

**INTERCONNECT DRIVER DESIGN FOR LONG WIRES IN
FIELD-PROGRAMMABLE GATE ARRAYS**

by

Edmund Lee

B.A.Sc., University of Toronto, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Electrical and Computer Engineering

THE UNIVERSITY OF BRITISH COLUMBIA

June 2006

© Edmund Lee, 2006

Abstract

Designers of field-programmable gate arrays (FPGAs) are always striving to improve the performance of their designs. As they migrate to newer process technologies in search of higher speeds, the challenge of interconnect delay grows larger. For an FPGA, this challenge is crucial since most FPGA implementations use many long wires.

A common technique used to reduce interconnect delay is repeater insertion. Recent work has shown that FPGA interconnect delay can be improved by using unidirectional wires with a single driver at only one end of a wire. With this change, it is now possible to consider interconnect optimization techniques such as repeater insertion.

In this work, a technique to construct switch driver circuit designs is developed. Using this method, it is possible to determine the driver sizing, spacing and the number of stages of the circuit design. A computer-aided design model of the new circuit designs is developed to assess the impact they have on the delay performance of FPGAs. Results indicate that, by using the presented circuit design technique, the critical path can be reduced by 19% for short wires, and up to 40% for longer wires.

Table of Contents

Abstract	ii
Table of Contents	iii
List of Tables	v
List of Figures	vi
Acknowledgements	ix
Chapter 1 Introduction	1
1.1 Motivations and Objectives	4
1.2 Contributions.....	5
1.3 Overview.....	6
Chapter 2 Background	7
2.1 FPGA Overview.....	7
2.1.1 <i>Routing</i>	9
2.1.2 <i>FPGA CAD Flow</i>	12
2.1.3 <i>VPRx</i>	13
2.2 Interconnect Design Theory.....	15
2.2.1 <i>Deep-Submicron Interconnect</i>	15
2.2.2 <i>Interconnect Delay Models</i>	18
2.2.3 <i>Interconnect Driver Design</i>	20
2.3 Remarks	24
Chapter 3 Circuit Design of Unidirectional Interconnect	25
3.1 Design of Unidirectional FPGA Switch Drivers	25
3.1.1 <i>Components of FPGA Switch Drivers</i>	26
3.1.2 <i>Switch Driver Design Goals</i>	32
3.1.3 <i>Switch Driver Design Parameters</i>	34
3.2 Rapid Design Space Exploration	36
3.2.1 <i>Analytical Delay Model</i>	36
3.2.2 <i>Design Space Sweeps</i>	38
3.2.3 <i>Results</i>	39
3.2.4 <i>Summary</i>	43
3.3 Accurate Switch Driver Circuit Delay Modeling	44
3.3.1 <i>Delay Modeling through Circuit Characterization</i>	45
3.3.2 <i>Circuit Characterization Process</i>	47

3.3.3	<i>Characterization Results & Analysis</i>	47
3.3.4	<i>Delay Concatenation</i>	53
3.3.5	<i>Verification of Delay Concatenation</i>	54
3.4	Design Parameter Search	55
3.4.1	<i>Circuit Construction</i>	55
3.4.2	<i>Segment Length Sweep</i>	57
3.5	Circuit Design Results & Analysis	60
3.5.1	<i>Multiplexing Intervals</i>	63
3.5.2	<i>Distributed Buffering</i>	64
Chapter 4 CAD Circuit Architecture Modeling and Evaluation.....		67
4.1	Goals of the CAD Circuit Evaluation	67
4.1.1	<i>Comments on Area Overhead</i>	68
4.2	Experimental Methodology	69
4.3	Circuit Architecture Modeling	70
4.3.1	<i>Early Turn Modeling (ETM)</i>	70
4.3.2	<i>Distributed Buffering</i>	78
4.3.3	<i>Fast Paths</i>	80
4.3.4	<i>Multiplexer Delay Modeling</i>	82
4.3.5	<i>Circuit Characterization for VPRx</i>	83
4.4	Experimental Results	88
4.4.1	<i>Driver Designs Modeled in VPRx</i>	88
4.4.2	<i>Benchmark Logic Circuits</i>	90
4.4.3	<i>Experiments</i>	92
4.4.4	<i>Critical Path Delay Results</i>	93
4.4.5	<i>Turn Count Analysis</i>	99
4.4.6	<i>Runtime</i>	104
Chapter 5 Conclusions and Future Work.....		106
5.1	Future Work	107
5.1.1	<i>Circuit Design</i>	108
5.1.2	<i>Future Work for CAD</i>	109
References		111
Appendix A - Wire Models.....		114
Appendix B - VPRx Results		117

List of Tables

Table 3.1 - Comparison of Multiplexer Architectures.....	29
Table 3.2 - Driver Design Parameters.....	35
Table 3.3 - Typical Parasitics in Deep Submicron Process Technology	37
Table 3.4 - Delay-Driven Results	40
Table 3.5 - AreaDelay-Driven Results.....	41
Table 3.6 - Characterized Sections	46
Table 3.7 - Best Delay-per-millimeter for Various Sections	51
Table 3.8 - Concatenation Verification Results (180nm).....	54
Table 3.9 - Concatenation Verification Results (90nm).....	55
Table 3.10 - Distributed Driver Design Results for 180nm.....	61
Table 3.11 - Lumped Driver Design Results for 180nm	61
Table 3.12 - Driver Design Results for 90nm.....	62
Table 3.13 - Lumped Driver Design Results for 90nm	62
Table 4.1 - Lumped Driver Designs Used for Experiments	90
Table 4.2 - Distributed Driver Designs Used for Experiments.....	90
Table 4.3 - Benchmark Circuit Array Sizes.....	92
Table 4.4 - Overview of Experiments.....	93
Table 4.5 - Critical Path Results	94
Table 4.6 - Turn Count Changes Due to Addition of ETM.....	100
Table 4.7 - Turn Count Changes Due to Addition of Fast Paths	101
Table 4.8 - Turn Count Changes Due to Addition of Distributed Features.....	104

List of Figures

Figure 1.1 - Routing Example of a Net in an ASIC.....	2
Figure 1.2 - Routing Example of a Net in an FPGA.....	2
Figure 1.3 - Example of FPGA Interconnect.....	3
Figure 1.4 - Example of a Switch Driver Path Delay Profile	4
Figure 2.1 - FPGA Architecture with Switch Block Detail.....	9
Figure 2.2 - Representation of a Wire in a Bidirectional Architecture.....	10
Figure 2.3 - Tristate Driver Example.....	10
Figure 2.4 - Representation of a Wire in a Unidirectional, Single-Driver Architecture...	11
Figure 2.5 - Unidirectional Driver Example.....	11
Figure 2.6 - Example of a Routing Resource Graph [6].....	14
Figure 2.7 - Deep-submicron Parasitic Capacitances	17
Figure 2.8 - Elmore Delay Example [12].....	19
Figure 2.9 - RC Model of a Buffer	21
Figure 3.1 - Architectural Location of FPGA Switch Drivers.....	26
Figure 3.2 - FPGA Switch Driver Components.....	26
Figure 3.3 - Level-Restoring Circuit with 2:1 NMOS Pass Transistor Multiplexer.....	27
Figure 3.4 - Binary Tree Multiplexer.....	28
Figure 3.5 - Flat Multiplexer.....	28
Figure 3.6 - 2-Level Multiplexer	29
Figure 3.7 - Multiplexer Fast Path	30
Figure 3.8 - Ubiquitous CMOS Buffer	32
Figure 3.9 - Driver Example.....	32
Figure 3.10 - Path Delay Profile	34
Figure 3.11 - Block Diagram Identifying Driver Design Parameters.....	35
Figure 3.12 - Elmore Model of Buffer & Wire Delay	37
Figure 3.13 - Parameters Being Swept	38
Figure 3.14 - Design Space Sweep Pseudo Code	39

Figure 3.15 - Areadelay-Driven Minimum-Delay Wire Distribution Plot for a 4mm Wire	42
Figure 3.16 - Areadelay-Driven Minimum-Delay Wire Distribution Plot for a 4mm Wire Zoomed in on L2 Axis.....	43
Figure 3.17 - Programmable Driver Example.....	45
Figure 3.18 - Testbench Configuration.....	47
Figure 3.19 - Delay vs. Buffer size and Wirelength for 180nm 1x1x nomux Design.....	48
Figure 3.20 - Delay vs. Wirelength for Different Buffer sizes for 180nm 1x1x nomux Design - Wirelength Axis.....	48
Figure 3.21 - Delay/mm vs. Buffer size and Wirelength for 180nm 1x1x nomux Design	49
Figure 3.22 - Delay/mm vs. Wirelength for Different Buffer sizes for 180nm 1x1x nomux Design - Wirelength Axis.....	50
Figure 3.23 - Buffer size Selection for 180nm 1x1x nomux	52
Figure 3.24 - Buffer size Selections for 180nm 1x1x nomux Design	53
Figure 3.25 - Circuit Concatenation Example	54
Figure 3.26 - Driver Construction Template.....	56
Figure 3.27 - L0 Sweep for a 2mm Wire in 180nm 1x1x for N=3 Stages.....	57
Figure 3.28 - Buffer sizes Used for L0 Sweep in 180nm 1x1x	58
Figure 3.29 - Multi-N L0 Sweep for a 2mm Wire in 180nm 1x1x.....	59
Figure 3.30 - Multi-N L0 Sweep for a 4mm Wire in 180nm 1x1x.....	60
Figure 3.31 - Multiplexing Intervals for various Technologies.....	63
Figure 3.32 - Path Delay Profile Plots for 180nm 1x1x for 1mm-4mm.....	66
Figure 4.1 - Experimental Methodology Flow	69
Figure 4.2 - Early Turns.....	71
Figure 4.3 - Original Routing Resource Graph of VPRx.....	73
Figure 4.4 - Routing Resource Graph for VPRx with ETM Enabled.....	73
Figure 4.5 - Wire Delay Calculation Example.....	75
Figure 4.6 - Routing Resource Graph with ETM and Distributed Buffers Shown.....	78
Figure 4.7 - Modeling a Distributed Driver with Different Architectural Wirelengths....	79

Figure 4.8 - The Effect on the Path Delay Profile of Modeling Circuits with Different Architectural Wirelengths	80
Figure 4.9 - Signal Path through a Switch Block Using Normal or Fast Paths	81
Figure 4.10 - Multiplexer Fanin Delay with a 2mm Wire 180nm 1x1x	83
Figure 4.11 - How Switch Circuit Delay is Modeled in VPRx	83
Figure 4.12 - VPRx Path Delay Profile Extraction.....	86
Figure 4.13 - Path Delay Profiles for 2mm Wire in 180nm 1x1x.....	87
Figure 4.14 - Delay Breakdown for 0.5mm Wire.....	97
Figure 4.15 - Delay Breakdown for 2.0mm Wire.....	98
Figure 4.16 - Delay Breakdown for 3.0mm Wire.....	98
Figure 4.17 - Early Turn Routing Example for L4 Wires.....	101
Figure 4.18 - Routing Choices Due to Fast Paths with Early Turns in an L4 Architecture	103

Acknowledgements

I would to like thank my academic supervisors Dr. Guy Lemieux and Dr. Shahriar Mirabbasi for their advice, teachings and support throughout my graduate degree. I am very fortunate to have two professors who have taught me a great deal about research and the technical aspects of our field.

I would also like to thank the faculty and staff of the SOC lab for making the lab a wonderful academic home. Thanks for making me feel welcome and always offering assistance. My gratitude goes out to all the members of the FPGA and analogue research group for their many helpful insights and of course, excellent company.

I am grateful for the use of the Westgrid computing resources at UBC. Much of the research in this thesis was facilitated through the use of Westgrid.

Finally, I would like to thank my family for their constant support and enthusiasm.

Chapter 1

Introduction

Field-programmable gate arrays (FPGAs) are large integrated circuits comprised of blocks of programmable logic interconnected with programmable routing circuits. The demand for increasing their performance has driven FPGA designers in search of the latest process technologies. With each new technology generation, FPGAs have grown larger and increasingly dense, providing more logic with a smaller feature size. As one can expect, the wiring demands of these devices have also increased.

In deep submicron process technologies, interconnect has been identified as one of the most critical challenges facing integrated circuit designers [1]. In an FPGA, this is even more important, as 60-80% of the delay is caused by the interconnect [2]. Although extensive efforts have been made on interconnect optimization by means of repeater insertion for ASIC designs, few studies have investigated the optimization of circuit design for FPGA interconnect.

Techniques used in general ASIC interconnect optimization cannot be directly applied because the FPGA interconnect design problem is different in nature. Fortunately, due to its rigid structure and point to point nature, the topology of FPGA interconnect does not possess the complex fanout trees found in ASIC designs as seen in Figure 1.1.

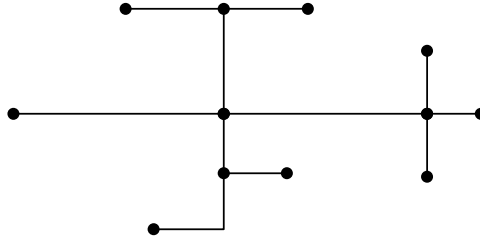


Figure 1.1 - Routing Example of a Net in an ASIC

Instead, routing resources in an FPGA are made up of long straight wires which make up the predetermined paths of the routing resource architecture, as shown in Figure 1.2. This simplification is welcome as it considerably reduces the complexity of the circuit design problem. However, it is not without its own challenges.

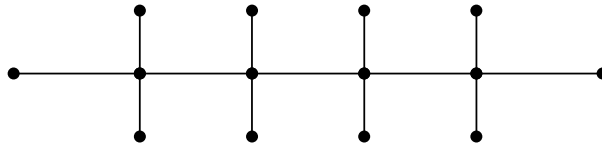


Figure 1.2 - Routing Example of a Net in an FPGA

Like an ASIC, wires still fanout to numerous points in FPGAs, but whether or not a fanout is used in an FPGA interconnect is not known until *after* a user circuit is fully implemented. During operation, the path of a signal can occupy either a part or the entire length of a wire. Our experiments have demonstrated that 50% to 87% of signal paths in an FPGA routing solution leave the wire before arriving at the end. Such turns are called **early turns** (Figure 1.3). In order to assess early turn delays, this work will often consider the delay to several points along the interior of a wire. In general, these delays are referred to as the **midpoint delays** of a wire. To the author's knowledge, the concept of early turns and midpoint delay has not been previously examined in FPGA research.

The other major difference between FPGA and ASIC interconnect is that the FPGA interconnect must be programmable. This requirement introduces multiplexer circuits which can

adversely affect delay. The closed form models often used in the development of general interconnect optimization techniques are less accurate at modeling such circuits, making it difficult to apply the techniques from previous work on FPGAs. While closed-form analytical techniques are useful for rough approximations, they are not accurate enough to compare significantly different circuit implementations. This accuracy is vital to obtaining the best possible speed performance from FPGAs.

Figure 1.3 presents an example of an FPGA interconnect made up of wires, multiplexers and drivers. This research attempts to address the problem of wire delay in FPGAs by developing an accurate approach to design and evaluate interconnect driver circuits for FPGAs.

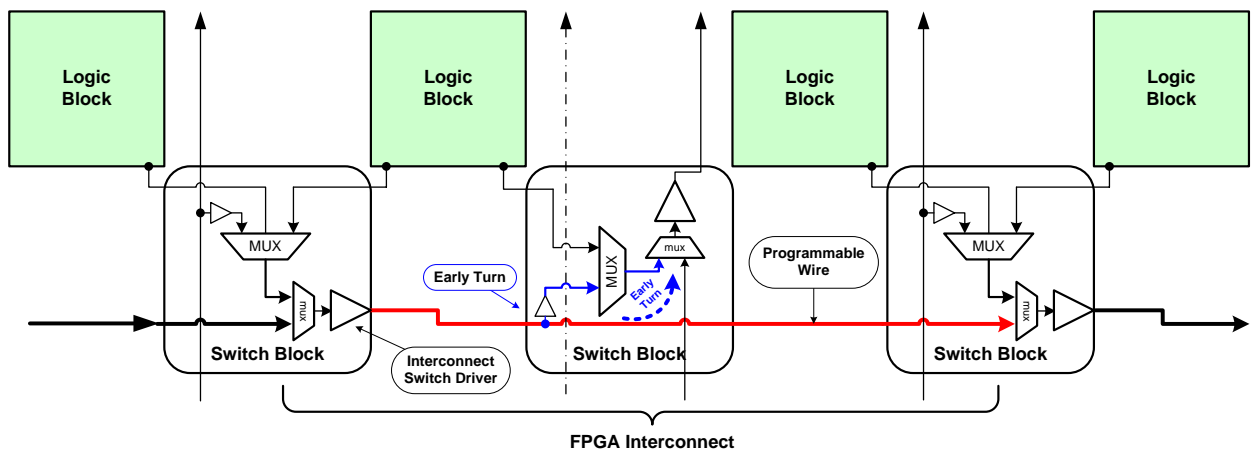


Figure 1.3 - Example of FPGA Interconnect

By taking advantage of the recent shift to FPGA architectures with a single driver per wire [3], it is possible to reduce midpoint delays, in addition to end-to-end (or **endpoint**) delay, to speed up performance in FPGAs through the use of distributed driver designs. As an example, Figure 1.4 presents a distributed driver design which could be used to implement the interconnect switch driver and the programmable wire in Figure 1.3. This sample driver circuit is made up of 2 distributed drivers of size B0 and B1. Using a **path delay profile** (PDP), the delay of the signal can be examined from its origin to the end of the wire, or to any point in between. PDPs for two

designs, a lumped driver and a distributed driver, are shown in Figure 1.4. It can be seen that the midpoint delays of the first half of the wire are significantly faster than midpoint delays of the lumped-design PDP. This indicates that distributed driver design has the potential to improve the delay of the early turn shown in Figure 1.3.

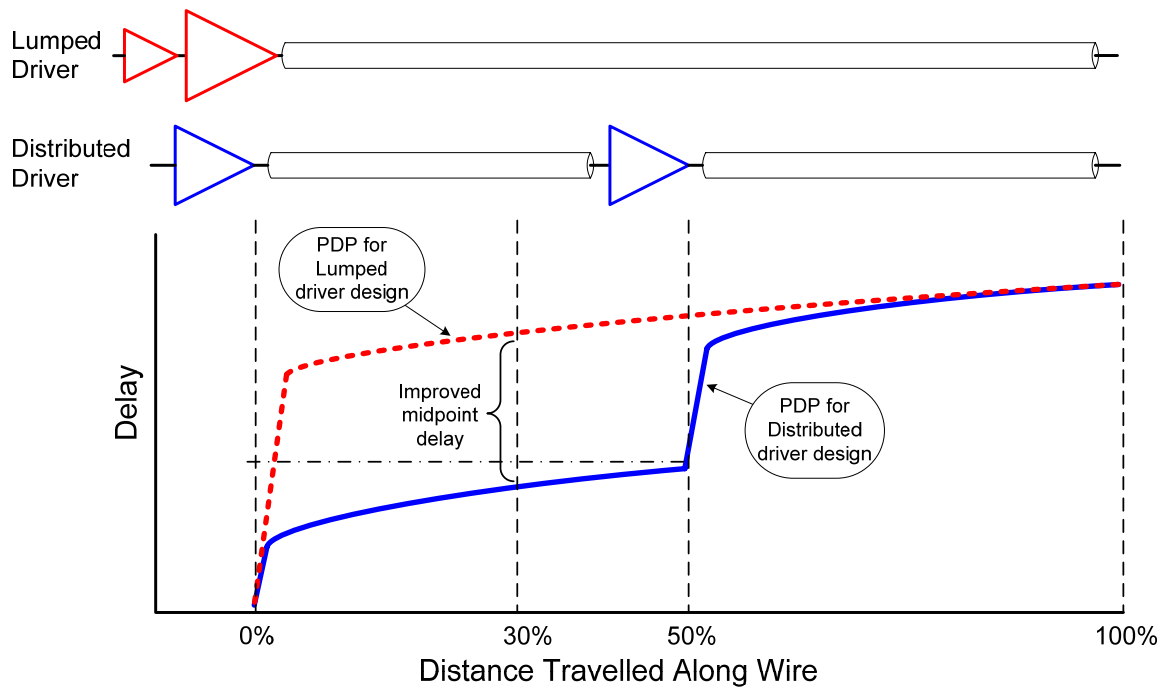


Figure 1.4 - Example of a Switch Driver Path Delay Profile

Using modified CAD tools, it is possible to model the improvements from these circuit designs. Our results demonstrate that critical path delay due to these improved circuit designs can be reduced by up to 46%.

1.1 Motivations and Objectives

In the past, it was not possible to consider interconnect optimization techniques, such as repeater insertion, on FPGA wires because a long wire was shared by multiple tri-state drivers located at different points along the interconnect. In [3], it was shown that implementing directional wires with a single, lumped driver at the beginning of the wire improves both the

delay and area efficiency of an FPGA architecture. One way to take further advantage of directional wires that was not explored in [3] is to uniformly insert additional repeaters in order to reduce the delay of the wire.

The impact on FPGA performance from the use of additional repeaters is, as yet, unclear. Distributed buffering, has potential to improve not only endpoint delay, but midpoint delays as well. This is of particular benefit to FPGA designs as signals often turn off a wire before reaching the end. However, the only way to determine how much of a wire is used is by routing circuits on the FPGA using CAD tools. A large component of this research is to assess circuit design options using a CAD model which accurately considers the impact of early turns on critical path delay.

1.2 Contributions

The contributions of this research are a circuit design methodology and evaluation of interconnect drivers for long wires in FPGAs to improve midpoint delays and end-to-end delay. Key findings of this work can be organized into three categories: circuit design, FPGA architecture and CAD modeling.

Circuit Design

- A circuit design methodology for FPGAs was produced which when given a fixed wirelength, can determine the number of buffers, size of buffers and spacing between buffers to achieve near-optimal delay.
- Using the circuit design approach, it is shown that distributed buffering is effective at reducing delay for wires, but only if the wires are of sufficient length (greater than 2mm in a 180nm technology with a minimally spaced and minimally sized wire)

FPGA Architecture

- Increasing the length of the wire between multiplexers (switch boxes) can improve the signal velocity and achieve near-ASIC interconnect speeds.
- Turns at the end of a wire (normal turns) are not critical. As fast paths and proper turn modeling are introduced, the frequency of normal turns decreases. Also, 50-87% of turns are before the end of the wire. These facts suggest that it may be possible to remove or reduce frequency of normal turns in the architecture.
- Fast paths through the switch block multiplexer were verified to improve critical path delay by up to 8% for short architectural wirelengths.

CAD Modeling

- FPGA CAD tools which are capable of improved modeling were developed and used to evaluate proposed circuit designs. The improved modeling alone resulted in a 10% improvement in critical path delay.
- Distributed buffering yields a modest delay improvement of about 3%.

1.3 Overview

This thesis is composed of 5 chapters. Chapter 2 starts with an overview of FPGA architecture and CAD, and presents concepts of interconnect design theory. Chapter 3 presents the circuit design of FPGA interconnect drivers by providing detail on the development of a driver design methodology. Chapter 4 describes the modeling improvements incorporated into FPGA CAD tools which were used to assess the circuit designs produced in the previous chapter. Finally, Chapter 5 summarizes the conclusions drawn throughout the thesis and provides suggestions for future work.

Chapter 2

Background

In this chapter, the background information for this thesis is presented. The first half presents an overview of FPGA architecture and the supporting CAD flow. Particular emphasis will be placed upon the topics related to routing.

The second half of this chapter is focused on interconnect design theory. This section presents methods used for designing and optimizing interconnects in deep-submicron integrated circuits. Important fundamentals such as device parasitics, wire models and interconnect driver design techniques will be described in detail.

2.1 FPGA Overview

An FPGA is an integrated circuit equipped with programmable logic and programmable routing resources. The reconfigurable elements allow an FPGA to be programmed after fabrication to implement virtually any digital logic function. The majority of FPGAs provide programmable logic using lookup tables (LUTs). An individual k-input lookup table, or k-LUT, is capable of implementing any k-input combinational logic function. In order to support sequential logic, flip flops are placed at the LUT output; this combination is referred to as a basic logic element (BLE). In most modern FPGAs, BLEs are grouped together in larger blocks called configurable logic blocks (CLBs) and are configured using SRAM memory elements.

Connectivity between logic blocks is achieved through the programmable routing resources. These resources are made up of metal tracks arranged in channels running vertically and horizontally across the FPGA. A channel is made up of a number of tracks, typically referred to as the **channel width**. A track is made up of wire segments of fixed length. These wire segments are placed end-to-end to span the length of the channel. The **architectural** or **logical length** of a wire is defined by the number of CLBs it spans. The physical length of a wire is equal to the logical length times the physical width of the CLB layout tile. Wires are connected to each other using **switch blocks**, and to the logic blocks using **connection blocks**.

Figure 2.1 presents a typical mesh or island-style FPGA architecture which is assumed in this work. An example of an architectural length 2 wire, also denoted as L2, is indicated. The connection blocks are labeled as C blocks and the switch blocks are labeled as S blocks.

This work is focused on the transistor-level circuits inside the switch blocks which are located at the intersection of horizontal and vertical channels. These blocks contain multiplexers which connect tracks together across the intersection in predefined patterns [4]. The switch block contains large buffers that are used to drive the long metal traces which make up the wire segment. These buffers occupy considerable area in the switch block which represents a significant proportion (roughly 1/3) of overall FPGA area [4].

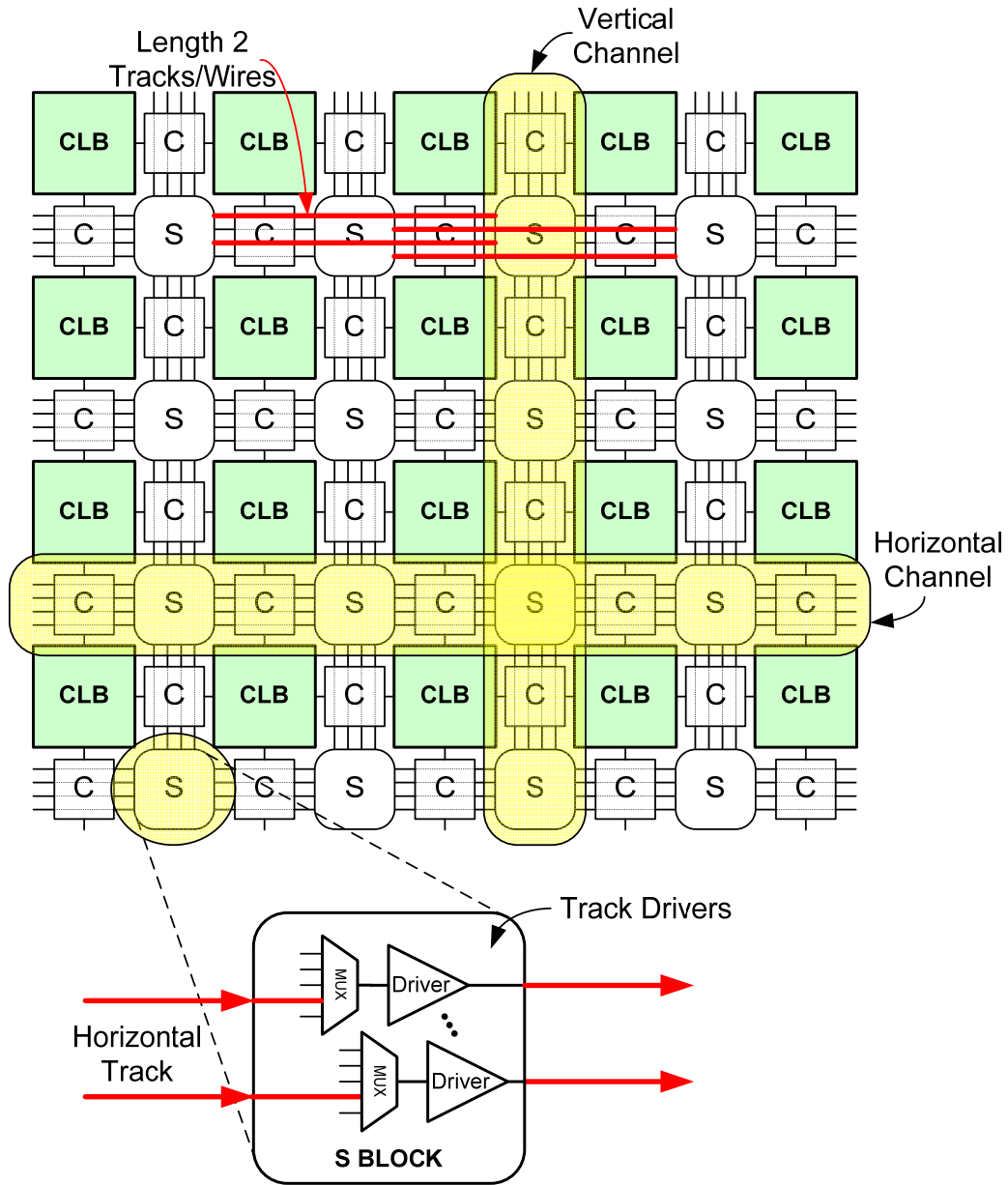


Figure 2.1 - FPGA Architecture with Switch Block Detail

2.1.1 Routing

The functional design of these interconnect circuits is governed by the routing resource architecture. The routing resource architecture defines the precise connections and turns a signal may follow in the routing resource network. There are two main routing resource architectures:

bidirectional and **unidirectional**.

Bidirectional Architecture

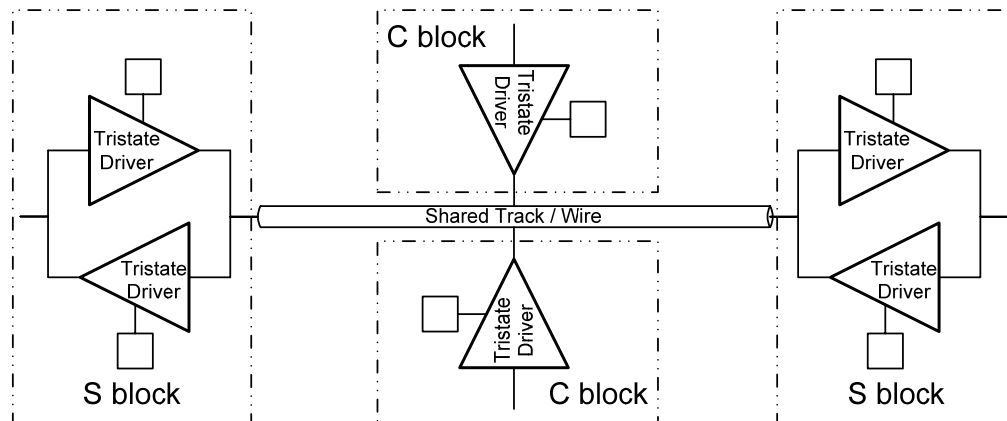


Figure 2.2 - Representation of a Wire in a Bidirectional Architecture

In a bidirectional routing network (Figure 2.2), a wire can transmit a signal in either direction. This approach provides a more flexible routing network which allows efficient use of available metal tracks. This means that the drivers of these wires must be tristate drivers so they can be disabled when not in use.

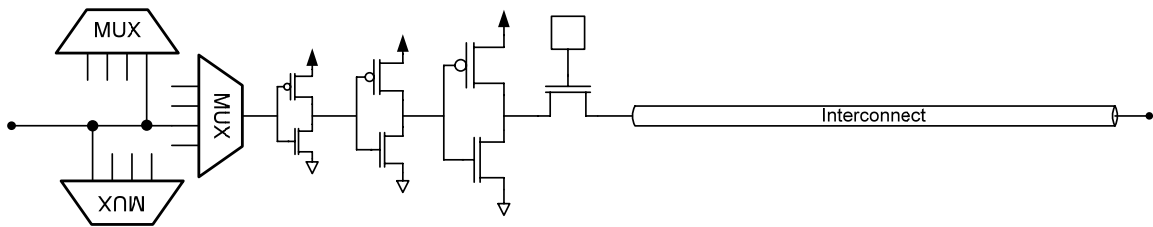


Figure 2.3 - Tristate Driver Example

A common approach to building tristate buffers in FPGAs is to use an NMOS passgate placed at the output of the regular driver (Figure 2.3) [5]. The use of this design confines the layout of the driver near to the point where the driver is connected to the wire. This means the driver design cannot be distributed along the wire. The output passgate affects both speed and area negatively by adding resistance to the output drive, producing a V_T drop in the signal swing, and adding area to the circuit layout. Furthermore, since only one of the tristate drivers connected to

each wire can be enabled after programming, this approach causes a significant waste of active area.

