Analytical Models for Accelerating FPGA Architecture Development

by

Joydip Das

B.Sc., Bangladesh University of Engineering and Technology, 1996

M.Sc., The University of Louisiana at Lafayette, 2006

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

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Electrical and Computer Engineering)

The University Of British Columbia

(Vancouver)

October 2012

Abstract

Field-Programmable Gate Arrays (FPGAs) are widely used to implement logic without going through an expensive fabrication process. Current-generation FPGAs still suffer from area and power overheads, making them unsuitable for mainstream adoption for large volume systems. FPGA companies constantly design new architectures to provide higher density, lower power consumption, and faster implementation. An experimental approach is typically followed for new architecture design, which is a very slow and computationally expensive process. This dissertation presents an alternate faster way for FPGA architecture design.

We use analytical model based design techniques, where the models consist of a set of equations that relate the effectiveness of FPGA architectures to the parameters describing these architectures. During *early stage architecture investigation*, FPGA architects and vendors can use our equations to quickly short-list a limited number of architectures from a range of architectures under investigation. Only the short-listed architectures need then be investigated using an expensive experimental approach.

This dissertation presents three contributions towards the formulation of analytical models and the investigation of capabilities and limitations of these models.

First, we develop models that relate key FPGA architectural parameters to the depth along the critical path and the post-placement wirelength. We detail how these models can be used to estimate the expected area of implementation and critical path delay for user-circuits mapped on FPGAs.

Secondly, we develop a model that relates key parameters of the FPGA routing fabric to the fabric's routability, assuming that a modern one-step global/detailed router is used. We show that our model can capture the effects of the architectural parameters on routability.

Thirdly, we investigate capabilities and limitations of analytical models in answering design questions that are posed by the FPGA architects. Comparing with two experimental approaches, we demonstrate that analytical models can better optimize FPGA architectures while requiring significantly less design effort. However, we also demonstrate that the analytical models, due to their continuous nature, should not be used to answer the architecture design questions related to applications having ‘discrete effects’.

Preface

[1] (Chapter 3) A.M. Smith, J. Das, S.J.E. Wilton. Wirelength Modeling for Homogeneous and Heterogeneous FPGA Architecture Development. In Proc. of the ACM/SIGDA international symposium on Field programmable gate arrays, pages 181-190, February 22-24, 2009.

[2] (Chapter 3) J. Das, A. Lam, S.J.E. Wilton, P. Leong, W. Luk. Modeling Post-Techmapping and Post-Clustering FPGA Circuit Depth. In Proc. of the International Conference on Field Programmable Logic and Applications, pages 205-211, September 2009.

[3] (Chapter 4) J. Das, S.J.E. Wilton. An Analytical Model Relating FPGA Architecture Parameters to Routability. In Proc. of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pages 181-184, February 2011. (short paper)

[4] (Chapter 3) J. Das, A. Lam, S.J.E. Wilton, P. Leong, W. Luk. An Analytical Model Relating FPGA Architecture to Logic Density and Depth. In IEEE Transactions on Very Large Scale Integration. Vol. 19, issue 12, pp. 2229-2242, December 2011.

[5] (Chapter 5) J. Das, S.J.E. Wilton. Accelerated FPGA Architecture Design: Capabilities and Limitations of Analytical Model. In Proc. of the International Conference on Field-Programmable Technology, pages 1-8, December 2011.

[6] (Chapter 4) J. Das, S.J.E. Wilton. An Analytical Model for FPGA Routability and its Applications. *Submitted* to ACM Transactions on Reconfigurable Technology and Systems, April 2012.

The research contributions presented in this thesis have previously been published in journals and conference proceedings. The majority of the Chapter 3 has been published in [1], [2] and [4]. In [1], I developed the wirelength model for homogeneous architecture that was extended by the co-authors for modeling wirelength for heterogeneous architectures. In [4], I used the work from the co-authors to model logic density; the rest of this publication was based on my work. Parts of Chapter 4 was published in [3] and an extended version was submitted for review [6]. A preliminary version of Chapter 5 was published in [5] and an extended version is under preparation.

In [1], I have collaborated with Dr. Alastair M. Smith of Imperial College, UK (currently at PA Consulting Group, UK). In [2] and [4], I have used guidance and manuscript editing from Dr. Wayne

Luk of Imperial College, UK and Dr. Philip Leong of Chinese University of Hong Kong, Hong Kong (currently at University of Sydney, Australia). In all cases, I conducted the research and prepared the manuscripts with guidance and manuscript editing from my supervisor Dr. Steven J.E. Wilton.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

FPGA	Field-Programmable Gate Array
CAD	Computer-Aided Design
BLE	Basic Logic Element
LUT	Look-Up Table
ASIC	Application Specific Integrated Circuit
SB	Switch Block
CB	Connection Block
VPR	Versatile Place-and-Route, a commonly-used academic FPGA place-and-route tool
VTR	Verilog-to-Routing project, an academic synthesis flow
DOE	Design of Experiments, originally proposed by Sir Ronald Fisher in 1926 for the agriculture domain, and later used by the designers and the industry leaders of several domains
VEB	Virtual Embedded Block
DPG	DOE-based Pareto-point generation, a technique proposed to investigate Pareto points of a FPGA architecture by using DOE-based technique
PB	Plackett-Burmann set of experiments, used to investigate the effects of input parameters on output parameters in DOE-based techniques
EB	Embedded Block
GALS	Globally Asynchronous Locally Synchronous
PCB	Printed Circuit Board
GPU	Graphics Processing Unit

Acknowledgements

The work in this dissertation was performed under the supervision of Dr. Steven J.E. Wilton. I am fortunate to have worked with Dr. Wilton. His unlimited guidance in the direction of my research helped me to carry out my research as well as to prepare this dissertation. His detailed oriented approach helped me in each stage of my research. Working with him made me much more organized and much more focused than I had been ever before. I am also thankful to him for his guidance in my professional development.

I would also like to express thanks to my qualifying, departmental and final examination committee members: Dr. Tor Aamodt, Dr. Sathish Gopalakrishnan, Dr. Alan Hu, Dr. Shahriar Mirabbasi and Dr. Konrad Walus. Their comments and feedback improved the quality of my research and my dissertation.

I would like to thank the Natural Sciences and Engineering Research Council (NSERC) of Canada and Altera Toronto Technology Center for funding my Ph.D. research. I thank Dr. Fahmida N. Chowdhury of the University of Louisiana for her support during my MS program and for encouraging me to pursue further research. I thank the colleagues at the Systems-On-Chip laboratory in the University of British Columbia for their support during my Ph.D. program. I thank my current employer Achronix Semiconductor Corporation for giving me the industry exposure in a dynamic and challenging environment.

I am thankful to my parents, Dharma Das and Kazal Das as well as my three sisters and their little ones: Aishi, Aira and Auritro; for their support. I thank my wife Reshmi Chowdhury for her extreme support and encouragement throughout my Ph.D. program as well as for helping me maintain calmness during my submission deadlines. Finally, I am thankful to my little one Aattoja Chowdhury Das for keeping me upbeat over the last three years. Posting Aattoja's updates and pictures on Facebook provided me with the much-needed breaks during this time.

Chapter 1

Introduction

1.1 Motivation

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated devices that can be used to implement logic without going through an expensive fabrication process. FPGAs have evolved considerably since their introduction in 1985. Originally used for prototyping and small glue logic replacement, these devices are now used to implement entire systems containing memory, embedded processors, and other functionality. Advanced manufacturing technologies provide an unprecedented number of transistors, and FPGA vendors have used these transistors to create new logic, routing and embedded block architectures. Unlike Application Specific Integrated Circuits (ASICs), FPGAs are pre-fabricated devices. This requires the FPGA vendors to make FPGA devices suitable for a wide range of applications. Much of the improvement in FPGA technology is a result of improvements in FPGA architecture. The architecture of an FPGA refers to the structure and interconnection of the configurable elements inside the device. New architectures are designed to provide higher density, lower power consumption and/or faster circuit implementations. FPGA manufacturers expend tremendous effort evaluating architectural enhancements for every generation of their devices, and this activity shows no signs of diminishing [10, 188].

During the design of a new FPGA, each architectural enhancement has to be evaluated to determine whether it should be incorporated in the new device. Typically-used evaluation metrics are area, delay, power and routability. This evaluation is typically done using an experimental approach [21] with the help of detailed area, delay, and power models [11, 21, 116, 119, 141]. Due to the presence of numerous interacting architecture parameters, the experimental evaluation of FPGA architectures by sweeping the

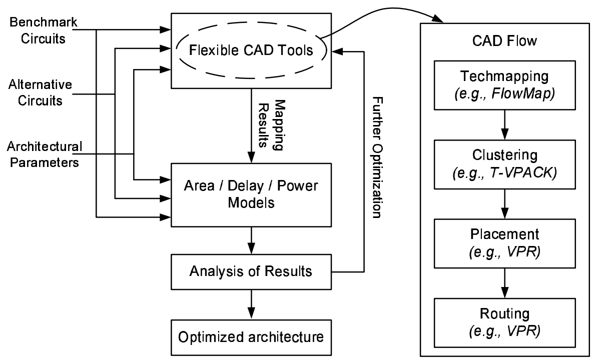


Figure 1.1: Fully Experimental Approach for New FPGA Architecture Design

parameters can be slow and computationally expensive, and limits the number of alternative architectures that can be considered. The consequence of this is the limited ability of FPGA companies to explore new structures that may lead to more efficient FPGAs.

Analytical models that describe some aspects of an architecture may complement the experimental approach to accelerate the architecture investigation. Analytical models relate parameters describing an FPGA architecture to area, delay, or power efficiency. These usually take the form of simple expressions, and thus searching for efficient architectures can be fast and the need for time-consuming experiments is reduced. The focus of this thesis is to accelerate the process of new FPGA architecture design by using analytical models. More specifically, this dissertation *develops analytical models to describe area efficiency, delay efficiency and routability of FPGA architecture variants, as well as to investigate the capabilities and the limitations of the developed models.*

1.2 FPGA Architecture Design

Since the accelerated FPGA architecture design process is central to this thesis, we present a summary of how new architectures are designed. We first present an overview of a typical experimental design approach. The major aspects of analytical models are presented next along with the discussion about how these models can accelerate architecture design process.

