multiplexer leg count increases, the unroutes tend to disappear. The points in red highlight the cost enhancements for which the unroutes are zero. As our first goal is to eliminate unroutes, for the remaining metrics, we will only consider architectures for whom the the unroutes are zero. Our next main aim is to minimize the degradation in critical path delay.

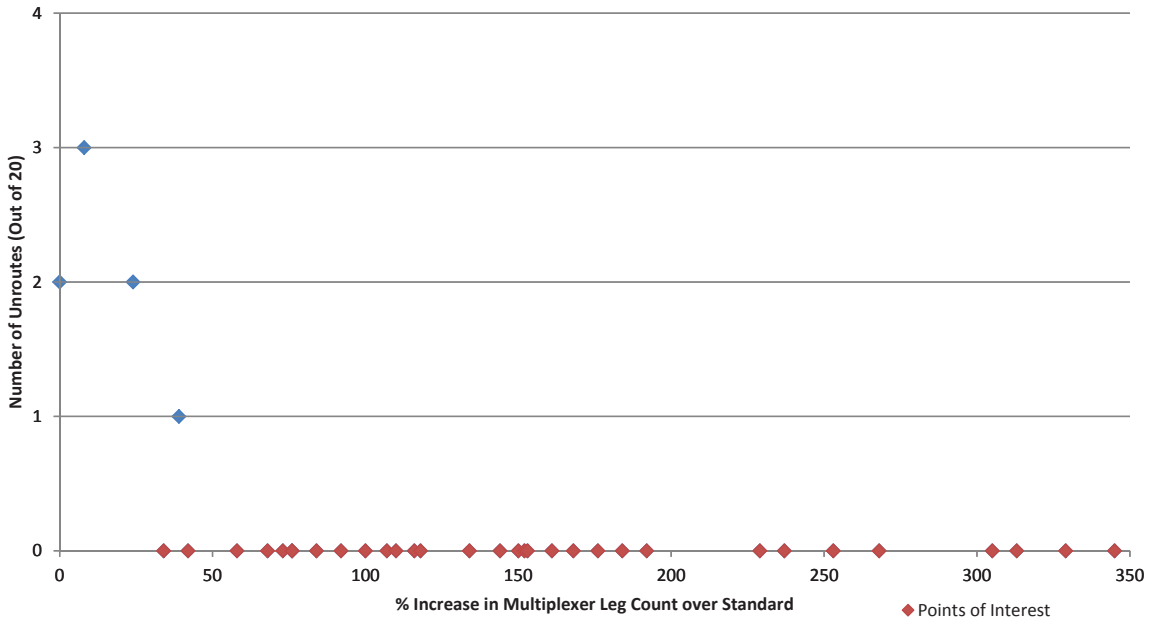


Figure 4.26: Number of Unroutes as a function of increasing multiplexer legs for Perturbed Pin-to-Wire Routing

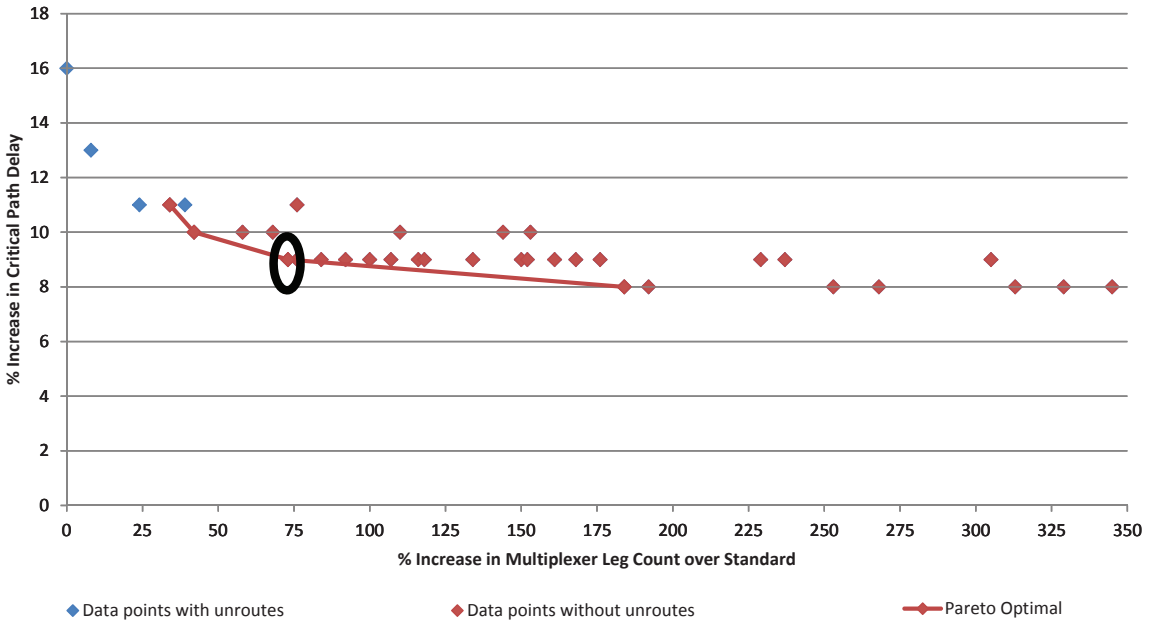


Figure 4.27: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Multiplexer Leg Count, for critical path delay metrics