Unidirectional Architecture

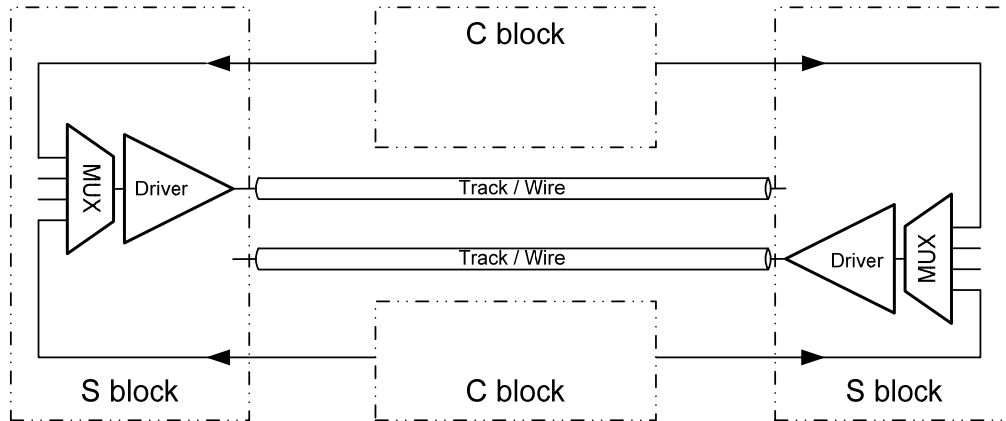


Figure 2.4 - Representation of a Wire in a Unidirectional, Single-Driver Architecture

In a unidirectional routing network (Figure 2.4), each track can only transmit data in one direction. This topology reduces the flexibility of the routing resources and suggests that the channels contain pairs of wires. Despite this restriction, work done in [3] demonstrates that this approach is more efficient in terms of area and provides improved delay over the bidirectional architecture.

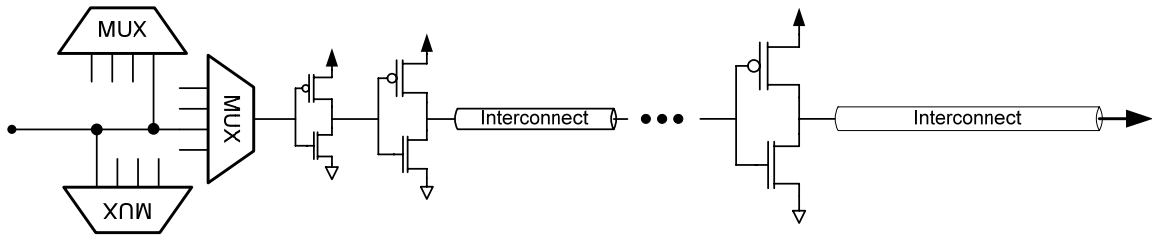


Figure 2.5 - Unidirectional Driver Example

An additional restriction known as **single-driver** wiring ensures each wire is only driven by one driver, as opposed to multiple drivers as in the bidirectional architecture (Figure 2.5). This simplifies the routing network, eliminating the ability to connect at arbitrary points in the middle

of the wire. Instead, CLB outputs can only connect to the starting-points of nearby wires. A key benefit of this is that tristate operation is no longer required. Note, however, that the buffers which make up the driver no longer have to be in the physical vicinity of the source. Instead, they can be located at various positions along the length of the wire. In this research, it is shown that the relaxation of this constraint can produce an improvement in the delay of the wire segment, particularly when the wire is of sufficient length.

2.1.2 FPGA CAD Flow

Before a user's logic circuit can be implemented on an FPGA it must under go certain processing steps known as the FPGA design flow to map the circuit onto an FPGA device. The FPGA CAD flow is comprised of 5 main steps: **synthesis**, **technology mapping**, **clustering**, **placement** and **routing**.

The first two steps in the CAD flow are synthesis and technology mapping. In these steps, the circuit is converted from a hardware description language into a network of FPGA-specific logic blocks which implement the functionality of the original circuit. After this point, logic blocks in the network are grouped together into clusters during the clustering step. This step controls the number of BLEs which are packed into a CLB and can be used as a rough method to manipulate the overall size of the circuit implementation on an FPGA. The following steps are placement and routing. Placement determines the locations for each CLB on an FPGA device. Routing uses detailed information of the FPGA routing resources to efficiently connect all the clusters together and implement the connections between logic blocks. In this work, particular emphasis is placed on developing the model of the routing resources used in the routing step.

FPGA CAD Experimental Methodology

The standard FPGA CAD experimental methodology involves running a suite of user circuits as benchmarks through the CAD flow multiple times, each time modifying the CAD step under study. In this work, the first 4 CAD flow steps are applied once on each benchmark circuit. The final step, routing, is applied multiple times. The first time the router runs, it searches for the lowest channel width which can successfully route the circuit design. Once this value is determined, it is increased by one complete set¹ of directional tracks to produce a new larger channel width. The router is run again, but this time it only routes the design once using the new calculated channel width. This gives some flexibility to the router, which tends to improve the quality (delay performance) of the routing results.

2.1.3 VPRx

For the place and route steps in the CAD flow, the academic tool VPR [6] is used. A heavily modified version, known as VPRx [3, 4], is used because it supports unidirectional wiring. Both VPR and VPRx use the same core routing and delay calculation algorithms. In the following two subsections, details on the routing resource graph and the VPRx delay model is provided.

Routing Resource Graph

VPRx models all routing paths in the FPGA using a **routing resource graph**. This data structure represents all possible connections which can occur in the FPGA routing network. In essence, the routing resource graph is a directed graph of wires, switches and pins at different locations on the FPGA. In the graph, wires and pins are represented as nodes while switches

¹ The number of tracks in a set is equal to twice the architectural length. Further details can be found in [3].

between wires are represented as directed edges connecting nodes. Figure 2.6 presents the routing resource graph for a set of logic blocks connected by two wires. In this example, the bidirectional switch on wire 1 is modeled as a pair of directed edges, where the directional switch on wire 2 has only one edge.

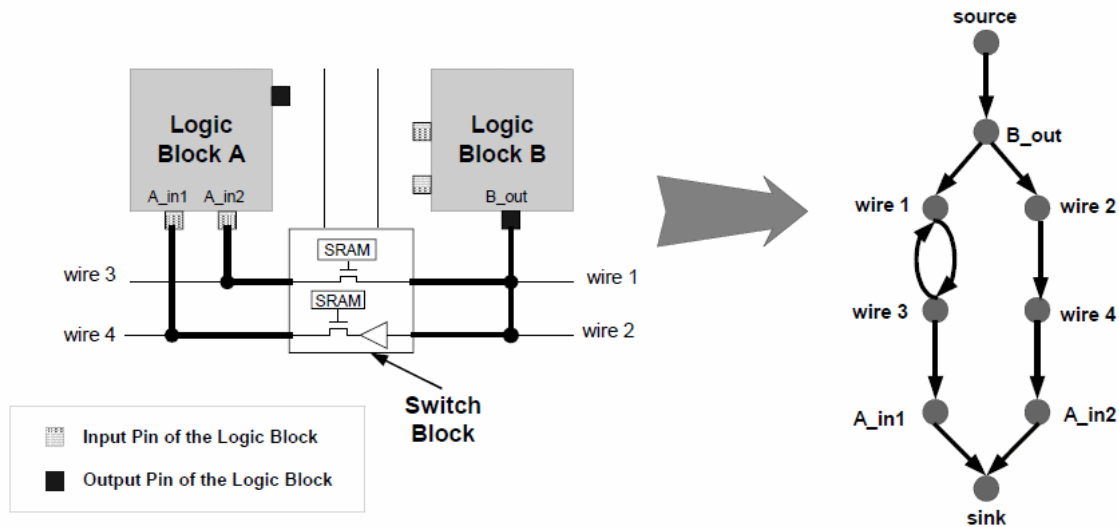


Figure 2.6 - Example of a Routing Resource Graph [6]

Delay Calculation

The routing algorithm used in VPRx is a modified version of the Pathfinder routing algorithm [6, 7]. This algorithm uses the Elmore delay calculation [8] as the primary metric to optimize the delay of routing paths. For this reason, it is important to ensure that the delay calculation is accurate. Fortunately, the routing resource graph is designed to facilitate the calculation of the delay of a signal path through the graph. Each node in the graph contains the capacitance and resistance of the wire being modeled. Similarly, each switch edge contains the delay of the switch, its input capacitance, output capacitance and its ability to drive an RC load in the form of an equivalent resistance. VPRx uses an incremental Elmore approach to calculate the delay [6].

The Elmore delay to a given node in an RC tree can be calculated by iterating over all the capacitances in the tree. For a signal path with no branches, this computation is straightforward. As the router expands along the routing resource graph, the delay of the next node is computed incrementally by adding the contribution of its parasitics to a running delay value. The equation used at each node is $t_{del} = t_{del} + R_{upstream} \times C_{node}$. The value of $R_{upstream}$ is increased as the nodes are added to the routing solution. This approach works well for calculating the approach to the end of an RC tree where there are no fanouts to add extra capacitive loading. Greater detail on Elmore delay calculations will be described shortly in Section 2.2.2

2.2 Interconnect Design Theory

Interconnect design is an increasingly important consideration for integrated circuits built on deep-submicron process technologies. In this section, background on interconnect design is presented to provide the reader with an understanding of tools and techniques used to design circuits which drive interconnects. Topics such as interconnect models and interconnect driver design techniques are provided to ensure that the reader is familiar with the concepts in the subsequent chapters.

2.2.1 Deep-Submicron Interconnect

Interconnect in deep-submicron process technologies has several important issues that affect the performance and design of high speed circuits. Problems such as signal integrity, inductive coupling, IR drop and electromigration are among the many growing challenges which face IC designers. However, the most prevalent challenge for interconnect design in deep-submicron design is signal delay.

The primary factors controlling interconnect delay are wire parasitics. The parasitics of interest, resistance (R) and capacitance (C), are physical properties of the wire. The resistance and capacitance of an interconnect act like an RC load in the signal path which causes propagation delay. The amount of resistance and capacitance is determined by the physical dimensions of the interconnect and the materials used.

Parasitic Resistance

The resistance is determined from the cross-sectional area of the interconnect. A larger area implies a lower resistance. However, as technology shrinks, wires in nanometer technologies tend to be thinner than before. The overall effect is an increase in parasitic resistance for a minimum width wire. For example, the resistance of a minimum sized wire in 90nm is roughly twice the resistance of a minimum sized wire in 180nm.

Improved materials such as copper interconnect have been introduced in order to reduce interconnect resistance. At most, this provides a one-time improvement; the resistance continues to increase as wire geometries continue to shrink. In the meantime, the most straightforward solution is to increase the wire width in order to reduce resistance. Unfortunately, this is not always possible because an increase in wiring density is needed to keep up with the increase in logic density as transistors are scaled.

Parasitic Capacitance

Parasitic capacitance of an interconnect is caused by coupling with neighboring conductors. The amount of capacitance is related to the ratio of the conductive areas facing each other to the distance separating the two conductors. Figure 2.7 shows a typical construction of a deep-submicron interconnect with the most dominant parasitic capacitances labeled. Plate capacitance

is caused by the area at the top and bottom of the wire. In earlier technologies, this value was the dominant factor. However, with the narrower wires in nanometer technologies, the coupling capacitance has grown to be the major capacitance contribution.

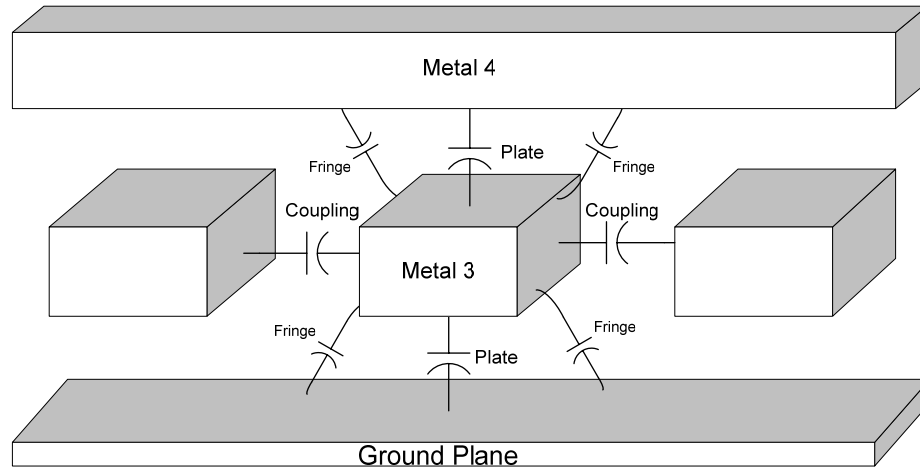


Figure 2.7 - Deep-submicron Parasitic Capacitances

As with resistance, advanced materials such as low-k dielectric insulators are being introduced to lower the parasitic capacitances. However, in cases where a designer requires even lower capacitance, the easiest solution is to increase the distances between wires. With the increase in importance of coupling capacitance, wire spacing has become an even more effective tool at reducing delay. However, the benefits of increasing the wire spacing are limited by diminishing returns and can only be used at the cost of losing interconnect density.

Inductance

One important parasitic effect which is omitted in this research is that of inductive effects. Severe inductive effects include overshoots and undershoots in signal waveforms. These signal integrity faults can potentially manifest themselves as glitches or worse, as false transitions at the end of wires. Inductance is caused by electrical loops formed on integrated circuits that generate

corresponding magnetic fields. The interaction of magnetic fields on large integrated circuits can cause inductive interference which is very difficult to locate and solve.

In the past, inductance was not a serious concern since the resistive component of a wire dominated the impedance. However, now that the frequencies are increasing, the effect of the inductive component is growing.

In this work, inductance was ignored for several reasons. The first reason is that it is very difficult to predict the effective interconnect inductance because it depends strongly on the overall construction of the integrated circuit. Unlike capacitance, which is based strongly on neighboring features, inductive effects have a much larger spatial range. The second reason is that inductive effects do not severely affect the signal or propagation delay of interconnects. As shown in [9], the worst case inductance would cause an 8% reduction in the delay.

2.2.2 Interconnect Delay Models

In order to design interconnect circuits one must be able to model the delay of a signal traveling along the wire. Interconnect delay models range from a simple lumped capacitance model to complex transmission line representations which are good at modeling high-order effects but are computationally intensive [10]. For the purposes of this research, a **distributed RC circuit model** is used with two computational models: Elmore delay and HSPICE delay.

Elmore Delay

There are many approaches used to compute the signal delay of a wire. Different methods trade off accuracy with computation speed. The fastest and most common approach is the Elmore delay [8]. This method offers high fidelity with a very fast runtime. Despite this, it is important to understand that the Elmore delay is only a first order approximation of the true

delay of an RC network. Other methods based on the Elmore delay offer improved accuracy with similar runtime [11]. However, these techniques do not have the ability to accurately model complex circuits such as pass transistors, level-restorers, or other active elements.

Elmore delay can be computed using a straightforward algorithm described in [12]. For any RC tree, the delay to node i can be computed using

$$\tau_i = \sum_k (C_k \times R_{ik})$$

C_k is the capacitance at node k and R_{ik} is the sum of all the resistances from the source to node k that are **in common** with the path from the source to node i .

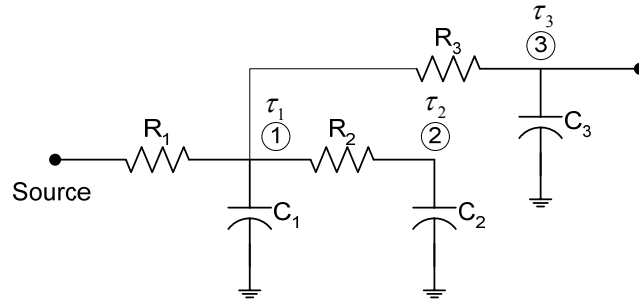


Figure 2.8 - Elmore Delay Example [12]

Figure 2.8 shows an example of a circuit from [12]. The Elmore delay to each node is calculated as follows.

$$\tau_1 = R_1 C_1 + R_1 C_2 + R_1 C_3$$

$$\tau_2 = R_1 C_1 + (R_1 + R_2) C_2 + R_1 C_3$$

$$\tau_3 = R_1 C_1 + R_1 C_2 + (R_1 + R_3) C_3$$

Using this method, the Elmore delay for a distributed RC model of a wire made up of N stages is expressed as $RC \frac{N+1}{2N}$, where R is the total resistance and C is the total capacitance. For large

values of N , this expression becomes $\frac{RC}{2}$. This can also be written as $\frac{rcl^2}{2}$, where r and c are values of resistance and capacitance per unit length and l is the length of the wire ($R = rl$, $C = cl$). This demonstrates that the delay of the wire is a quadratic function of its length.

SPICE-Level Simulation

In order to achieve high accuracy when modeling more complex driver circuits, it is necessary to use the HSPICE circuit simulator [13]. A SPICE simulator is capable of modeling a larger range of electrical effects while providing a higher level of confidence in the quantitative results in comparison to first-order approaches such as Elmore delay.

HSPICE is capable of modeling interconnects as transmission lines using a 2D field solver. However, these approaches have lengthy runtimes. Instead, wires are modeled in HSPICE using numerous distributed RC stages which produce an error of less than 3% if proper care is taken [10]. The value of resistance and capacitance can be extracted from process technology documents and/or first characterizing the RC manually using the 2D field solver in HSPICE. Further details on interconnect parasitics are provided in Appendix A.

2.2.3 Interconnect Driver Design

The delay of a wire is a quadratic function of its length. This delay can be reduced to a linear one by inserting additional buffers along the wire. Careful sizing and positioning of the buffers can further improve the delay. In this section, several approaches used to design buffers to improve interconnect delay are described.

Driver Modeling

In addition to modeling interconnect, it is important to be able to model delay of a driver circuit in order to determine the overall delay of the driver and interconnect. One common approach is to model a buffer using a resistance and capacitance. This method of using an RC time constant to represent a buffer, allows the delay of the buffer to be easily included in an Elmore delay computation.

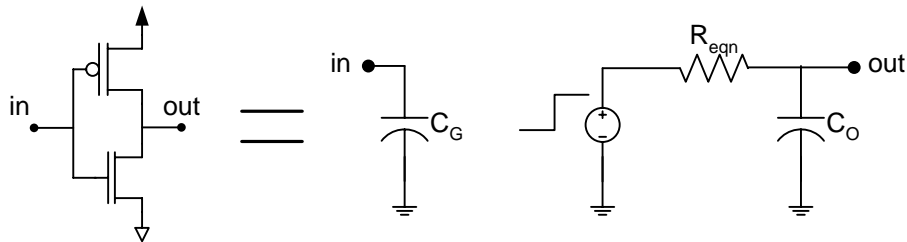


Figure 2.9 - RC Model of a Buffer

In this approach, a buffer is modeled by an effective resistance or on-resistance, a gate capacitance and an output capacitance as shown in Figure 2.9. The gate capacitance is seen as a load on the previous stage, while the output stage is modeled as a step input driven through a resistance. This buffer model can be combined with an interconnect model to form an RC network which can be evaluated using the Elmore delay.

Using this simplistic method has some drawbacks, the most notable being that it cannot accurately model the effects of reduced signal swing or smaller input slew. Also, the accuracy of this model is strongly dependant on the value of the effective resistance. Since the on-resistance of a transistor in operation is not constant, deriving an accurate effective resistance can be tricky.

Although this approach is often used, there are other methods to model the delay of a buffer. [14, 15] uses an alpha-law model [16, 17] which is based on the I-V curve of the transistor model. Although this approach yields more accurate results, it involves complicated Laplace domain

calculations. For first-order calculations, use of the RC model was continued due to its simplicity and existing implementation in FPGA CAD tools.

Repeater Insertion

A common technique for improving delay performance on long wires is to use repeater insertion [10, 12, 18-20]. Conventionally, this involves placing single stage repeaters, at uniformly spaced intervals along the wire. One derivation of this approach is based on a careful application of the Elmore delay model as shown in [12]. Results show that the number of stages required to drive a wire of length l is

$$N = \sqrt{\frac{rc l^2 / 2}{R_{eqn} (C_J + C_O)(1 + \beta)}}$$

The size of each buffer (relative to a minimum sized buffer) is

$$M = \sqrt{\frac{R_{eqn} c}{C_G (1 + \beta) r}}$$

where r and c are the resistance and capacitance per unit length, respectively. R_{eqn} is the effective resistance of the driving transistor. C_G and C_O are the input and output capacitance of a minimum size buffer. Finally, β is the ratio of PMOS to NMOS sizes used in the buffer.

This repeater insertion technique is a simple solution to the interconnect problem. Unfortunately, the approach has some drawbacks. The resulting equations compute the optimum spacing and sizing of the repeaters based solely on the process characteristics. In practice, it should also depend upon the circuit preceding the interconnect. Furthermore, the optimum repeater size is usually quite large and unrealistic. Also, start driving this large load, a chain of cascaded inverters is typically used to drive the first stage [21]. However, the delay of this initial cascaded buffer arrangement can represent a significant fraction of the overall delay [22] and this

is ignored in the equations above. Previous work in [22] has shown that it is possible to integrate this cascaded buffer with the preceding logic, however, this approach does not apply well to FPGA switches due to the limited amount of active logic in the switch block.

An alternative design approach is non-uniformly inserted repeaters [22]. This technique uses buffers of increasing size to drive progressively longer wire segments. This particular work uses a geometric relationship for the sizes and wirelengths of successive segments. Although [22] offers a change from the classical uniformly-spaced buffer insertion approach, it still relies on a constrained solution space. In one component of this research, a more general design space will be considered to determine if further improvements can be gained from a fully non-uniform design.

Other Buffer Insertion Optimization Approaches

A substantial amount of work has been done on the development of interconnect circuit design optimization techniques. Many recent studies are focused on delay minimization using closed-form expressions [1, 9, 14, 18, 23-25], but there also exists some work which uses SPICE level analysis [22, 26]. In addition to delay, other important metrics such as power and area reduction are considered in [14, 18, 24, 27].

Although much of the previous work assumes CMOS buffers, there are other more exotic circuit architectures which can reduce interconnect delay [28-30]. Regardless of what circuit topology is used, it is clear that interconnect delay is a significant problem that will continue to be studied.

2.3 Remarks

In this work, FPGA architecture and interconnect circuit optimization are combined. FPGA architecture introduces both constraints and simplifications to the general interconnect problem. The requirement of programmability places constraints on the circuit design and, in particular, it introduces complications with the delay model used in most closed-form expressions. However, the rigid and programmable structure of an FPGA allows the designer to avoid the more general problem of inserting buffers into interconnect trees. To this day, there has been an enormous amount of work done in both interconnect circuit optimization and FPGA circuit design. But to the best of our knowledge, no previous work has been attempted to combine the two concepts to produce optimized interconnects for FPGAs.

Circuit Design of Unidirectional Interconnect

This chapter develops a circuit design technique for an FPGA switch driver. The first step is to revisit the problem of FPGA switch circuit design with a particular focus on designing for single-driver routing. From this discussion, design objectives and parameters are obtained. The next step is a rapid exploration of the design space which provides insights that facilitate the development of an accurate circuit delay modeling technique. Finally, this technique is used to determine the values of the design parameters which make up a complete FPGA switch driver design.

3.1 Design of Unidirectional FPGA Switch Drivers

In this section the problem of FPGA switch driver design is defined for use in a single-driver routing architecture. In the FPGA architecture shown in Figure 3.1, the switch driver for an architectural length 2 wire is represented as a part of the switch block. Although this is logically valid, the switch driver can be physically distributed along the entire wire. But before that can be discussed, it is important to investigate the design of the components which make up an FPGA switch driver.

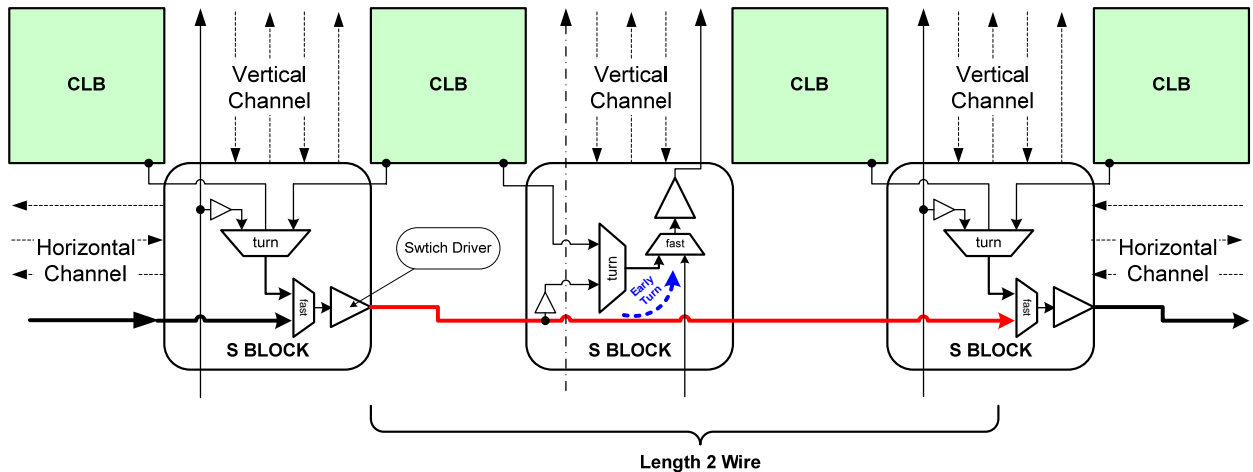


Figure 3.1 - Architectural Location of FPGA Switch Drivers

3.1.1 Components of FPGA Switch Drivers

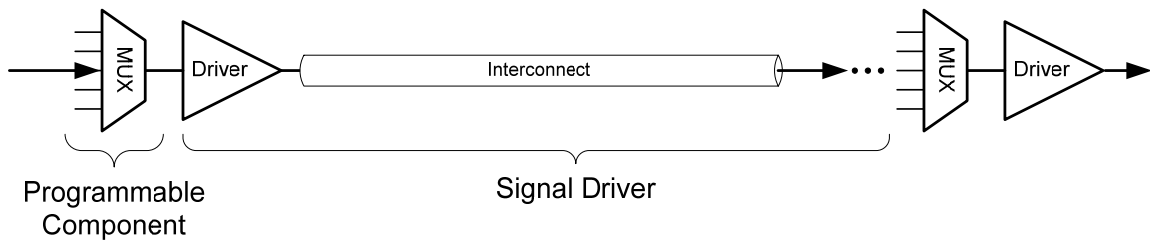


Figure 3.2 - FPGA Switch Driver Components

A block diagram of the switch driver is shown in Figure 3.2. A switch driver consists of a programmable component and a signal driving component. The programmable component allows a variety of sources to access the driver and is typically implemented using a multiplexer. The driver circuit transmits the signal down the wire to surrounding logic blocks and I/O pads. What follows is a description of some of the possible implementations of the multiplexer and the driver.

Multiplexer Design

The use of a multiplexer allows the device to select the signal to be driven from a variety of inputs of the switch block. Since multiplexers are prevalent throughout an FPGA, it is important

to appreciate that their circuits contribute to a large portion of the chip area. For this reason, area-efficient multiplexer designs are preferred in FPGA devices.

A multiplexer can be built using either active CMOS gates or passgates. The former uses CMOS logic gates to achieve full rail-to-rail signals. Although this approach is convenient, it is inefficient in terms of area and delay, especially as the fanin of the multiplexer increases. The latter design uses pass transistors or transmission gates to select the desired signal. This approach is commonly used because it provides good area and delay performance. Multiplexers using NMOS pass transistors have the smallest area, but suffer from reduced signal swing which causes downstream gates to operate slowly and leak at steady state. One solution to this problem is to use a level-restoring circuit, shown in Figure 3.3, integrated with the driver circuit. Unfortunately, this technique introduces non linear models into the circuit, making it difficult to use an analytical approach to compare with other circuits.

One compromise between area and delay is to use the CMOS transmission gate. This design is fast and provides a full swing signal to drive downstream gates. However, CMOS transmission gates require more than twice the area of an NMOS passgate due to the use of PMOS transistors.

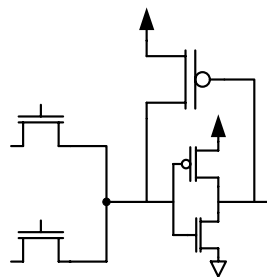


Figure 3.3 - Level-Restoring Circuit with 2:1 NMOS Pass Transistor Multiplexer

Once the passgate implementation is determined, the next step is to consider the multiplexer architecture. For multiplexers built using pass transistors, there are two common architectural choices.

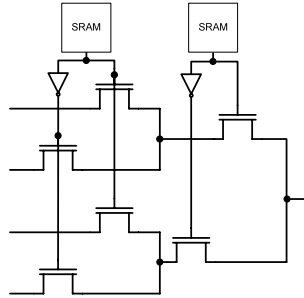


Figure 3.4 - Binary Tree Multiplexer

The first option is the fully-encoded binary tree structure. This design fully encodes the control signal input to make efficient use of the configuration RAM and reduce the area overhead. The signals are selected using a binary tree structure as shown in Figure 3.4. For an N -input multiplexer of this architecture, the number of configuration RAM bits used is equivalent to the number of levels in the tree, hence $\lceil \log_2 N \rceil$. The signal delay in this design is proportional to the number of levels squared, or $\lceil \log_2 N \rceil^2$.

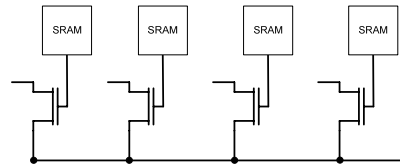


Figure 3.5 - Flat Multiplexer

An alternative approach is the flat, decoded multiplexer as in Figure 3.5. This architecture uses one configuration RAM bit and one passgate for each signal input. This design requires the largest SRAM area overhead, but limits the delay for any signal to that of one passgate plus the loading effect of additional junction capacitance from neighboring transistors. The number of transistors used in this design is equal to the number of signal inputs.

Comparison of Multiplexer Architectures (N = fanin of MUX)			
Parameter	Fully Encoded Multiplexer	Flat Decoded Multiplexer	2-Level Multiplexer
Delay Levels	$\lceil \log_2 N \rceil$	1	2
Loading (junction capacitance)	$\lceil \log_2 N \rceil$ (distributed)	N-1 (lumped)	$\sqrt{N} \times 2$
Configuration RAM	$\log_2 N$	N	$2\sqrt{N}$
Passgates Required	$2N - 2$	N	$(N + \sqrt{N})$

Table 3.1 - Comparison of Multiplexer Architectures

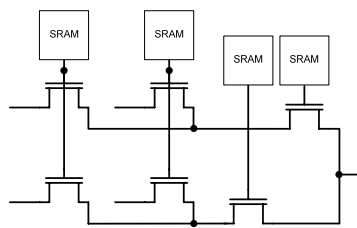


Figure 3.6 - 2-Level Multiplexer

A combination of the aforementioned designs can be seen in the Stratix II architectural paper [31]. This approach uses a combination of encoded and decoded designs to produce a tree with multiple branches. This design attempts to reduce the delay by limiting the number of levels a signal must pass to two levels. For a 16:1 multiplexer, this design requires about half the amount of configuration RAM cells as the flat decoded multiplexer but 25% more passgates. Since configuration RAM is usually quite large in comparison to a passgate, this design can still result in area savings. Table 3.1 compares the characteristics of the three multiplexer designs.

For this work, the two-level passgate architecture was selected because it reduced delay without incurring a large increase in area. Also, the delay of this design remains fairly linear with increasing fanin due to its constant depth. This linearity simplifies modeling of the multiplexer delay during FPGA architectural explorations where the fanin depends on the channel width.