1.2.1 Experimental Approach

Figure 1.1 illustrates a typical experimental design flow. To design a new architecture using an experimental approach, FPGA architecture designers and/or FPGA vendors start with a set of prospective architectures. These architectures are characterized by the parameters related to the logic fabric that implements the circuit functions and the interconnect (routing) fabric that connects the elements. To evaluate each architecture, the architects collect benchmark circuits, make required modifications to the Computer-Aided Design (CAD) tools, and run the benchmark circuits through the CAD flow. The experimental results are evaluated empirically to fine-tune the architectural parameters for the next iterations. Such empirical evaluation demands a high number of experimental runs. There are several challenges associated with such a fully experimental approach:

1. To properly exercise an architecture, many benchmarks are required. If the choice of benchmark circuits is insufficient, it is possible to create an architecture that is tuned for specific circuits rather than one suitable for a wide range of customers.

2. Each of the benchmark circuits must be mapped to all potential variants of the architecture under investigation; each mapping can take several hours using modern CAD tools. This slow progress limits the number of alternative architectures that can be considered, and thus limits the ability of FPGA companies to explore new structures that may lead to more efficient FPGAs.

3. Optimizing the architectures by repeated iterations through a complete CAD flow is time-consuming and computationally expensive. This is compounded by the fact that the designers often optimize the architectures on an ad-hoc basis since they do not have any detailed insight about the behavior of the architectures under investigation.

4. To investigate a set of new architectures, the architects may need to modify the existing CAD tools for each architecture, which is a time consuming process.

1.2.2 Analytical Model-Based Approach

To accelerate the development of new FPGA architectures by addressing the challenges listed in Section 1.2.1, there have been recent efforts to develop a set of analytical models that relate the architectural parameters to each other, and to the evaluation matrix [41, 58, 59, 61, 70, 90, 104, 168].

Figure 1.2 presents a conceptual overview of analytical models. The inputs to these models are FPGA architectural parameters. The architectural parameters may consist of logic fabric parameters as well as the routing fabric parameters. The inputs also include a limited number of circuit parameters, such as Rent

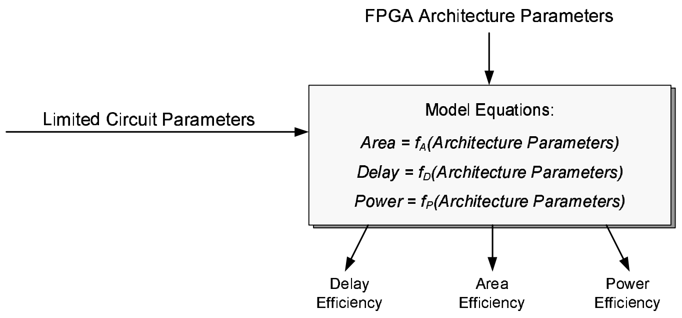

FPGA Architecture Parameters
Limited Circuit Parameters
Model Equations:
Area = f_A(Architecture Parameters)
Delay = f_D(Architecture Parameters)
Power = f_P(Architecture Parameters)
Delay Efficiency
Area Efficiency
Power Efficiency

Figure 1.2: Analytical Models: Conceptual Overview

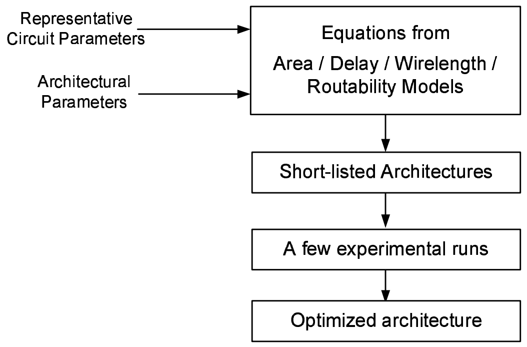

Representative Circuit Parameters
Architectural Parameters
Equations from Area / Delay / Wirelength / Routability Models
Short-listed Architectures
A few experimental runs
Optimized architecture

Figure 1.3: Analytical Model-Based Design of New FPGA Architectures

co-efficient[1], and the number of two-input gates in the circuit. The models consist of a set of equations to relate these architectural parameters to the evaluation metrics, such as area of implementation, critical path delay, routability and power.

This thesis envisages that the analytical models to complement the traditional experimental approach, and accelerate the architectural investigation in three ways:

1. Understanding the relationships between architectural parameters enables *early-stage architecture development* [70] in which the analytical models can be used to quickly search the design space. This is illustrated in Figure 1.3, where the design space is first pruned using model equations. Once a promising region of architectures has been identified, a traditional experimental approach can be used to further fine-tune the short-listed architectures. This will significantly accelerate the FPGA architecture design process, and will allow the study of a wider range of architectures,

[1]The Rent coefficient is a measure of the complexity of the interconnect pattern in a circuit, defined by the relationship between number of logic gates and required input/output pins [107].

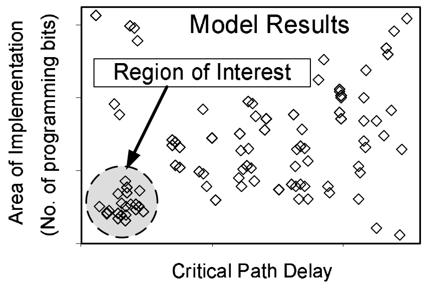


Figure 1.4: Early Stage Architecture Investigation Using Analytical Models

since the designers do not require to develop or modify the CAD tools for each architecture under investigation.

2. The FPGA architecture designers do not require benchmark circuits during early-stage architecture investigation. Instead, they may use a range of representative values for the limited number of circuit parameters that the model equations require.

3. The development of a body of theory describing the behavior of architectures will encourage an understanding of *why* certain architectures work well, and may eventually provide bounds on the capabilities and efficiencies of programmable logic. Such insights will also help the designers to better optimize the architectures using an experimental approach.

1.2.3 Example of Using Analytical Model Based Design Technique

Figure 1.4 uses fictitious results to illustrate how analytical models can be used to optimize an FPGA architecture with respect to area and delay. Each data-point on this plot represents one particular architecture. Values on the horizontal axis and the vertical axis represent the critical path delay and the area of implementation respectively. For each architecture, designers can use analytical models to quickly estimate the values for area and delay. For the best area-delay trade-off, only the architectures on the bottom left corner of Figure 1.4 that are enclosed by a circle will be identified as the promising ones, and will be experimentally evaluated to identify the best architecture with respect to design goals.

1.3 Research Approach

This section presents the assumptions and the guiding principles used in this thesis to develop and validate the analytical models.

1.3.1 Assumptions

The models in this thesis assume a homogeneous clustered architecture. We also discuss the possibilities of extending our models to capture the behavior of heterogeneous architectures with embedded blocks. Furthermore, creating a complete analytical model that relates all architectural parameters is complex. To make our task tractable, we employ the approach described in [61, 104], which parallels the step-by-step CAD flow used in the FPGA design flow. For each CAD stage, we develop a stage model that isolates the impact of the CAD stage on quantities related to the area and delay of the final implementation. These stage models are then combined in an overall model. Following this construction, area and delay models are divided into five stages, namely (1) technology mapping stage, (2) packing stage, (3) placement stage, (4) routing stage and (5) physical design stage.

1.3.2 Guiding Principles

Three principles guide the development of the analytical models that we present in this dissertation:

1. We endeavor to develop models by deriving relations analytically, without relying on curve-fitting or experimental techniques. This ensures that the models capture the essence of programmable logic, and is not limited to a particular CAD flow or tool suite.

2. We wish to derive models that are as independent of the circuit (to be implemented on FPGA) as possible. This makes the models of this dissertation different from estimation studies, in which the goals are to predict the area, speed, or power for given circuits. That being said, it is impossible to completely ignore the impact of specific circuits; hence a few circuit parameters are used by the models. All of the circuit parameters used in this thesis are available during early stage evaluation and do not require to run the circuit through any stage of an expensive CAD flow.

3. We attempt to balance complexity with accuracy. The simple model equations will provide more insight into architectural trade-offs than complex expressions. Such insights will help designers to effectively fine-tune an architecture under evaluation.

1.4 Research Contributions

To summarize the previous sections, the goal of this thesis is to *develop the analytical models and to investigate the capabilities and limitations of such models*. This dissertation presents three contributions towards this goal:

1. We develop analytical models that relate key FPGA architectures to the post technology mapping and the post-clustering depth of the critical path as well as the post-placement wirelength. Parts of this work have been published in two conference proceedings [60, 168] and in a journal [61].

2. We develop an analytical model that relates key FPGA parameters to the fabric's routability, assuming that a modern one-step global/detailed router is used. This work has been published in a conference proceeding [58] and has been submitted for publication in a journal [57].

3. We investigate on the capabilities and the limitations of analytical models in answering design questions that the FPGA architects are typically interested in. This work has been published in a conference proceeding [59].

The next three subsections briefly discuss these three contributions. The details are presented as individual chapters in this dissertation.

1.4.1 Analytical Models Relating FPGA Architecture to Area and Delay

The first contribution of this thesis is *developing analytical models that relate key FPGA architecture parameters to area and delay*. More specifically, this thesis presents models for post-technology mapping depth, post-packing depth and post-placement wirelength. These models can be used in conjunction with the previously proposed models to quickly evaluate area and delay performances of an FPGA architecture.

The challenges tackled in developing these models were to balance simplicity with accuracy and to make the models independent of CAD tools and as independent of circuit parameters as possible. Chapter 3 presents the details on derivation and validation of these models.

1.4.2 Analytical Model Relating FPGA Architecture to Routability

The second contribution of this dissertation is *developing an analytical model to relate FPGA logic and routing fabric parameters to the consequent routability, when a modern single step global/detailed router is used to map the circuits on FPGAs*. We define routability as the probability that all nets can be routed on a given FPGA device. Although a range of studies exist to estimate routability for both ASIC and FPGA devices [25, 35, 68, 99, 120, 152, 163], to our knowledge only one of them [25] is focused on relating FPGA architectural parameters to routability. The work of [25] models the FPGA routability for the detailed stage of a two-stage router, where the routing paths for the detailed stage are pre-determined by the global router. The novelty of our work is that we model routability for a single-step global-detailed router, where many possible routing paths are simultaneously considered by the routing algorithm; such a single-step router is most commonly used for modern FPGAs.