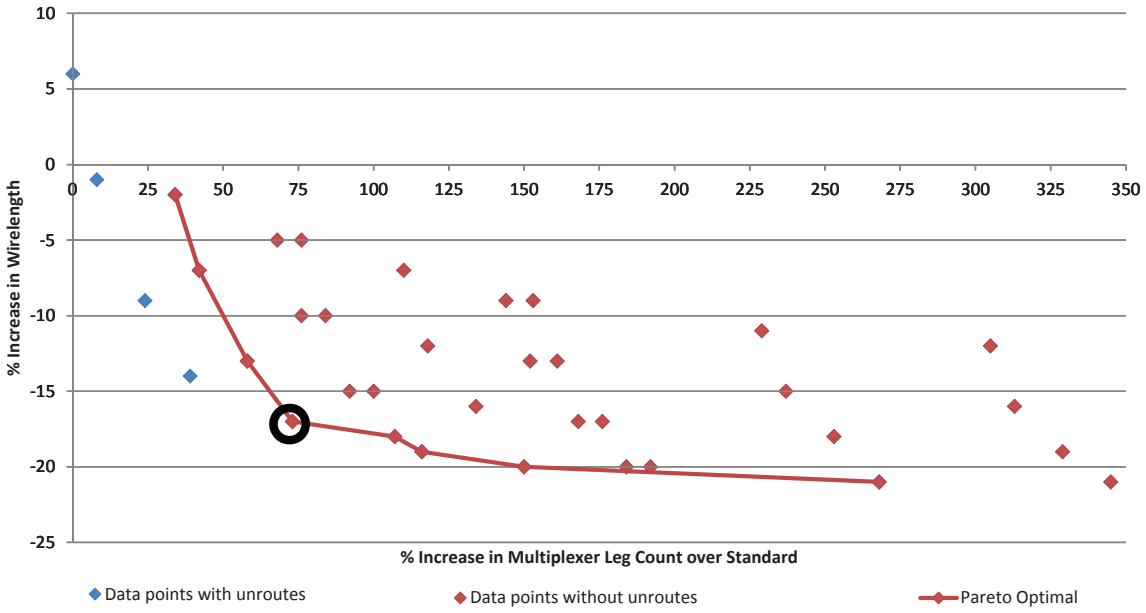


Figure 4.28: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Multiplexer Leg Count, for wirelength metrics

Figure 4.27 illustrates how the degradation in critical path delay varies with respect to increasing multiplexer leg count. In that figure, the data points in red highlight the cost enhancements for which the unroutes are zero. We plot the Pareto Optimal curve for those data points, and using that pick an enhancement which gives maximum reduction in degradation of critical path delay at a relatively low cost. Figure 4.27 highlights the desired data point using a black circle, which corresponds to an increase in multiplexer leg count of 73%. The Pareto Optimal curve demonstrates that going beyond this cost enhancement yields diminishing returns. The 73% increase in multiplexer leg count corresponds to Fs=4 and Fc_{in}=0.40. Next, we repeat this procedure for the wirelength metric, whose plot is illustrated in Figure 4.28. From the Figure we can observe that the 73% cost enhancement (highlighted using black circle), works for the wirelength metric as well. Albeit, as the graph indicates, at higher costs, we could get more reduction in the wirelength metric. However, that would require paying a high cost for a small gain.

Next, we repeat the procedure for the heap push and pop count metrics. Figures 4.29