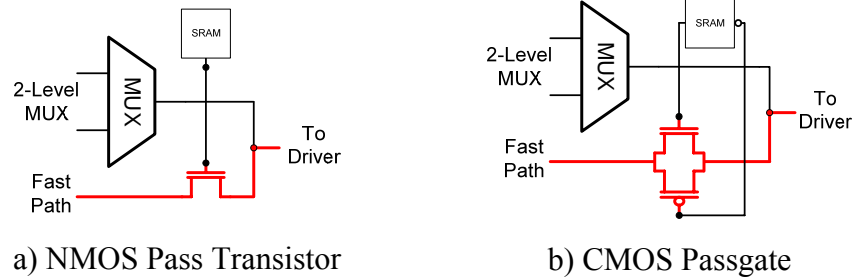


Figure 3.7 - Multiplexer Fast Path

Multiplexer “Fast Path”

Regardless of which multiplexer design is used, it will always be a significant source of delay in the circuit. To further exacerbate this problem, directional routing architectures require multiplexers which are larger than those found in bi-directional routing architectures. The multiplexers used have large fan-in (e.g., 20:1, 40:1) resulting in large propagation delays. In order to avoid this, an isolated path called the “**fast path**” was created. Also referred to as the “fast input” in [31], this path allows one signal to bypass the majority of the multiplexer inputs and arrive at the driver input after going through one passgate as shown in Figure 3.7. When the fast path is taken, the remainder of the multiplexer design can be ignored since it is shielded by the disabled passgates. Since it is expected that this is the common case for high-speed signals that must cross the chip, the buffer design will be optimized for this fast path by reducing the multiplexer structure to a simple 2:1 multiplexer.

Multiplexer Design Results

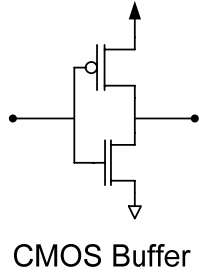
The delay of three multiplexer implementations (NMOS pass transistor, NMOS pass transistor with level restorer, and the CMOS transmission gate) were compared using HSPICE. Circuit simulations of the fast path were performed using a 2-1 multiplexer driving a single buffer of various sizes. Sweeping the size of the passgate transistors yielded minimal

improvements for the end-to-end delay of the multiplexer-buffer-wire path, as a result minimum-sized transistors were used in order to conserve area.

Simulation results also confirmed that the CMOS transmission gate yielded the fastest delay. In addition, this option offers full signal swing, eliminating the need for any level restoring circuitry. Because of this, a CMOS transmission gate was used with the 2-level multiplexer and fast path. The final design is shown in Figure 3.7b.

Driver Designs

The component following the multiplexer is the driver circuit. The purpose of this circuit is to “drive” the multiplexed signal down the wire. Although there are many ways to implement a signal driver, our focus is restricted to designs using CMOS **inverters**, also referred to as a **buffer** or a **repeater**. The standard CMOS inverter is built with one NMOS and one PMOS transistor (Figure 3.8). The PMOS to NMOS transistor size ratio is dependant on the input signal and therefore related to the multiplexer design. For example, a buffer following an NMOS pass transistor multiplexer will require a much larger NMOS transistor to sense the weak ‘1’ input signal [4]. In this work, a 2:1 PMOS to NMOS sizing ratio is assumed for simplicity and because weak input signals do not occur with the CMOS transmission gate.



CMOS Buffer

Figure 3.8 - Ubiquitous CMOS

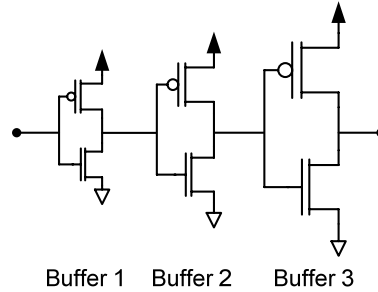


Figure 3.9 - Driver Example

Buffer

A **driver** is made up of one or more buffers connected in series as shown in Figure 3.9. The buffers in this design are progressively larger in size. Since physical distance between the buffers is very small, it is referred to as a “**lumped**” driver design. An alternative approach is to space the buffers apart along the length of wire they must drive. This is referred to as a “**distributed**” driver design. The key unknowns regarding driver design are the transistor sizes of the buffers, the number of buffers, and the distance between buffers. Within a driver, the combination of a buffer and a wire is referred to as a “**stage**”. Often, the length of a wire following a buffer is used to indicate the spacing between buffers, and it is also referred to as the wirelength of the stage.

3.1.2 Switch Driver Design Goals

As mentioned in the background, the elimination of tristates allows single-driver interconnect architectures to use distributed buffering and subsequently, to consider midpoint delay as well as endpoint delay.

Distributed Buffering

Conventional VLSI practices suggest that distributed drivers are the best design for minimizing end-to-end delay of an interconnect [10]. However, in order for a distributed driver to become beneficial, the wire must be long enough and resistive enough to take advantage of

the additional repeaters. One of the goals of this work is to determine how long must a programmable interconnect be, before distributed buffering can benefit an FPGA.

Modeling Early Turns

Another benefit of distributed drivers for FPGAs is the improvement of early turn delays. As mentioned in the introduction and shown in the following Chapter, place and route results demonstrate that early turns occur very often in directional FPGAs with long wires. Intuitively, distributed driver designs can offer improved early turn delay since all the inverters are not lumped at the front of the wire. However, in order to guarantee this, an ideal design would have to improve delay to all points along the wire, in addition to the endpoint delay.

The delay to all points along a wire is shown in a “**path delay profile**” (PDP). This metric will be used as a qualitative tool to determine if a circuit design can offer improved early turn delay. For example, two PDPs are shown in Figure 3.10. Both circuits have similar delay to the end of the wire. However, the signal of circuit A arrives before the signal of circuit B for the majority of locations along the length of the interconnect, particularly at points between 0 and 0.75mm. This suggests that circuit A would yield better performance in an FPGA architecture if there are turns before 0.75mm.

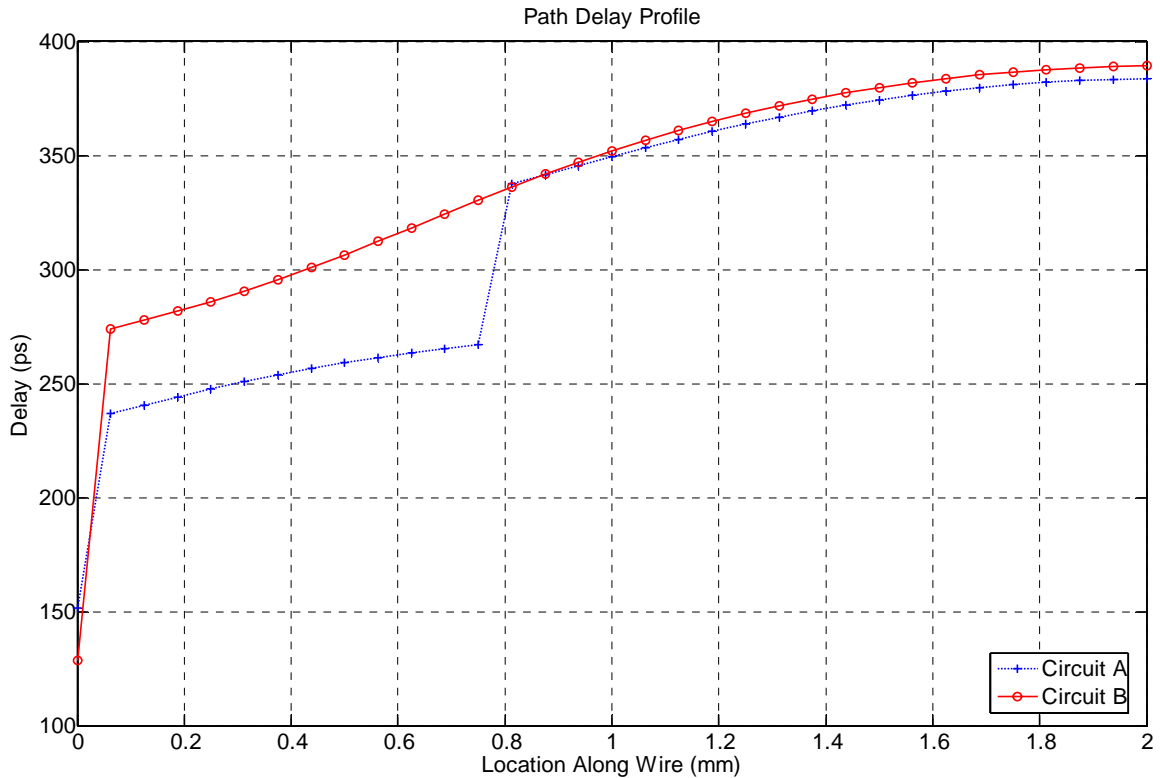


Figure 3.10 - Path Delay Profile

In summary, the circuit design goals of the switch driver are to take advantage of unidirectional interconnect architectures to optimize for midpoint and end-to-end delay. In particular, this work will attempt to determine if distributed driver designs are beneficial for use in FPGA interconnect. It will identify what wirelengths benefit in end-to-end delay from a distributed driver approach and using the PDP, identify designs that improve early turn delays.

3.1.3 Switch Driver Design Parameters

This section summarizes the key design parameters of the circuit design problem. The design parameters are listed in Table 3.2. Figure 3.11 presents a block diagram indicating the various design parameters.

Driver Design Parameters		
Parameter	Symbol	Description
Total Wirelength	L	Length of interconnect. Architecturally, this is the distance between multiplexers (typically in mm)
Number of Driver Stages	N	Number of buffer-wire stages which make up the total interconnect, including the programmable component as the first stage
Buffer Sizing	B_i	Size of the buffer i , normalized to a minimum sized buffer
Buffer Spacing	L_i	Length of wire following buffer i (typically in mm or in % of L)

Table 3.2 - Driver Design Parameters

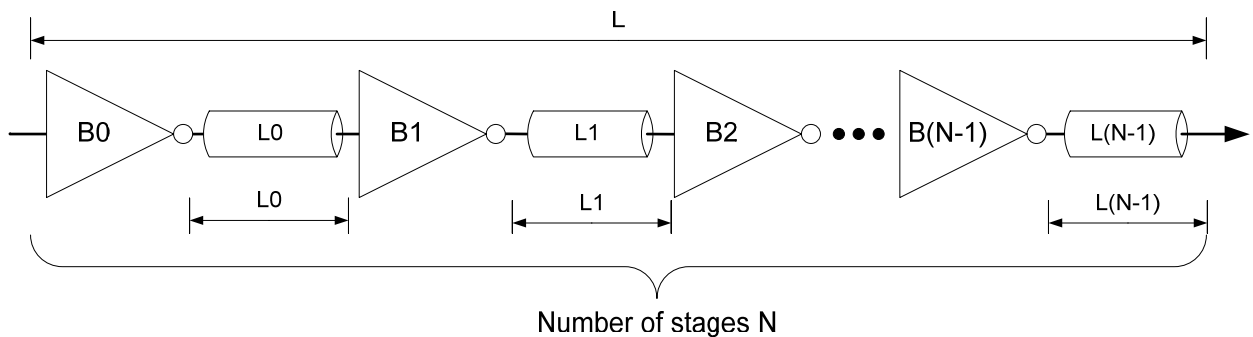


Figure 3.11 - Block Diagram Identifying Driver Design Parameters

Interconnect Characteristics

Thus far, the geometry issues of interconnect have not been discussed. In most FPGAs, long interconnects are manufactured on mid-level layers (e.g., metal 3 in a 6 metal process). Therefore this work assumes that metal layer 3 is used to build the wires. Two combinations of wire widths and spacings were considered for 180nm and 90nm process technologies: 1x minimum width/1x minimum spacing (denoted as 1x1x) and 2x minimum width/2x minimum spacing (2x2x). Results are presented for designs built using 180nm 1x1x and 90nm 2x2x interconnects. These combinations are chosen because they represent the range of delays achievable, as 180nm 1x1x is the slowest and 90nm 2x2x is the fastest.

3.2 Rapid Design Space Exploration

By developing a model based on the design parameters from the previous section, the general problem of sizing and placing buffers on a wire can be explored. A quick-to-compute Elmore model is created and used to rapidly explore the design space of a system with three buffers. The results from this exploration suggest that distributed buffering can improve results at certain wirelengths. Furthermore, the results indicate that the design space is fairly insensitive to small changes in buffer size and wirelength, which allows for some flexibility in choosing an optimal design.

3.2.1 Analytical Delay Model

In this study, the design parameters will be exhaustively swept using a simple Elmore delay model. The difference between this exploration and previous work [14, 22, 23, 32] is that the search here does not impose relationships between subsequent inverters. Most approaches constraint the size of successive inverters to be equal or related to one another based on a geometric series. Unfortunately, by not introducing any constraints, the design space becomes very large and unwieldy. To reduce the dimensionality of the design space, the number of stages is restricted to 3 stages as in previous FPGA switch driver designs [3, 33]. Another constraint is that the size of the first inverter is fixed to minimum. This is done because the Elmore model approach does not take into account the input capacitance of the first gate. Also, using a smaller sized buffer will minimize the impact of loading on the preceding circuitry.

The delay model uses standard VLSI techniques [12] presented in the background. Figure 3.12 shows a buffer of size s driving a wireload of length l and a downstream buffer of size s' .

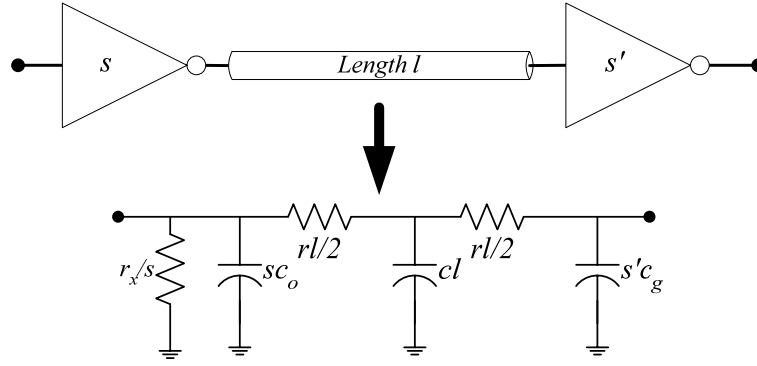


Figure 3.12 - Elmore Model of Buffer & Wire Delay

The Elmore delay equation for the wire has the time constant:

$$\tau = \frac{r_x}{s} (c_o s + c l + c_g s') + \frac{r c l^2}{2} + (r l \times c_g s') \quad (1)$$

where r and c are the resistance and capacitance per unit length of a wire, c_o and c_g are the output and input capacitance of a minimum sized buffer and r_x is the equivalent resistance of a minimum-sized transistor. Although the value of r_x depends on the type of transistor used, most approaches do not distinguish the value. In this work, the Elmore-delay computation code distinguishes between a rising and falling scenario and takes the average of the delays. Table 3.3 lists typical values of the parasitics in a 180nm process technology.

Parasitics in 180nm Process Technology		
Parameter	Description	Typical Values [12]
c_o	Output capacitance	1 fF/ μ m
c_g	Gate Capacitance	2 fF/ μ m
c	Wire capacitance per unit length (min width & spacing)	0.1-0.25 fF/ μ m
r	Wire resistance per unit length (min width)	125-300 m Ω / μ m
r_p	Equivalent resistance of a PMOS transistor	30 k Ω / μ m
r_n	Equivalent resistance of an NMOS transistor	12.5 k Ω / μ m

Table 3.3 - Typical Parasitics in Deep Submicron Process Technology

The total delay through the chain of inverters is calculated by summing up the delay through all three stages. Typical Elmore delay modeling applies a $\ln(2) \cong 0.69$ factor to τ to calculate 50% propagation delay, however, it was found that a 1.0x factor was more accurate due to the

non-ideal (ramp) inputs used to drive the circuits [12]. With this model, the delay of a three-stage driver for any given combination of inverter sizes, inverter spacings and total wirelength can be calculated.

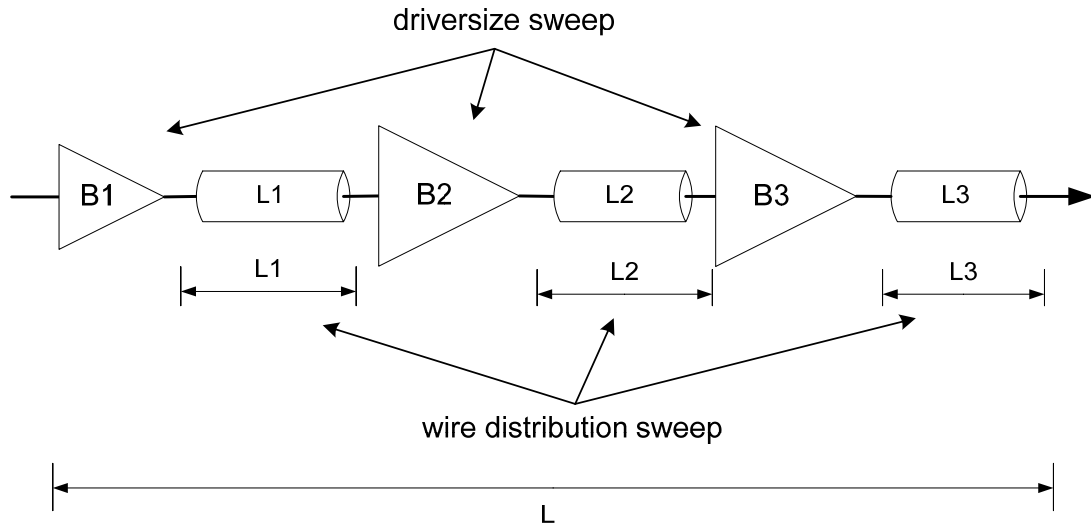


Figure 3.13 - Parameters Being Swept

3.2.2 Design Space Sweeps

Using the general delay model, a set of nested sweeps was used to search the design space for a variety of wirelengths. The inner sweep is a driver-size sweep and the outer sweep is a wire-distribution sweep. The driver-size sweep calculates the delay for all possible combinations of a set of predetermined buffer sizes. From the sweep, the buffer sizes which produce the smallest delay and the smallest area-delay product can be determined. Similarly, the wire-distribution sweep generates the best buffer spacing configuration for each wirelength setting. The pseudo-code for this exhaustive search is shown in Figure 3.14. This pseudo-code was implemented in Matlab. The `calculate_delay_of_()` function computes the Elmore delay as described in the previous section (3.2.1).

```

driver_sizes_weep(wirelengths_array [w1 w2 w3]) {
  /* mindelaymetric can represent delay or areadelay */
  mindelaymetric = largenumber;
  for all b1sizes [b1] {
    for all b2sizes [b2] {
      for all b3sizes [b3] {
        /*build circuit with buffersizes [b1 b2 b3]
          and wiredistributions [w1 w2 w3] */
        circuit = build_circuit( [b1, b2, b3], [w1 w2 w3] );
        delaymetric (b1,b2,b3) = calculate_delay_of( circuit );
        /* Grab the best design */
        if (delaymetric (b1,b2,b3) < mindelaymetric) {
          mindelaymetric = delaymetric (b1,b2,b3);
          bestcircuit = circuit;
        }
      }
    }
  }
  return bestcircuit;
}

wire_distribution_sweep(wirelength) {
  /* mindelaymetric can represent delay or areadelay */
  mindelaymetric = largenumber;
  for all L1lengths [L1] {
    for all L2lengths [L2] {
      L3 = wirelength - L1 - L2;
      circuit = driver_sizesweep([L1 L2 L3])
      delaymetric (L1,L2) = calculate_delay_of( circuit );
      /* Grab the best design */
      if (delaymetric (L1,L2) < mindelaymetric)
        delaymetric = delay (L1,L2);
        bestcircuit = circuit;
    }
  }
  return bestcircuit;
}

```

Figure 3.14 - Design Space Sweep Pseudo Code

3.2.3 Results

Table 3.4 presents the best wire distribution, buffer sizes and resulting delays for wirelengths ranging from 1mm to 16mm in a 180nm process using wires with 1x minimum spacing and 1x minimum width. The best wire distribution is shown as three values which represent the length of wire following buffer 1, 2 and 3, respectively, these values are normalized to the total wirelength and sum to 1.0. Similarly, best buffer sizes are listed as the size of buffer 1, 2 and 3, respectively. The delay for the best design is shown in column 4 and the delay for the corresponding lumped design is shown in column 5. The final column indicates the performance

difference between the two designs. For example, the 2.5mm design is made up of 1x, 7x and 38x buffers followed by wirelengths which make up 0%, 15% and 85% of the total wirelength, respectively. This design has a delay of 379.8ps. In comparison, the best 3 stage lumped design for a 2.5mm design would have a delay of 382.3ps, approximately 1% slower.

With only three stages in this design, it is unlikely that any wires as longer than 4-5mm will even be considered. Data for wirelengths up to the 16mm long are shown because that is when the design becomes fully uniform. The most interesting region is around 2-3mm, where the best designs begin to shift from lumped designs to distributed designs.

Delay-Driven Results									
<i>Wirelengths</i>	<i>Best wire distribution</i>			<i>Best Buffer Sizes</i>		<i>Delay</i>	<i>Best delay for Lumped Design</i>	<i>Performance Difference</i>	
1mm	0.00	0.00	1.00	1	4	21	185.4ps	185.4ps	0%
2mm	0.00	0.00	1.00	1	5	36	305.8ps	305.8ps	0%
2.5mm	0.00	0.15	0.85	1	7	38	379.8ps	382.3ps	1%
2.8mm	0.00	0.25	0.75	1	8	37	423.5ps	434.3ps	2%
3mm	0.00	0.25	0.75	1	8	39	453.2ps	471.5ps	4%
4mm	0.00	0.35	0.65	1	10	39	613.5ps	687.4ps	11%
5mm	0.00	0.40	0.60	1	12	39	796.6ps	956.6ps	17%
6mm	0.00	0.40	0.60	1	12	39	1004ps	1281ps	22%
7mm	0.00	0.45	0.55	1	14	36	1237ps	1662ps	26%
8mm	0.00	0.45	0.55	1	14	36	1496ps	2098ps	29%
9mm	0.00	0.45	0.55	1	16	39	1781ps	2590ps	31%
10mm	0.00	0.45	0.55	1	16	39	2094ps	3138ps	33%
12mm	0.00	0.45	0.55	1	18	39	2802ps	4401ps	36%
16mm	0.00	0.50	0.50	1	22	36	4539ps	7596ps	40%

Table 3.4 - Delay-Driven Results

One concern with the delay driven analysis based on Elmore delay calculations is that buffer sizes tend to increase rapidly. A commonly used method to reducing the area usage is to use the area-delay product metric to improve the tradeoff between area and delay. The results in Table 3.5 are determined using the area-delay metric.

AreaDelay-Driven Results					
<i>Wirelengths</i>	<i>Best wire distribution</i>	<i>Best Areadelay Buffer Sizes</i>	<i>AreaDelay driven Delay</i>	<i>Best AreaDelay driven delay For Lumped Design</i>	<i>Performance Difference</i>
1mm	0.00 0.00 1.00	1 3 13	194.7ps	194.7ps	0%
2mm	0.00 0.00 1.00	1 3 16	334.6ps	334.6ps	0%
2.5mm	0.00 0.00 1.00	1 3 17	422.1ps	422.1ps	0%
2.8mm	0.00 0.35 0.55	1 6 15	469.2ps	484.1ps	3%
3mm	0.00 0.40 0.60	1 7 14	502.8ps	528.2ps	5%
4mm	0.00 0.45 0.55	1 8 14	677.3ps	803.4ps	16%
5mm	0.00 0.45 0.55	1 8 14	884.3ps	1117ps	21%
6mm	0.00 0.50 0.50	1 9 13	1117ps	1507ps	26%
7mm	0.00 0.50 0.50	1 9 12	1397ps	1936ps	28%
8mm	0.00 0.50 0.50	1 9 12	1688ps	2454ps	31%
9mm	0.00 0.50 0.50	1 9 12	2007ps	3085ps	35%
10mm	0.00 0.50 0.50	1 9 12	2354ps	3747ps	37%
12mm	0.00 0.50 0.50	1 8 11	3214ps	5140ps	37%
16mm	0.00 0.50 0.50	1 8 10	5200ps	8843ps	41%

Table 3.5 - AreaDelay-Driven Results

Once again, results indicate that as the wirelengths increase beyond 2-3mm, a distributed arrangement begins to demonstrate improvement over a lumped design. It is interesting to note that as in the delay driven results, slightly non-uniform distributed solutions are good for certain wirelengths. However, the range of wirelengths with non-uniform solutions shrinks from 2.5mm-12mm to 2.8-5mm through the use of the area-delay metric. These wire distribution results suggest two conclusions:

- 1) The first two driver stages should be lumped together ($L = 0$)
- 2) For wires longer than 2mm (in 180nm with minimum wire width and spacing), the design should be distributed, and wire segment lengths in subsequent stages should be approximately equal

The first result is strongly tied to the initial assumption that the first buffer should be of minimum size. However, as mentioned earlier, this is a realistic assumption since the first stage is a sense stage. This reinforces the notion that ASIC interconnect designers should consider

tapering logic which proceeds large interconnect drivers [22]. The second conclusion verifies that common design approaches which assume equally spaced repeaters are valid. To further understand these conclusions, it is beneficial to examine figures 3.15 and 3.16.

Figure 3.15 shows the effects of wire distribution on delay. In the figure, the wirelengths of the stages are represented by L1, L2 and L3, respectively. Since $L1 + L2 + L3 = 1.0$, it is sufficient to plot only L1 and L2, as the third length can be determined from their sum (i.e., $L3 = 1 - L1 - L2$). The plot shows that the effect of L1 has a large influence on the delay, where as the effect of L2 has little effect on the delay. This confirms that to minimize delay, L1 should be fixed at 0 and L2 can be selected for minimum overall delay.

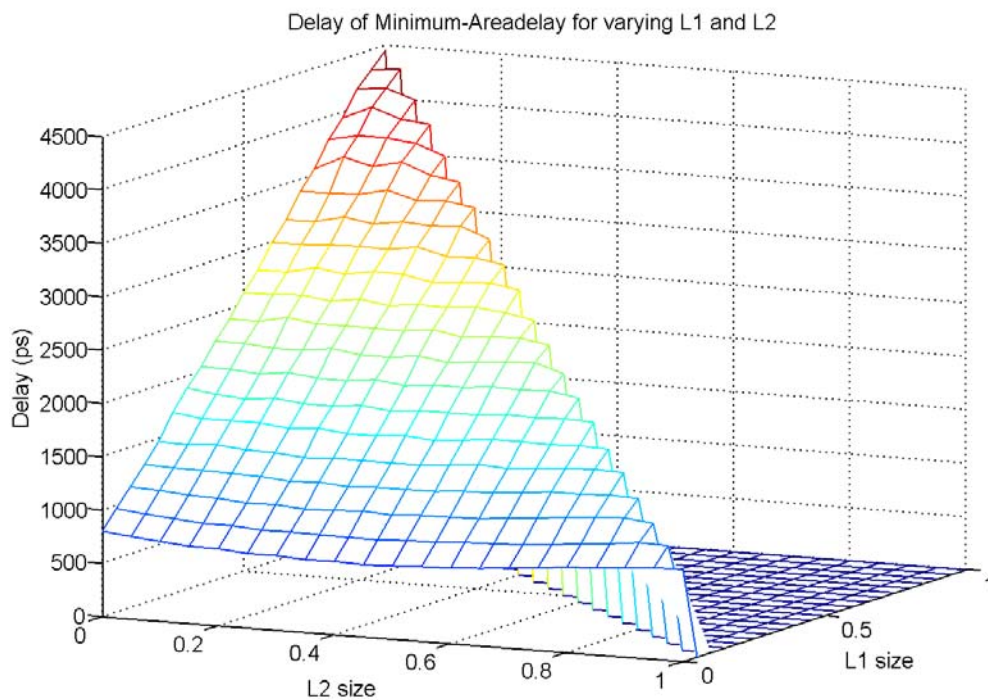
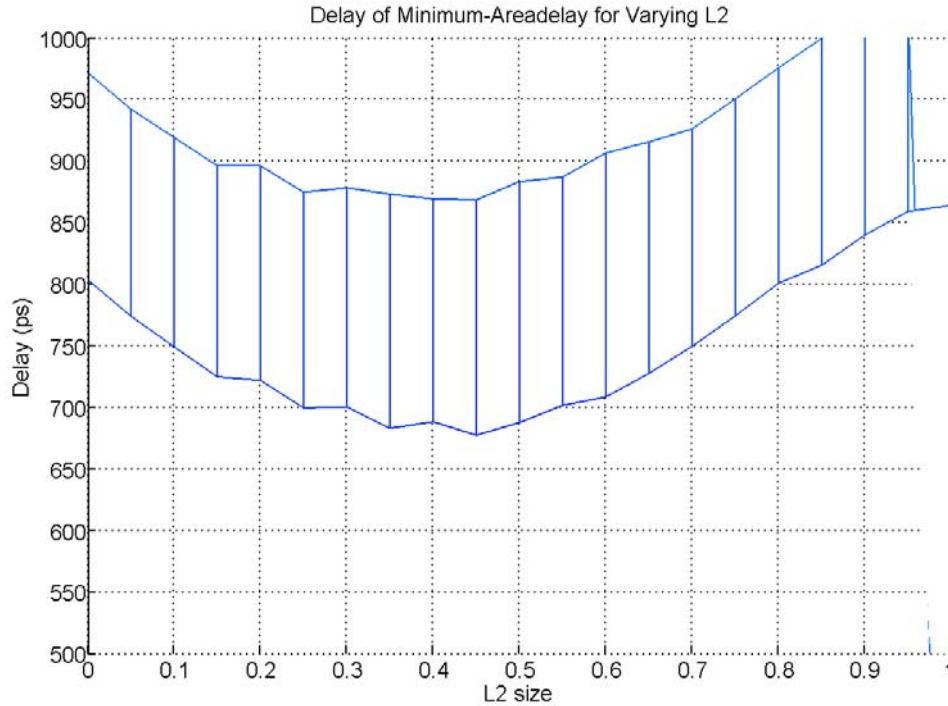


Figure 3.15 - Areadelay-Driven Minimum-Delay Wire Distribution Plot for a 4mm Wire



**Figure 3.16 - Areadelay-Driven Minimum-Delay Wire Distribution Plot for a 4mm Wire
Zoomed in on L2 Axis**

Figure 3.16 presents the same plot as Figure 3.15, but it is enlarged from the perspective of the L2 axis. It shows that L2 can easily be stretched from 35% (683ps) to 50% (688ps) with only a 0.7% change in minimum delay. This suggests that uniformly dividing the wire between stages 2 and 3 can achieve near-minimum delay, supporting the notion that a uniform spacing of the subsequent stages is an acceptable choice for design. Moreover, it reassures the designer that the design is robust to minor shifts in the placement of the latter driver stages, provided they are already within the vicinity of the optimal configuration.

3.2.4 Summary

Results from the design space exploration provided an understanding of where distributed buffering becomes applicable and how much tolerance the design has for deviations. Although

certain distributed designs are non-uniform, the tolerance in delay to minor shifts in the design suggests that uniform spacing still provides an adequate solution.

One apparent limitation to this analysis is that the number of stages is fixed. However it is unlikely that introducing additional stages will change these conclusions. Placing an additional stage in the design would continue to result in a uniform arrangement with shorter wire segments. Furthermore, adding unnecessary stages will only result in increasing the overall delay of the design.

Using results from this study it is possible to make good assumptions which will help to reduce the design space of the general problem and develop a more detailed circuit model with fewer unknowns. Conclusions based on the wire distribution results suggest that the circuit model can be divided into two sections; a lumped front end and a distributed back end. The lumped front end will contain two buffers, starting with the minimum sized buffer to sense the input. Since this work is for programmable interconnects the front end will also need to include a multiplexer. For this reason, this section is referred to as the **multiplexer stage**. The distributed back end, referred to as the **distributed stages**, will be made up of *multiple* driver stages consisting of *buffer-wire segments* of equal length. This is an important constraint because the ability to assume that the back end of the circuit is uniformly distributed greatly reduces the design complexity. Now that the circuit sections are defined, our attention turns to the development of a more detailed delay model.

3.3 Accurate Switch Driver Circuit Delay Modeling

The rapid design space exploration used a simplified model to calculate the delay of the circuit. Unfortunately, this model lacks the ability to accurately predict the effects of passive devices such as passgates, model the initial input buffer loading, or properly account for reduced

drive strength due to a weak input signal. These inaccuracies make it difficult to incorporate and compare the delay of different multiplexer designs. This section addresses the problem of how to determine the overall delay of the switch driver circuit accurately and rapidly, by taking advantage of the design space constraints determined in the previous section.

3.3.1 Delay Modeling through Circuit Characterization

Figure 3.17 provides an example of a circuit which contains elements that are difficult to evaluate and compare using an Elmore model. However, these complex elements in multiplexer designs such as pass transistors and level restorers can be accurately modeled and compared using HSPICE. The major drawback to HSPICE is that its runtime is long and care must be taken in order to avoid a lengthy optimization process.