The major challenge in deriving our routability model was to capture the effect of many possible routing paths that are not independent of each other. To tackle this challenge, we observe that our problem is related to the problem of estimating the reliability of a *multi-terminal stochastic network*. For given network constraints, the reliability of such a network is measured by the existence of at least one useful communication path between the terminals (out of many possible communication paths). This is analogous to our problem, which is to model the routability with given FPGA architecture constraints, while assuming that the routing of the nets may use any of the many possible available paths.

Finding an exact solution for the reliability of a stochastic network is *NP*-hard [16], and earlier studies use graph-theoric techniques to find bounds for the reliability of such a network [69, 142, 155, 156, 159]. In this dissertation, we also use graph-theoric techniques in modeling the upper-bound of routability for a single-step FPGA combined router.

Chapter 4 presents the details on derivation and validation of the routability model.

1.4.3 Accelerated FPGA Architecture Design: Capabilities and Limitations of Analytical Models

The final contribution of this dissertation *investigates the capabilities and limitations of analytical models*. This contribution has two parts, related to the use of analytical models in two stages of a design flow: (a) the very initial "back-of-the-envelope" stage and (b) the "parametric sweep" stage.

During the initial stage, architects wish to identify the architectural parameters that will affect density and performance of the resulting device. The first part of this contribution investigates whether analytical models can provide such insight.

The second part investigates whether the analytical models can be effectively used during parameteric sweep when the architectural parameters are swept to evaluate their effects on resulting area or delay. Since analytical models can quickly evaluate architectural choices, use of analytical models in this stage is expected to significantly reduce design time. The study of the second part is motivated by two issues that are related to analytical model based design techniques:

1. First, studies on analytical modeling consider a limited number of architectural parameters in isolation, and make important assumptions in deriving model equations. This issue motivates us to study if the analytical models, when combined together, can effectively answer the architecture questions that simultaneously evaluate a wide range of architectural parameters with respect to multiple evaluation metrics. The significance of this issue is emphasized by Yan et al. [191]. That work observes that experimental techniques proposed by separate studies make a range of assumptions when estimating evaluation metrics for an FPGA implementation. Yan et al. finds that when these studies are used simultaneously, resulting architecture conclusions are highly sensitive to the assumptions made by each study.

2. Secondly, to our knowledge, the analytical models previously presented are continuous in nature. The models therefore may not capture the effects of architecture choices when implementing the applications having "discrete effects". For instance, the Basic Logic Element (BLE) of an FPGA is characterized by the number of inputs that it can take. A BLE with six inputs can efficiently implement a multiplexer with four inputs and two select signals. However, in implementing such a multiplexer, a BLE with five inputs will be inefficient and a BLE with seven inputs will not offer additional benefits. The analytical models, due to their continuous nature, may not capture these effects. This may limit the usefulness of analytical models when designing an FPGA device that targets selected domain(s) of applications.

The second part of the final contribution of this thesis investigates the effects of these two issues. More specifically, we ask the following two architecture questions:

1. What are the optimal values of the architectural parameters for a general-purpose FPGA?

2. For an optimized application-specific FPGA architecture, what are the values of the architectural parameters?

To investigate the effectiveness of analytical models, we compare the model results with two experimental techniques: a sequential optimization technique and a Design of Experiments (DOE)-based experimentation technique. The first technique is often used for FPGA optimization while the second one is used in a wide range of domains including manufacturing and agriculture. The focus of this work is to determine if the model-based technique can identify similar (or better) architectural configurations, with respect to area and/or delay, when compared with the experimental technique. If the conclusions are similar (or better), the model-based technique will be a valuable addition to the architecture design process, since it can evaluate a wide range of architectures in significantly shorter period of time.

The investigation results from this contribution will help the architecture designers in understanding the types of design questions that can be effectively answered by the analytical model-based techniques, as well as the limitations of the analytical model-based techniques in answering certain design questions.

Chapter 5 presents details on this investigation.

1.5 Thesis Organization

Chapter 2 provides background information on FPGA architecture and CAD flow, and presents earlier work related to our work. Chapter 3 presents our first contribution related to the development of analytical models for area and delay. The second contribution, development of routability model while considering a single-step combined router is presented in Chapter 4. Chapter 5 presents our third contribution, which is to investigate the capabilities and limitations of analytical models in designing new FPGA architectures. Finally, Chapter 6 summarizes the contributions, concludes this thesis and presents the short-term and long-term avenues for future work.

Chapter 2

Background and Related Work

This chapter presents background of this dissertation and puts this dissertation into the context of earlier work. In Section 2.1 and 2.2 respectively, we review the architecture of typical island-style FPGAs and the CAD flow used to implement circuits on FPGAs. In Section 2.3.1, we review experimental techniques that can be used for FPGA design space exploration. Section 2.3.2 presents the analytical models that have been proposed for ASIC domain, and are relevant to the work presented in this dissertation. In Section 2.3.3, we present the analytical models proposed by other researchers for the FPGA domain. The objective of these models is to evaluate FPGA architectures without going through an experimental flow. Section 2.3.4 presents previously published techniques that *experimentally* estimate the area, delay and routability for FPGA implementation. Finally, Section 2.4 summarizes this chapter.

2.1 Review of FPGA Architecture

2.1.1 Island-style FPGA Architecture

The *Island-style* FPGA architecture [21] is the most commonly used architecture both in academia and industry. This architecture has been proven to scale with Moore's Law [40] and has been used by leading FPGA vendors, such as Altera and Xilinx. In this thesis, our models are developed for and validated for island-style architectures. Figure 2.1 presents a typical island-style FPGA architecture. The architecture consists of three fundamental components: configurable logic resources, a configurable interconnect fabric, and I/O resources [21]. FPGA architecture parameters that we investigate are associated with these components and are presented in Table 2.1.

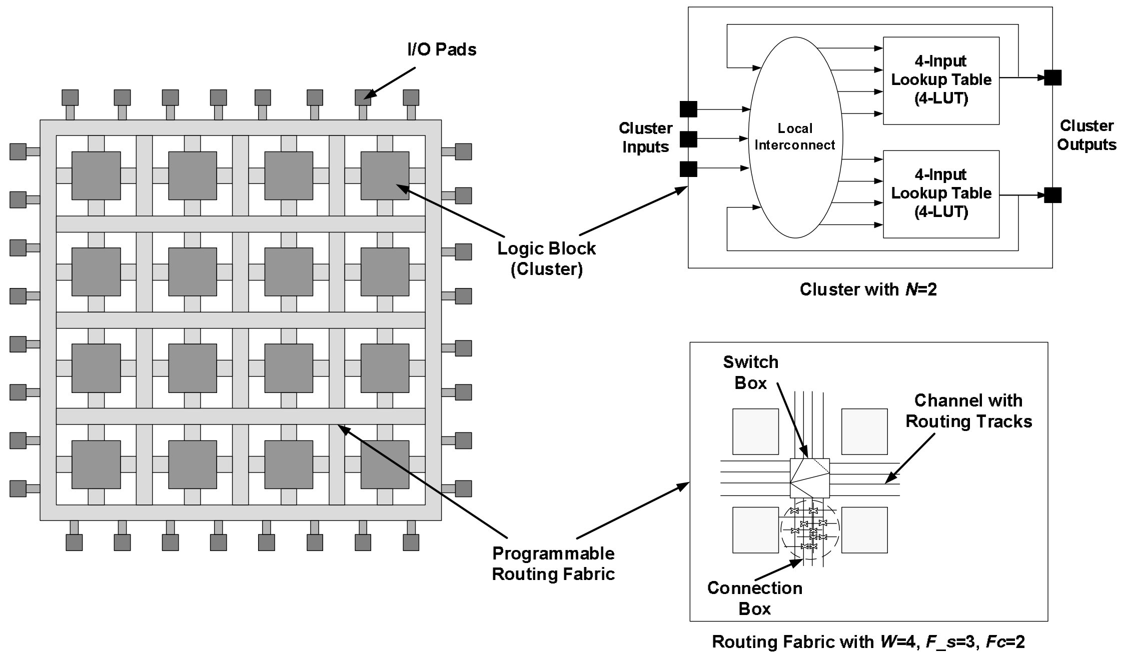

I/O Pads
Logic Block (Cluster)
Programmable Routing Fabric
Cluster Inputs
Local Interconnect
4-Input Lookup Table (4-LUT)
4-Input Lookup Table (4-LUT)
Cluster Outputs
Cluster with N=2
Switch Box
Channel with Routing Tracks
Connection Box
Routing Fabric with W=4, F_s=3, Fc=2

Figure 2.1: Typical Island-Style FPGA Architecture

2.1.1.1 Configurable Logic Resources

Configurable logic resources of an FPGA consist of Look-Up Tables (LUTs), Basic Logic Elements (BLEs) and clusters. LUTs are characterized by the number of inputs *K*. Each *K*-LUT is capable of implementing a logic function having up to *K* inputs and a single output. A *K*-LUT can be configured to implement a specific function by storing the corresponding truth table in the LUT's configuration memory bits. Together with a flip-flop at its output, a LUT forms a *BLE*.

Multiple BLEs are grouped into *clusters*. Clusters are characterized by the number of BLEs that they contain (*N*). Figure 2.1 shows a cluster with *N*=2. The BLEs within a cluster are connected by

Table 2.1: Architectural Parameters that We Investigate (Model Inputs)

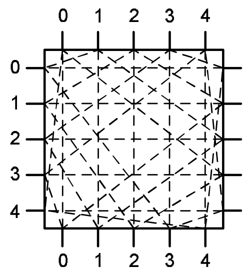
Logic Fabric Parameters	
K	Lookup table size
N	Cluster (logic block) Size
I	Inputs per cluster
Routing Fabric Parameters	
W	Channel width (tracks per a routing channel)
$F_{c_{out}}$	Source connection box flexibility
$F_{c_{in}}$	Sink connection box (CB) flexibility
F_s	Switch box flexibility
N_x	FPGA grids in columns
N_y	FPGA grids in rows

local interconnect, and we refer to these connections as *intra-cluster* connections. The BLEs that are not within the same cluster are connected by using a configurable interconnect fabric, a description of which will follow. We refer to such connections as *inter-cluster* connections. Intra-cluster connections are much faster than inter-cluster connections and incur less area overhead. Packing closely-connected BLEs into clusters therefore minimize delay and reduce interconnect area.