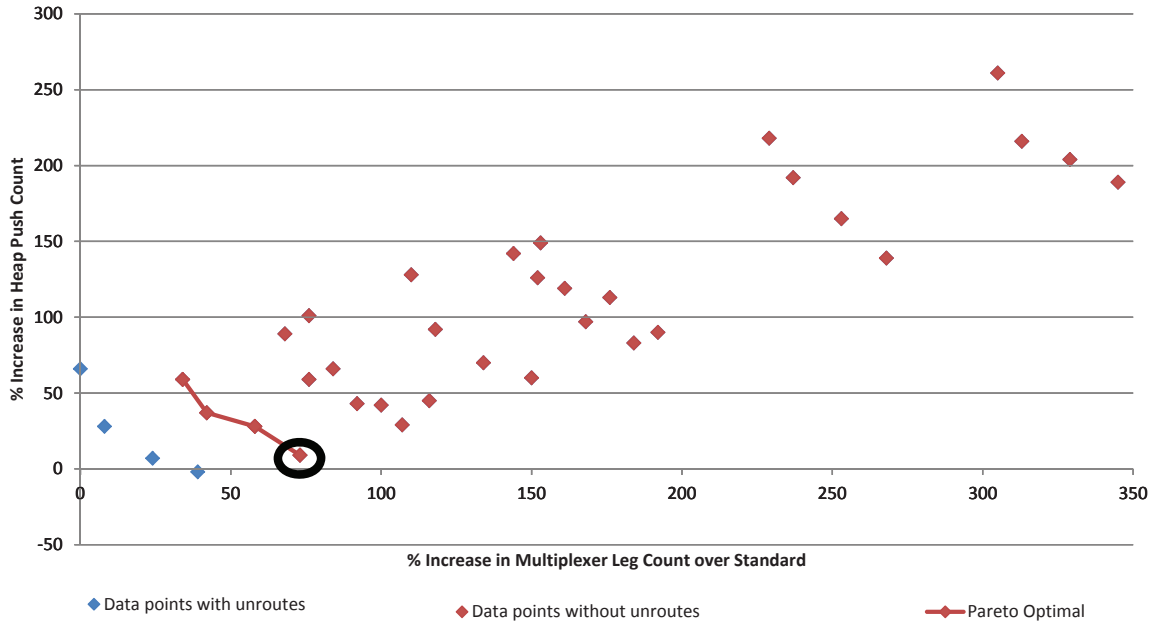


Figure 4.29: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Multiplexer Leg Count, for heap push count metrics

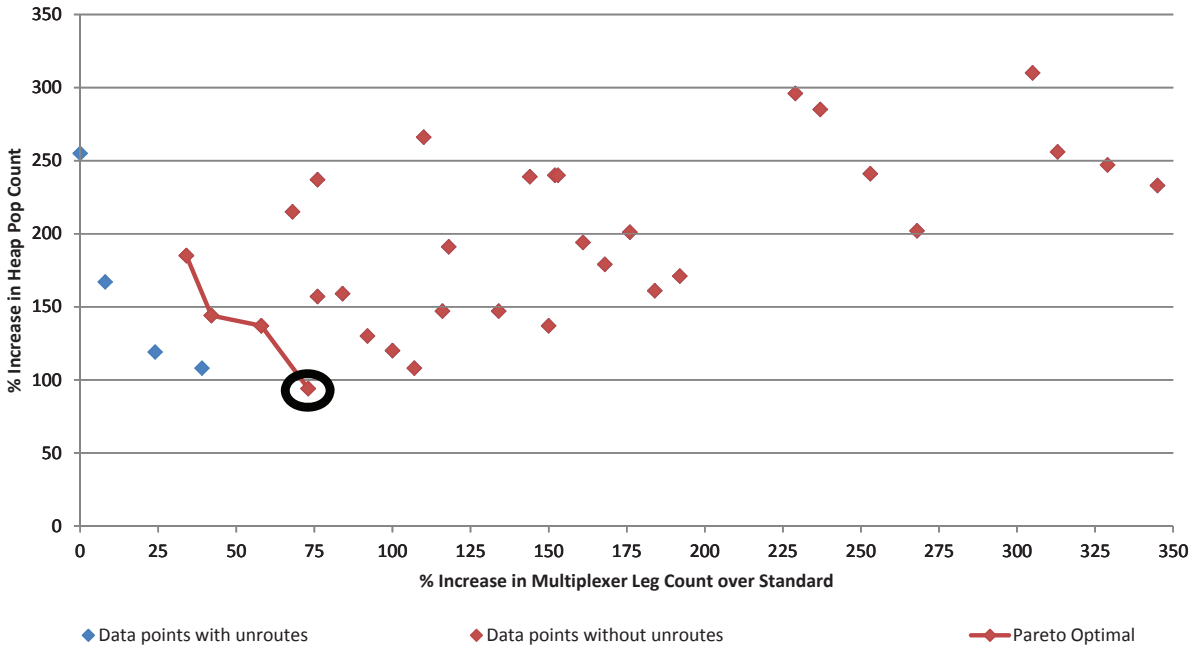


Figure 4.30: Percentage Increase in Geometric Mean of Enhanced Perturbed over Base as a function of increasing Multiplexer Leg Count, for heap pop count metrics

and 4.30 illustrate how increasing multiplexer leg count influences the heap push and pop count metrics respectively. As is highlighted in both graphs, the 73% cost enhancement works well for these metrics as well. So, using these plots, we can conclude that to eliminate unroutes, and minimize degradation in critical path delay and wirelength, Fs should be maintained at 4 and Fc_{in} at 0.40.