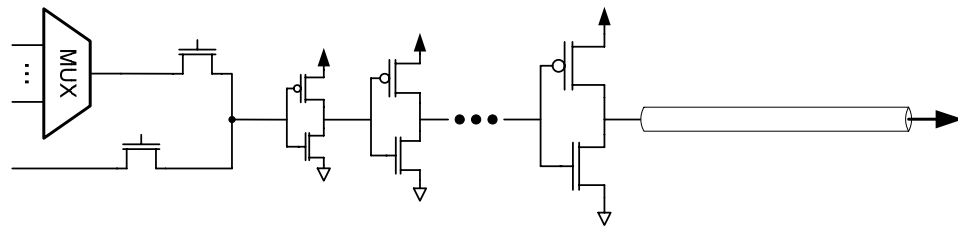


Figure 3.17 - Programmable Driver Example

In this section, the delays of different circuit stages are *characterized* using HSPICE. This data is then loaded into an array which acts as a lookup table for delay calculations. The circuit stages characterized represent either a *multiplexer stage* or a *distributed stage* of the overall circuit design. There are several different types of multiplexer stages that are defined by the type of multiplexer used and the driver. These designs are summarized in Table 3.6.

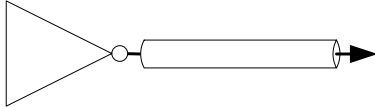
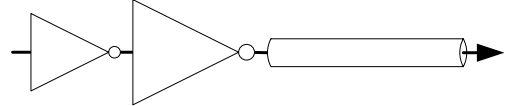
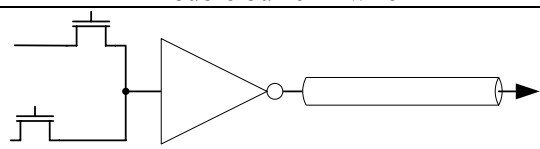
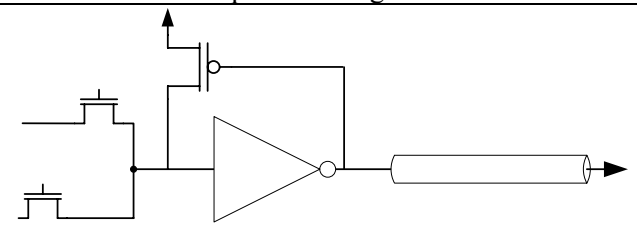
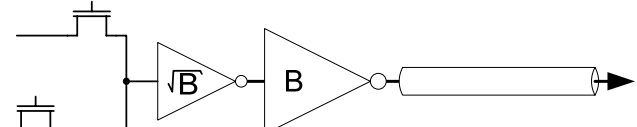
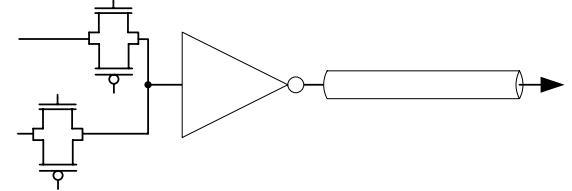
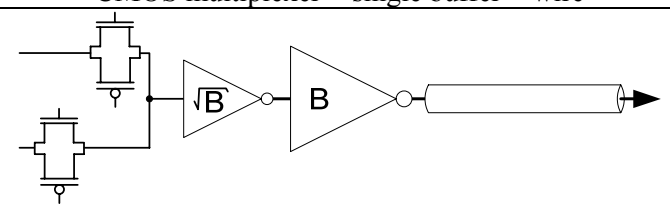
Various Characterized Stage Types		
Characterized Sections	Description	
Distributed Stage Types	nomux	 <p>Single buffer + wire</p>
	dblbuffer	 <p>Double buffer + wire</p>
Multiplexer Stage Types	wmux	 <p>NMOS multiplexer + single buffer + wire</p>
	wmuxl	 <p>NMOS multiplexer with level restorer + single buffer + wire</p>
	wmuxsqrbuffX	 <p>NMOS multiplexer + square root sized buffer + buffer + wire</p>
	wcmux	 <p>CMOS multiplexer + single buffer + wire</p>
	wcmuxsqrbuffX	 <p>CMOS multiplexer + square root sized buffer + buffer + wire</p>

Table 3.6 - Characterized Sections

3.3.2 Circuit Characterization Process

The first step in the characterization process is to build a parameterized SPICE circuit model (SPICE deck) which allows for using variable buffer size and wirelengths. The parameterized circuit model is then plugged into a testbench circuit which is built to model the input and output conditions of the circuit. This testbench arrangement is shown in Figure 3.18.

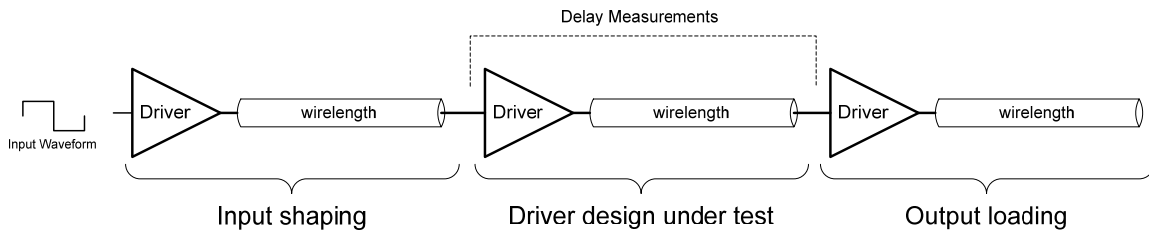


Figure 3.18 - Testbench Configuration

Next, a Matlab interface to HSPICE is developed. This function is capable of modifying the SPICE deck parameters, launching the spice job and collecting the desired data. Additional Matlab scripts were written to perform two dimensional HSPICE based sweeps on the buffer sizes and the wirelengths of each circuit type. These scripts call the underlying HSPICE simulator to generate delay data and load the lookup table array. In the following section, this data is examined to identify certain characteristics about the technology and produce design information.

3.3.3 Characterization Results & Analysis

The characterization process produces a three dimensional mesh plot of delays as shown in Figure 3.19. In this plot, the x-axis represents the buffer size in multiples of minimum size buffers and the y-axis represents wirelength in mm. The delay in picoseconds is produced along the z-axis for each buffer size-wirelength pair.

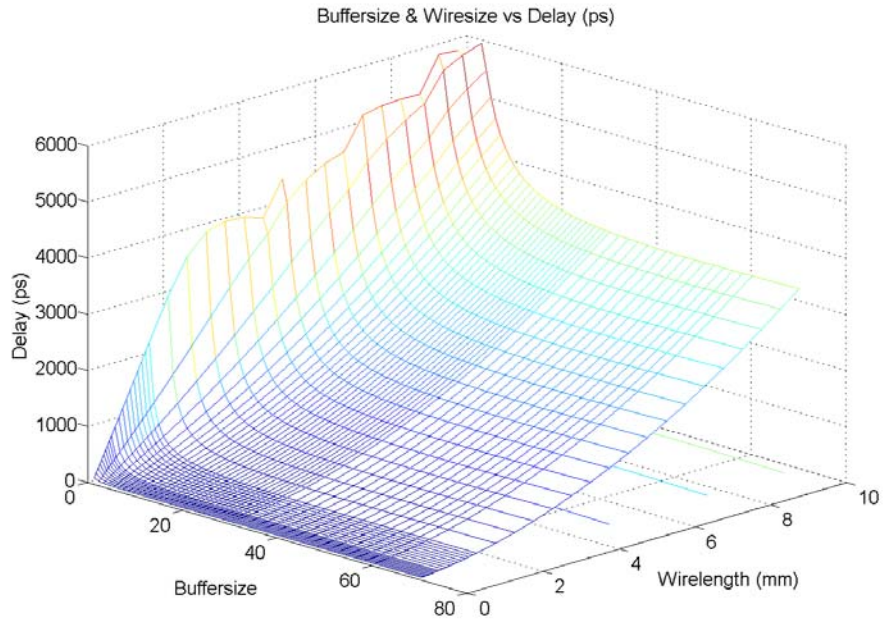


Figure 3.19 - Delay vs. Buffer size and Wirelength for 180nm 1x1x nomux Design

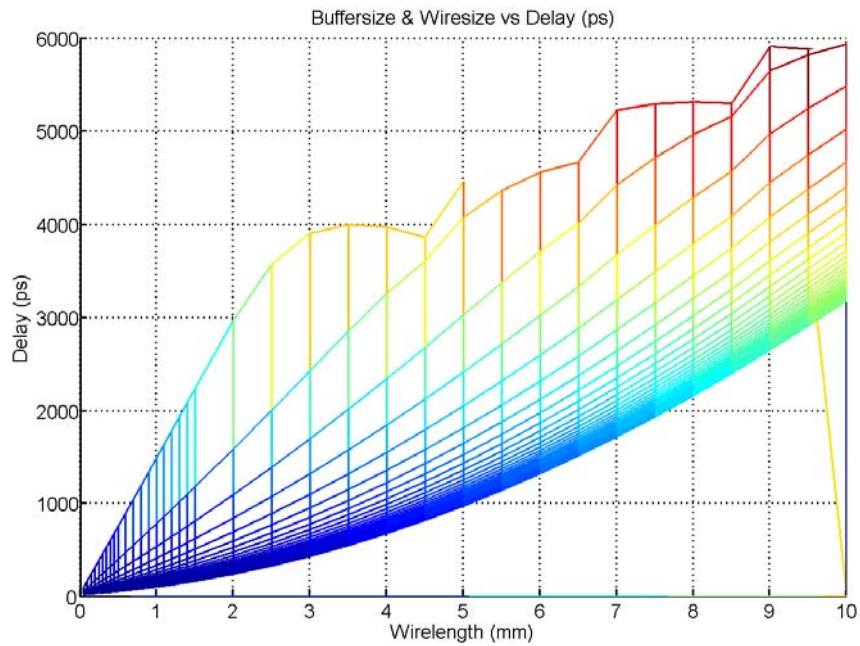


Figure 3.20 - Delay vs. Wirelength for Different Buffer sizes for 180nm 1x1x nomux Design

- Wirelength Axis

Since the plot is in 3D, it helps to view it from various perspectives to help identify local minima. Figure 3.20 presents the delay from the wirelength axis perspective. It simply shows

that the minimum delay is achieved with the shortest wire, which is a fairly intuitive result. However, using a different metric, it is possible to identify which wirelength achieved the fastest end-to-end signal propagation speed. This metric is referred to as the “**delay-per-millimeter**” and can be calculated by dividing the delay by the distance traveled. A similar metric is described in [34].

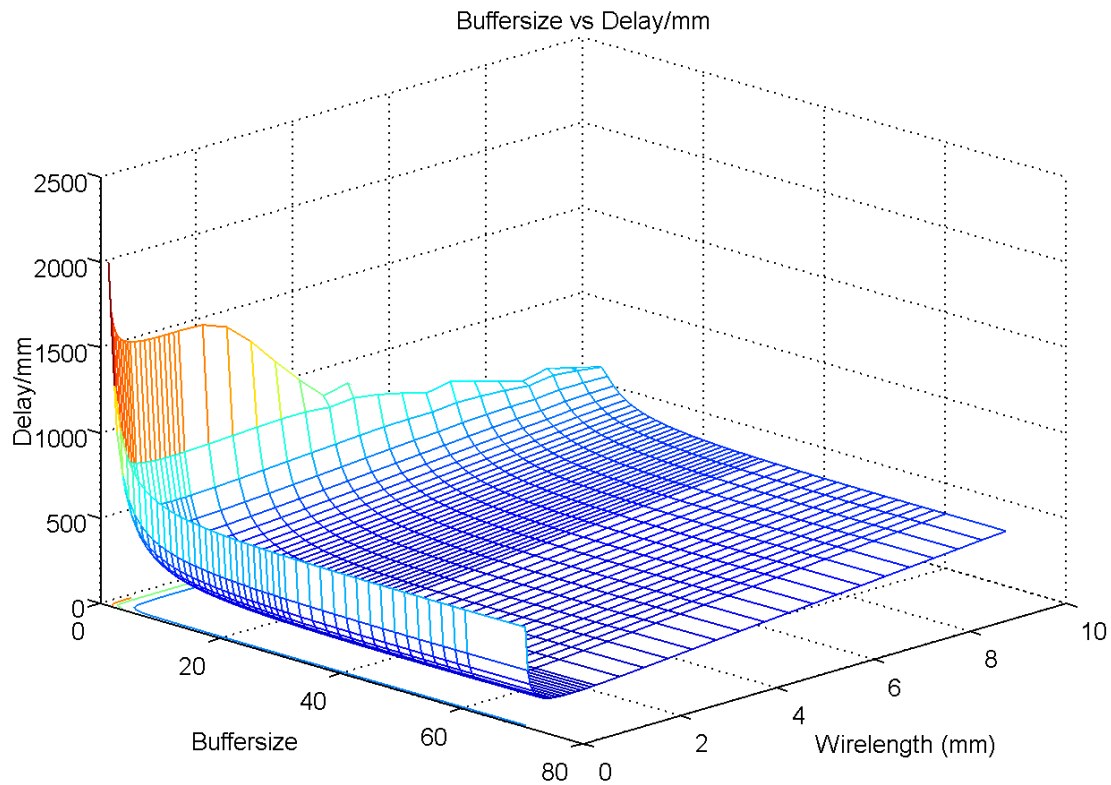
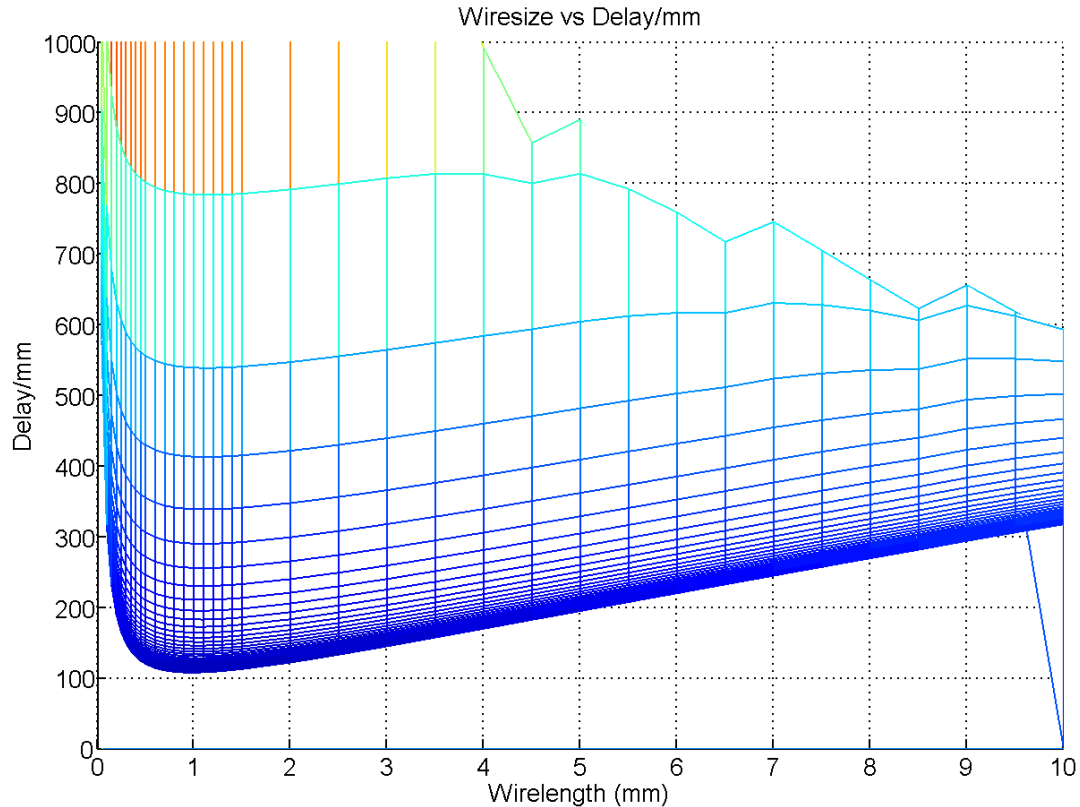


Figure 3.21 - Delay/mm vs. Buffer size and Wirelength for 180nm 1x1x nomux Design

Figure 3.21 presents a 3D plot similar to Figure 3.19. The difference is that the z-axis now presents the delay in terms of delay-per-millimeter.



**Figure 3.22 - Delay/mm vs. Wirelength for Different Buffer sizes for 180nm 1x1x nomux
Design - Wirelength Axis**

As before, Figure 3.22 presents a view of Figure 3.21 from the wirelength axis. This perspective clearly indicates that in this technology, using wires approximately 1mm in length will achieve the best delay-per-millimeter.

For the distributed (non-multiplexed) stages, the wirelength corresponding to the minimum delay-per-millimeter represents the optimal distributed drive stage wirelength. This is the spacing that would be used by an ASIC uniform repeater insertion process. In a given process technology, this spacing represents a lower bound on the delay-per-millimeter of ASIC interconnect. However, it is important to keep in mind that this only applies to designs using the same metal layer, wire width and pitch.

Best Delay-per-millimeter Values for Various Circuit Sections			
Process Technology	Characterized Stages	1x spacing 1x width Delay (ps/mm)	2x spacing 2x width Delay (ps/mm)
180nm	nomux	108	69.1
	dblbuffer	179	-
	Wmux	322	-
	Wmuxl	340	-
	wmuxsqrbuffX	215	-
	Wcmux	262	-
	wmuxsqrbuffX	207	138
90nm	nomux	90.9	57.9
	wmuxsqrbuffX	199	131

Table 3.7 - Best Delay-per-millimeter for Various Sections

Table 3.7 summarizes some of the delay-per-millimeter values for the different characterized sections. Of the multiplexer stage types in 180nm, **wmuxsqrbuffX** was the fastest design and is assumed for the remainder of the work. This circuit is made up of a CMOS passgate multiplexer followed by a two-stage lumped driver as shown in last row of Table 3.6 (page 46). The size of the first buffer is equal to the square root of the size of the second buffer. From this point forward, any discussions involving the “size” of this driver will be referring to the size of the second buffer. Since this design was selected and because characterization is a time-consuming process, only the nomux circuit and the wmuxsqrbuffX circuit were characterized for other technology configurations, as shown in Table 3.7.

Selecting Buffer Sizes

The characterization data also provides valuable information which is used for accurate selection of buffer sizes. As an example, Figure 3.23 presents buffer size curves for wirelengths 0.1mm to 4.0mm for the 180nm 1x1x nomux stage. Buffer sizes which yield the minimum delay at each wirelength are marked with x's and connected with another line.

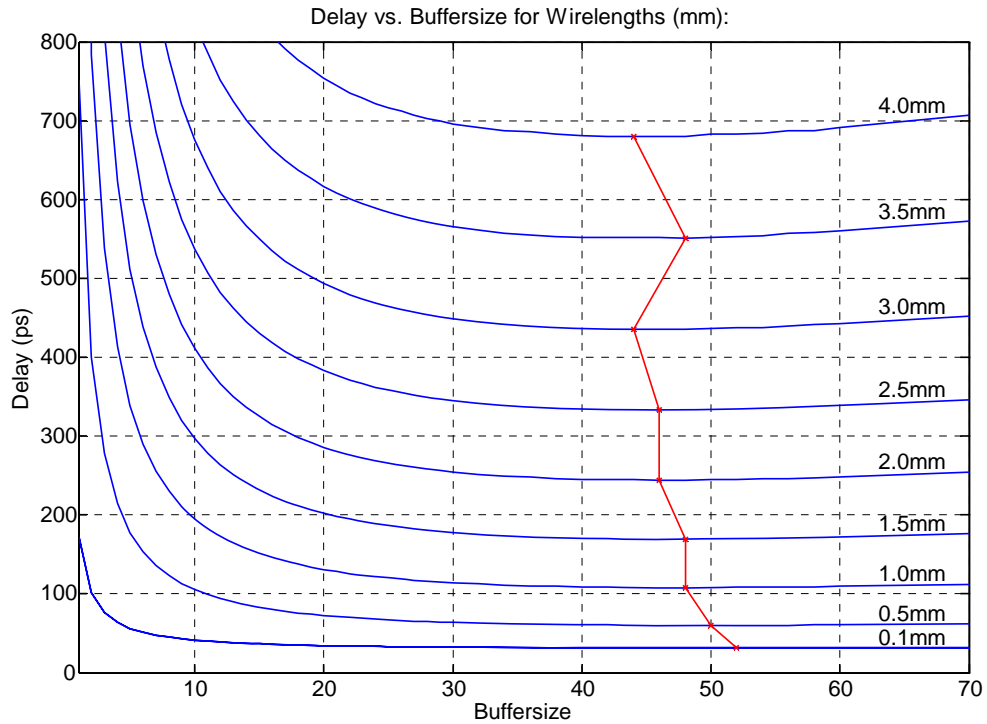


Figure 3.23 - Buffer size Selection for 180nm 1x1x nomux

Since the delay curve is quite shallow towards the minimum delay, it is common practice to use a buffer size corresponding to a modest delay increase instead of the minimum delay to save significant transistor area. In this work, the buffer size corresponding to a delay which is 10% above the minimum delay was extracted. Figure 3.24 presents buffer size data for both minimum delay and the 10%-off minimum delay for the 180nm 1x1x nomux design.

Note that the curve representing the minimum-delay buffer sizes is not smooth. This is due to the coarseness of the characterization data and the fact that the delay surface is very flat. This sometimes leads to minor variations in the results of future computations based on these buffer sizes.

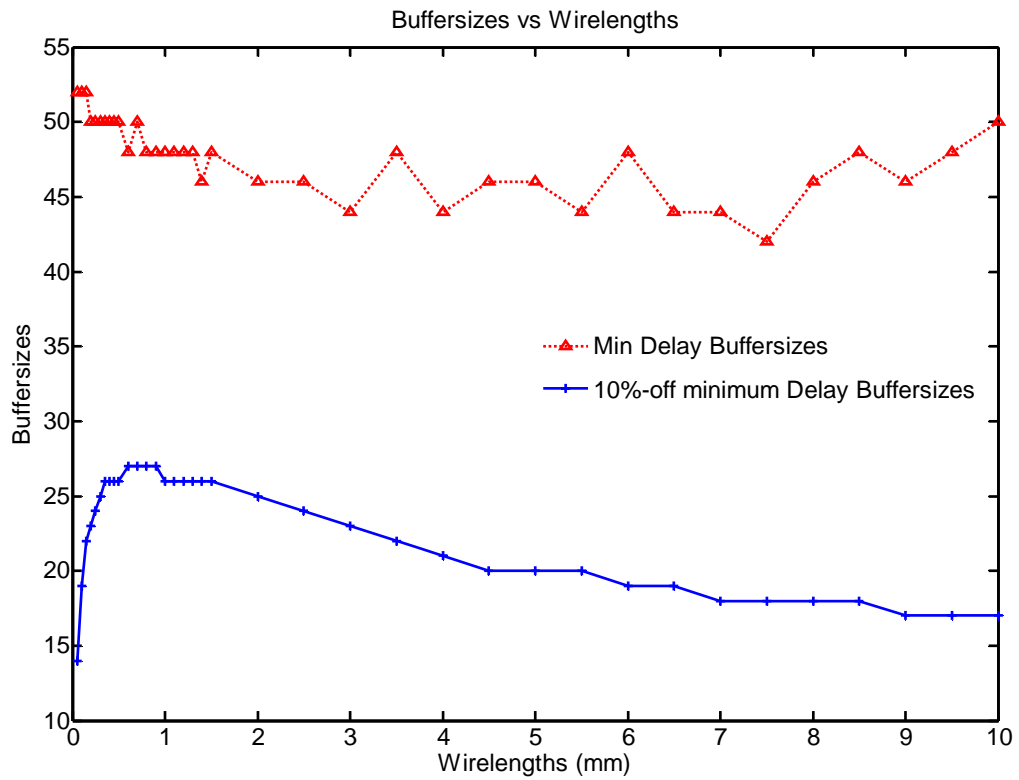


Figure 3.24 - Buffer size Selections for 180nm 1x1x nomux Design

3.3.4 Delay Concatenation

In this section, a technique is described which uses the pre-characterized data as a lookup table to produce the delay of an arbitrary circuit driver design. This technique assumes that the overall interconnect circuit design is constructed through some combination of the pre-characterized stages. Figure 3.25 presents an example of how stages can be concatenated together to produce a total interconnect driver solution. Since all the internal sections of the circuit are fully buffered, it is reasonable to assume the delay is additive. This assumption will be verified in the following section.

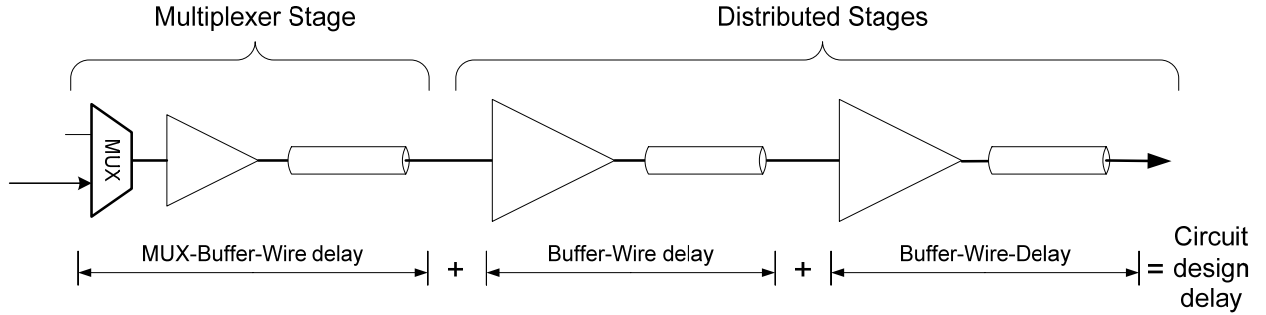


Figure 3.25 - Circuit Concatenation Example

3.3.5 Verification of Delay Concatenation

In order to verify accuracy of the concatenation approach several circuit designs were built and their delays were computed using both HSPICE and the concatenation technique. As suggested in section 3.2.4, the circuits are built using two sections, the multiplexer stage and the distributed stages which are made up of identical buffer-wire drive stages. The circuit designs had 2 to 5 stages and total wirelengths ranging from 2.0mm to 3.0mm. These ranges were chosen because they represent the range of the design space where distributed driver designs begin to outperform lumped designs. The delays of the test circuits are shown in Tables 3.8 and 3.9.

180nm 1x spacing 1x width						
Total Wirelength (mm)	Number of Stages	Multiplexer Stage Length (mm)	Distributed Stage Buffer-wire Segment Length (mm)	Concatenated Delay/mm	HSPICE Delay/mm	% error
2.0	2	0.80	1.20	198	192	3.2%
	3	0.15	0.93	196	203	-3.4%
	4	0.15	0.62	202	210	-4.1%
3.0	2	1.50	1.50	178	172	3.6%
	3	0.60	1.20	171	168	1.7%
	4	0.36	0.88	170	170	0.2%
	5	0.36	0.66	174	174	0.0%

Table 3.8 - Concatenation Verification Results (180nm)

90nm 2x spacing 2x width						
Total Wirelength (mm)	Number of Stages	Multiplexer Stage Length (mm)	Distributed Stage Buffer-wire Segment Length (mm)	Concatenated Delay/mm	HSPICE Delay/mm	% error
2.0	2	0.80	1.20	110	108	1.9%
	3	0.15	0.93	103	105	-1.6%
	4	0.15	0.62	103	105	-2.2%
3.0	3	0.60	1.20	95	94	0.5%
	4	0.30	0.90	91	91	-0.2%
	5	0.26	0.69	90	91	-1.0%

Table 3.9 - Concatenation Verification Results (90nm)

The results show that the concatenated delay model can achieve delay estimates within 4% of HSPICE. The accuracy of this approach appears to degrade when the stage lengths become extremely short (under 200um). Table 3.9 presents similar results based on a 90nm technology. Here, the accuracy is improved to within 2.2% of HSPICE.

3.4 Design Parameter Search

In the previous section, a circuit modeling method capable of rapidly and accurately determining the delay of a circuit design based on lookup tables was presented. This section describes an approach which uses the aforementioned circuit modeling technique to determine values for the buffer sizes, number of stages and buffer spacings used to construct a complete FPGA switch driver.

3.4.1 Circuit Construction

Results from section 3.2.4 showed that it is possible to construct a driver design using two separate sections: the multiplexer stage and the distributed stages. This simplification substantially reduces the search space, particularly when considering higher N values (number of stages), by making it possible to only consider 2 wirelengths and 2 buffer sizes. Figure 3.26

presents a general template of how a driver circuit will be constructed using the aforementioned restrictions. (Recall from Section 3.1.3 that the symbols B_i represent the size of the buffer driving a wire of length L_i).

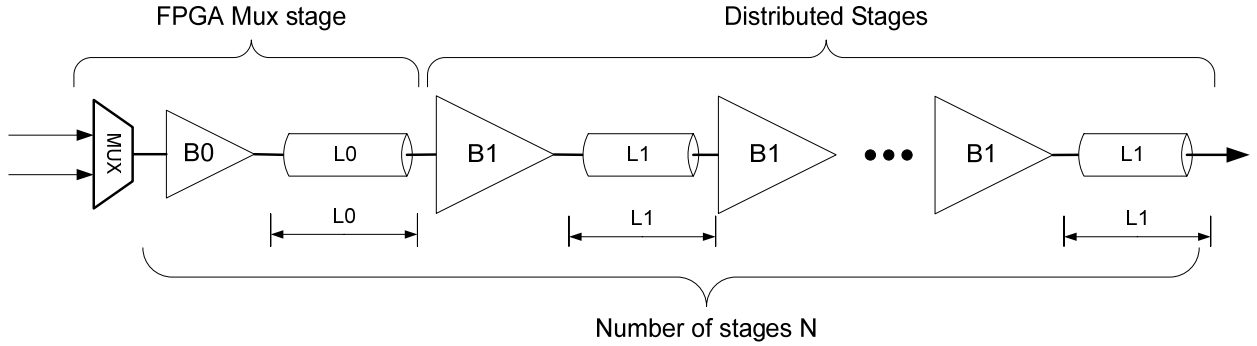


Figure 3.26 - Driver Construction Template

The multiplexer stage in Figure 3.26 is made up of the `wcmuxsqrbuffX` stages described in the characterization step (mux-buffer-wire). The distributed stages can be constructed using one or more buffer-wire pairs (nomux stages) to make up a uniformly distributed buffer arrangement.

Driver Delay Equation

Using the template described above, several equations can be written to relate the lengths and delays of the entire circuit design. The first equation relates the stage wirelengths to the number of stages (N) and the total wirelength (L).

$$L = L0 + (N - 1) \times L1 \quad (2)$$

This equation will primarily be used to solve for $L1$, given any L , N and $L0$. Once $L0$, $L1$, L and N are known, the delay of each buffer-wire segment can be extracted from the pre-characterized data, by looking up the appropriate buffer size and delay corresponding to a wirelength $L0$ or $L1$. The final design delay is calculated as the sum of the segment delays.

$$delay = d_{mux}(L0) + (N - 1) \times d_{distrib}(L1) \quad (3)$$

Where $d_{circuittype}(x)$ is the delay of wirelength x for a circuit stage determined by the pre-characterized lookup table data.

Given a total wirelength (L) and a number of stages (N), this process of calculating the wirelength, looking up the delay and buffer size must be performed for all values of L0 and L1. However, since L0 and L1 are related, the search degenerates into a single sweep referred to as the segment length sweep, or the L0 sweep.

3.4.2 Segment Length Sweep

The segment length sweep involves sweeping the length of the first stage (L0) to determine the best delay for a particular configuration of total wirelength (L) and number of stages (N). Plotting the delay of the resulting circuit design against L0 reveals a curve which contains a minimum. Figure 3.27 presents the effect of changing L0 on the delay. In this example the best delay-driven circuit design should have L0=0.15mm to achieve a delay just over 390ps.