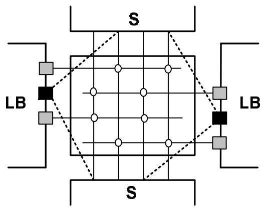
Configurable logic resources of island style architectures are further broken down into two categories: general logic blocks and embedded specialized blocks. Examples of embedded blocks are memory blocks, Digital Signal Processing (DSP) blocks and embedded configurable CPUs. FPGAs containing only general logic blocks are known as *homogeneous FPGA architectures*. The term *heterogeneous FPGA architecture* refers to an architecture that contains embedded blocks in addition to general logic blocks. Such blocks are also known as *coarse-grained logic blocks*. Recent studies demonstrate that incorporating embedded blocks into an architecture can significantly save area [103]. In industry, Altera first deviated from island-style homogeneous architecture by introducing hard-wired memory into an FPGA architecture [7, 165]. Xilinx, Microsemi and Achronix have also incorporated embedded blocks into their products [1, 2, 189].

Finding the optimal number of inputs to each LUT is an active area of research. While earlier studies found the optimal number of inputs to be four or five [5], recent studies reported the optimal LUT size to be 6 for modern technology [122]. Altera introduced an adaptive logic module (ALM) that was based on a large “fracturable” LUT with 6 inputs; the fracturable LUT could be partitioned into two smaller LUTs [92, 115]. A fracturable LUT is parameterized by two parameters k and m, where k represents the size of the base LUT and m represents the extra inputs available for implemented smaller LUTs; a 6,2 fracturable LUT implements a 6-LUT with total 8 input pins [115].

The optimum number of LUTs in a cluster N and the number of inputs per cluster I have also gained substantial attention [6, 116]. Comparing the experimental results for cluster size from 1 to 10, Ahmed and Rose [6] found that the cluster size of 3 to 10 presented the best area-delay product for LUT size of 4 to 6. Li et al. [116] found that the power-delay product decreased as cluster size N increased. However, this work also found that the logic power consumption increased with increasing N and this increase became dominant beyond N=12; N=12 was identified as the optimal cluster size with respect to power-delay product [116]. As for the optimal value of inputs per cluster I, Ahmed and Rose [5] presented an



(a) Wilton switch block [182]
($W = 5$, $F_s = 3$)



(b) Typical connection block [70, 112]
($W = 4$, $F_{c\_in} = 2$, $F_{c\_out} = 2$)

Figure 2.2: Switch Block and Connection Block Detail

empirical relationship between N, K and I:

$$I = (K/2) \cdot (N+1). \tag{2.1}$$

In finding the optimal values for N, K and I, experimental approach was followed by the studies discussed above.

2.1.1.2 Configurable Interconnect Fabric

Figure 2.1 illustrates the location of configurable interconnect (routing) fabric within an FPGA architecture. The configurable routing fabric provides connections among clusters and I/O resources, and consists of wires, programmable switches and connection blocks.

The wire segments that connect the clusters are arranged in horizontal and vertical *channels* surrounding the logic clusters. Each routing channel contains W parallel tracks; we refer to W as the *channel width*. The width of channels in an FPGA device is typically consistent across the device and FPGAs are typically manufactured with the worst-case channel-width requirement in mind. In this thesis, we assume all channels to have the same width. One of the optimization goals of experimental CAD tools is to reduce the number of tracks in FPGA routing channels. Several works have aimed at reducing the channel-width requirement at the expense of an increased number of clusters [66, 176]. DeHon [66] showed that under-utilization of clusters (packing less BLEs than are allowed by cluster size) reduced the channel-width requirement at the expense of increased cluster-count. Tom and Lemieux [176] found that under-utilization of clusters only in routing-congested regions could offer a ‘graceful trade-off’ between channel-width and cluster-count.

At the intersection of each horizontal and vertical channel is a *switch block* that allows the wires to go straight, to turn directions, or to terminate. Each switch block connects the input wire segments to the wire segments in adjacent channels. The amount of connectivity in each switch block is quantified using the parameter F_s; F_s is the number of wires to which each input wire can connect. In Figure 2.2(a), wire-0 on the left side can connect to wire-0, wire-1 and wire-3 on right, top and bottom sides respectively. Thus $F_s = 3$ in this example. Several types of switch boxes have been presented and analyzed, including the Universal switch box [185], the Wilton switch box [182] and the Disjoint switch box [128]. Lemieux and Lewis [111] presented a detailed study on these switch blocks.

Each logic cluster is connected to the neighbouring wires using input and output *connection blocks*. The flexibility of these blocks is quantified by the parameters F_{c_in} and F_{c_out}. Each of these parameters indicate the proportion of the wires in the neighbouring channel to which each input (output) pin of the cluster can be connected. In the example of Figure 2.2(b), F_{c_in} and F_{c_out} are 50%.

To make our task tractable, we assume the *segment length* of routing fabric to be 1, implying that routing tracks span one logic block before being terminated at switch blocks. Betz and Rose [20] observe that routing architectures with higher segment lengths are superior in terms of both delay and routing area. Modern FPGAs provide routing tracks that span more than one logic block before terminating. For instance, Altera's Stratix II family provides routing tracks with segment lengths of 1 and 4 for both horizontal and vertical wires, segment length of 16 for vertical channels only and segment length of 24 for horizontal channels only [115].

2.1.1.3 I/O Resources

I/O resources on an FPGA provides the interface between the circuit mapped on the FPGA and the outside world. Modern FPGAs provide programmable I/O resources and optimize I/O resources with respect to range of design goals. Programmable I/O features provided by Altera Cyclone series include programmable current strength, programmable slew rate control and programmable delay [8]. Current generation FPGAs from Xilinx group the I/O resources into banks that are independently able to support different I/O standards; the Virtex-6 family FPGAs from Xilinx provide 9 to 30 I/O banks [189]. Actel (Microsemi) optimizes the I/O features of their IGLOO PLUS family to meet the needs of I/O-intensive, low-power applications [132]. The I/O frame of Achronix's picoPIPE fabric ensures that Data Tokens enter the asycnronous picoPIPE core at a clock edge; every Data Token leaving picoPIPE is also clocked out at a clock edge by the same I/O frame [1].

2.1.2 Application-Specific FPGAs (ASFPGAs)

The FPGA architecture explained above targets the implementation of a large range of applications (circuits). Such flexibility makes FPGAs ideal for prototyping and low-volume production. However, this high degree of flexibility is associated with area and delay penalties. For circuits implemented by using only LUTs and flip-flops, Kuon and Rose [102] found that FPGA implementations were 40X larger and 3.2X slower on average when compared to standard cell implementations. Recent works attempt to design reconfigurable architectures that can target a specific application or a domain of applications. Such architectures will ensure less area and delay overhead at the expense of reduced flexibility. Holland and Hauck investigated domain-specific CPLDs [88] and presented automated tool-flow to generate domain-specific CPLDs [89]. Hammerquist and Lysecky [85] found that ASFPGAs provided significant savings in terms of area, delay and energy when compared to the FPGAs designed for average benchmarks.

2.1.3 Other FPGA Architectures

FPGAs that differ from synchronous island-style architectures have also been proposed by vendors and academic researchers. Microsemi (previously Actel) uses row-based architecture for its FPGAs [133]. They use Flash-memory based FPGA fabric [83]. Such an architecture has rows of programmable logic cells separated by horizontal wiring channels; horizontal channels provide the inputs to the cells. To mitigate the effects of latency and metastability owing to global clocks, Royal and Cheung [148] proposed the Globally Asynchronous Locally Synchronous (GALS) FPGA architectures. Other academic studies further investigated the feasibility of asynchronous FPGA architectures [97, 172]. In industry, Achronix Semiconductor Corporation has used asynchronous architectures for FPGAs. Their architecture redefines the concept of ‘Data-Token’ [1]. In contrast to conventional logic where a data-token is a logic value at the clock edge, this asynchronous architecture combines data and clock edge to form a data-token. The Spacetime Architecture proposed by Tabula Inc. increases the density and performance of 2D architectures through time multiplexing [173]. The Spacetime hardware is capable of dynamically reconfiguring itself to perform multiple operations in a single user clock cycle. A recent work from Grant et al. [82] proposed the *Malibu* architecture that incorporated embedded blocks into homogeneous architectures. Malibu architecture adds time-multiplexed coarse-grained processing elements to the typical fine-grained

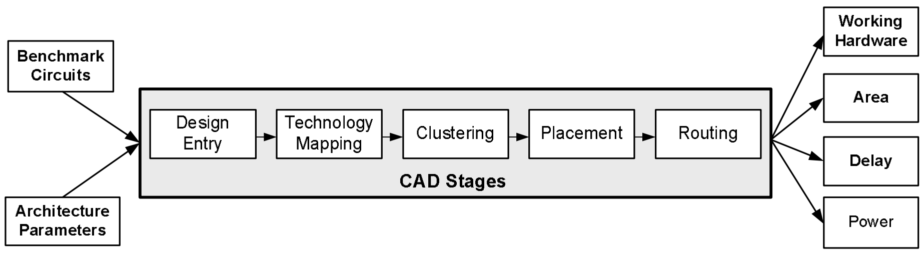


Figure 2.3: FPGA Design Flow

clusters; the original logic clusters and interconnect fabric are not time-multiplexed. By time-multiplexing the coarse-grained elements, the area cost for these elements are divided over many cycles. Several works in academia [3, 28, 177] have also proposed hierarchical architectures that are different from cluster-based hierarchy that we have explained in previous sub-sections. In hierarchical architectures, the basic level-1 block consists of a few logic blocks (LUTs) and I/O blocks [28]. Level-2 block consists of level-1 blocks, local routing for level-1 blocks and global routing for connections to level-3 blocks. Upper level blocks are formed in a similar way. Such hierarchical architectures are not widely used due to scalability issues [40].

2.2 Review of FPGA CAD Flow

2.2.1 CAD Stages

Figure 2.3 shows a typical FPGA CAD flow, consisting of five stages: design entry, technology mapping, packing (clustering), placement and routing. A brief summary of previous works on these stages is included below. Further information can be found in the survey on FPGA design automation by Chen et al. [32].