Now, we repeat this procedure for the Dispersed Perturbed pin-to-wire routing case. Figures 4.32, 4.33, 4.34 and 4.35 give the summary of results. Figure 4.31 plots the number of circuits that failed to route (out of twenty total benchmark circuits), as a function of increasing multiplexer leg count. On repeating our previous analysis, we observe that for the Dispersed Perturbed case, paying an extra leg cost of 116% yields our desired results. From Table 4.4, we can see that this cost corresponds to a Fs of 6 and a Fc_{in} of 0.40. The data points corresponding to this cost enhancement are highlighted using a black circle in Figures 4.32, 4.33, 4.34 and 4.35.

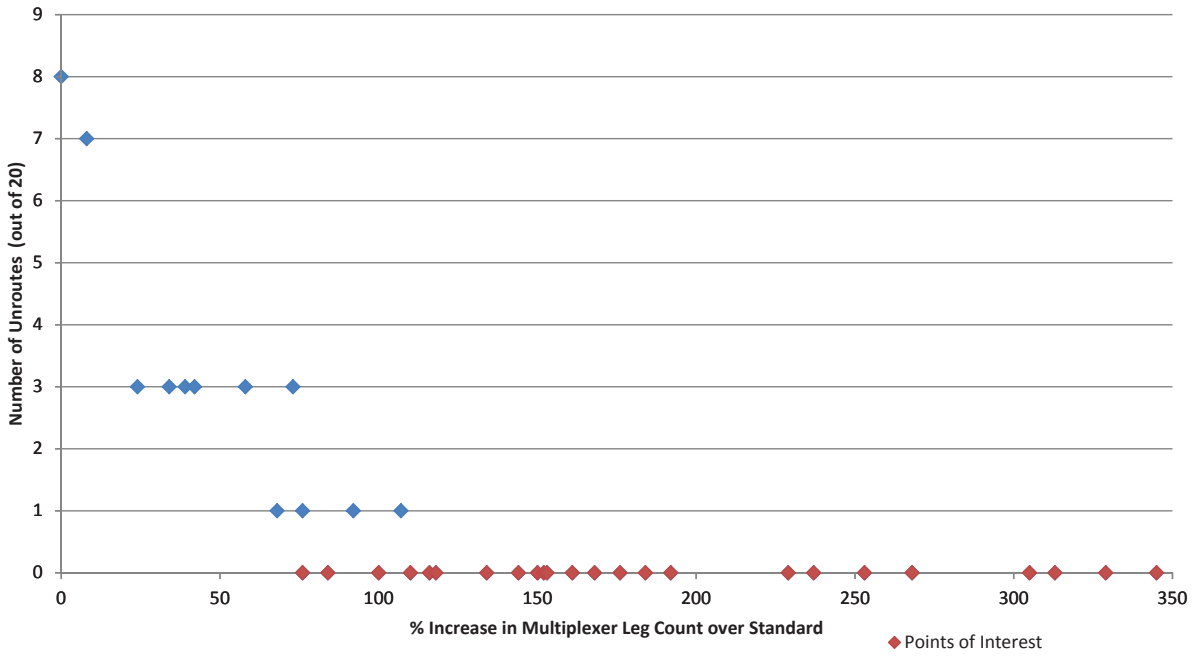


Figure 4.31: Number of Unroutes as a function of increasing multiplexer legs for Dispersed Perturbed Pin-to-Wire Routing