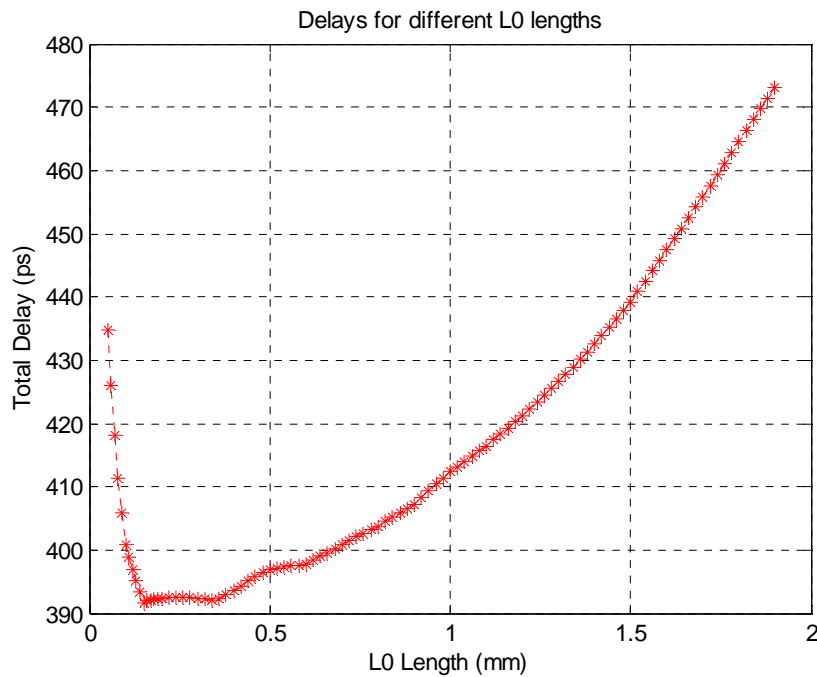


Figure 3.27 - L0 Sweep for a 2mm Wire in 180nm 1x1x for N=3 Stages

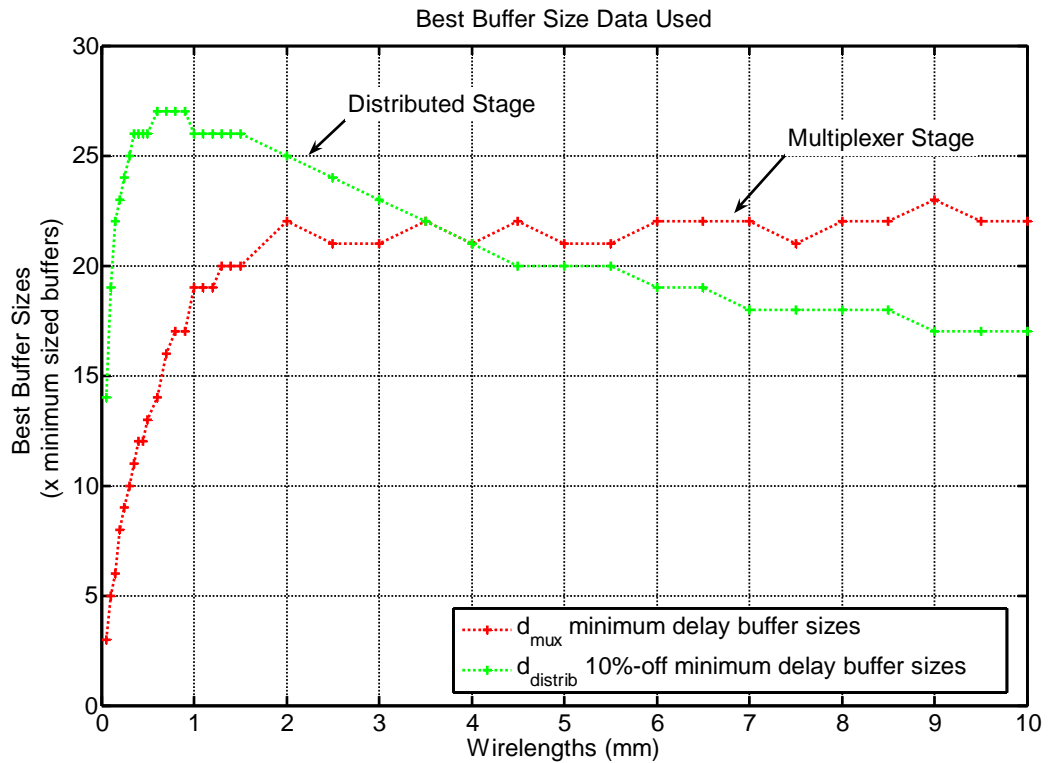


Figure 3.28 - Buffer sizes Used for L0 Sweep in 180nm 1x1x

It is noted that the L0 sweep in Figure 3.27 is not completely smooth and contains some kinks in the curve. This is due to changing buffer sizes according to the data shown in Figure 3.28. In this figure, the curve for the distributed stage gives the buffer size B1 to use with wirelength L1, while the curve for the multiplexer stage gives the buffer size B0 to use with the wirelength L0. As discussed in 3.3.3, the discrete jumps in Figure 3.28 are due to the flat delay curve and the coarseness of the characterization data. However, because the delay curves are insensitive with respect to buffer sizes, this does not interfere with establishing a circuit design which achieves close to minimum delay.

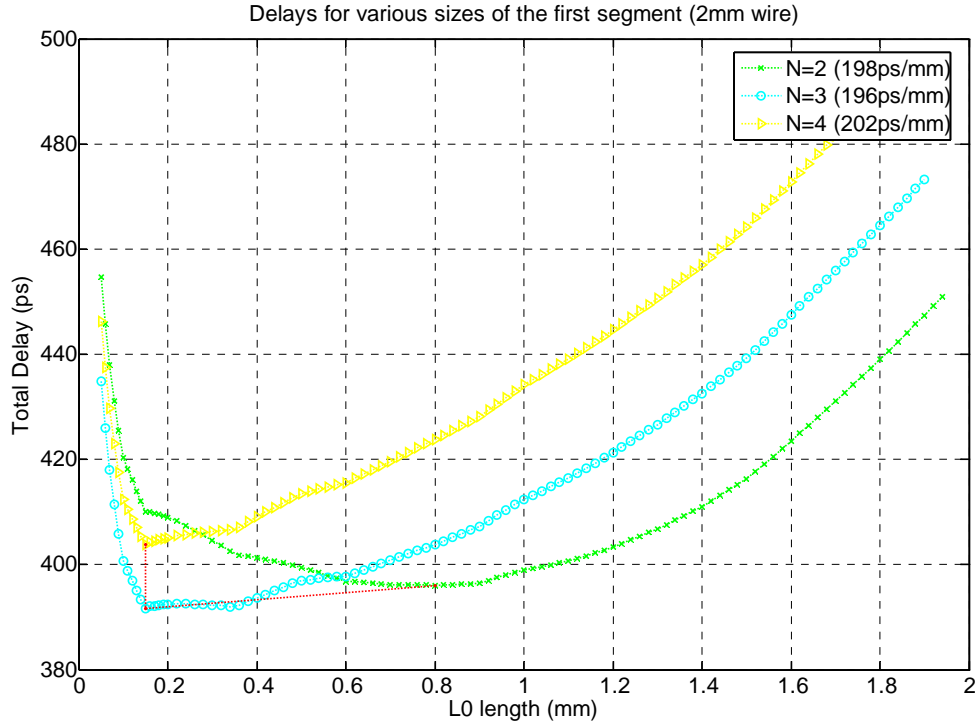


Figure 3.29 - Multi-N L0 Sweep for a 2mm Wire in 180nm 1x1x

Multi-Segment Length Sweeps

Using L0 plots it is possible to compare designs with different numbers of stages (N). An example of a multi-N L0 sweep is shown in Figure 3.29. This plot shows that there is a value of N which yields the delay curve containing a minimum delay. In this example, the optimal design

parameters are: $L = 2mm$, $N = 3$, $L_0 = 0.15mm$ and $L_1 = \frac{L - L_0}{N - 1} = 0.925mm$. Buffer sizes are extracted from Figure 3.28 to yield $B_0 = 6$ from the minimum delay curve, and $B_1 = 22$ from the 10%-off minimum delay curve.

In Figure 3.29 the effect of the number of stages appears to have a minimal impact on the best delay, however, this is dependant on the wirelength and technology. Figure 3.30 presents a similar plot, except for a 4mm long wire. In this case, the effect of changing N has a larger impact on the overall delay, particularly between N=2 and N=3.

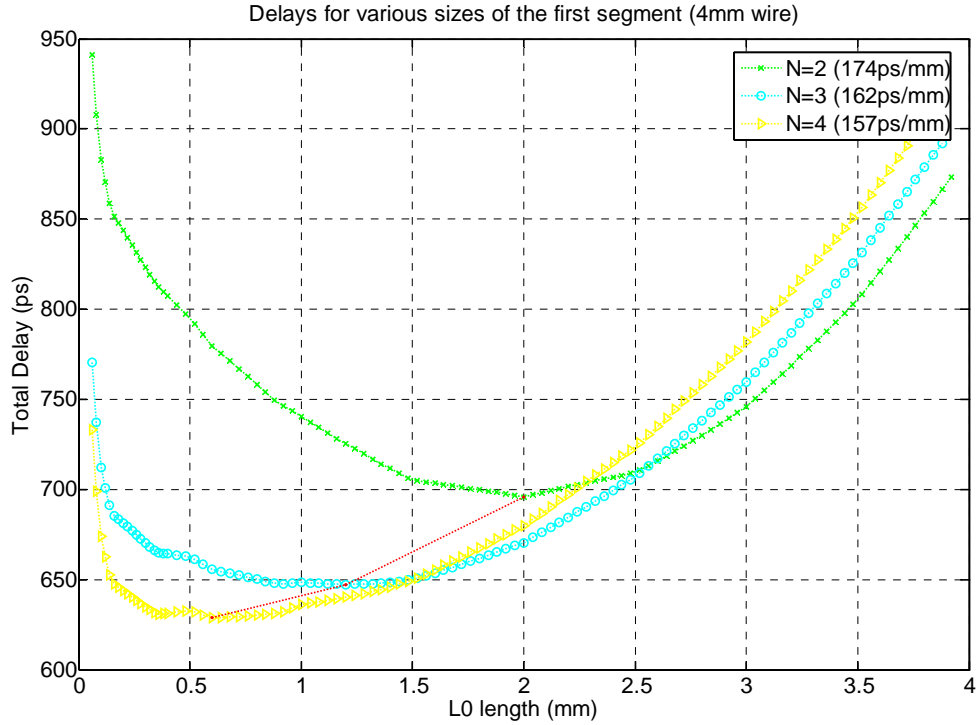


Figure 3.30 - Multi-N L0 Sweep for a 4mm Wire in 180nm 1x1x

3.5 Circuit Design Results & Analysis

The multi-N L0-sweep was performed for a variety of total wirelengths (L) and number of stages (N) to produce optimal circuit designs for various total wirelengths. Each solution provides the four parameters required to build the circuit. For the multiplexer stage, the length is L_0 and the buffer size is B_0 . For the distributed stages, there are $N-1$ buffer-wire segments with B_1 sized buffers and wires of length L_1 . This arrangement was depicted in Figure 3.26 (page 56).

Results of the multi-N L0-sweep for different wirelengths are presented in Tables 3.10 and 3.12. Circuit designs built in 180nm process technology with wires at 1x minimum width and 1x minimum spacing are shown in Table 3.10. Similar results for 90nm with wires at 2x minimum width and 2x minimum spacing are shown in Table 3.12. In addition to the six design parameters used to construct the circuit, the estimated delay-per-millimeter is shown. For each wirelength,

the design which yields the best delay (ps/mm) is in bold. It is interesting to note that the L1 lengths of the fastest designs are closest to the length of wire shown to achieve the minimum delay-per-millimeter in the delay characterization process.

180nm - 1x spacing 1x width						
Wirelength (mm)	Number of Stages (N)	Multiplexed Stage		Distributed Stage Buffer-wire Segment		Delay (ps/mm)
		Driver Size B0 (x min)	Length L0 (mm)	Buffer Size B1 (x min)	Length L1 (mm)	
0.5	2	3.0	0.05	14.0	0.45	414
	3	3.0	0.05	14.0	0.23	462
1	2	6.0	0.15	22.0	0.85	266
	3	5.0	0.10	19.0	0.45	283
	4	5.0	0.10	19.0	0.30	306
2	2	17.0	0.80	27.0	1.20	198
	3	6.0	0.15	22.0	0.93	196
	4	6.0	0.15	22.0	0.62	202
3	2	20.0	1.50	26.0	1.50	178
	3	14.0	0.60	27.0	1.20	171
	4	11.2	0.36	26.0	0.88	170
	5	11.2	0.36	26.0	0.66	174
4	2	22.0	2.00	25.0	2.00	174
	3	19.0	1.20	26.0	1.40	162
	4	14.0	0.60	27.0	1.13	157
	5	11.2	0.36	26.0	0.91	157

Table 3.10 - Distributed Driver Design Results for 180nm

Lumped Design Results 180nm - 1x spacing 1x width			
Wirelength (mm)	Number of Stages (N)	Buffer Sizes (x min)	Delay (ps/mm)
0.5	2	3.74, 14.0	408
1	3	4.0, 10.0, 30.0	260
2	3	4.0, 9.0, 35.0	192
3	3	3.3, 11.1, 37.0	184
	4	2.6, 6.7, 17.4, 45.0	186
4	3	3.4, 11.5, 39.0	194
	4	2.7, 7.1, 18.9, 50.0	191

Table 3.11 - Lumped Driver Design Results for 180nm

90nm - 2x spacing 2x width						
Wirelength (mm)	Number of Stages (N)	Multiplexed Stage		Distributed Stage Buffer-wire Segment		Delay (ps/mm)
		Driver Size B0 (x min)	Length L0 (mm)	Buffer Size B1 (x min)	Length L1 (mm)	
2	2	31.0	0.80	49.0	1.20	110
	3	12.0	0.15	43.0	0.93	103
	4	12.0	0.15	43.0	0.62	103
	5	12.0	0.15	43.0	0.46	105
3	3	27.0	0.60	50.0	1.20	95
	4	19.0	0.30	48.0	0.90	91
	5	17.2	0.26	47.1	0.69	90
	6	17.2	0.26	47.1	0.55	91
4	5	22.0	0.40	49.0	0.90	85
	6	19.0	0.30	48.0	0.74	84
	7	19.0	0.30	48.0	0.62	84
	8	17.4	0.26	47.2	0.53	85

Table 3.12 - Driver Design Results for 90nm

Lumped Design Results 90nm - 2x spacing 2x width			
Wirelength (mm)	Number of Stages (N)	Buffer Sizes (x min)	Delay (ps/mm)
2	3	4.0, 16.2, 65	115
3	5	2.6, 6.6, 16.8, 43.0, 110.0	115
4	5	2.6, 6.8, 17.7, 46.1, 120.0	125

Table 3.13 - Lumped Driver Design Results for 90nm

Data for the best lumped driver designs is shown in Tables 3.11 and 3.13 for comparison with the distributed driver design results. These circuits are designed by assuming a geometric relationship between each successive driver and running an HSPICE-based search on the final drive stage for a range of N values. Once the final drive stage is determined, the design is optimized by hand to reduce any delays caused by a non-ideal drive stage ratio. Since the delay of the lumped design is strongly dependant on the final drive stage, rigorous tuning of the intermediate buffers is not crucial.

Analysis of the driver design results yield several useful conclusions regarding FPGA architecture and distributed buffering.

3.5.1 Multiplexing Intervals

In Figure 3.31 the delay-per-millimeter of the programmable interconnect designs in 180nm process technology can be compared to the best ASIC delay found in Table 3.7. It is expected that the programmable multiplexed design will always be slower than the ASIC equivalent. However, as the length of the programmable interconnect grows the delay-per-millimeter of the design improves. This has significance to FPGA architecture since the length of the programmable interconnect is essentially the distance between multiplexers in the FPGA device, or the “multiplexing interval.”

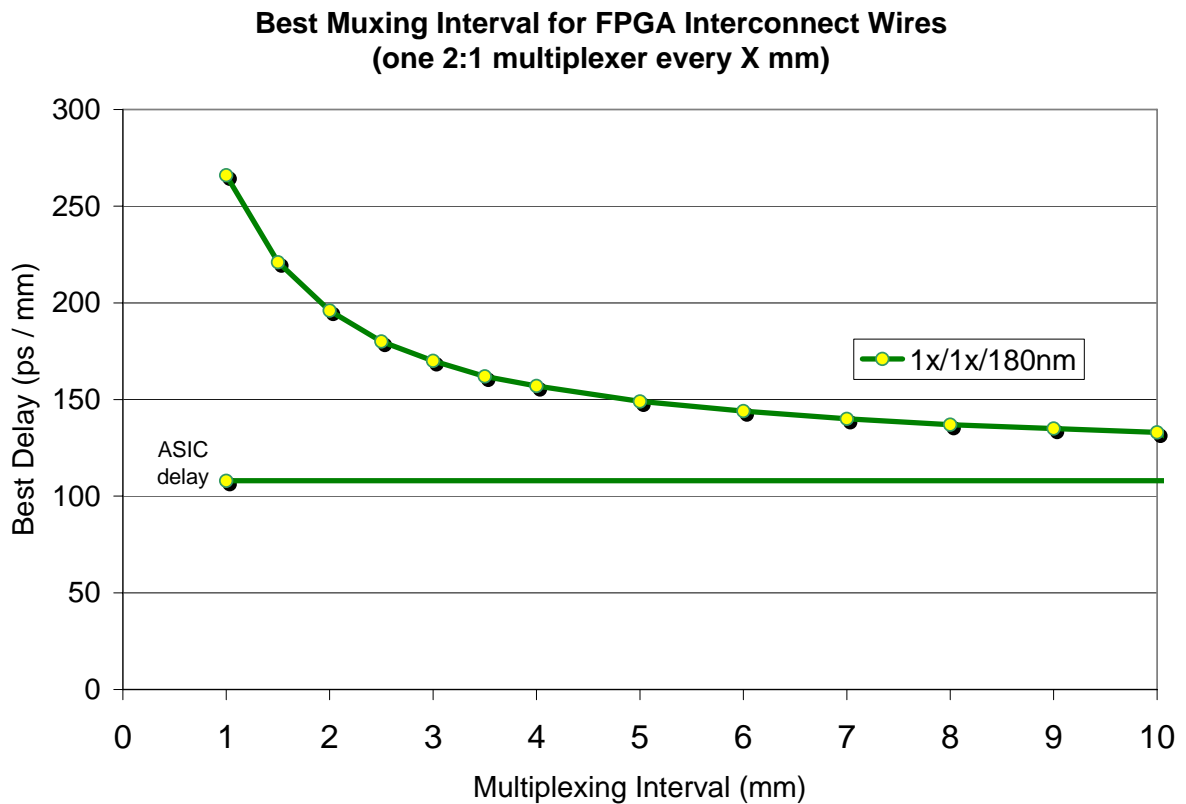


Figure 3.31 - Multiplexing Intervals for various Technologies

This means that an FPGA architect can choose to select a longer multiplexing interval in order to achieve a delay-per-millimeter which is sufficiently close to that of an ASIC. Identifying the best physical wirelength is important for FPGA architecture because it can be translated directly into an architectural wirelength when the tile size is known. Although Figure 3.31 suggests that using the longest possible wire can produce the smallest delay, it is important to note that this is often impractical due to many other tradeoffs. For instance, the larger logical lengths that come about from longer wires will increase the minimum channel width required to ensure the start of a wire is located at all CLBs. In contemporary FPGAs, 3mm [35] is a reasonable length for long wires.

3.5.2 Distributed Buffering

Using the driver design results, the suitability of distributed buffering can be assessed. The delay of the distributed driver designs in Tables 3.10 and 3.12 will be compared with those of the lumped designs using path delay profiles. PDPs provide a qualitative assessment of the benefits of distributed driver design on midpoint delays in addition to a direct comparison with the end-to-end delays.

Figure 3.32 presents PDPs for 0.5mm to 4mm for a 180nm process. From the plots it is clear that the distributed driver designs offer improved end-to-end performance for longer wires. In this technology, wirelengths beyond 2mm benefit from distributed designs when compared to lumped designs. In the case where end-to-end delay is roughly equal, distributed designs can offer smaller midpoint delays. For example, a 2mm distributed N=2 design uses roughly the same total buffer size area as the 2mm lumped N=3 design and both have the same end-to-end delay. However, the distributed design offers smaller delay to early turns up to 800 μ m down the wire. At 800 μ m, the distributed design offers a turn delay which is approximately 18% faster than the lumped design. The actual net benefit of achieving this early midpoint delay is not clear

without proper modeling and simulation using a real CAD router and timing analysis. This is the objective of the next chapter.

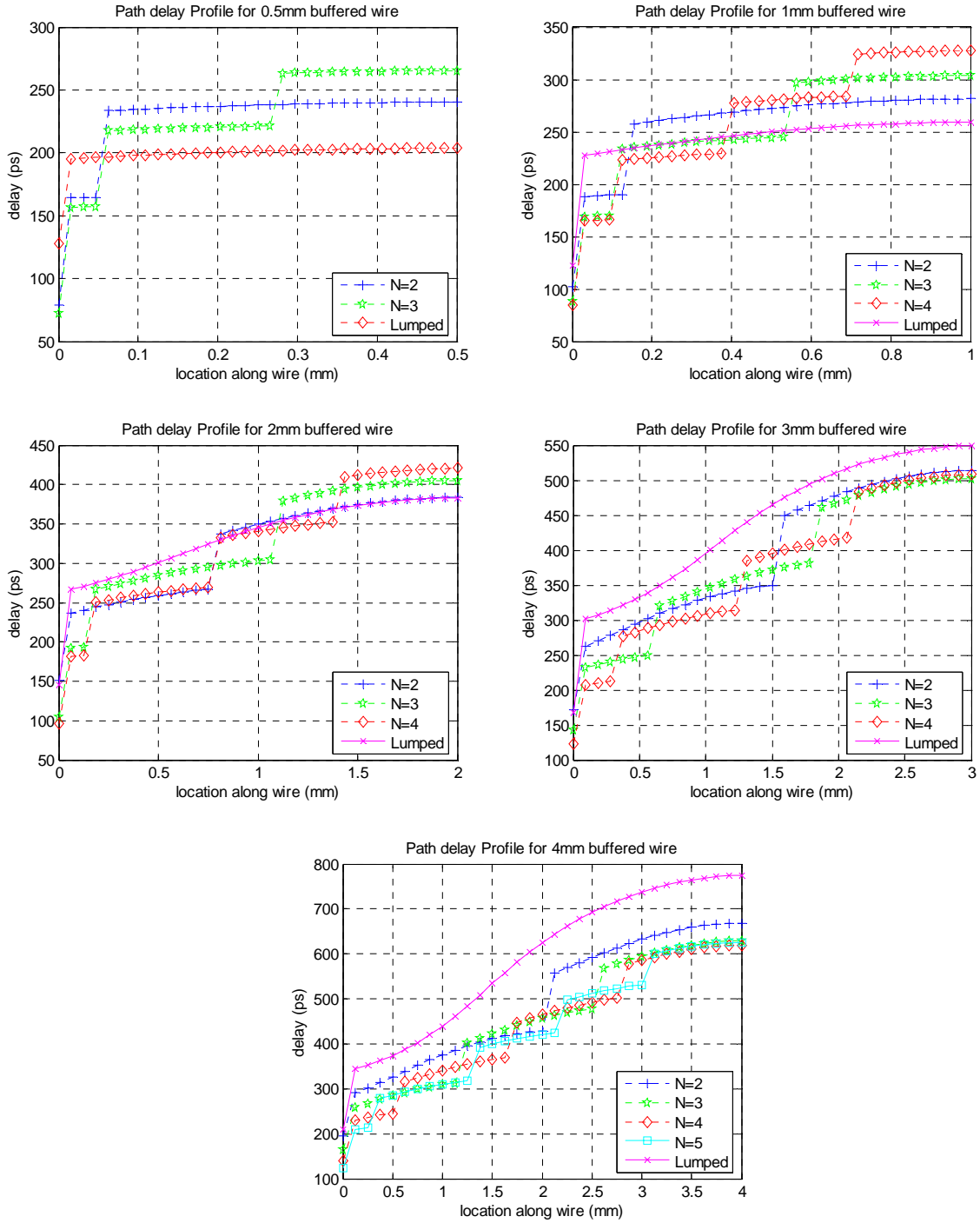


Figure 3.32 - Path Delay Profile Plots for 180nm 1x1x for 1mm-4mm

CAD Circuit Architecture Modeling and Evaluation

In the previous chapter a method to design FPGA switch drivers was presented. Given the nature of FPGAs it was not clear how the proposed designs would improve performance. In this chapter, an FPGA CAD tool, VPRx, is modified to model the proposed switch driver circuits in order to evaluate their performance. It begins with a discussion on the goals of the circuit evaluation, and proceeds to develop the model of the proposed circuits in VPRx. Later sections will present critical path delay results of using the new switch designs in an FPGA.

4.1 Goals of the CAD Circuit Evaluation

In the previous chapter, the design with the best delay was constructed using a 2:1 CMOS passgate multiplexer followed by 2 inverters and then by a uniformly distributed driver. In this chapter, that circuit design is modeled in VPRx to measure the impact of early turns and the new design on delay. Using a router which is aware of precise delays of early turns, it is investigated how often early turns are used and whether a performance improvement can be attained using circuits which offer reductions in midpoint delay. This requires that the CAD tools be capable of a) modeling the midpoint delays during routing, and b) making decisions based on this information.

Another architectural question involves the effect of adding fast paths to the FPGA routing resources. The new circuit design uses a simplified high-speed fast path to operate near maximum speed for straight connections. To determine the effect of these changes, they must be evaluated in the FPGA CAD framework as well.

Modification of CAD tools to fully support the proposed circuit designs is the first goal of this chapter. The second goal of the chapter is to use the modified CAD tools to implement benchmark circuits using a CAD framework to determine the overall improvement in FPGA performance.

4.1.1 Comments on Area Overhead

The goal of this thesis is to obtain the best possible delay performance from long wire interconnect. As such, area is only a secondary concern. To mitigate concerns over area overhead, we note two things. First, by stepping back delay by 10% from optimal, considerable area is saved. Second, the number of high-performance long wires in an FPGA device is expected to be a small fraction of the total amount of interconnect. Hence, even though large buffers are used, they appear infrequently (few long wires) and are spaced far apart (approximately 1mm apart). For this reason the remainder of this thesis will not be concerned with measuring area overhead.

4.2 Experimental Methodology

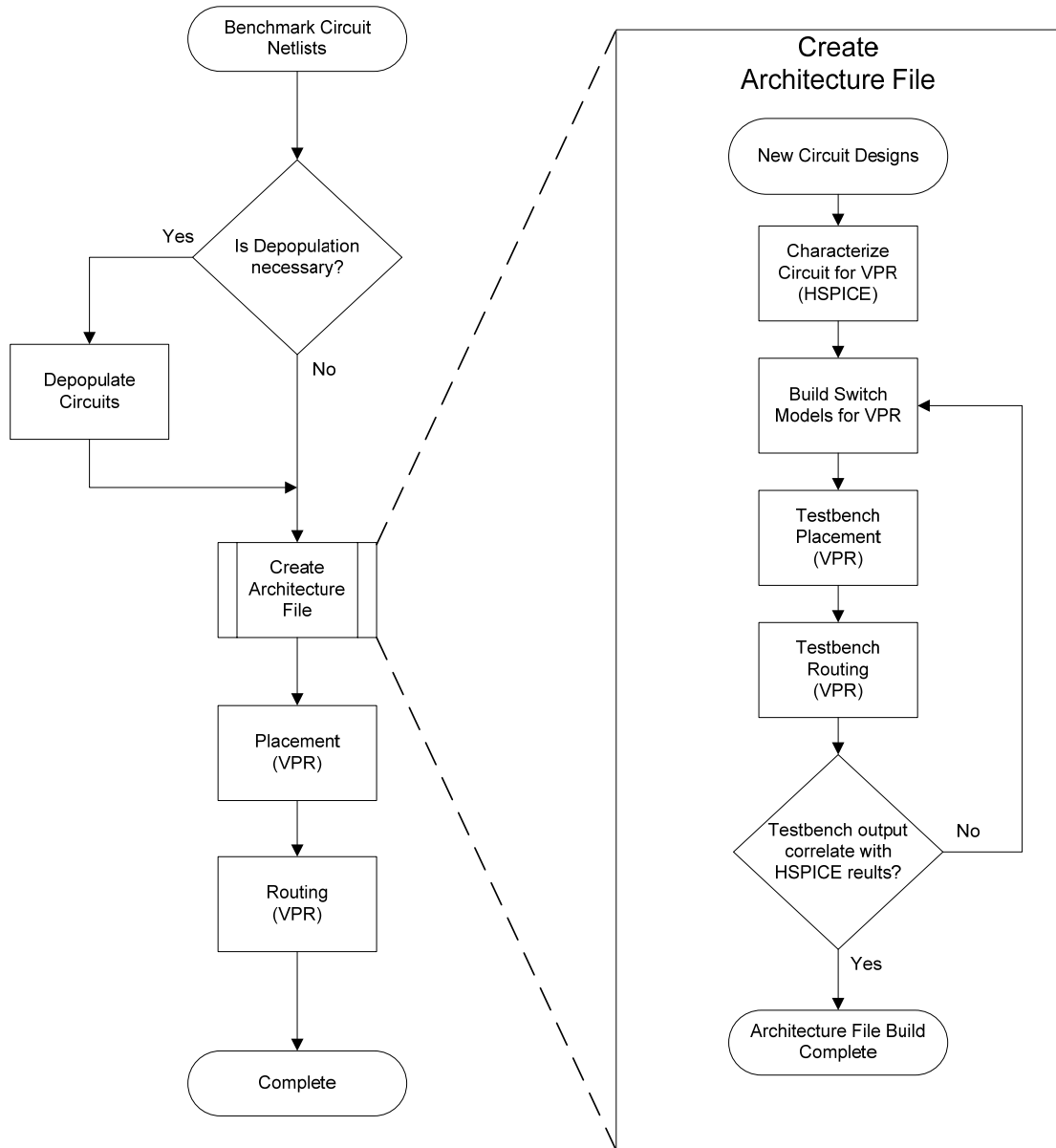


Figure 4.1 - Experimental Methodology Flow

The experimental methodology shown in Figure 4.1 follows the flow described in Chapter 2. Steps are added to depopulate circuits to generate larger FPGA arrays for use with longer wires. Also shown, is a process to generate architecture files based on the proposed circuit designs. Details on this process are provided in section 4.3.5.

4.3 *Circuit Architecture Modeling*

In order to evaluate the new circuit designs using FPGA CAD tools, proper support needs to be added to the CAD models. In particular, VPRx must be able to use early turns and fast paths while accurately assessing their respective costs. Modifications made to VPRx can be broken down into *circuit modeling* and *router awareness*.

Circuit modeling allows accurate representation of the circuit designs in VPRx. The existing wire model built into VPRx lacks the resolution required to distinguish the benefits of the new designs from the previous designs. In order to increase the detail of the wire model and to model new circuit designs, VPRx was modified to model wires made up of multiple wire-segments and driver designs using distributed buffering. Another VPRx model that required revision is the multiplexer delay model. Since the two-level flat multiplexer design has delay properties which differ from the original tree-based design, a new multiplexer modeling scheme is introduced in section 4.3.4.

The second component of the modified CAD tool is router awareness. This requires that the CAD algorithms are capable of understanding and exploiting the routing options provided by the new circuit models. For instance, the router should be able to recognize the presence of a fast path and take advantage of it when applicable. In the following sections, detailed discussions of the new features added to VPRx are presented.

4.3.1 **Early Turn Modeling (ETM)**

In an FPGA, a *turn* occurs when a signal leaves a wire for another destination. In a unidirectional architecture, a turn which occurs at the end of a wire is referred to as a *normal turn* since it uses the entire interconnect resource. When a turn takes place before reaching the end of the wire it is referred to as an *early turn*. An example of the two types of turns is provided

in Figure 4.2. The reason why it is important to distinguish the two is because the delay of an early turn can be substantially smaller than that of a normal turn.

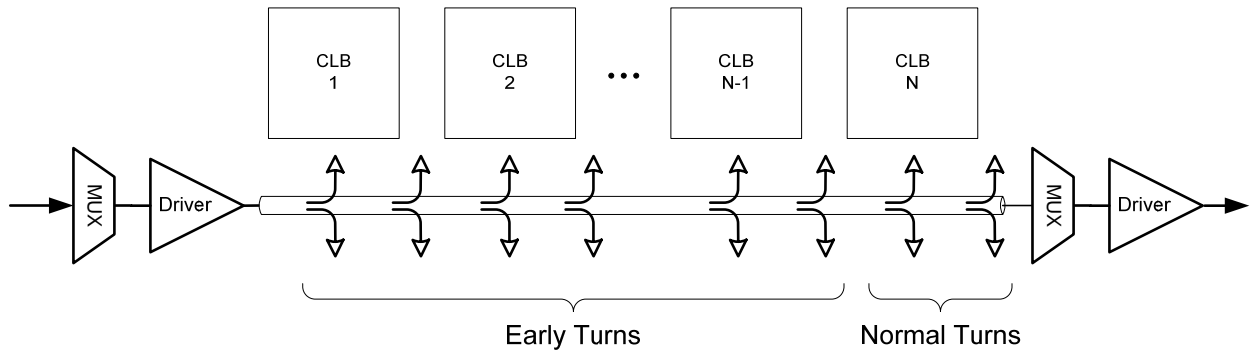


Figure 4.2 - Early Turns

Initial turn counting experiments indicated that early turns occurred often in unidirectional FPGA designs. Unfortunately, in the existing VPRx tool the delay of such turns is not modeled accurately because both normal turns and early turns are treated as if they have the same delay.