2.2.1.1 Design Entry

The design entry phase is the first step in the CAD flow. During design entry, the user specifies the circuit that will be implemented on the FPGA. Design entry methods can be broadly divided into two categories: schematic capture based techniques and programming language based techniques [165]. The schematic capture based techniques typically use a dataflow/synchronous dataflow paradigm. Programming language based approaches include VHDL [14], Verilog [171], C, and MATLAB-based programs [135, 137].

2.2.1.2 Technology Mapping

In this phase, an optimized netlist is mapped into BLEs consisting of K-LUTs and flip-flops. Many studies have focused on different aspects of technology mapping [18, 23, 30, 31, 34, 50, 52, 66, 76, 95, 174], and the commonly used and referenced mapping algorithms include DAGMap [34], FlowMap [50], DAOMap [31] and the ABC Mapping tool [18]. Different technology mapping tools target area of implementation, depth (delay), power consumption and routability. Despite extensive work on logic optimization and technology mapping, for large circuits with 'known' optimal solutions for these stages, existing techniques were found by Cong and Minkovich [51] to perform very poorly. That work argued that there was still much room for research on FPGA synthesis algorithms.

2.2.1.3 Packing (Clustering)

During packing, the BLEs from the technology mapping phase are packed together to form *clusters*. Since the intra-cluster connections are faster than the inter-cluster connections, packing algorithms attempt to pack as many closely connected BLEs in a cluster as possible. Numerous studies have proposed techniques for clustering to optimize density [17, 21, 127, 162], speed [21, 53, 127, 158], power [141, 162] and routability [162] of FPGA implementations.

2.2.1.4 Placement

The placement stage of a CAD flow positions the packed clusters from the packing stage on the FPGA physical locations. Betz et al. [21] identified three major classes of placement algorithms that are used in the FPGA domain: (a) partitioning-based min-cut approaches, (b) analytic approach followed by local iterative improvement and (c) simulated annealing based approaches. To improve the resultant placement solutions, several publications have attempted to incorporate early predictions for post-routing delay, wirelength and routability into placement algorithms [15, 124–126]. The run-time for placement is a major concern for modern FPGAs since the order of run-time of a placement stage grows faster than linear with the growth in the number of clusters [40].

2.2.1.5 Routing

Finally, the routing stage determines which programmable switches in an FPGA need to be turned on to connect all required input and output pins of the clusters [21]. A routing algorithm can be timing-driven or delay oblivious, targeting speed and routability respectively. Since the routing delay makes up the majority of the overall delay of an FPGA implementation, timing-driven algorithms are usually preferred over delay oblivious ones [21].

Two types of routers have been widely studied. The first type of router is a *two-step router* consisting of "global" and "detailed" steps. During the global routing step, the router assigns sequences of channels to route the nets. The detailed routing step finds a route using this sequence. Several studies have proposed algorithms for global [145, 153] and detailed stages [24, 113] of two-step router. The second type of router, *single-step combined global-detailed router* routes the nets between the post-placement logic clusters in a single step [21]. A combined router considers all available routing resources simultaneously and therefore provides more routing flexibility. Chapter 4 of this dissertation provides further details on these two types of routers.

A recent work from Rubin and DeHon [149] argued that FPGA routing was not a solved problem despite the quality of VPR/Pathfinder. That work found that variations in initial conditions in Pathfinder might cause 17-110% variations in critical paths. Rubin and DeHon [149] proposed techniques to reduce delay noise to 1-13%.

2.2.2 CAD Run-time

With shrinking process sizes, the capacity of FPGA devices is increasing and the CAD run-time is becoming a major concern for FPGA users. Several studies focus on improving the run-time of CAD tools. The majority of these works focus on the most computationally expensive stages of a CAD flow: placement and routing [39, 80, 81, 121, 151]. Sankar and Rose [151] demonstrated that 52X placement-time improvement could be achieved for a 100,000-gate circuit at the expense of a 33% area penalty. This technique used multiple-level clustering and a constructive placement followed by annealing-based iterative improvement. Chin and Wilton [39] showed that FPGA architectures could be designed in a way that will reduce the time required for placement and routing stages. Ludwin et al. [121] detailed two parallelization strategies used within Altera's Quartus II FPGA placer. These two strategies benefit from multi-core processors by using a pipelined and parallel-moves approach, and offer 1.3X speedup on a two-core processor and 2.2X speedup on a four-core processor [121]. Wang and Lemieux [181] proposed a timing-driven parallel placer based on simulated annealing algorithm that can produce deterministic results. The proposed placer divides the FPGA into regions and each region is further divided into subregions. The threads investigating a cluster will sequentially iterate through all cluster-locations within its own region to find a possible swap. Only one sub-region is placed at a time to ensure determinism. Gort and Anderson [81] presented a new routing approach combined with a modified routing architecture and reported a 34% reduction in router run-time while incurring 3% area overhead with no delay penalty. The proposed router consists of two stages. Rather than assigning signals to single wire sigments as in a typical router, the first stage assigns signals to a *group of wire segments* using the Pathfinder algorithm, forcing Pathfinder to terminate early. The second stage uses a Boolean satisfiability solver (SAT) to assign each signal to one specific wire segment contained within the corresponding group.

Chin and Wilton [38, 42] also investigated the memory storage requirement for routing algorithms. These studies take the advantage of regularity in FPGA architectures by dividing FPGA into tiles; rather than storing the detailed routing resource graph (RRG) for a device.

2.3 Related Work

This section puts this dissertation into the context of earlier studies. We first present earlier studies on experimental design space exploration. We next present analytical models-based work in ASIC domain that is relevant to this dissertation. Earlier analytical models for FPGAs are presented next. Finally, we present the experimental techniques that have been previously proposed for estimating wirelength, delay and routability of FPGA implementations. In subsequent chapters, when presenting our contributions, we further differentiate our work from these earlier studies.

2.3.1 Experimental FPGA Design Space Exploration

This section presents previous work related to FPGA design space exploration (DSE). During design space exploration, the architects aim to optimize FPGA architectural parameters with respect to one or more design goals, such as area, delay and power. This section presents three experimental flow that may be used to optimize FPGA architectures.

2.3.1.1 Versatile Place-and-Route (VPR)

The most commonly used academic CAD flow is the VPR flow. VPR was developed at the University of Toronto [19, 21] and has been widely used by the research community. Although the original version of VPR has been developed to investigate homogeneous architectures, the newer versions [122, 123, 147] can investigate architectures with embedded blocks.

The inputs to VPR are a technology mapped netlist and a text or XML file describing the architecture. The architecture files describe an architecture and include area and delay constants related to the logic elements and the routing elements. VPACK, an associated tool in VPR, translates a technology mapped design to a clustered netlist. VPR places this netlist on an FPGA using a simulated annealing based method. VPR then performs either global routing or combined global/detailed routing using the Pathfinder [129] negotiated congestion algorithm.

In VPR, area is modeled by counting the number of minimum size transistors [21]. VPR uses the *Elmore delay model* to estimate the delay of a routed connection. Several studies have presented power estimation and power reduction techniques for FPGAs [11, 12, 22, 26, 47, 78, 84, 105, 106, 116, 117, 141, 154, 178].

Hammerquist and Lysecky [85] demonstrated that VPR could also be used to optimize application-specific FPGAs for a target domain of applications. The extended version of VPR, coined as Verilog-to-Routing (VTR), can capture designs at higher level of abstraction (HDL or higher) and pass them through synthesis stages to generate the routing solution for FPGA implementations [147].

2.3.1.2 Virtual Embedded Block (VEB)

The island-style architecture assumed in VPR only crudely approximates commercial FPGAs and advanced optimization features are not supported in VPR. To address these limitation, Ho et al. [87] proposed a methodology to optimize the Embedded Blocks (EBs) into commercial devices. This new methodology uses the concept of VEBs that may be virtually added to an existing FPGA architecture for rapid assessment of the consequent effects. This method helps evaluate the impact of introducing new embedded blocks even before the blocks are modeled in an experimental CAD tool.

In this methodology, prospective embedded blocks, such as multipliers, are captured into VEBs. Circuits containing VEBs are then placed and routed using the vendor tools; the VEBs are considered as black boxes during synthesis. Post-routing performance results from vendor tools are then used to investigate the impact of the embedded block. While Ho et al. [87] used VEB to capture the effects of multipliers, this technique was later used to investigate the impacts of other embedded blocks. For instance, Chong and Parameswaran [43] used this technique to investigate the performance of a flexible floating-point unit on Xilinx Virtex-II FPGA.

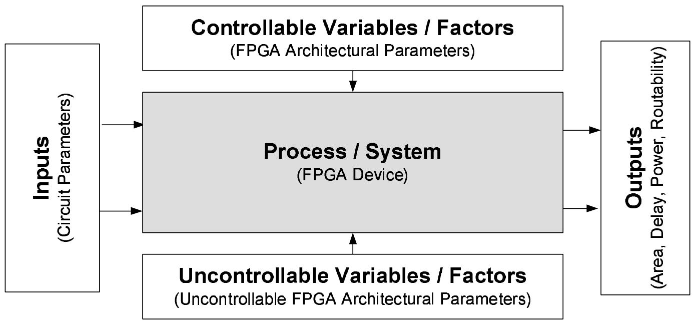

Controllable Variables / Factors
(FPGA Architectural Parameters)
Inputs
(Circuit Parameters)
Process / System
(FPGA Device)
Outputs
(Area, Delay, Power, Routability)
Uncontrollable Variables / Factors
(Uncontrollable FPGA Architectural Parameters)

Figure 2.4: FPGA Architecture Evaluation Using Design of Experiments (DOE)-Based Technique

2.3.1.3 Design of Experiments (DOE)

Originally proposed by Ronald Fisher in 1926 for use in the agriculture sector [75], the DOE based approach has been used in many other sectors [13, 157]. The goal of this approach is to understand the behavior of the *outputs* (responses) of a *process* with respect to the *inputs* (factors). In the context of FPGA architecture exploration, the FPGA architecture can be considered as a process, with architectural parameters (such as LUT size, channel width) being the factors or inputs and evaluation metrics (such as area, delay) being the responses or outputs. This conceptual framework is illustrated in Figure 2.4. DOE essentially aims to evaluate the effects of inputs on outputs using a minimal number of experiments. DOE-based techniques typically use a maximum of three levels (values) for each inputs: maximum, median and minimum.