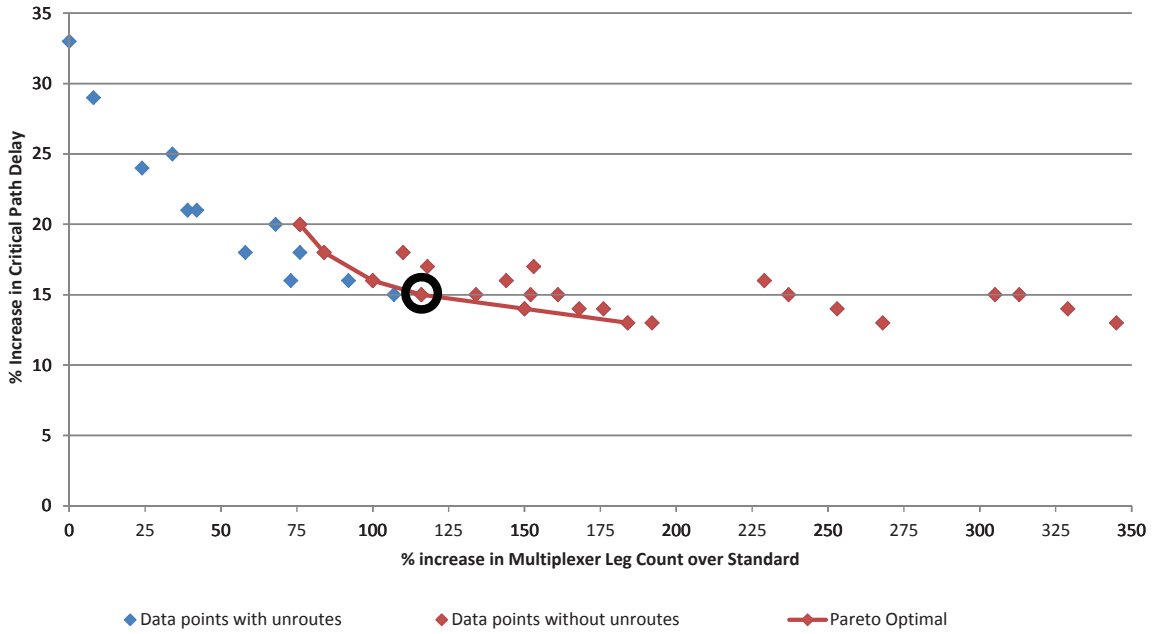


Figure 4.32: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Multiplexer Leg Count, for critical path delay metrics

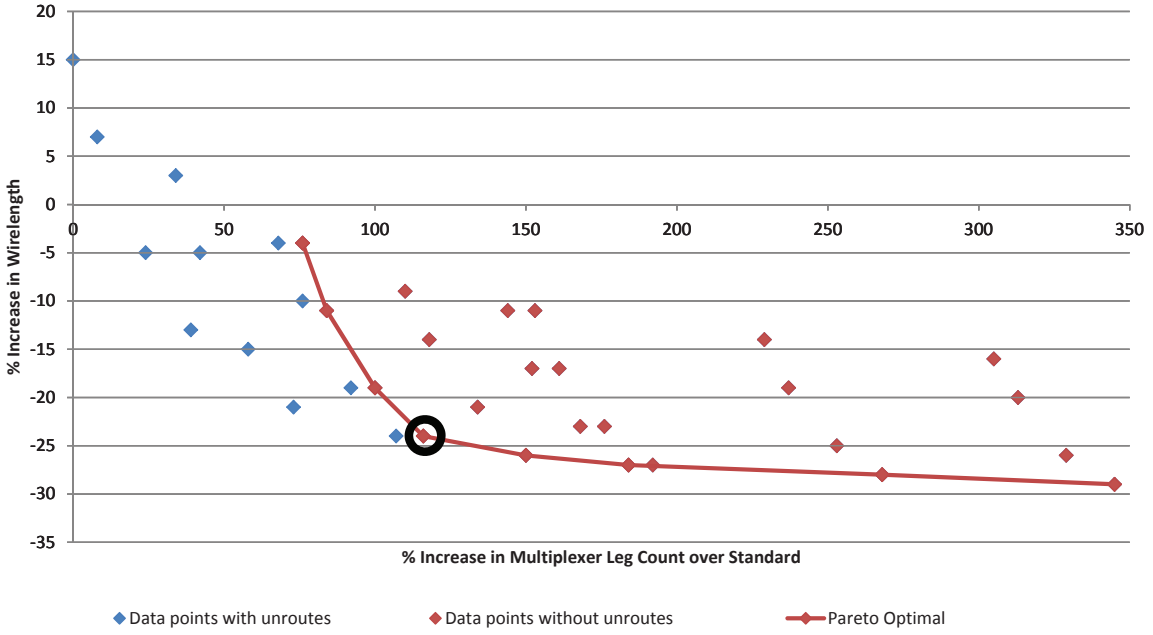


Figure 4.33: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Multiplexer Leg Count, for wirelength metrics