Turn Modeling Error

Treating the delay of all turns equivalently introduces error into the timing calculations. This affects the accuracy of the critical path calculation and the metrics used during the routing search. An idea of the error that can occur in the delay of the wire can be obtained by using an Elmore tree calculation as follows.

The delay of a normal turn is the delay to the end of the wire: $\frac{R_{wire} C_{wire}}{2}$

The delay of an early turn occurring at $\frac{1}{4}$ of the wirelength:

$$\begin{aligned}
 &= \frac{R_{wire}/4 \cdot C_{wire}/4}{2} + \frac{R_{wire}}{4} \cdot \frac{3C_{wire}}{4} \\
 &= \frac{7R_{wire} C_{wire}}{32}
 \end{aligned}$$

In this example, the normal turn delay is roughly 2.3 times larger than the early turn delay. To account for this, VPRx assumes that all turns occur at the halfway point of the wire and estimates all turns to have delay $\frac{R_{wire}C_{wire}}{4}$. In this work, VPRx was modified to model early turns in order to compute wire delay more accurately based on the actual turn location².

As the wirelength increases the potential difference in delay between an early turn and a normal turn grows larger. Hence, from a circuit design modeling standpoint, ignoring early turns cannot be tolerated when longer wires are used.

Proper early-turn modeling allows the timing analyzer to extract more accurate delays for the critical path. It also allows the router to make informed decisions on whether or not to use early turns instead of normal turns. Most importantly, early turn modeling makes it possible to assess the benefits of circuits offering improved delay to early turns.

Routing Resource Graph Modifications

A new option was added to the VPRx code called “**Early Turn Modeling**” or **ETM**. Implementation of ETM involved building a new routing resource graph which could accommodate the level of detail appropriate for modeling early turns at the routing-resource graph level.

² When VPRx is not using early turn modeling, timing calculations revert to their original behavior.

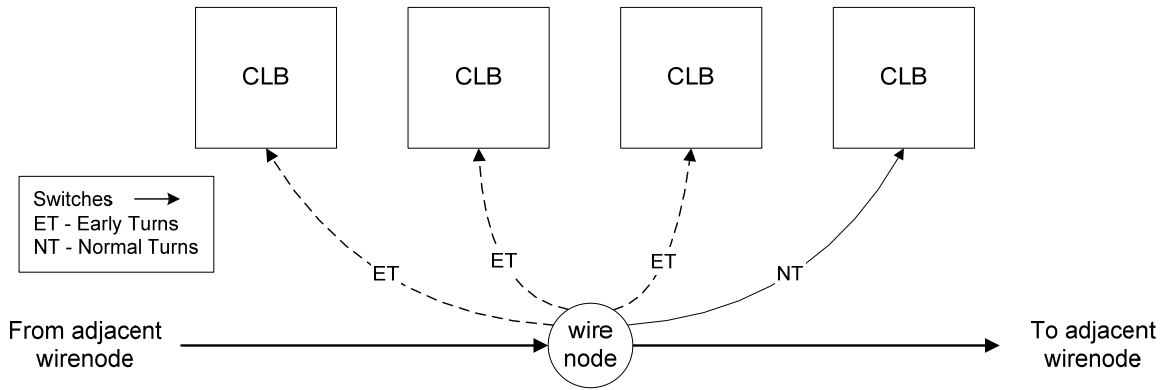


Figure 4.3 - Original Routing Resource Graph of VPRx

In the original VPRx, wire segments are modeled as discrete elements represented by nodes in the routing resource graph. An example highlighting where early turns are possible is shown in Figure 4.3. ETM was implemented by breaking up the existing wires into sub-wires; each sub-wire spans one CLB. To model this in the routing resource graph, the nodes are split apart into ETM sub-nodes and strung together using special ETM edges to form a multi-segment wire model. Once the new ETM sub-nodes are created, the edges representing early turns are connected directly to the closest ETM sub-node. Figure 4.4 shows the resulting routing resource graph.

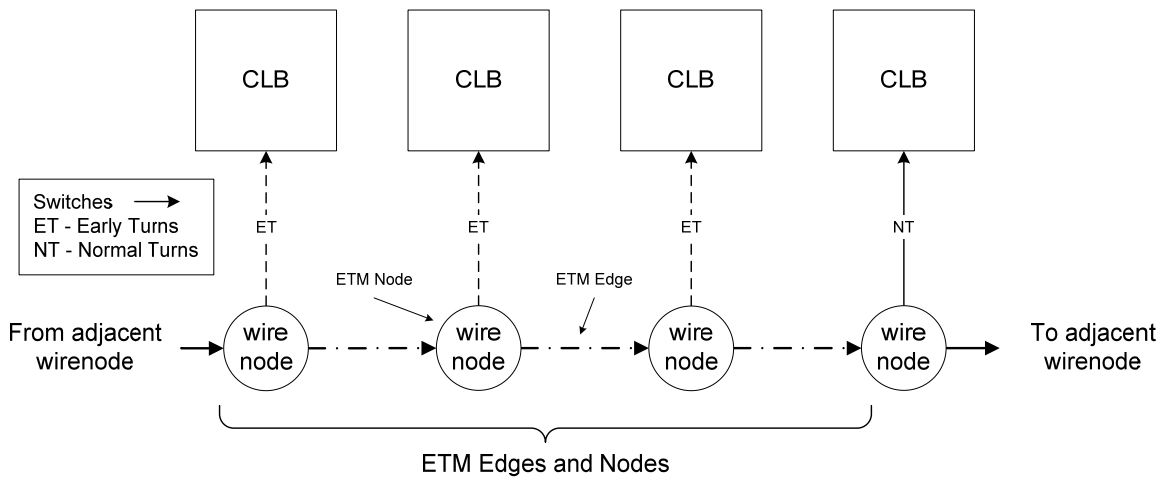


Figure 4.4 - Routing Resource Graph for VPRx with ETM Enabled

The result of this modification is a larger routing resource graph with more nodes and more edges, but it is now capable of modeling all possible turns. Although it is possible to obtain additional detail by breaking the wires into even smaller segments, this increases the complexity of the routing resource graph unnecessarily. The cost for the accuracy of ETM is increased runtime and memory use by the router. The amount that the routing resource graph grows is directly related to the architectural wirelength of the wire being modeled. For example, an L4 wire architecture ETM adds 3 additional nodes and wires for every existing wire. Similarly for an L16 architecture, enabling ETM adds 15 nodes and 15 wires for each existing wire, expanding the routing resource graph by 15 times. Note, however, this is roughly equivalent to the overhead needed if the architecture used L1 wires everywhere.

ETM Delay Calculations

The introduction of ETM requires some modifications to the router delay estimation code. Like before, the router must be able to calculate the incremental delay from any point in the ETM routing resource graph to its neighbors. However, this time the neighbor can either be the start of a new wire or an adjacent sub-wire in the middle of an ETM wire. Before ETM, during the routing wavefront expansion, the only thing that is known to the router is the return path to the source. Now, there are situations where the router must consider downstream information from as-yet-unvisited nodes of the routing resource graph. This scenario occurs when the router is considering adding the first segment of a long wire to the routing solution. The effect of the latter segments of the wire is manifested in the concept of **downstream capacitance**.

ETM requires that the router is able to calculate the delay to all midpoints of the wire both accurately and incrementally. The following derivation demonstrates how to calculate the delay

to the midpoint of a wire using Elmore tree analysis. Afterwards, the resulting equation is manipulated into a form which is practical for incremental calculation.

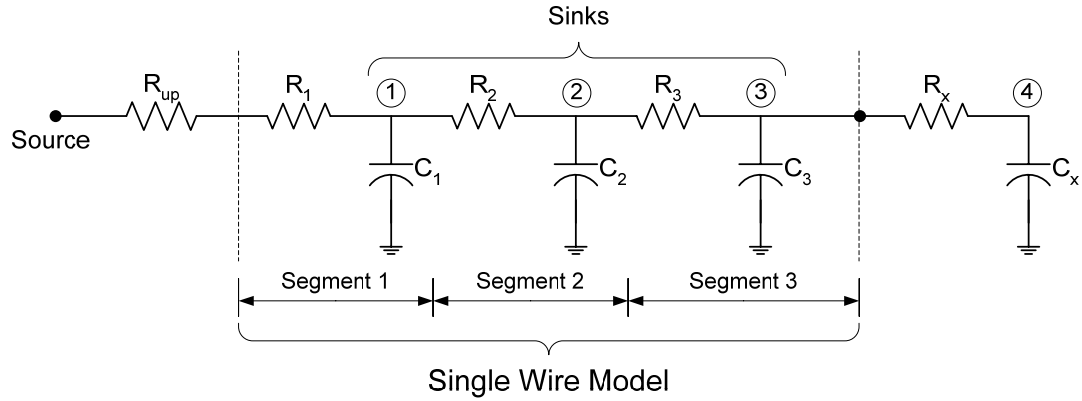


Figure 4.5 - Wire Delay Calculation Example

Figure 4.5 presents an example of a wire model made up of three lumped RC segments. The wire is driven through an upstream resistance R_{up} . The end of the wire is connected to an RC load represented by R_x and C_x . In this example, the router is expanding only along the wire made up of segments 1 through 3 and it is *not aware* of the RC load located at the end of the wire.

The delay from the source to each of the three labeled nodes can be determined using an Elmore tree calculation:

$$\text{Delay to node 1: } \tau_1 = (R_{up} + R_1)(C_1 + C_2 + C_3)$$

$$\text{Delay to node 2: } \tau_2 = (R_{up} + R_1)C_1 + (R_{up} + R_1 + R_2)(C_2 + C_3)$$

$$\text{Delay to node 3: } \tau_3 = (R_{up} + R_1)C_1 + (R_{up} + R_1 + R_2)C_2 + (R_{up} + R_1 + R_2 + R_3)C_3$$

Rewriting the equation for τ_2 gives us:

$$\begin{aligned} \tau_2 &= (R_{up} + R_1)C_1 + (R_{up} + R_1 + R_2)(C_2 + C_3) \\ &= (R_{up} + R_1)(C_1 + C_2 + C_3) + R_2(C_2 + C_3) \\ &= \tau_1 + R_2(C_2 + C_3) \end{aligned}$$

Similarly, for τ_3 :

$$\begin{aligned}\tau_3 &= (R_{up} + R_1)C_1 + (R_{up} + R_1 + R_2)C_2 + (R_{up} + R_1 + R_2 + R_3)C_3 \\ &= (R_{up} + R_1)C_1 + (R_{up} + R_1 + R_2)(C_2 + C_3) + R_3C_3 \\ &= \tau_2 + R_3C_3\end{aligned}$$

In the above equations, it is common to have sum of all the capacitances starting from the current node to the end of the wire (e.g., $C_1 + C_2 + C_3$ for node 1). This sum is referred to as “**downstream capacitance.**”

It is interesting to look at the wire in terms of segments since each segment represents a sub-wire in the ETM model. Thus the delay to node 1 is effectively the delay of segment 1. This can be rewritten as:

$$\tau_{firstsegment} = (R_{upstream} + R_{segment})C_{downstream} \quad (4)$$

For the remaining sinks on the wire, the delay of the previous sink can be used to form the basis of the delay. This is apparent in the rearrangement of the equations. The delay is calculated using the delay to the previous sink plus the product of the resistance to the current sink and the downstream capacitance. Again in terms of wire segments, the delay of a mid-wire segment is:

$$\tau_{midwiresegment} = \tau_{previoussegment} + R_{segment}C_{downstream} \quad (5)$$

Using these two equations, it is possible to incrementally calculate the delays to the midpoints of an ETM wire. When the router encounters the first segment in an ETM wire, it will calculate the delay using Equation (4). As the router examines the adjacent segments, it will determine the delays using Equation (5).

It is important to note that the above equations only apply once the router is aware it is working on an ETM wire. The concept of downstream capacitance only applies internally to an ETM wire. This is because once an ETM segment is included in the routing solution, then all the

downstream capacitance must be included along with it. However, it is not apparent what will be included in the routing solution from beyond the routing resource nodes which make up the wire. This is why the previously described equations do not include the effect of R_x and C_x ; they are not part of the current wire being examined and so the router cannot assume they will be part of the solution. However, it is important to make sure that if node 4 does become part of the solution, the delay equation will still be correct. According to the Elmore tree calculation the delay of node 4 is:

$$\begin{aligned}\tau_4 &= (R_{up} + R_1)C_1 + (R_{up} + R_1 + R_2)C_2 + (R_{up} + R_1 + R_2 + R_3)C_3 + (R_{up} + R_1 + R_2 + R_3 + R_x)C_x \\ &= \tau_3 + (R_{up} + R_1 + R_2 + R_3 + R_x)C_x\end{aligned}$$

However, since node 4 is not an ETM node, the router will use the conventional non-ETM equation described in section 2.1.3:

$$\tau_4 = \tau_3 + (R_{upstream} + R_x)C_x \quad (6)$$

This equation (6) applies only if $R_{upstream}$ is equal to $R_{up} + R_1 + R_2 + R_3 + R_x$. Fortunately, this is the case, as the value of $R_{upstream}$ is constantly being updated at each stage of an ETM wire even though it is not being used for ETM delay calculation.

Accurate Delay Timing using ETM

In the example presented above, the calculations discussed were “forward” looking estimations. This means that the router was attempting to produce a delay estimate based on the best available knowledge. However, the intermediate delay values calculated using this approach are not always accurate, due to the fact that the full signal routing path is unknown. This can be seen in the previous example where τ_4 is accurate (includes C_x) but τ_1 , τ_2 and τ_3 have not yet been update to include C_x . VPRx resolves this problem by eventually performing a full reverse

traversal of the routing tree for each source after the complete net has been formed. It builds a parallel routing tree which fully encompasses any downstream capacitance at all points in the tree. This allows a more accurate timing analysis which provides intermediate delay numbers throughout the tree. ETM modifications were made to the routing tree construction code in order to make it aware of the additional wiring detail. The extent of the modifications involved adding correct downstream capacitance to nodes modeling early turns.

4.3.2 Distributed Buffering

Once the ETM modifications are made to the routing resource grid, implementation of distributed buffering is the next logical step. The fact that a wire can be modeled using multiple sub-wires makes it possible to insert buffers in-between sub-wires to construct a distributed buffer design.

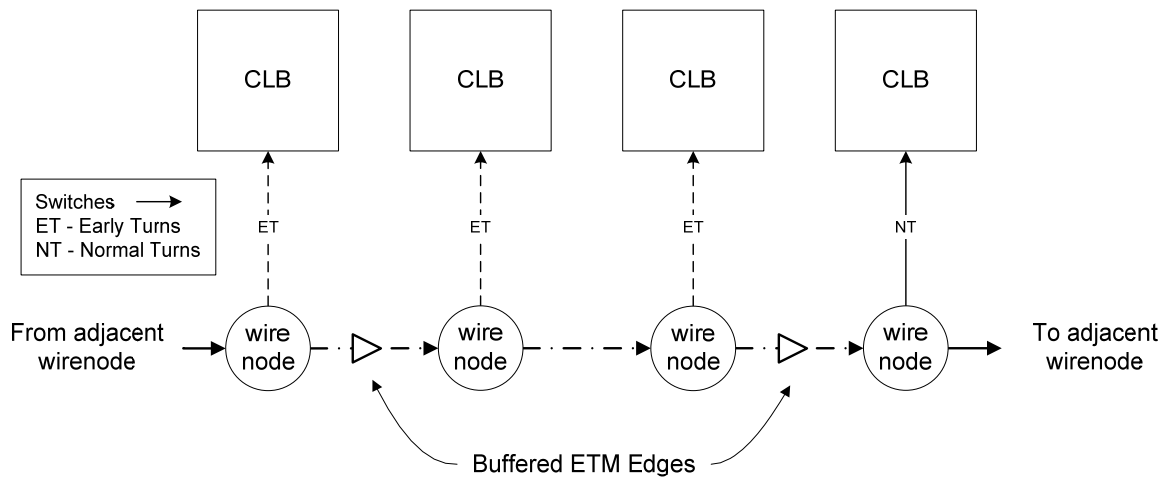


Figure 4.6 - Routing Resource Graph with ETM and Distributed Buffers Shown

One drawback to this approach is that often the prescribed locations for the buffers from the “best” circuit design do not align precisely with the ETM sub-wire segment divisions. This causes the buffer locations of the driver designs modeled in VPRx to be slightly different from

the actual “best” circuit design specifications. The amount of deviation that occurs is related to the architectural wirelength of the wire. Larger architectural wirelengths provide more “resolution” to model a distributed driver. For instance an L16 wire will be composed of 16 ETM wire segments, whereas in comparison, an L4 wire will only be made up of 4 ETM wire segments. Figure 4.7a illustrates how a distributed driver design is implemented in an L16 wire model. Alternatively, Figure 4.7b presents the design implemented in an L4 wire model.

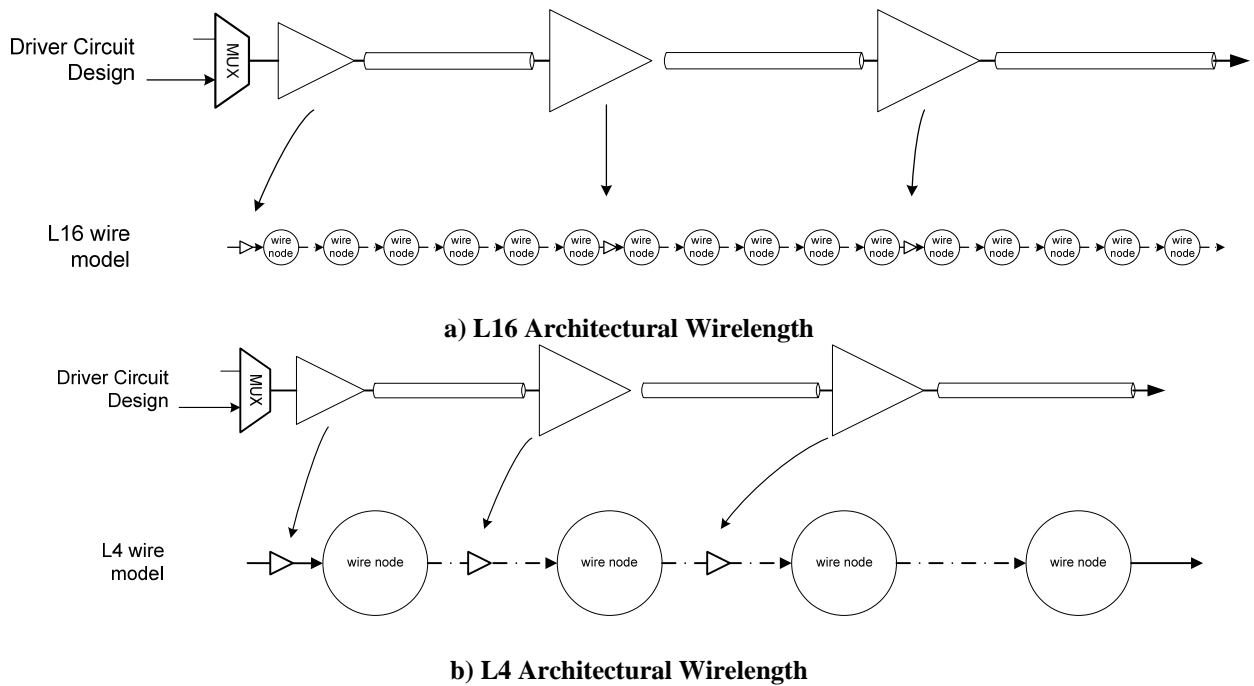


Figure 4.7 - Modeling a Distributed Driver with Different Architectural Wirelengths

Figure 4.8 illustrates the effect that architectural wirelength has on the distributed buffering model. As suggested by the PDP shown in Figure 4.8, larger architectural wirelengths can more closely match the “best” circuit designs found in Chapter 3. However, according to the results from the design space exploration in 3.2.3, small deviations from the “best” design point can be tolerated since overall delay is relatively insensitive to the precise location of downstream buffers.

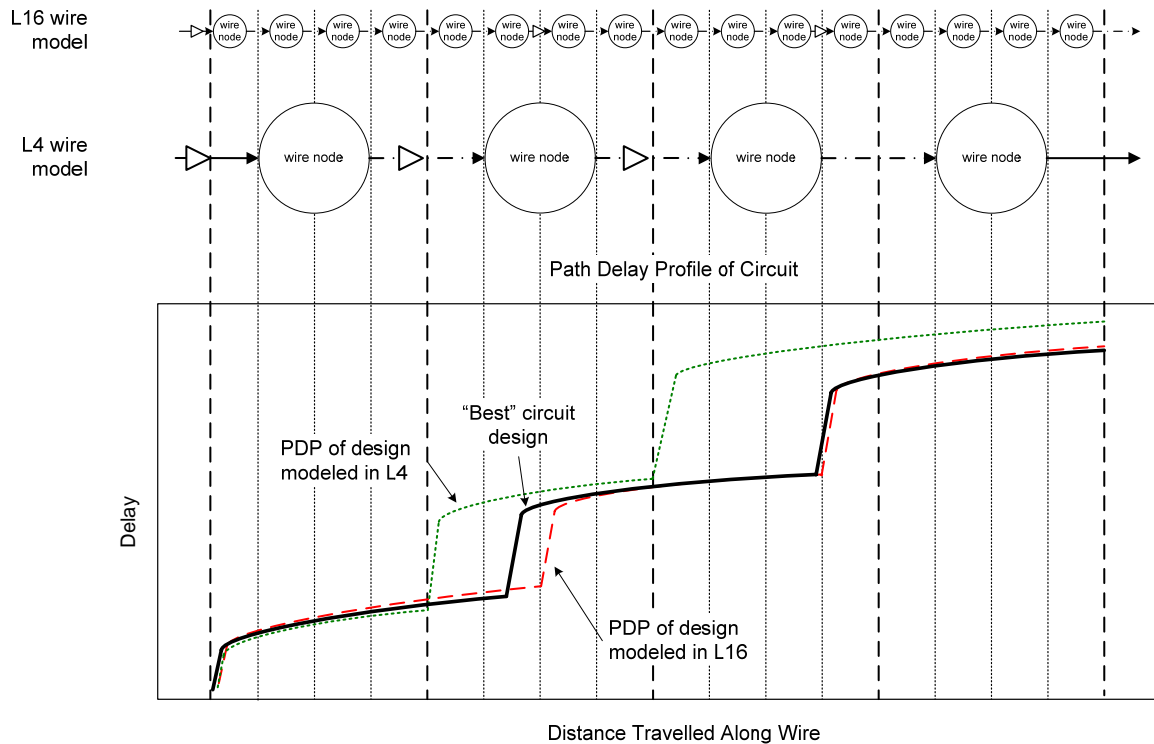


Figure 4.8 - The Effect on the Path Delay Profile of Modeling Circuits with Different Architectural Wirelengths

4.3.3 Fast Paths

The modified VPRx is capable of modeling the fast path used by the new circuit designs. This option enables VPRx to be aware of fast paths and model them accordingly. Recall from Chapter 3 that the fast path is a fast multiplexer bypass designed to aid cross-chip signal performance. It allows a signal which is traveling straight to bypass the wide fan-in multiplexers and take a faster path to the buffer input.

In order to implement this feature in VPRx, the edge in the routing resource grid representing the path of adjacent wires is modified with the smaller delay of the fast path circuits. Figure 4.9 demonstrates how the implementation of the fast path affects the routing resource graph.

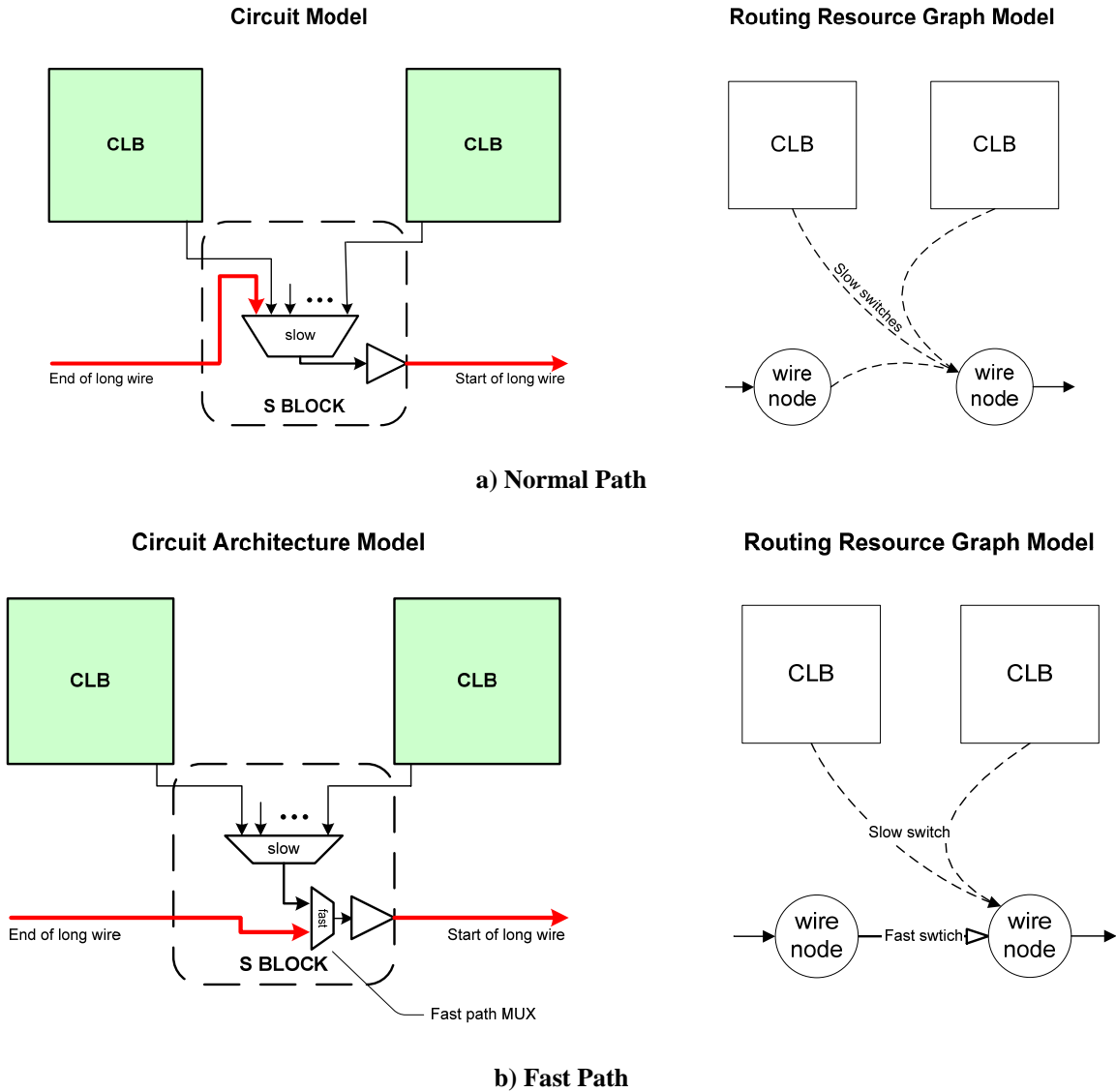


Figure 4.9 - Signal Path through a Switch Block Using Normal or Fast Paths

In Figure 4.9a, all the switches use the multiplexer and hence share the same switch design, indicated as “slow switches”. In Figure 4.9b, two switch types are indicated, a fast switch and a slow switch. Since there can only be one fast edge per switch driver, all remaining edges which represent turns must use the slow multiplexer delay model. Details regarding this delay model are presented in the following section.

4.3.4 Multiplexer Delay Modeling

Chapter 3 introduces a 2-level flat multiplexer design for use with routing switches. In order to model this multiplexer design in VPRx, a new multiplexer modeling scheme is used.

In VPRx, the size (fanin) of the routing multiplexer in each switch is determined automatically. The size may vary depending on architectural parameter such as architectural wirelength or channel width. As the size of the multiplexer changes, so does its delay. The VPRx delay model accounts for these changes and provides a delay estimate for each instance of a multiplexer based on the fanin required by the architecture.

The current VPRx multiplexer delay model assumes a fully encoded multiplexer tree. As shown in Chapter 3, this multiplexer architecture has a delay that is quadratic to the depth of the tree. Unlike the previous design, the 2-level flat design has a constant depth. When fanin increases, additional delay is caused by the junction capacitance of the added passgate devices. Experiments indicate that the delay of the 2-level flat design is actually linearly related to the logarithm of the fanin in the region of interest. Using this relationship, an accurate delay model of this revised multiplexer architecture is achieved. Figure 4.10 presents an example of this model using a 180nm 1x1x process for a 2mm wire driven by a N=2 distributed driver design.

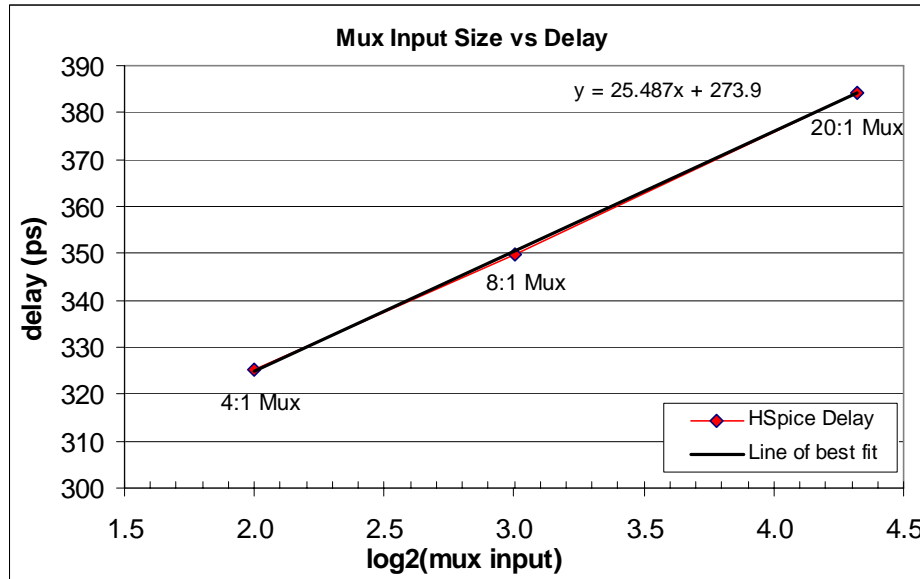


Figure 4.10 - Multiplexer Fanin Delay with a 2mm Wire 180nm 1x1x

4.3.5 Circuit Characterization for VPRx

Up to this point, only wire modeling in VPRx has been described. Since the wire models have known parasitics, characterizing them for VPRx can be achieved by entering their capacitance per micron and the resistance per micron into the architecture file. The next step is to characterize the buffers so that they can be modeled in this framework.

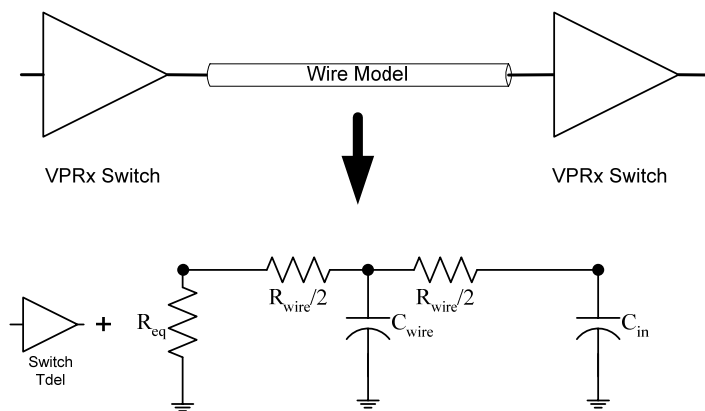


Figure 4.11 - How Switch Circuit Delay is Modeled in VPRx

In VPRx, the general term used to refer to a circuit driving a wire is a **switch**. The circuit can be a single buffer or multiple buffers lumped together used to drive a wire in the routing resource graph. In order to model a switch, a combination of fixed delays and equivalent resistances is used. A switch has several characteristics relevant to this discussion:

- Tdel: Intrinsic switch delay
- R_{eq}: Equivalent Output Resistance
- C_{in}: Input Capacitance

Each of these characteristics is used by VPRx to calculate the delay of the signal. Figure 4.11 shows how these characteristics are used to model a switch using an RC circuit. The delay is calculated using the following equation:

$$Delay = Tdel + \left(R_{eq} + \frac{R_{wire}}{2} \right) \times C_{wire} + \left(R_{eq} + \frac{R_{wire}}{2} + \frac{R_{wire}}{2} \right) \times C_{in} \quad (7)$$

Each of the parameters in the above equation are determined for use with VPRx as follows:

Wire Parasitics (R_{wire}, C_{wire})

The wire parasitics are obtained from process geometries and technology documents. Details are provided in Appendix A.