To our knowledge, the study by Sheldon and Vahid [157] has been the only one that uses DOE-based technique for FPGAs. Sheldon and Vahid [157] used a DOE-based experimental technique to optimize the parameters of the configurable components (such as cache with configurable size or associativity) that were incorporated into modern pre-fabricated FPGAs. In contrast to [157], we use a DOE-based experimental technique in optimizing the design of *new* FPGA architectures.

2.3.1.4 Regression-Based Techniques

Several studies have explored the use of regression-based techniques in exploring the design space of general-purpose computer architectures [109], Graphics Processing Units (GPUs) architectures [96] and FPGA architectures [136]. These studies simulate a limited part of the full design space. Based on the simulation results, estimators are developed for the target evaluation metrics. Such estimators can be used to investigate the effects of the architectural choices on the evaluation metrics. For example, Lee and Brooks [109] derive application-specific performance and power models for applications executed on microrpocessors. Jia et al. [96] uses a step-wise regression method to investigate the effects of GPU parameters on the runtime of specific applications. Nepal et al. [136] uses regression based method to investigate the effects of algorithmic and hardware parameters on the performance of FPGA accelerators with the target domain of applications being real-time image processing. Each of these three previous studies use simulations on a limited number of architectural configurations from a large design space when deriving the models for the evaluation metrics.

The regression-based methods are dependent on applications (benchmarks) that they use for modeling. For instance, Jia et al. found that it was necessary to fine-tune their model for a specific application (AES) to address the application-specific issues. When designing new architectures, the CAD tools need to be in place to identify any such issues that may affect the quality of the developed models. Furthermore, models developed using regression-based techniques are dependent on the results from the simulations run on the existing architectures. This latter property may prevent them from capturing the effects of radically different architectures.

Chapter 5 of this dissertation demonstrates that analytical models have limitations in answering certain design questions, and advocates the use of analytical models as a supplement to the experimental approach. Regression-based techniques can be used for experimentations in such cases.

2.3.2 Analytical Models for ASICs

Researchers in the ASIC community have created several analytical models, some of which have also been used by the FPGA community. Many of these models use the well-known Rent's rule that relates the average number of pins per module T and average number of blocks per module G. Landman and Russo [107] describes the following relationship between T and G for a partitioned circuit:

$$T = t \cdot G^p \tag{2.2}$$

where in a given partition, T is the average number of pins per module and G is the average number of blocks per module. t and p respectively represent the Rent coefficient and the Rent exponent. In this formulation, T represents the number of internal pins in the partitioned design (as opposed to the number of external pins). Details on Rent's rule and the estimation of the Rent exponent may be found in [44, 46, 56].

El Gamal [68] presented a model for a non-programmable chip that related the area required for routing to the total number of pins in the logic gates. This model was later used in designing FPGAs [146]. Other researchers have also presented models for RC and RLC interconnect that can be used for delay modeling [33, 64, 65, 150].

In 1979, Donath presented a wirelength estimation model based on the hierarchical partitioning and placement method [67]. Donath modeled an upper bound on the average wirelength. Feuer presented a model that provided the wirelength distribution rather than the upper bound [73]. More recently, Stroobandt enhanced Donath's model to take into account the fact that an optimal placement favored short interconnection paths in physical architectures as opposed to longer paths [169, 170]. Stroobandt's estimation method also incorporated multi-terminal nets, external connections and three-dimensional physical architecture parameters. Studies by Davis et al. [62, 63] presented stochastic models for the distribution of local, semi-global and global wiring requirements based on occupancy probability. In that work, the circuit was divided into cells for wirelength modeling. Zarkesh-Ha et al. extended this latter model for heteregenous architectures [196]. Several studies used these wirelength distribution models for different purposes, such as efficient channel assignment for wires [98], determination of yield for wire cuts and bridges [45] and defining the boundary for RC lines that might be replaced with transmission lines [94]. In [46], Christie and Stroobandt presented a comprehensive study on Rent-based wirelength distribution models presented above. A recent study by Lanzerotti et al. [108] evaluated the performance of the wirelength models. Comaparing the model outputs with data extracted from modern chips, [108] found that although a few models exhibited a similar shape for the wirelength distribution graphs, they typically underestimated the number of connections with large wirelengths.

In [4], Agrawal presented an analytical technique to model routability for Printed Circuit Boards (PCBs) by showing that the Lee-Moore routing algorithm could be identified as a *percolation process* [1]. Agrawal showed that percolation theories could be used to analytically estimate the probability of routing nets through the cells on a PCB. In this study, each cell along the routing path was assumed to have a pre-defined probability of blocking a connection through them (obstacles). This study has found that at a certain blocking probability, routability from the source to the sinks (of the nets) suddenly drops from

[1]A percolation process is the one that is similar to the phenomenon, in which a liquid introduced to a porous medium (such as stone) percolates through the atoms of the medium and attempts to fully wet the medium.

very high value (close to 1) to zero.

2.3.3 Analytical Models for FPGAs

In this sub-section, we present previous work that use analytical models either to optimize FPGA architectures or to relate architectural parameters to evaluation metrics. These works are different from the *estimation* techniques for implementation area, delay and routability; a summary of estimation techniques will be presented in a later section.

2.3.3.1 Modeling Area and Routability

Fang et al. [70] related the routing parameters of an FPGA to the minimum expected channel width. When deriving the model, this work assumed the average post-placement point-to-point wirelength to be constant at 4.43 and incorporated a few empirical relations. Lam et al. [104] related the architectural parameters to the number of LUTs, number of clusters and used inputs per cluster required to implement a design on an FPGA [104]. These two models are further discussed in Chapter 3. Gao et al. [77] proposed models to relate area and delay to LUT size. This work can be used only for unclustered LUT-based FPGAs. Brown et al. related the parameters describing the FPGA routing fabric to the routability of a design [25]. For a two-step router, Brown et al. modeled the routability for the detailed phase while assuming that the paths for the nets had been pre-defined by the global phase. We further discuss this model in Chapter 4.

2.3.3.2 Modeling Wirelength, Delay and Energy

Smith et al. [168] presented an analytical model for the post-placement wirelength for homogeneous FPGAs and extended this model for heterogeneous FPGAs. The wirelength model for homogeneous architecture is part of this dissertation and is presented in Chapter 3. This model is extended to heterogeneous FPGAs in [168] by considering three effects related to a heterogenous architecture: (a) placement constraints in heterogeneous FPGAs, (b) existence of dead-blocks [2], and (c) higher number of pins in embedded blocks as compared to the number of pins in logic clusters. Singh and Marek-Sadowska [161] presented an interconnect planning technique based on Rent's rule that could be used to allocate optimal segment lengths (1 to 4) across an FPGA architecture. This technique attempted to match the Rent coefficient of a clustered design (circuit) with the Rent coefficient of the underlying architecture. Their

[2]Dead-blocks are the blocks in an architecture that are left unused. [168]

work utilized previous works on circuit fanout and net-length distributions from the ASIC domain [196][3] to identify the optimized segment-length distribution for an architecture. Area-delay product for an architecture derived using this study was reported to be 10% better than a Xilinx-like device. Feng and Kaptanoglu [72] have used the concept of entropy to estimate an input interconnect block's (IIB) routing flexibility. This work enables designers to analytically evaluate different IIBs without going through an expensive place-and-route stage. Feng and Greene [71] has presented an interconnect entropy model to bound the number of required programming bits. Hung et al. [90] presented a physical delay model for FPGAs. Taking the process-dependent values for resistance and capacitance as well as FPGA architectural parameters as inputs, their work modeled intra-cluster delay and inter-cluster delay for a given FPGA architecture. A recent work by Rajavel and Akoglu [144] has related the FPGA architectural parameters to the energy consumption of FPGA devices. The authors of [144] used models presented in this dissertation as a basis for their model.

2.3.3.3 Modeling CAD Runtime

Using the characteristics of the CAD algorithms and the previously published analytical models, Chin and Wilton [41] modeled the effects of architectural parameters on the runtime for the placement and the routing stages of a CAD flow. They further demonstrated how designers could trade off the area of implementation to reduce the place-and-route run-time for an FPGA implementation.

2.3.3.4 Architecture Optimization by Using Convex Programming Tools

Recent studies have used convex programming tools to utilize the analytical models in optimizing FPGA architectures [164, 166, 167]. These studies use the analytical models presented in this dissertation and in Lam et al. [104] with the physical area and delay models for fast early-stage architecture evaluation. More specifically, these studies use a geometric programming (GP) framework to concurrently optimize both high-level (architectural) and low-level (transistor sizing) parameters. These optimization studies express the equation-based analytical models in the formats that are amenable to the GP framework [164].

[3]The work in [196] by Zarkesh-Ha et al. presented closed-form expressions for fan-out distribution of a random logic network.

2.3.4 Experimental Techniques for FPGA Wirelength, Delay and Routability

This section presents the studies on *estimating* the FPGA evaluation metrics: wirelength, delay and routability. These earlier studies estimated the evaluation metrics for given circuits implemented on given architectures. In contrast, this thesis presents equation-based expressions for the relationship between FPGA architectural parameters and evaluation metrics.

2.3.4.1 Wirelength and Interconnect Distribution

Several works estimate interconnect and wirelength for an FPGA implementation [15, 91, 131, 138]. For a given circuit implemented on an FPGA with given logic fabric parameters, Balachandran and Bhatia [15] estimated: (a) the pre-placement average wirelength for all nets, (b) the individual wirelength for nets and (c) the channel width required to route the circuits. Estimations of wirelength and maximum channel width in [15] used the information on circuit characteristics. Pistorius and Hutton [139] used Feuer's model [73] to the post-placement wirelength. Using partial least squares regression and software quality matrix on a high level C-like circuit description, Meeuws et al. [131] estimated the number of interconnects and wirelength. Although the number of interconnects estimated by that work followed the experimental results, the estimated wirelength was as much as 31% higher. Meeuws et al. [131] argued that the estimated wirelength was still acceptable, since the estimation had been made before the synthesis stage and could therefore guide the synthesis and later stages of the CAD flow.