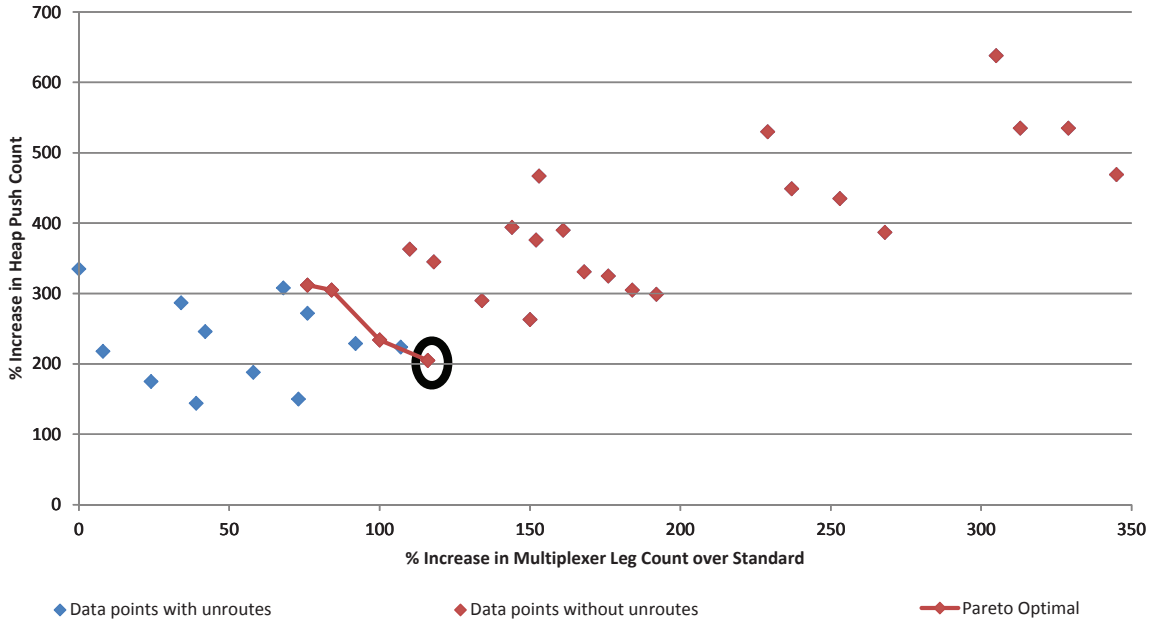


Figure 4.34: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Multiplexer Leg Count, for heap push count metrics

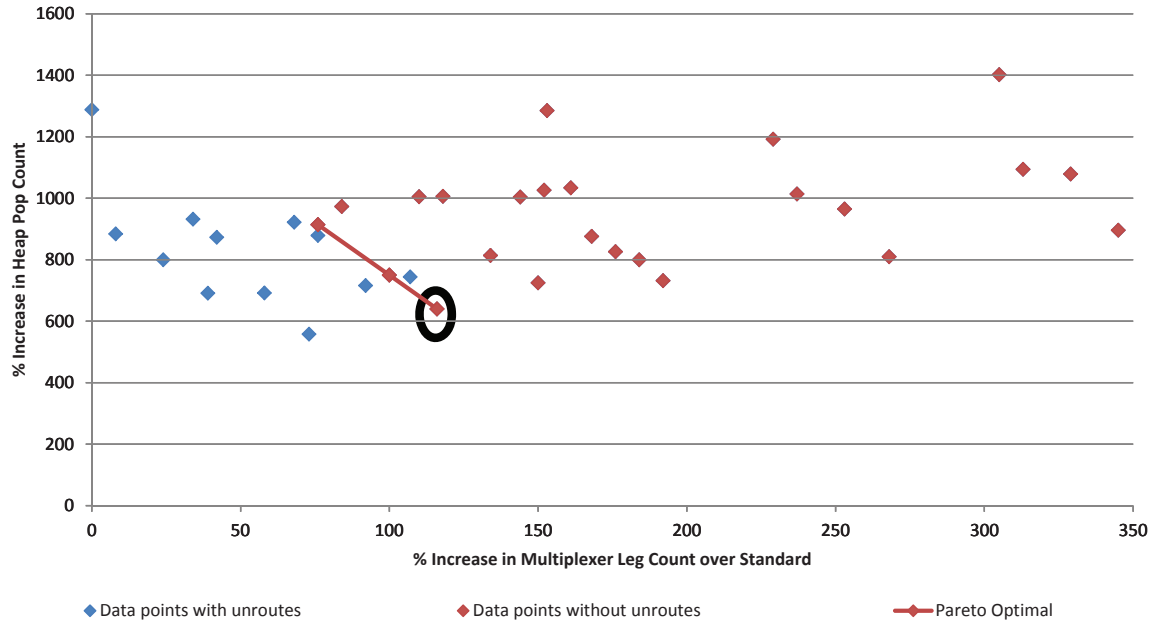


Figure 4.35: Percentage Increase in Geometric Mean of Enhanced Dispersed Perturbed over Base as a function of increasing Multiplexer Leg Count, for heap pop count metrics

4.6 Summary

In this chapter we measured the impact of routing architecture on pin-to-wire routing. We did this by individually modifying each of Fs, Fc_{in} and Fc_{out}. In general, we discovered that adding a small amount of flexibility better enables pin-to-wire routing while reducing the gap between pin-to-pin and pin-to-wire routing. However, adding too much flexibility either worsens the gap or shows no improvement in the gap in router effort between pin-to-pin and pin-to-wire routing. We also presented an approach for selecting a low cost architecture enhancement that yields maximum reduction in degradation at the lowest cost. We showed that maintaining Fs in the range of 4-6 and Fc_{in} at 0.40 gives us the desired benefits.