Intrinsic Buffer Delay (T_{del})

T_{del} is also known as the intrinsic delay of the driver circuit. It is effectively the delay of the driver without a load. Note that this value includes the effect of the output capacitance of the output buffer. Using HSPICE, the value of T_{del} can be measured through simulation.

Input Capacitance (C_{in})

The input capacitance is process specific. It is characterized using an HSPICE simulation which determines input capacitance for a variety of gate sizes.

Equivalent Resistance (R_{eq})

Unlike the aforementioned characteristics, the output resistance is not measured directly by HSPICE. Instead, it is used as a model fitting parameter to correlate HSPICE with VPRx delay calculations. Hence, the concept of an equivalent resistance is only used to model timing and does not represent a physical value. Calculation of the output resistance begins with a delay value obtained from an HSPICE simulation. Substituting this value into equation (7) yields an equation which has one unknown: R_{eq} . By solving for the equivalent resistance the value used to represent the drive strength of the buffer is obtained.

The above approach provides initial values for a VPRx architecture file. In the following section, an iterative calibration loop used to tune the architecture file is described. In this process the initial architecture file is used by VPRx to generate calibration data. This data is subsequently used to further tune the accuracy of the system.

VPRx Circuit Model Calibration Results

In order to verify that the architecture files used by VPRx accurately represent HSPICE delays, a custom testbench circuit shown in Figure 4.12 was constructed. The delays at several

points in this path were computed by VPRx to construct a PDP. This VPRx-PDP was compared to a similar one generated using HSPICE.

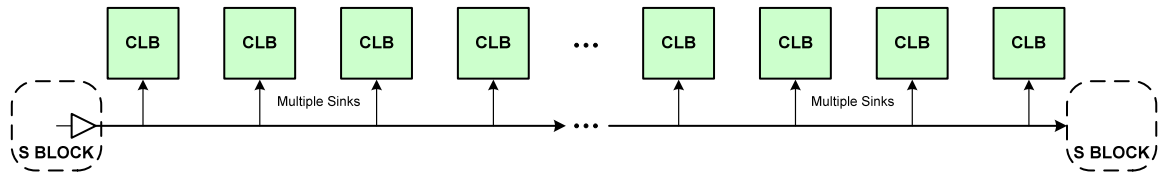
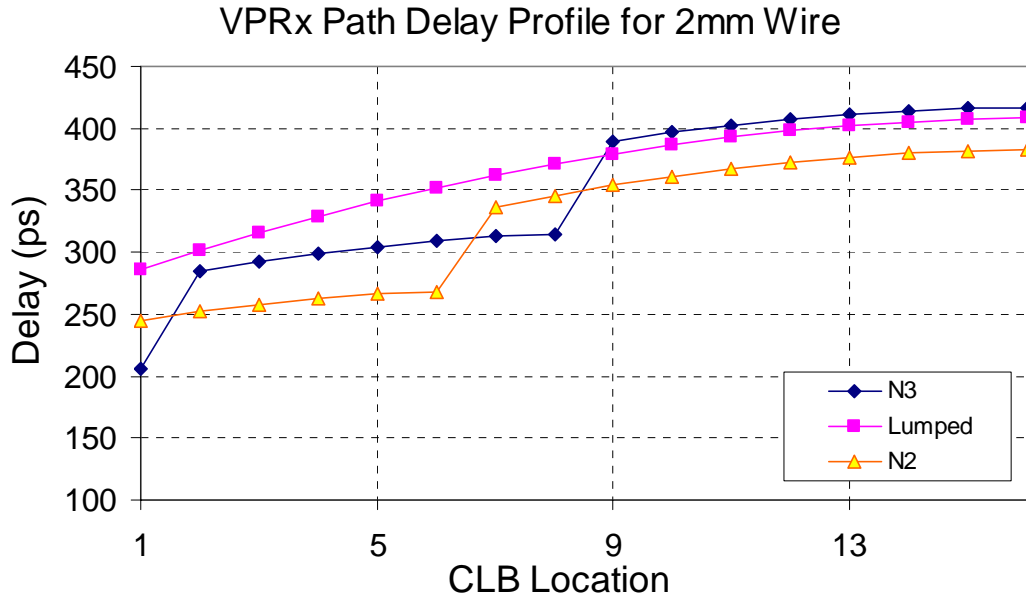
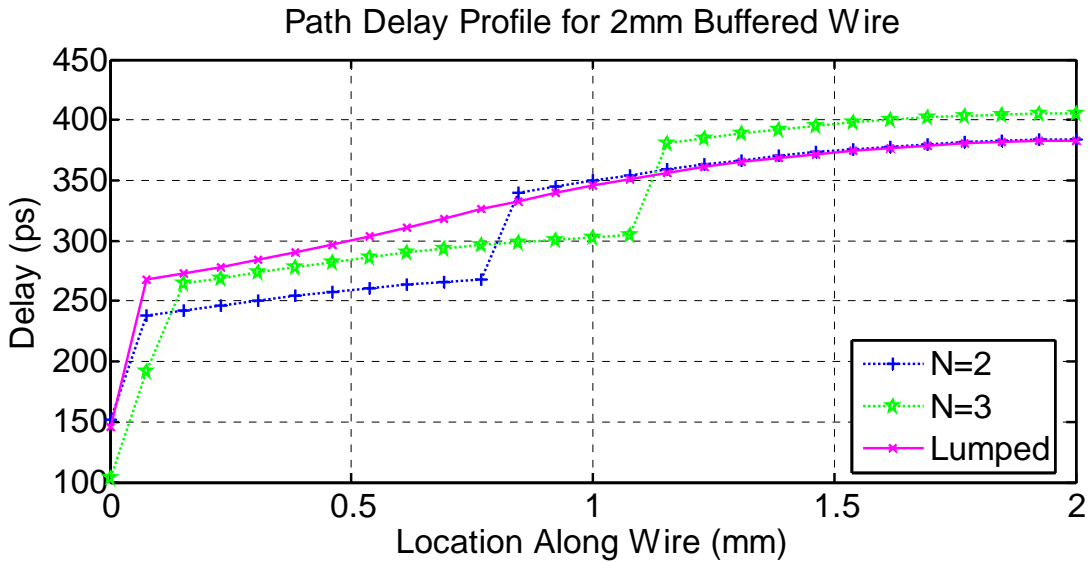


Figure 4.12 - VPRx Path Delay Profile Extraction

Since the VPRx model is much simpler than the HSPICE model, it is expected that there will be differences in the outputs. Although absolute delay values can differ, fidelity must be ensured by preserving relative values. This means that end-to-end delays must be ranked accurately according to the original HSPICE PDP. Also, the midpoint rankings must behave similarly. The delay values of the beginning and end of each step in the PDP are examined and compared. This ensures that the relative shapes of the PDPs are consistent with one another.



a) VPRx Generated PDP



b) HSPICE Generated PDP

Figure 4.13 - Path Delay Profiles for 2mm Wire in 180nm 1x1x

Any major discrepancies between the two PDPs are resolved by tuning the parameters such as Tdel and Req in the architecture file. Each time the architecture file is adjusted, a new VPRx PDP is generated and this process is repeated.

Figure 4.13 compares a calibrated VPRx PDP with the HSPICE equivalent for a 2mm wire in a 180nm technology using 1x1x wires. Although the overall shape and ranking of the two curves is preserved, one can ascertain that there are certain differences in the output. The most notable one is the change in the delay of the lumped design. It appears that the Lumped delay is larger in the VPRx-PDP. One reason for this is that the model in VPRx contains architectural knowledge that the HSPICE model is missing. In VPRx, the effect of switch drivers which tap off the wire is added to the wire capacitance. Since the HSPICE model was constructed before the architectural length was specified, it does not model these additional loads. As expected, the VPRx PDP delays increase as a result of these loads. Notice however, that the increase is larger for the lumped design since all the additional wire load is added to the output of one buffer. In the distributed design, the additional load is spread among multiple buffers. This demonstrates another advantage of distributed designs.

4.4 Experimental Results

Using the modified VPRx developed in the previous section, 20 MCNC benchmark circuits are placed and routed several times using different combinations of modeling configurations. The different modeling configurations are described in the experiments subsection, but first the routing switch designs which are being modeled will be presented.

4.4.1 Driver Designs Modeled in VPRx

Three physical interconnect lengths were chosen to be modeled in VPRx. For each length, an optimal buffer sizing and spacing was determined according to Chapter 3.

The first length chosen is 2.0mm. This was selected because the 180nm design results from Chapter 3 suggested that 2.0mm designs are the crossover point where distributed buffer designs

demonstrate potential improvements over lumped buffers. In modern FPGAs, such as Altera's Stratix which is implemented in 130nm, long wires are approximately 3mm in length [35]. Furthermore, the PDPs of Chapter 3 indicate that distributed buffering begins to offer improvements in endpoint delay as well as midpoint delays at 3.0mm. For these reasons, a 3.0mm wire is considered.

The third wirelength which is examined is the 0.5mm design. This length is interesting because it represents the shorter wires (e.g., L4 wires). It also provides a point of comparison with previous work optimized for L4 wires.

Driver design results show that the 0.5mm wirelength is too short to take advantage of distributed buffering. As a result, for 0.5mm only a lumped circuit design will be used. Using the lumped design, benefits due to ETM, the 2-level multiplexer design and use of the fast path are evaluated. For the longer 2.0mm and 3.0mm wirelengths, distributed drivers are compared to lumped driver designs. The driver designs to be evaluated are summarized in Table 4.1 and Table 4.2.

Lumped Driver Designs		
Wirelength (mm)	Number of Stages (N)	Driver Sizes (x min)
0.5	2	3.74, 14
2	3	4, 9, 35
3	3	3.33, 11.1, 37

Table 4.1 - Lumped Driver Designs Used for Experiments

Distributed Driver Designs					
Wirelength (mm)	Number of Stages (N)	Multiplexed Stage		Distributed Stage Buffer-wire Segment	
		Driver Size B0 (x min)	Length L0 (mm)	Buffer Size B1 (x min)	Length L1 (mm)
2	2	17.0	0.80	27.0	1.20
	3	6.0	0.15	22.0	0.93
3	4	11.2	0.36	26.0	0.88

Table 4.2 - Distributed Driver Designs Used for Experiments

Each physical wirelength also corresponds to an architectural wirelength in VPRx. The 0.5mm wire is modeled as an L4 architectural length wire made up of 4 CLB tiles of 125 μ m. This is roughly equivalent to the wirelength in FPT04 [3] which was approximately 0.46mm.

The 2.0mm and 3.0mm designs are modeled as wires with an L16 architectural length. For the 2.0mm design, the CLB tiles of this architecture are assumed to be 125 μ m. However, for the 3.0mm design, the CLB tiles are assumed to be 187.5 μ m long.

4.4.2 Benchmark Logic Circuits

The 20 largest MCNC benchmark circuits were used for each experiment. Although these circuits are large enough to properly utilize the L4 architectural wirelength, they are not large enough to take advantage of L16 wires. To increase the size of the array used, the amount of CLB packing was decreased so as to spread out the circuit design. Circuits used for the L16 experiments are packed such that only 1 LUT per CLB is used. This produces larger array sizes

which are large enough for experimentation with longer wires. Since the purpose of this research is to assess the interconnect circuit design and not the packing efficiency of the clustering tool, this is an acceptable solution.

Table 4.3 presents the resulting array sizes used by each benchmark circuit. Since the arrays are square, only one dimension is listed. Columns 3 and 5 indicate the number of end-to-end wires which would be required to cross the entire array. Despite packing the CLBs to only 1/8 of the maximum logic capacity, it is still difficult to generate array sizes which would utilize the same number of end-to-end wires in the L16 array as there are in the L4 array.

For all experiments using L4 wires, the same placement is used for each MCNC circuit. Similarly, all L16 experiments share identical placements. This is necessary to remove noise generated by different placement solutions. In theory, delay could be further improved by placing each circuit with the additional knowledge provided by the ETM, but this is not explored in this work.

Architectural Wirelengths of Driver Circuit Designs				
Benchmark Circuits	L4 (0.5mm)		L16 (2.0 & 3.0mm)	
	Array Size	Number of L4 wires required to cross array	Array Size	Number of L16 wires required to cross array
alu4	14	3.5	40	2.5
apex2	16	4.0	44	2.8
apex4	13	3.3	36	2.3
bigkey	18	4.5	42	2.6
clma	33	8.3	92	5.8
des	21	5.3	40	2.5
diffeq	14	3.5	39	2.4
dsip	18	4.5	38	2.4
elliptic	22	5.5	61	3.8
ex1010	25	6.3	68	4.3
ex5p	12	3.0	33	2.1
frisc	22	5.5	60	3.8
misex3	14	3.5	38	2.4
pdc	25	6.3	68	4.3
s298	16	4.0	44	2.8
s38417	29	7.3	81	5.1
s38584.1	29	7.3	81	5.1
seq	15	3.8	42	2.6
spla	22	5.5	61	3.8
tseng	12	3.0	33	2.1

Table 4.3 - Benchmark Circuit Array Sizes

4.4.3 Experiments

Experiments can be divided into two groups: Lumped experiments and distributed experiments. Table 4.4 presents the different experiments performed.

Lumped experiments are performed on lumped circuit designs for each wirelength. These experiments demonstrate the incremental benefit of adding ETM and the fast path.

The distributed experiments apply to the longer wire circuits (2.0mm and 3.0mm) which have distributed circuit designs. These experiments demonstrate the benefits of distributed buffering.

Note that the distributed designs can only be evaluated when ETM is enabled.

Overview of VPRx Experiments							
Circuit Architecture	Lumped					Distributed	
Experiment	FPT04 Baseline	Lumped	Lumped +Fast	Lumped +ETM	Lumped +ETM +Fast	Distributed +ETM	Distributed +ETM +Fast
Wirelengths (mm)	0.5	0.5	0.5	0.5	0.5	-	-
	2.0	2.0	2.0	2.0	2.0	2.0	2.0
	3.0	3.0	3.0	3.0	3.0	3.0	3.0
Early Turn Modeling (ETM)	No	No	No	Yes	Yes	Yes	Yes
Distributed Buffering	No	No	No	No	No	Yes	Yes
Fast Path	No	No	Yes	No	Yes	No	Yes

Table 4.4 - Overview of Experiments

Table 4.4 also shows the wirelengths which apply to each type of experiment. The column indicating ‘baseline’ experiments is unique in that it uses the exact same driver design for 0.5mm, 2.0mm and 3.0mm wirelengths. This design was presented in FPT04 [3] and is optimized for an L4 wire of approximately 0.5mm in length. Although not optimized for 2.0mm or 3.0mm wires, the same buffer sizes will be used as a baseline. Hence, the results will also demonstrate the extent of performance loss that can occur if a circuit is not appropriately designed for the wirelength. In all other cases the driver design is chosen as the “best” possible for the given wirelength according to Table 4.1 and Table 4.2.

4.4.4 Critical Path Delay Results

Critical path delay is used as the performance metric used in each experiment. Table 4.5 presents the average critical path delay results for the 20 MCNC benchmark circuits. All values are normalized to the baseline design. Absolute delays for each circuit are provided in Appendix B.

Normalized Critical Path Delay Results							
Circuit Architecture	Lumped					Distributed	
Experiment	FPT04 Baseline	Lumped	Lumped +Fast	Lumped +ETM	Lumped +ETM +Fast	Distributed +ETM	Distributed +ETM +Fast
0.5mm (L4)*	1.0 (20ns)	0.90	0.82	0.88	0.81 (16.2ns)	-	-
2.0mm (L16)*	1.0 (31ns)	0.73	0.70	0.69	0.65	0.67 (N=2) 0.67 (N=3)	0.63 (N=2) (19.5ns)
3.0mm (L16)*	1.0 (38ns)	0.70	0.67	0.63	0.60	0.56 (N=4)	0.54 (N=4) (20.5ns)

* L4 packs CLBs until full, L16 packs only 1 LUT per CLB to spread out the circuit over a larger array

Table 4.5 - Critical Path Results

Each column in Table 4.5 provides insights on the benefits of each circuit feature combination. Below, the change in critical path caused by the proposed circuits, ETM, addition of the fast path and distributed buffering is discussed.

One important limitation to note is that the results in this study are based on an FPGA architecture with a single wirelength in the routing architecture. This restriction arises from the fact that VPRx does not support heterogeneous routing resource wirelengths.

Improved Circuit Design Results

The change from the FPT04 design to the lumped circuit design involves two differences: the introduction of the 2-level flat multiplexer and buffer resizing to optimize delay. Results from the 0.5mm wirelength demonstrate that the new lumped circuit design produces about 10% improvement in critical path delay. Most of this improvement is due to the multiplexer design. In [31], a similar multiplexer design is cited to improve delay by 5%.

The 2.0 and 3.0mm results indicate larger gains of up to 30%. This is mostly because the FPT04 circuit was not optimized for such long wirelengths. This stresses the importance of using properly designed circuits for different wirelengths.

Early Turn Modeling Results

The benefits of ETM can be observed by comparing the results in the Lumped column with the Lumped+ETM column. Addition of ETM shows minor improvements of 2.2% (0.02 points), for the L4 architectural wirelength. ETM produces larger improvements from 5.4% (0.04 points) up to 10% (0.07 points) on the longer L16 architectural wirelength wires.

These results appear reasonable as the L4 architecture provides only three ETM nodes for each wire. In comparison, the L16 architecture provides better detail and more opportunities to take advantage of early turns by adding 15 ETM nodes for each wire. Furthermore, these results are consistent with the idea that longer wires benefit more from ETM due to the larger error which occurs with longer wires.

Fast Path Results

The impact of the fast path is determined by adding the Fast Path option to the Lumped and Lumped+ETM experiments. Results show a consistent improvement in critical path delay ranging from 8.9% (0.08 points) on the L4 0.5mm wire to 4.1% (0.03 points) for the longer L16 2.0 and 3.0mm wires. Intuitively, adding a fast path to the architecture will improve delay by an amount related to the number occurrences of the fast path. As described in section 4.4.2, the array size of the L4 wires contains more wires arranged end-to-end and therefore, has a larger occurrence of fast paths. In comparison, the L16 design has fewer instances of the fast path due

to the combination of array size, channel width and track staggering arrangements of an L16 wire.

One other reason for the smaller improvement from fast paths on the longer wires can be attributed to the fact that the delay of the longer wires is large compared to the fast path savings. Since the fast path reduces the delay through the circuit by a constant amount, the relative impact of this improvement is larger for smaller L4 wires with a lower wire delay.

Distributed Buffering Results

Improvements due to distributed buffering are presented in the final two columns of Table 4.5 which are only applicable to the long 2.0mm and 3.0mm wires. The design improvements differ between the 2.0mm wire and the 3.0mm wire due to key differences in the circuit designs.

Compared to the lumped circuit design with ETM enabled, distributed buffering on the 2.0mm wire is able to reduce critical path delay by about 2.8% (0.02 points). Recall from Chapter 3 that the PDP of the 2.0mm driver circuit design shows that the end-to-end delays are the same as the lumped designs delay, but the midpoint delays are improved. This experiment demonstrates that the improvements to midpoint delays can be used by early turn modeling to provide a modest increase in performance.

In the 3.0mm design, the distributed circuit offers improved end-to-end delays in addition to reduced midpoint delays. The overall result is a larger reduction in delay just over 11% (0.07 points). This demonstrates that distributed buffering can improve overall critical path delay, but not substantially until the wire is long enough.

Combined Gains

When all the improvements are combined, the cumulative gains can be substantial. For the 3.0mm wire, distributed buffering combined with the fast path produces a 46% improvement over the original FPT04 design driving an equivalently long wire. Since the FPT04 circuit design is not optimized for a 3.0mm wire, it makes sense to compare the final distributed design with the Lumped design for 3.0mm. The results for this comparison show that a distributed circuit design can still offer a 16 percentage point or 23% improvement when combined with the fast path. A breakdown of the improvements on a per-circuit basis are provided in Figures 4.14 to 4.16.

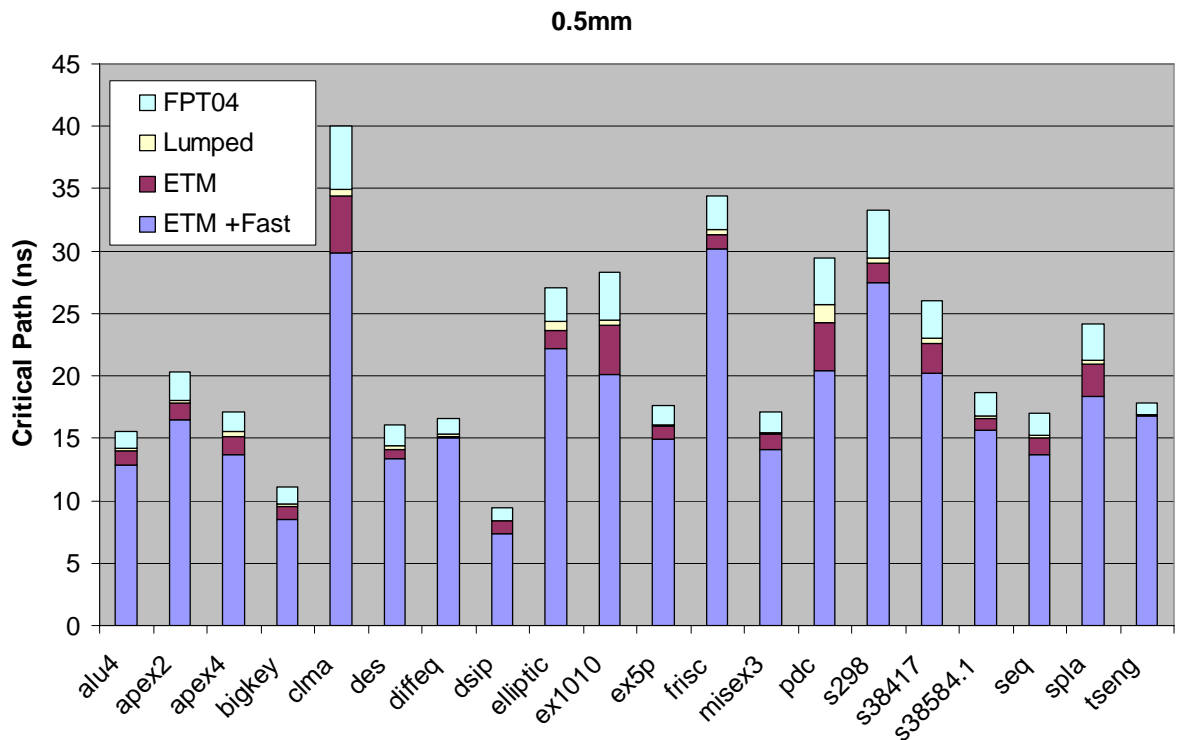


Figure 4.14 - Delay Breakdown for 0.5mm Wire

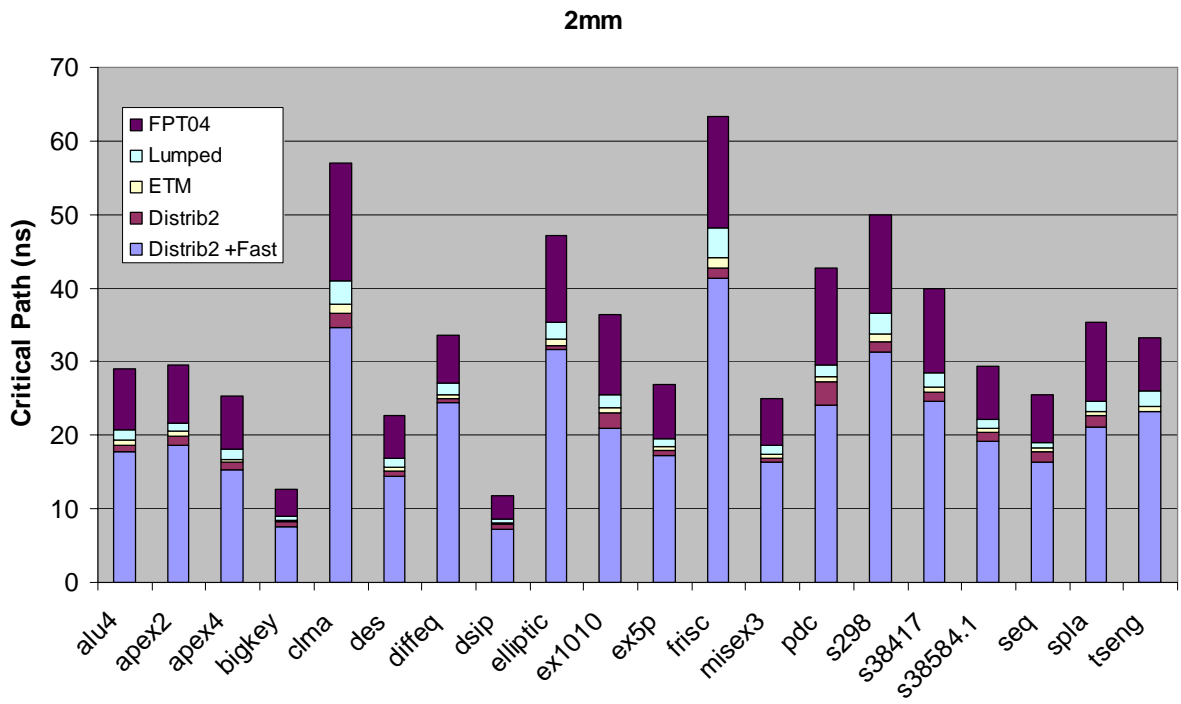


Figure 4.15 - Delay Breakdown for 2.0mm Wire

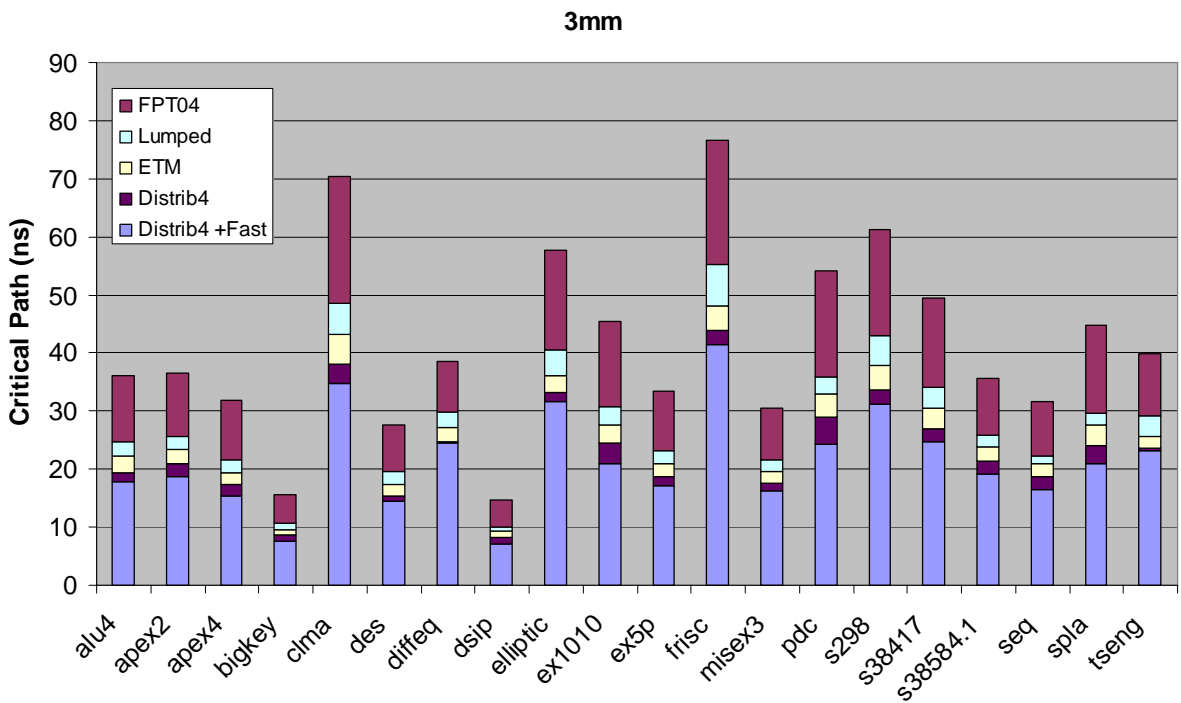


Figure 4.16 - Delay Breakdown for 3.0mm Wire

4.4.5 Turn Count Analysis

By keeping track of the number of turns in the final routed solution, it may be possible to better understand the impact of adding early turn modeling, distributed buffering and fast paths to the routing resource graph. Each routed solution has turn data which can be divided into three categories: *Early Turns*, *Normal Turns* and *Straight Throughs*. *Early Turns* represent turns which occur before the end of the wire. *Normal Turns* occur at the end of the wire and *Straight Throughs* represent the cases where the routing does not turn at all³. The sum of these three values is the total number of turns.

In almost all the scenarios which follow, the addition of new features had little effect on the total turn count. This is likely because the number of turns in a routing solution are dominated by the placement and the circuit connectivity relative to the FPGA architecture. Turn counts can also be affected by other factors such as placement and congestion. Although this section attempts to explain some of the trends observed in the results, it is difficult to draw concrete conclusions without further examination of the router operation.

³ Note that when fast paths are enabled, the number of straight throughs represents the number times the fast path is used.

Effect of Adding ETM on Turn Counts				
Designs	Average Total Number of Turns	Average Early Turns	Average Normal Turns	Average Straight Throughs
FPT04 0.5mm	7617	57.0%	26.0%	16.9%
+ ETM	7604	57.7%	26.2%	15.2%
Lumped 0.5mm	7587	57.3%	25.7%	16.0%
+ ETM	7597	58.9%	25.1%	15.0%
FPT04 2.0mm	10986	87.4%	7.0%	5.0%
+ETM	11041	87.3%	7.5%	4.4%
Lumped 2.0mm	10978	87.6%	6.9%	4.9%
+ETM	11029	89.0%	6.4%	3.7%
Lumped 3.0mm	10983	87.5%	6.9%	5.0%
+ETM	11046	88.8%	6.6%	3.8%

Table 4.6 - Turn Count Changes Due to Addition of ETM

Effect of Early Turn Modeling on Turn Counts

Table 4.6 presents the effects of ETM on turn counts for the three lumped circuit designs. For each circuit, two rows of data are shown. The first row provides the turn count data with no features enabled. The second row, indicated by +ETM, provides the turn data for circuits routed with ETM enabled.

The table shows that the addition of ETM yielded slight increases in early turns for all cases except the FPT04 design running on long wires. The fact that the number of early turns increased by a small amount while the critical path decreased suggests that the improvements due to ETM are caused by a combination of changes. The biggest factor is most likely increased modeling accuracy, followed by smarter local choices in routing. An example of smarter routing is given in Figure 4.17. It presents a scenario where two routes are considered equivalent in a non-ETM L4 architectural wirelength model. Route B is slightly faster than route A, but only by virtue of the early turn. It is important to note that although the delay improves, the number of turns does not. This helps to explain why the number of total turns does not change substantially through the addition of ETM.