2.3.4.2 Routability

Several studies have attempted to predict the routability of an FPGA implementation [27, 35, 99, 120, 175, 194]. Some of these techniques have been initially proposed for ASIC domain, but can be used to estimate FPGA routability as well. fGREP, presented by Kannan et al. [99], estimated the routability of placed circuits by using the routing alternatives that were available for each net. Logic block fanout was used as the measure for the routing alternatives in [99]. RISA [35] was based on a wiring distribution map. This map was used to analyze supply versus demand for routing resources that belonged to a region. RISA can be used for FPGAs by considering routing channels as regions. Lou's method [120] used a stochastic model for nets in estimating routability. In this technique, the chip is divided into regions, and the demand for routing the nets of a placed circuit is calculated for each region. For a region, the demand for a net will be dependent on the ratio of the number of shortest paths within that region and the total number of shortest paths that the net can use. Demand for a net is restricted only for the regions that lie within the bounding box of the net. To implement Lou's method for FPGAs, a region can be mimicked by an FPGA tile. Yang et al. [194] and Chan et al. [27] used Rent's rule to estimate routability of pre-placed circuits. Yang et al. [194] argued that the use of estimated routability during placement would contribute to a congestion-free layout.

2.3.4.3 Routing Delay

Techniques for estimating FPGA routing delay have been presented by several studies [100, 125, 126, 135]. These techniques focus on *estimating* delay that is dependent on a particular FPGA architecture and/or a particular set of user circuits. Manohararajah et al. [125, 126] presented a simple early timing model that used a lookup table with pre-recorded interconnect delays as a function of connection types. Manohararajah et al. [125, 126] used this model for physical synthesis. They found that the criticality values (for nets) computed from this model were 'almost as good' as the ones obtained from placement results. Nayak et al. [135] presented an estimator for area and delay that could be used for applications described in MATLAB.

2.3.4.4 Comments on Rent Co-efficient

While several analytical models and estimation techniques rely on the Rent's Rule for effectiveness, recent studies have identified the difficulty in measuring the Rent coefficient and the Rent exponent. Dambre et al. [56] finds that the Rent parameters can be measured in different ways, and the resulting Rent parameters may be far apart. For an FPGA implementation, Pistorius and Hutton [139] argued that the pre-placement partitioning-based Rent parameter measurement might be biased and "less natural", and suggested to sample the final placement regions when measuring the Rent exponent.

2.4 Summary

This chapter first presents a short description of FPGA architecture and related CAD tools, and discuss previously published studies aimed at improving different components of FPGA architectures and/or CAD tools. We then discuss previous work related to the contributions of this thesis. We present previously published studies on analytical models targeting FPGA implementations. We also present the analytical models that have been proposed for ASIC implementations, and are relevant to our work. We further present previous works on experimentally estimating selected evaluation metrics. While presenting our contributions in Chapter 3, 4 and 5, we will further detail some of these previous studies.

Chapter 3

Analytical Models Relating Architecture to Area and Delay

This chapter describes analytical models that relate the architectural parameters of an FPGA to area of implementation and critical path delay. More specifically, the models relate the logic fabric of FPGA architectures (cluster size, Look-Up Table (LUT) size and inputs per cluster) to the average wirelength after placement as well as the circuit depth after technology mapping and clustering. We show that these models, when combined with the previously published models [70, 90, 104], can be used to investigate the effects of architectural choices on area and delay. We also present results to validate the effectiveness of our models. Our models are validated using Versatile Place-and-Route (VPR) 5.0 [21, 122].

The remainder of this chapter is organized as follows. Section 3.1 presents the framework and assumptions that we use. Section 3.2 and 3.3 respectively detail our works on modeling area and delay respectively. Section 3.4 presents the validation results for our models as well as a discussion of the results. Finally, Section 3.5 summarizes this chapter.

The majority of this chapter has been published in [60, 61, 168].

3.1 Framework

We first describe the architectural and circuit assumptions that we make in this chapter and present the flows for modeling area of implementation and critical path delay. We then present the logic utilization model [104] that we use to develop the models in this chapter. Finally, we introduce the physical delay

Table 3.1: Model Parameters

Model Inputs	
Architectural Parameters: Logic Fabric	
K	Lookup table size
N	Cluster (logic block) Size
I	Inputs per cluster
Circuit Parameters:	
p	Rent coefficient
n_2	Number of 2-LUTs in a given circuit
d_2	Maximum depth of 2-LUT netlist of a given circuit
Inputs from Logic Utilization Model [104]	
n_k	Expected number of K-LUTs needed to implement a given circuit
n_c	Expected number of clusters needed to implement a given circuit
c	Expected number of LUTs packed in each cluster ($c = n_k/n_c$)
i	Expected number of inputs used in each cluster
o	Expected number of outputs used in each cluster
Input from Physical Area Model [70]:	
W	Expected channel width required to map a circuit
Inputs from Physical Delay Model [90]	
t_{intra}	Expected intra-cluster delay along critical path of a circuit implementation
t_{inter}	Expected inter-cluster delay along critical path of a circuit implementation
Other Input Parameters:	
γ	Average number of inputs *not* used in each LUT*
f_{max}	Maximum fanout of all nets in a circuit (From the work of [195])
f_{avg}	Average fanout of all nets in a circuit (From the work of [195])

Model Outputs:	
Implementation Parameters:	
d_k	Expected post-technology mapping depth of a circuit implemented on FPGA
d_c	Expected post-packing depth of a circuit implemented on FPGA
l_{avg_placed}	Expected post-placement average wirlength

* We detail later how we experimentally obtain γ.

models for area and delay [70, 90], which can be used with our models to estimate area of implementation and critical path delay.

3.1.1 Architectural and Circuit Assumptions

We assume an island-style FPGA architecture, as detailed in Chapter 2. Additionally, we make the following architectural and circuit assumptions in deriving our models:

1. We assume homogeneous FPGA architectures. In other words, we do not consider the presence of embedded blocks when we derive the models. However, Smith et al. [168] show that our wirelength model for homogeneous architectures can be extended to model the post-placement average wirelength of heterogeneous architectures.

2. While we aim to make our models as independent of circuit parameters as possible, we can not completely ignore the circuit properties and our models use three circuit dependent parameters.

First, we find that the Rent coefficient p has significant impact on the implementation of a circuit on FPGA. Secondly, to investigate the area efficiency of an FPGA implementation, we must assume a value for the number of 2-input gates n_2 in the original circuit; we define area efficiency by the ratio of the final area for an FPGA implementation and this parameter n_2. Finally, the estimation of delay efficiency requires the critical path depth of the original circuit. We use the maximum depth of 2-LUT netlist d_2 to capture this information and define the delay efficiency as the ratio of the implemented circuit's critical path depth and d_2.

As mentioned above, we try to minimize the number of circuit parameters in our model. The suitability of a general purpose FPGA architecture should not depend on circuit parameters, since we want our FPGA to implement as wide variety of circuits as possible. However, it may be possible to use our model to investigate application-specific FPGAs; in this case, the circuit parameters can be used to investigate the suitability of an architecture over a restricted range of these parameters.

In Table 3.1, we list those parameters, used by the models in this chapter.

3.1.2 Stages of Area and Delay Models

From Chapter 1.3.1, we break up the task of modeling area and delay into five stages that mimic the stages of a typical CAD flow: technology mapping, packing (clustering), placement, routing and physical design.

3.1.2.1 Stages of the Area Model

Figure 3.1 presents the stages associated with modeling the implementation area for an application that is mapped on the FPGA architecture under investigation. Out of the stages shown in Figure 3.1, this thesis models the expected post-placement average wirelength l_{avg_placed} and the routability of an average application that is mapped on the FPGA architecture under investigation. [1] We also investigate whether the empirical relations can be used to model the post-routing critical path wirelength l_{cp_routed}. We use results from prior work [70, 104] for technology mapping and packing stages and results from [70] for physical model stage. The routability model is part of this dissertation and will be presented in Chapter 4.

[1]We use the term post-placement wirelength to refer to the distances of all connections in a placed circuit. The actual wirelength after routing may be longer than this quantity due to routing congestion.

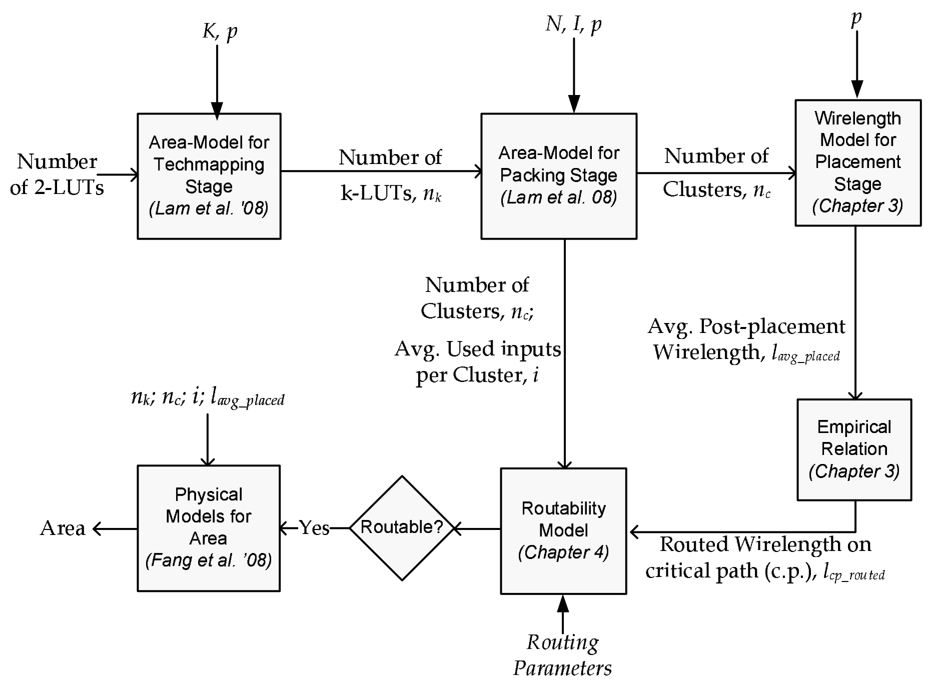
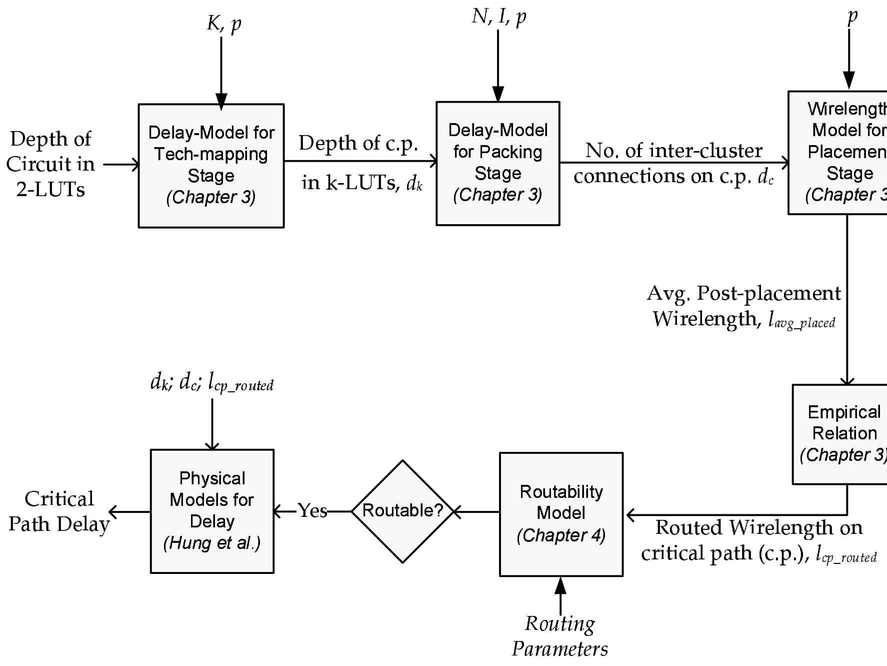