Chapter 5

Conclusion

The goal of this work was to measure the difficulty of pin-to-wire routing in FPGAs by measuring their impact on wirelength, critical path delay and router effort, as well as the impact of FPGA architecture on those same metrics. To that end, we made the following major contributions:

1. We designed experiments that measured the difficulty faced by a router and a routing architecture in face of pin-to-wire routing in a routing-by-abutment scenario. Compared to normal pin-to-pin routing, the wirelength degraded by 6%, critical path delay by 15% and the router effort increased by a factor of 3.5. There was also a loss of routability as two of the twenty benchmarks failed to route. Further, we realized that selection of the wire segment interface is very crucial to achieve good routability and must be done with extreme care. We believe that this also speaks to the difficulty of pin-to-wire routing.

2. We also measured the rate at which the difficulty of pin-to-wire routing increases as the quantity of pin-to-wire routing in a netlist increases. We achieved this by varying the fraction of nets in a netlist that were subjected to pin-to-wire routing, from 20% to 100%. Results indicate that as the amount of pin-to-wire routing in a netlist increases, the performance degradation for each metric increases. We

measured the percentage increase in geometric mean (over the entire benchmark suite) of pin-to-wire routing over pin-to-pin routing as a function of increasing pin-to-wire routing in a netlist for all metrics. Results showed that when the percentage of nets that were split increased from 20% to 100%, the percentage degradation in wirelength increased from 1% to 15%, and in critical path delay from 8% to 33%. Likewise, the router effort went from being 2x worse to 14x worse (as compared to pin-to-pin routing). Further, the routability of the circuits was also impacted by the amount of pin-to-wire routing, as the number of circuits that failed to route increased from zero at 20% net split to eight at 100% net split.

3. Next, we measured how circuits under high stress, or ones with high demand for routability, respond to pin-to-wire routing. We also measured how circuits under extremely low stress responded to pin-to-wire routing. To acquire these measurements, we repeated our previous experiments while varying the channel width from minimum to 50% higher than minimum. Results indicated that circuits under high stress find it difficult and at most times impossible to achieve routability when faced with pin-to-wire routing. Routability improves as track count increases, i.e. circuits under low stress achieve routability much easily when faced with pin-to-wire routing. Also, circuits under low stress levels suffer lower performance degradation than circuits under high stress levels, when subjected with pin-to-wire routing.

4. Having studied the impact of pin-to-wire routing on a specific router and routing architecture, we then explored the impact of routing architecture on pin-to-wire routing. The goal was to answer two questions: first, does increasing architecture flexibility reduce the performance gap between pin-to-wire and pin-to-pin routing and second, does increasing architecture flexibility better enable pin-to-wire routing. Accordingly, we measured the impact of enhancing our standard architecture by increasing each of Fs, Fc_{in}, and Fc_{out}. We observed that increasing Fc_{out} had

negligible impact on both pin-to-pin and pin-to-wire routing. For both Fs and Fc_{in}, we observed that increasing flexibility certainly better enabled pin-to-wire routing. Moreover, a minor increase in flexibility did reduce the gap between pin-to-pin and pin-to-wire routing for all metrics, but a large increase in flexibility either worsened or gave no improvement in the gap in router effort. Hence, it was found that a small increase in flexibility either Fs or Fc_{in} yielded the best results.

5. Our final step was to suggest an approach for selecting a cost effective enhancement, that can help architects determine which architecture would yield the most benefit at the lowest cost. Using that approach, we found that maintaining Fs in the range of 4-6 and Fc_{in} at 0.40 gave significant reduction in the performance degradation of wirelength and critical path delay, while maintaining a low degradation in router effort. These suggested Fs and Fc_{in} values correspond to a 73%-116% increase in architecture cost over the cost of the standard architecture.

To summarize, one can observe that pin-to-wire routing is no doubt difficult, but it is not impossible and with a careful selection of wire segment interface, can be done under certain circumstances. Also, a relatively low cost enhancement to the routing architecture can much better enable pin-to-wire routing while reducing the gap between pin-to-pin and pin-to-wire routing.

Recall that we had three motivations for pin-to-wire routing, namely: routing-by-abutment, partial-reconfiguration and the use of circuit elements within the routing fabric. As mentioned in Chapter 2, to enable the first two motivations, an alternative method to using pin-to-wire is the use of proxy logic. Given what we now know about pin-to-wire routing, we are in a better position to answer the following question: how does pin-to-wire routing compare to the proxy logic method? The answer to this question is complex, and depends on many factors. We believe our results indicate that when the amount of pin-to-wire routing needed is small, pin-to-wire routing would be cheaper (in terms of both area and delay) as compared to the proxy logic method. However, when

the amount of pin-to-wire routing required is large, and the architecture flexibility cannot be increased (as would be the general case), then the proxy logic approach would work better.