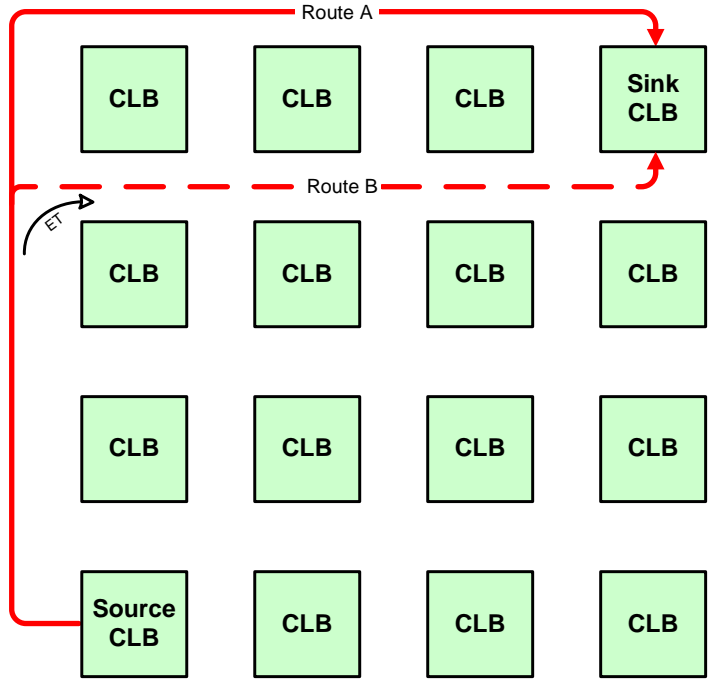


Figure 4.17 - Early Turn Routing Example for L4 Wires

Effect of Adding Fast Paths on Turn Counts				
Designs	Average Total Number of Turns	Average Early Turns	Average Normal Turns	Average Straight Throughs
Lumped 0.5mm	7587	57.3%	25.7%	16.0%
+ Fast	7591	57.3%	24.2%	17.5%
+ ETM & Fast	7698	59.2%	20.8%	18.9%
Lumped 2.0mm	10978	87.6%	6.9%	4.9%
+ Fast	10908	88.4%	6.4%	4.5%
+ ETM & Fast	11057	88.4%	5.1%	5.3%
Lumped 3.0mm	10983	87.5%	6.9%	5.0%
+ Fast	10913	88.3%	6.5%	4.5%
+ ETM & Fast	11073	88.2%	5.2%	5.5%

Table 4.7 - Turn Count Changes Due to Addition of Fast Paths

Effect of Fast Paths on Turn Counts

Table 4.7 presents the effects of fast paths on turn counts for the three lumped circuit designs. For each circuit, three rows of data are shown. As in the previous table, the first row is the turn count data with no features enabled. The +Fast row indicates the changes in turn counts caused

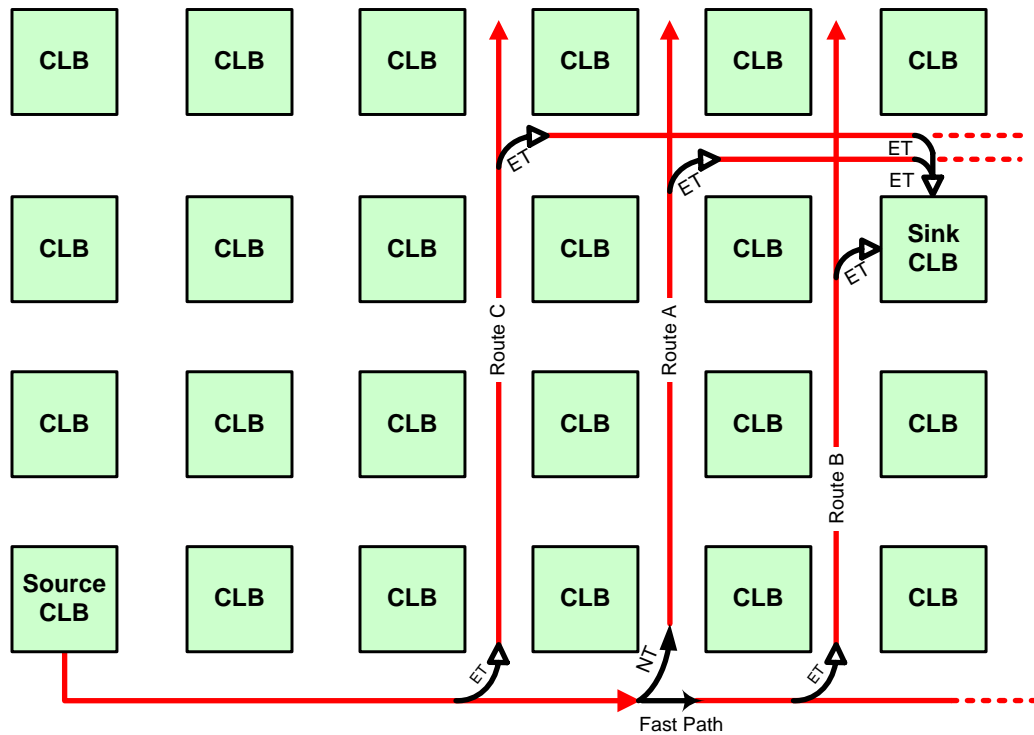
by only enabling fast path modeling. The third row, indicated by “+ETM & Fast”, provides the turn data for circuits routed with both Fast Paths and ETM enabled.

Intuitively, one would expect that the introduction of fast paths would increase the number of straight throughs. According to Table 4.7, it appears that this is exactly what is occurring in the L4 0.5mm wire. Adding the fast path increased average straight throughs, while the average normal turns decreased slightly. This suggests that normal turns are being replaced at the junctions with straight throughs. A simple example here would be a staircase route turning into an L shaped route.

In the case of the longer wires, the results are counter-intuitive. The average number of straight throughs and normal turns decrease, while the average number of early turns increases. For these longer wires, it appears that the lower cost of straight throughs causes more early turns to occur.

Once ETM is enabled, the results are different. The introduction of fast paths causes a definite increase in the number of fast paths used. A specific example of this is the case where the combination of a fast path and an early turn provide a routing option which is faster than a normal turn plus an early turn. This is shown in Figure 4.18, where three possible routing choices are illustrated. The table in the figure shows how these routing choices would be ranked depending on what options were enabled in VPRx, assuming there is no congestion. In the standard VPRx, without ETM or fast paths, all the routes would be considered equivalent in timing. If Fast Path is enabled, route B has an advantage because the fast path has a smaller delay than early or normal turns. If both Fast Path and ETM are enabled, the superiority of route B becomes clear through the benefits of early turns. Similarly, as the router becomes aware of the benefits of early turns, it is likely to initially favour route C over A during the routing search.

These examples help illustrate why normal turns are less popular once Fast Paths and/or ETM options are enabled.



Ranking of Routing Choices			
Routing	No ETM No Fast	No ETM + Fast	+ ETM + Fast
	A - "Normal Turn"	1	2
B - "Straight Through"	1	1	1
C - "All Early Turns"	1	2	2

Figure 4.18 - Routing Choices Due to Fast Paths with Early Turns in an L4 Architecture

Effect of Distributed Buffering on Turn Counts

Table 4.8 presents the turn count changes resulting from the addition of distributed driver designs. It is expected that the improved midpoint delays from the distributed driver designs would have promoted the use of early turns, increasing the early turn count even more. Instead, results indicate that the effect of adding distributed buffering does not have any significant effect on the overall turn count distribution. This suggests that the improvements in critical path delay are most likely due to reductions in the early turn delays and not changes to the routing solution.

Effect of Adding Distributed Buffering on Turn Counts				
Designs	Average Total Number of Turns	Average Early Turns	Average Normal Turns	Average Straight Throughs
Lumped 2.0mm +ETM	11029	89.0%	6.4%	3.7%
+ Distributed N2	11034	88.9%	6.5%	3.7%
or Distributed N3	11063	89.1%	6.3%	3.8%
Lumped 3.0mm +ETM	11046	88.8%	6.6%	3.8%
+ Distributed N4	11055	89.1%	6.4%	3.7%

Table 4.8 - Turn Count Changes Due to Addition of Distributed Features

One thing which has not been examined is where the early turns took place. It is possible that the location of the early turns is influenced by new circuit designs. Ideally, the router would be able to strategically choose the best location for an early turn based on the staggered delay profile of the distributed drive design. Unfortunately, without data indicating the location of early turns, no clear conclusions can be drawn.

4.4.6 Runtime

Although not a major focus of this work, runtime is an important factor which must be considered by all CAD tools. By introducing the detailed circuit models which enlarged the routing resource graph, the complexity and therefore runtime, of the routing problem increased considerably. The amount the routing resource graph grows is largely dependant on the architectural length of the wires being modeled. In this work, architectural length of L4 and L16 are used. For L4 designs, adding ETM increases runtimes by up to 3x. For the L16 designs, runtimes increase considerably, ranging from 3x up to almost 30x depending on the modeling options enabled in the experiment. The largest increases in runtime are observed in experiments involving *only* ETM.

An interesting observation is that all the experiments which had Fast Path enabled had runtimes only 3-16x larger than the original. It appears that providing a routing option as

compelling as the fast path helps to reduce the runtime of the routing process. In the standard configuration without the fast path, the router has to negotiate between three equally slow choices. Introducing the fast path makes the best choice obvious. This allows the router to postpone expansion of slower neighboring nodes enough to reduce the overall routing runtime. This observation is useful because it shows that runtime can be reduced by providing clear choices for the router to pursue.

Chapter 5

Conclusions and Future Work

As the industry moves towards faster clock speeds and smaller devices, the challenge of interconnect delay will always be present. For FPGA designers, this is a significant concern as the wiring demands of programmable interconnects are intense.

In this thesis, an attempt has been made to address the interconnect delay problem by investigating the design of programmable switch drivers for FPGAs. Our resulting circuit designs are based on routing architectures which were recommended by [3]. Prior to [3], FPGA routing architectures used shared wires that were driven from various points throughout the wire. This resulted in all FPGA drivers having tristate capability which restricted driver designs to lumped circuit architectures. In [3], it was shown that implementing directional wires with single-drivers can improve both the delay and area efficiency of an FPGA architecture. This thesis shows that by using directional wiring with single-drivers, it is possible to design circuits which can optimize the interconnect performance on FPGA wires. Optimized circuit designs are generated using a circuit design methodology which is capable of estimating the delay of a circuit design using SPICE generated delay data. The use of this method provides the flexibility and accuracy obtained from a SPICE-level simulator but has the advantage of shorter runtime.

By examining the PDP of the optimized circuits, it can be seen that distributed driver designs can offer more to FPGAs than just improved endpoint delay. In comparison to lumped driver

designs, distributed driver designs can improve early turns which occur before the end of the wire. Using an enhanced version of VPR capable of accurately modeling the new circuits, the performance of several circuit designs were evaluated using standard benchmarks. Results show that early turn improvements alone can reduce delay by a modest amount of about 3%. Overall, the effects of the new circuits are substantial. When the benefits from improved modeling, optimized circuit design and other enhancements such as fast paths and faster multiplexers are combined, reductions in critical path delay by as much as 46% are observed.

By examining the optimized circuit designs, several items which are useful to an FPGA architect are revealed. The first is that distributed buffering only outperforms lumped designs once wires are long enough. Results show that in 180nm technology, wires less than 2.0mm cannot reap the rewards of distributed buffering. The second discovery is that the length of the interconnect has particular influence on the best speed (delay-per-millimeter) at which the wire can transmit a signal. In the case of FPGAs, this means that using longer wires can help to achieve speeds closer to those found in general ASIC interconnect. This information is useful to an FPGA architect as it aids in the selection of wirelengths for long wires.

5.1 Future Work

This work has attempted to lay the preliminary ground work for further research into interconnect optimization for FPGAs. As long as FPGAs continue to use wires, approaches to reduce delay will be welcome. Since this research is mainly divided into two parts, recommendations are grouped into two categories: Circuit design and CAD.

5.1.1 Circuit Design

There are numerous choices involved in the circuit design approach. Some related to circuit design and other related to modeling. The following topics present some suggestions on future work related to the circuit design component of this work.

Advanced circuits

The SPICE simulator allows complex circuits to be simulated with great accuracy. This opens the door to a large variety of circuits which do not have an equivalent Elmore model. Low swing signaling circuits can offer reduced power consumption and higher performance. For noise immunity, one can consider the benefits of differential circuits as well.

Noise Modeling

Throughout this work, the effect of deep-submicron challenges were mentioned, but not directly addressed. Noise from inductance and coupling capacitance can impact performance and functionality of the circuits.

Coupling capacitance is typically modeled using the Miller Coupling Factor. In this work, it is assumed that there are no transitions on surrounding wires. Work done in [26] shows that the Miller Coupling Factor does not affect trends, but it will certainly affect the absolute values of the resulting design.

Similarly, modeling of inductance is recommended. Unfortunately, assessing the amount of inductances will be very difficult without prior knowledge of the IC layout. However, since the effects of worst case inductance are not substantial [9], it might be possible to explore a range of reasonable inductance values.

Process Variation

As feature sizes shrink the effect of process variations can be important. One study which examines the effects of process variations on the buffers insertion problem is [36]. This work is valuable to those considering further investigation of process variation effects on buffer insertion because the results show that the buffer insertion problem is “immune” to process variations [36].

Power and Area Modeling

In the buffer design problem, larger buffers mean more area and more power. In this work, power and area data is omitted although the SPICE based circuit design methodology can produce power data for the circuit designs. Further development of the circuit design methodology could introduce area and power awareness to the design flow.

5.1.2 Future Work for CAD

Area Modeling

Accurate area modeling from VPRx would provide an additional metric for comparison from the new circuit designs.

Heterogeneous Wiring

Since VPRx does not support multiple architectural wirelengths, the results in this work are based on single architectural wirelengths. A more realistic model should include multiple architectural wirelengths as they are present in modern FPGAs.

Detailed Turn Analysis

In this work turn counts were used to justify the importance of midpoint delays and to better understand the effects of the new circuit designs on the router. Although it is possible to identify

if an early turn occurs, it is not known *where*, on the wire, the early turn took place. Furthermore, since turn counts are computed by tracing individual sinks instead of examining an entire net, they do not encompass the actual utilization of a wire. Turn locations would aid designers by identifying exactly what part of the wire is most susceptible to improvements from a better PDP. Using complete turn data, it might even be possible to construct a PDP which would be ideal for FPGAs. Afterwards, an effort could be made to design a circuit to realize this ideal PDP.

Accurate Delay Lookup for the Router

Incorporating the PDP into VPRx would allow a more accurate method of delay computation instead of using the first-order Elmore model. This would also avoid any quantization errors introduced by modeling distributed buffers with ETM nodes.

Runtime Improvements for VPRx

The runtime of VPRx with ETM on long architectural length wires is very long. The main reason for this is the expansion of the routing resource graph. Any technique to reduce the number of nodes would be beneficial for runtime. One possibility is to join ETM nodes with similar delays. The largest changes in the PDP occur at the buffer locations. By collapsing the intermediate nodes, it will be possible to reduce the runtime complexity of the routing algorithm

Another potential improvement would be to add heterogeneous wire support in VPRx. In this way, a shorter set of wires can be added to the architecture, reducing the amount of long wires in the design. Since ETM is most beneficial for longer wires, additional reductions in runtime could be achieved by disabling ETM for the shorter wirelengths.

References

- [1] R. H. J. M. Otten, "Global Wires: Harmful?," in *ISPD '98: Proceedings of the 1998 international symposium on Physical design*. Monterey, California, USA: ACM Press, 1998, pp. 104-109.
- [2] M. Sheng and J. Rose, "Mixing Buffers and Pass Transistors in FPGA Routing Structures," in *International Symposium on Field Programmable Gate Arrays*. Monterey, California, 2001.
- [3] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and Single-Driver Wiring in FPGA Interconnect," in *IEEE International Conference on Field-Programmable Technology*, 2004, pp. 41-48.
- [4] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*. Boston: Kluwer Academic Publishers, 2004.
- [5] V. Betz and J. Rose, "Circuit Design, Transistor Sizing and Wire Layout of FPGA Interconnect," in *IEEE Custom Integrated Circuits*. San Diego, California, United States, 1999, pp. 171-174.
- [6] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Boston: Kluwer Academic Publishers, 1999.
- [7] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs " in *International Symposium on Field-Programmable Gate Arrays*, 1995.
- [8] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Applied Physics*, pp. 55-63, 1948.
- [9] K. Banerjee and A. Mehrotra, "Analysis of On-Chip Inductance Effects fo Distributed RLC Interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 904-915, 2002.
- [10] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits A Design Perspective*, 2nd ed: Prentice Hall, 2003.
- [11] A. I. Abou-Seido, B. Nowak, and C. Chu, "Fitted Elmore Delay - A Simple and Accurate Interconnect Delay Model," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 691-696, 2002.
- [12] D. A. Hodges, H. G. Jackson, and R. A. Saleh, *Analysis and Design of Digital Integrated Circuits: In Deep Submicron Technology*, 3rd ed: McGraw-Hill, 2004.
- [13] S. Inc., "HSPICE."
- [14] V. Adler and E. G. Friedman, "Repeater Insertion to Reduce Delay and Power in RC Tree Structures," in *Conference on Signals, Systems & Computers*, vol. 45. Pacific Grove, CA, 1997, pp. 607-617.
- [15] A. Nalamalpu and W. Burleson, "Repeater Insertion In Deep Sub-Micron CMOS: Ramp-Based Analytical Model and Placement Sensitivity Analysis," in *IEEE International Symposium on Circuits and Systems*. Geneva, Switzerland, 2000, pp. 766-769.

- [16] T. Sakurai and A. R. Newton, "A Simple Short-Channel MOSFET Model and its Applications to Delay Analysis of Inverters in Series-Connected MOSFETs," in *IEEE International Symposium on Circuits and Systems*. New Orleans LA, 1990, pp. 105-108.
- [17] T. Sakurai and A. R. Newton, "Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas," *IEEE Journal of Solid State Circuits*, vol. 25, pp. 584-594, 1990.
- [18] S. Dhar and M. A. Franklin, "Optimum Buffer Circuits for Driving Long Uniform Lines," *IEEE Journal of Solid State Circuits*, vol. 26, pp. 32-41, 1991.
- [19] H. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*: Addison-Wesley, 1990.
- [20] L. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," in *IEEE International Symposium on Circuits and Systems*. New Orleans, LA, USA, 1990.
- [21] H. B. Bakoglu and J. D. Meindl, "Optimal Interconnection Circuits for VLSI," *IEEE Journal On Electron Devices*, vol. 32, pp. 903-910, 1985.
- [22] S. Srinivasaraghavan and W. Burleson, "Interconnect Effort - A Unification of Repeater Insertion and Logical Effort," in *ISVLSI '03: Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 55.
- [23] V. Adler and E. G. Friedman, "Uniform Repeater Insertion in RC Trees," *IEEE Transactions on Circuits and Systems*, vol. 47, pp. 1515-1524, 2000.
- [24] V. Adler and E. G. Friedman, "Repeater Design to Reduce Delay and Power in Resistive Interconnect," *IEEE Transactions on Circuits and Systems*, vol. 45, pp. 607-617, 1998.
- [25] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze, "Accurate Estimation of Global Buffer Delay Within a Floorplan," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 1140-1147, 2006.
- [26] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater Scaling and Its Impact on CAD," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 451-464, 2004.
- [27] K. Banerjee and A. Mehrotra, "A Power-Optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs," *IEEE Transactions on Electron Devices*, vol. 49, pp. 2001-2007, 2002.
- [28] M. R. Greenstreet and J. Ren, "Surfing Interconnect," in *IEEE International Symposium on Asynchronous Circuits and Systems*, 2006.
- [29] A. Maheshwari and W. Burleson, "Differential Current-Sensing For On-Chip Interconnects," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 1321- 1329, 2004.
- [30] A. Nalamalpu, S. Srinivasan, and W. Burleson, "Boosters for Driving Long Onchip Interconnects - Design Issues, Interconnect Synthesis, and Comparison With Repeaters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 50-62, 2002.
- [31] 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 *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. Monterey, California, USA: ACM Press, 2005.
- [32] N. Mohamed and S. Yvon, "Optimal Methods of Driving Interconnections in VLSI Circuits," in *IEEE International Symposium on Circuits and Systems*, 1992, pp. 21--24.
- [33] G. Lemieux and D. Lewis, "Circuit design of routing switches," in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. Monterey, California, USA: ACM Press, 2002.
- [34] S. Sood, "A Novel Interleaved and Distributed FIFO," in *Electrical and Computer Engineering*, vol. Masters of Applied Science. Vancouver: University of British Columbia, 2005.
- [35] D. Lewis: Altera, private communication, 2005.
- [36] L. Deng and M. D. F. Wong, "Buffer Insertion Under Process Variations for Delay Minimization," in *IEEE/ACM International conference on Computer-Aided Design*. San Jose, CA 2005, pp. 317-321.

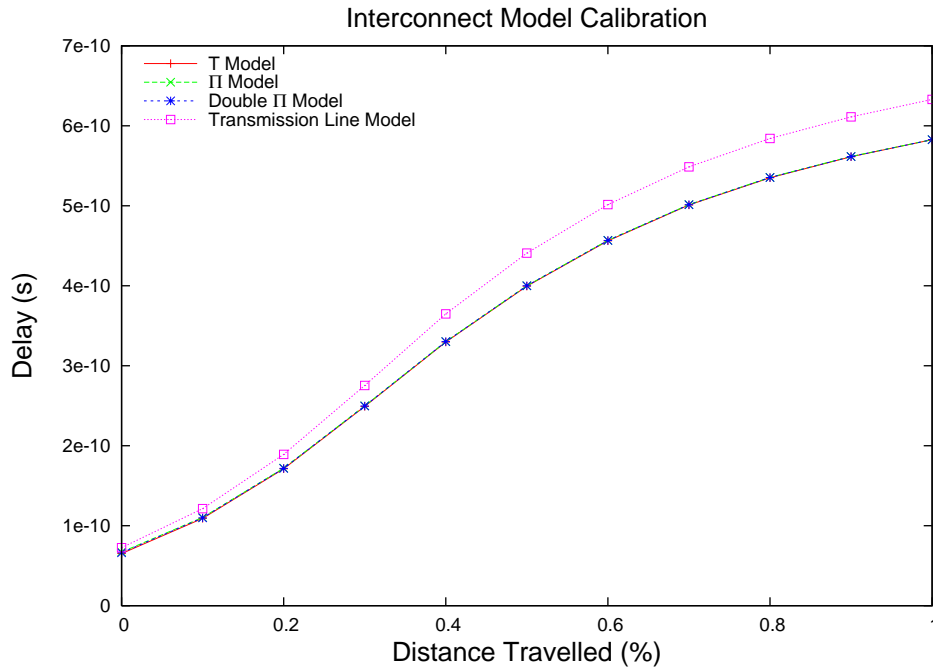
Appendix A - Wire Models

The purpose of this section is to demonstrate how parasitic parameters for wire models can be obtained for use with HSPICE, and how these values can affect delay. Typically, parasitic of interconnects are provided by the foundry, however these documents are not always available to researchers. Fortunately, it is possible to use the physical geometries of the interconnect to determine the wire resistance and parasitic capacitance of interconnect.

Using the HSPICE 2D field solver it is possible to build⁴ a transmission line model of an interconnect using data provided by the foundry, such as dielectric values, wire dimensions, spacing and geometries, and metal conductivity. The field solved transmission line is then used to generate a path delay profile for a simple driver design. Similar PDPs are generated using T-models, π -models and double- π models. The capacitance values of these models are determined by adjusting them until the PDP of the transmission line matches the PDP of the π models. Wire resistance is a straightforward calculation using conductivity and wire geometries. An example of the PDPs is shown below.

Through comparison with known data, this method was shown to be an acceptable method to determine interconnect parasitics.

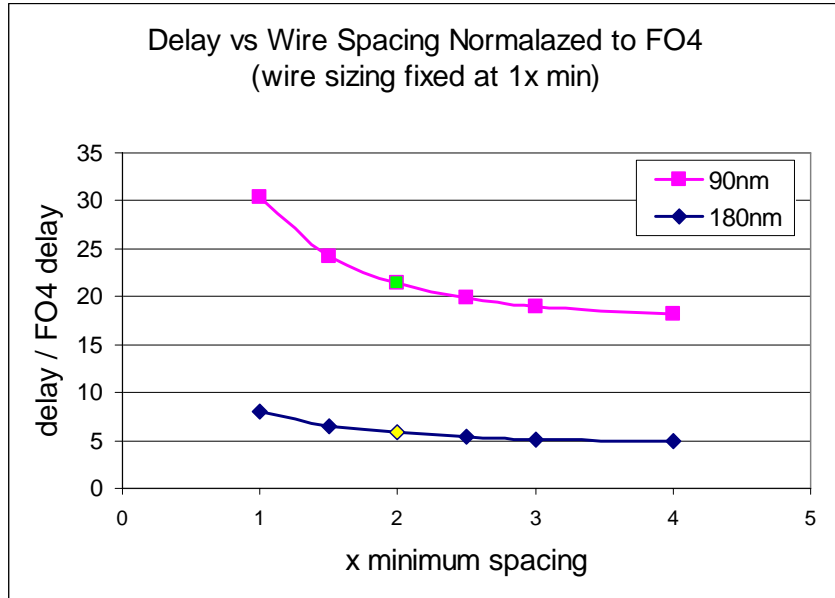
⁴ Details of the field solving technique can be found in the SOC CAD document “Interconnect Modeling in Spice.doc”.



Effects of Spacing and Wire Sizing

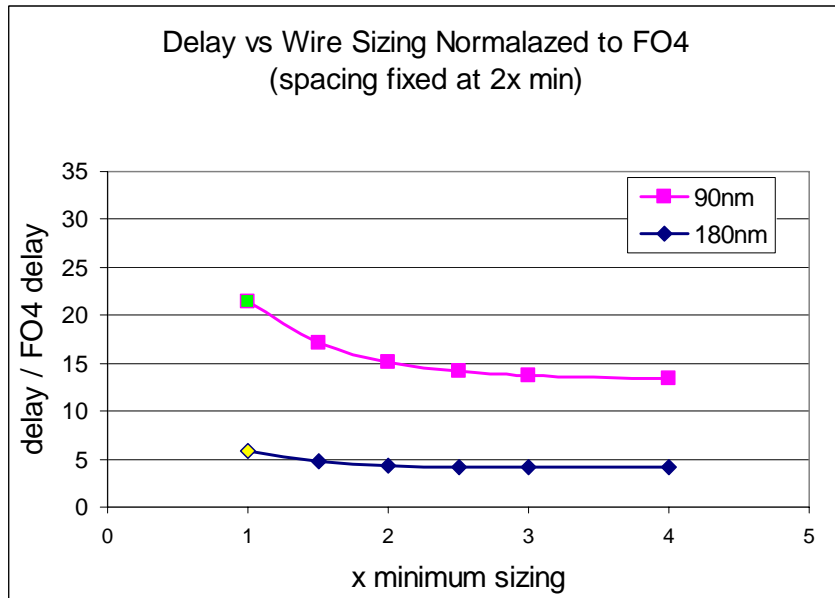
In addition to performing the above characterization, the effects that wire spacing and sizing have on delay were briefly examined. The trends are predictable, but results are useful to guide the selection of sizes when considering delay. The following simulations were performed on a 20x buffer driving a 4mm wire

Spacing



Sizing

The following wire sizing data was obtained by fixing the spacing at 2x from the above data.



Spacing and Wire Sizing can be used to achieve improvements of 60% end-to-end (going from 1x spacing 1x sizing to 2x and 2x, respectively).

Appendix B - VPRx Results

Critical Path Results from VPRx

0.5 mm L4 Critical Path Delay (ns)						
Benchmark	FPT04	FPT04 +ETM	Lumped	Lumped +Fast	Lumped +ETM	Lumped +ETM +Fast
alu4	15.6	15.1	14.2	13.3	14.0	12.9
apex2	20.3	19.5	18.1	16.5	17.8	16.5
apex4	17.1	16.4	15.5	13.8	15.2	13.7
bigkey	11.1	10.4	9.7	8.2	9.6	8.5
clma	40.0	37.9	35.0	29.8	34.4	29.9
des	16.1	15.1	14.4	13.6	14.1	13.4
diffeq	16.6	16.6	15.3	15.1	15.2	15.1
dsip	9.4	9.0	8.4	7.6	8.4	7.3
elliptic	27.0	25.3	24.4	21.9	23.6	22.2
ex1010	28.3	26.6	24.5	20.3	24.1	20.1
ex5p	17.6	17.0	16.1	14.9	15.9	14.9
frisc	34.5	32.7	31.8	30.8	31.4	30.2
misex3	17.1	16.4	15.5	14.1	15.3	14.1
pdc	29.4	26.7	25.7	22.2	24.3	20.4
s298	33.2	30.7	29.4	27.9	29.0	27.5
s38417	26.0	25.0	23.0	20.1	22.6	20.3
s38584.1	18.6	17.7	16.8	15.3	16.5	15.6
seq	17.0	16.6	15.2	14.1	15.0	13.7
spla	24.1	22.8	21.3	18.6	21.0	18.3
tseng	17.2	16.7	16.4	16.4	16.2	16.8

2.0mm L16 Critical Path Delay (ns)

Benchmark	FPT04	FPT04 +ETM	Lumped	Lumped +Fast	Lumped +ETM	Lumped +ETM +Fast	Distrib N2 (+ETM)	Distrib N3 (+ETM)	Distrib N2 (+ETM) +Fast
alu4	29.1	26.6	20.7	19.7	19.4	18.3	18.7	18.8	17.8
apex2	29.6	27.4	21.7	20.6	20.5	19.3	19.9	20.0	18.6
apex4	25.4	23.3	18.1	17.2	16.8	16.0	16.3	16.3	15.4
bigkey	12.6	11.8	9.0	8.3	8.4	7.8	8.2	8.1	7.5
clma	57.0	51.8	41.0	39.1	37.9	36.0	36.6	37.0	34.7
des	22.7	21.1	16.9	16.2	15.7	15.1	15.1	15.1	14.5
diffeq	33.6	31.8	27.1	26.7	25.5	25.1	24.9	24.5	24.5
dsip	11.7	10.7	8.5	8.0	8.2	7.5	7.9	8.0	7.2
elliptic	47.1	42.5	35.3	34.7	33.1	32.5	32.2	32.1	31.6
ex1010	36.3	33.5	25.4	23.1	23.7	21.6	23.0	23.0	20.9
ex5p	27.0	25.2	19.6	19.0	18.4	17.8	17.9	17.9	17.2
frisc	63.3	57.3	48.2	46.8	44.1	42.7	42.8	42.7	41.3
misex3	25.1	23.1	18.6	18.0	17.5	16.9	16.9	16.9	16.3
pdc	42.7	39.5	29.5	27.1	28.0	25.1	27.2	27.0	24.2
s298	50.0	45.9	36.5	35.3	33.7	32.4	32.7	32.5	31.3
s38417	39.9	36.5	28.6	27.0	26.6	25.5	25.9	26.0	24.6
s38584.1	29.4	27.7	22.2	21.5	21.0	19.8	20.4	20.3	19.2
seq	25.5	24.4	19.1	17.8	18.3	17.0	17.7	17.7	16.4
spla	35.4	32.7	24.5	22.8	23.2	21.7	22.6	22.7	21.0
tseng	33.3	30.2	25.9	26.0	24.0	24.0	23.3	23.2	23.2

3.0mm L16 Critical Path Delay (ns)							
Benchmark	FPT04	Lumped	Lumped +Fast	Lumped +ETM	Lumped +ETM +Fast	Distrib N4 (+ETM)	Distrib N4 (+ETM) +Fast
alu4	36.0	24.7	23.4	22.2	21.1	19.4	18.6
apex2	36.6	25.6	24.2	23.4	21.9	20.9	19.7
apex4	31.8	21.6	20.6	19.3	18.4	17.3	16.3
bigkey	15.5	10.7	9.9	9.6	8.9	8.6	8.0
clma	70.4	48.6	46.2	43.2	40.9	38.1	36.2
des	27.5	19.7	18.9	17.5	16.7	15.4	14.8
diffeq	38.6	29.9	29.4	27.1	26.6	24.8	24.4
dsip	14.7	10.0	9.5	9.3	8.6	8.3	7.6
elliptic	57.8	40.6	39.0	36.2	35.4	33.2	32.5
ex1010	45.5	30.7	28.2	27.6	25.1	24.5	22.4
ex5p	33.4	23.2	22.5	21.0	20.2	18.6	18.0
frisc	76.6	55.4	53.6	48.2	46.4	44.0	42.3
misex3	30.5	21.7	20.9	19.7	19.0	17.6	17.0
pdc	54.1	36.0	33.0	33.0	29.6	28.9	26.1
s298	61.4	43.0	41.4	37.8	36.4	33.7	32.5
s38417	49.5	34.0	31.9	30.6	29.0	27.1	25.8
s38584.1	35.6	25.7	25.0	23.8	22.3	21.3	20.1
seq	31.6	22.2	20.7	20.9	19.3	18.7	17.5
spla	44.7	29.7	27.1	27.7	25.4	24.2	22.5
tseng	39.9	29.2	29.2	25.7	25.7	23.5	23.5