5.1 Future Work

This work lays the basis which can be used by architects in the future to make guided decisions regarding the use and applications of pin-to-wire routing. In future, this work can be extended to be performed on architectures with heterogeneity and on bigger, industrial scale benchmarks. That would also address the question of the effect of the presence of structured carry chains on pin-to-wire routing, which may have a strong impact due to their relative inflexibility. It would also be interesting to explore how routing-by-abutment can be used in practical ways.

Bibliography

[1] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, pp. 288–298, Mar. 2004.

[2] J. Rose, V. Betz, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, Massachusetts: Kluwer Academic Publishers, 1999.

[3] P. Simpson and A. Jagtiani, "How to achieve faster compile times in high-density FPGAs," Altera, Jan. 2007. [Online]. Available: http://www.eetimes.com/design/programmable-logic/4015092/How-to-achieve-faster-compile-times-in-high-density-FPGAs

[4] "Xilinx Announces worlds highest capacity FPGA," Oct. 2011. [Online]. Available: http://www.eetimes.com/electronics-news/4230048/Xilinx-announces-world-s-highest-capacity-FPGA

[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," in *ACM/SIGDA FPGA*, ser. FPGA '05, 2005, pp. 14–20.

[6] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in Stratix-III^{TM} and Stratix-IV^{TM}," in *ACM/SIGDA FPGA*, ser. FPGA '09, 2009, pp. 33–42.

[7] "Stratix IV Device Handbook," Altera, Dec. 2011. [Online]. Available: http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf

[8] "7 Series FPGAs Overview," Xilinx, Mar. 2012. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[9] N. I. (NI), "Introduction to FPGA Technology: Top 5 Benefits," Apr. 2012. [Online]. Available: http://www.ni.com/white-paper/6984/en

[10] M. Guruswamy, R. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez, and L. Jones, "Cellerity: A fully automatic layout synthesis system for standard cell libraries," in *DAC*, June 1997, pp. 327–332.

[11] C.-T. Lin, D.-S. Chen, Y.-W. Wang, and H.-H. Ho, "Modem floorplanning with abutment and fixed-outline constraints," in *IEEE ISCAS*, vol. 6, no. 6214–6217, May 2005.

[12] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and J. Graf, "Wires on demand: Run-time communication synthesis for reconfigurable computing," in *FPL*, Aug. 2007, pp. 513–516.

[13] "Xilinx Partial Reconfiguration User Guide," Xilinx, July 2011. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/ug702.pdf

[14] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *IEEE FPT*, Dec. 2004, pp. 41–48.

[15] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *ACM/SIGDA FPGA*, 2009, pp. 133–142.

[16] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and routing tools for the Triptych FPGA," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

[17] R. Nair, "A Simple Yet Effective Technique for Global Wiring," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 6, no. 2, pp. 165–172, Mar. 1987.

[18] F. Rubin, "The Lee Path Connection Algorithm," *Computers, IEEE Transactions on*, vol. C-23, no. 9, pp. 907–914, Sept. 1974.

[19] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *Electronic Computers, IRE Transactions on*, vol. EC-10, no. 3, pp. 346–365, Sept. 1961.

[20] D. Koch, C. Beckhoff, and J. Torresen, "Zero logic overhead integration of partially reconfigurable modules," in *SBCCI*, 2010, pp. 103–108.

[21] Y. O. M. Moctar, N. George, H. Parandeh-Afshar, P. Ienne, G. G. Lemieux, and P. Brisk, "Reducing the cost of floating-point mantissa alignment and normalization in FPGAs," in *ACM/SIGDA FPGA*, Feb. 2012, pp. 255–264.

[22] S. Oldridge and S. Wilton, "Placement and routing for FPGA architectures supporting wide shallow memories," in *IEEE FPT*, Dec. 2003, pp. 154–161.

[23] S. J. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory," PhD thesis, University of Toronto, 1997. [Online]. Available: http://www.ece.ubc.ca/~stevew/papers/pdf/thesis.pdf

[24] "iFAR intelligent FPGA Architecture Repository," Feb. 2008. [Online]. Available: http://www.eecg.utoronto.ca/vpr/architectures/

[25] I. Kuon and J. Rose, "Area and delay trade-offs in the circuit and architecture design of FPGAs," in *ACM/SIGDA FPGA*, 2008, pp. 149–158.

[26] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide Version 3.0*, MCNC, Jan. 1991.

[27] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-vincentelli, "SIS: A System for Sequential Circuit Synthesis," Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, Department of Electrical Engineering and Computer Science University of California, Berkeley, Tech. Rep., May 1992.

[28] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *ACM/SIGDA FPGA*, 2000, pp. 203–213.

[29] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for fpgas," in *ACM FPGA*, 1995, pp. 111–117.

[30] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *Solid-State Circuits, IEEE Journal of*, vol. 26, no. 3, pp. 277–282, Mar. 1991.