K, p
N, I, p
p
Number of 2-LUTs
Area-Model for Techmapping Stage (Lam et al. '08)
Number of k-LUTs, n_k
Area-Model for Packing Stage (Lam et al. 08)
Number of Clusters, n_c
Wirelength Model for Placement Stage (Chapter 3)
Number of Clusters, n_c; Avg. Used inputs per Cluster, i
Avg. Post-placement Wirelength, l_{avg_placed}
n_k; n_c; i; l_{avg_placed}
Empirical Relation (Chapter 3)
Area
Physical Models for Area (Fang et al. '08)
Yes
Routable?
Routability Model (Chapter 4)
Routed Wirelength on critical path (c.p.), l_{cp_routed}
Routing Parameters

Figure 3.1: Flow for Modeling Area of Implementation

3.1.2.2 Stages of the Delay Model

Figure 3.2 presents the stages that are used in modeling critical path delay. Critical path in an implemented circuit defines the longest path and bounds the maximum clock frequency that can be used. We specifically develop analytical models for technology mapping and packing (clustering) stages. During technology mapping stage, the basic gates of a circuit are mapped into K-input LUTs. We develop model for the post-technology mapping depth d_k, which is the longest path on the technology mapped circuit in terms of the number of K-LUTs along the path. During packing stage, closely related K-LUTs are packed into clusters with the goal of reducing area and/or delay of implementation. We develop model for the post-packing depth d_c. This parameter represents the longest path on the packing solution in terms of the number of clusters that the nets of the critical path pass through. We also present the wirelength model for the placement stage and the routability model for the routing stage, and investigate the applicability of empirical relations in modeling the routed critical path wirelength. We use results from an earlier work [90] for the physical model stage.



K, p
N, I, p
p
Depth of Circuit in 2-LUTs
Delay-Model for Tech-mapping Stage (Chapter 3)
Depth of c.p. in k-LUTs, d_k
Delay-Model for Packing Stage (Chapter 3)
No. of inter-cluster connections on c.p. d_c
Wirelength Model for Placement Stage (Chapter 3)
Avg. Post-placement Wirelength, l_{avg_placed}
Empirical Relation (Chapter 3)
Routed Wirelength on critical path (c.p.), l_{cp_routed}
Routability Model (Chapter 4)
Routing Parameters
Routable?
Yes
d_k; d_c; l_{cp_routed}
Physical Models for Delay (Hung et al.)
Critical Path Delay

Figure 3.2: Flow for Modeling Critical Path (c.p.) Delay

3.1.3 Usage of Models from Prior Work

We now present summaries of the earlier works on analytical modeling [70, 90, 104] that we use in conjunction with the models presented in this chapter.

3.1.3.1 Logic Utilization Model

Our equations require an estimate of the number of logic elements and clusters required to implement a circuit. We use earlier work by Lam et al. [104] for this purpose. Table 3.1 lists the outputs of the Lam model that are used as inputs to the models in this chapter. A brief summary of the models from [104] are presented below.

Equation for number of K-LUTs needed to implement a circuit, n_k Lam et al. [104] estimates n_k using the following equation:

$$n_k = n_2 \cdot \sqrt[p]{\frac{3}{K+1-\gamma}}, \tag{3.1}$$

where, n_2, p and γ are as defined in Table 3.1.

Lam et al. [104] observed that during the technology mapping process, all K inputs are not always used in each K-LUT and the difference between the available and used LUT-inputs can be modeled as γ. The behavior of γ as a function of K is extremely consistent across the MCNC and the QUIP benchmarks, and that there is a linear relationship between K and γ ($\gamma = \frac{1}{4}K - \frac{1}{2}$). Developing an analytical expression for γ is left as future work.

Equation for number of clusters needed to implement a circuit, n_c Two types of architectures are considered in [104] to estimate n_c: N-limited architectures and I-limited architectures. The architectures with a low value of N and a high value of I are referred to as N-limited architectures. In N-limited architectures, all N slots in a cluster are typically filled. On the other hand, architectures with a low value of I and a high value of N are referred to as I-limited architectures. In I-limited architectures, not all N slots in a cluster can be filled due to constraints imposed by the number of inputs to each cluster I.

The boundary between N-limited and I-limited architectures is defined by the following condition [104]:

$$I < N^p \cdot \frac{K+1-\gamma}{1+\frac{1}{f_{avg}}}. \tag{3.2}$$

Clustering is I-limited if Inequality 3.2 holds.

To calculate the average fanout f_{avg} in Equation 3.2, we first estimate the maximum fan-out of the circuit f_{max} using a formula from the work of Zarkesh-Ha et al. [195]. Zarkesh-Ha et al. first describe the expression for the fan-out distribution, which is the expected number of nets in a circuit that have a given fan-out. This fan-out distribution is used to estimate f_{max} in that work [195]. We adopt this formula for the FPGA implementation of a circuit:

$$f_{max} = \left[(I+N)\cdot\frac{n_k}{N}\cdot(1-p)\right]^{(1/(3-p))} = \left[\left(\frac{I}{N}+1\right)\cdot n_k\cdot(1-p)\right]^{(1/(3-p))}. \tag{3.3}$$

In Equation 3.3, $(I+N)$ represents the total number of pins (both inputs and outputs) for each cluster and n_k/N represents the minimum number of clusters required to implement the circuit. The required number of clusters may be higher for I-limited clustering as shown later. Since Rent parameter $p < 1$, Equation 3.3 tells us that f_{max} will increase if we increase either n_k or the ratio I/N.

The average fanout f_{avg} can then be calculated using the following expression [195]:

$$f_{avg} = \frac{1 - (f_{max} + 1)^{(p-1)}}{1 - (f_{max} + 1)^{(p-2)} - \phi(p, f_{max})} - 1, \tag{3.4}$$

where $\phi(p, f_{max})$ is:

$$\phi(p, f_{max}) = \sum_{j=1}^{f_{max}} \frac{j^p}{j^2 \cdot (j+1)}. \tag{3.5}$$

Using Equation 3.4, Zarkesh-Ha et al. [195] show that most of the nets in a circuit are 2- or 3-terminal nets. Furthermore, Lam et al. [104] experimentally finds that f_{avg} from Equation 3.4 is only a weak function of f_{max}, and the use of the minimum number of clusters in Equation 3.3 leads to only a small error.

Finally, using n_k and f_{avg} from above, Lam et al. [104] estimated n_c for the N-limited and I-limited architectures as:

$$n_c = \begin{cases} \frac{n_k}{N} & \text{for } N\text{-limited clustering} \\ n_k \cdot \sqrt[p]{\frac{K+1-\gamma}{I\cdot(1+\frac{1}{f_{avg}})}} & \text{for } I\text{-limited clustering} \end{cases} \tag{3.6}$$

where K, N, I and γ are model inputs as defined in Table 3.1.

Average number of inputs used in each cluster, i Lam et al. [104] used the following expressions to model i:

$$i = \begin{cases} \frac{(K+1-\gamma)\cdot N^p}{1+\frac{1}{f_{avg}}} & \text{for } N\text{-limited clustering} \\ I & \text{for } I\text{-limited clustering} \end{cases} \tag{3.7}$$

where, f_{avg} is as in Equation 3.4 and the other parameters are listed in Table 3.1.

3.1.3.2 Physical Area and Delay Models Usage

We use a model from Fang and Rose [70] to estimate the channel width requirement for a given circuit. More specifically, we use the following equation to estimate the minimum channel-width required to map a circuit on a given FPGA architecture [70]:

$$W_{min\_model} = \frac{1}{U} \cdot \frac{i \cdot l_{avg\_placed}}{2}, \tag{3.8}$$

where, U is the utilization factor that is empirically found to be 0.71 by Fang and Rose [70]. Average used inputs per cluster i can be estimated by using Equation 3.7 above and we use our wirelength model to estimate l_{avg_placed}. (Fang and Rose [70] used a fixed empirical value for l_{avg_placed}.)

We use the models from Hung et al. [90] to estimate: (a) the delay within a cluster for a net, t_{intra} and (b) the delay between the clusters for a net, t_{inter}. That work uses a RC-based model and uses the Elmore delay method to model delay across pass transistor chains. The values of capacitances and resistances for buffers, level restorers and multiplexers are collected from HSPICE simulations. The intra-cluster (within cluster) delay in that work is dependent on cluster-size N, LUT-size K and inputs per cluster I. The inter-cluster (between clusters) delay is dependent on routing fabric parameters such as channel width W, connection box flexibility F_{c_in} and F_{c_out}, switch box flexibility F_s and segment length L as well as logic fabric parameter N. Both delay components are also dependent on the paramters related to process technologies such as a transistor's equivalent resistance, intrinsic capacitance and gate oxide capacitance. We omit the equations from that work, but they can be found in Hung et al. [90].

3.2 Modeling Area of Implementation

We now describe the derivation of our model that relates FPGA architectural parameters to area. We first present our models for post-placement average wirelength, and then show how this information can be used in the area model.

3.2.1 Modeling Post-Placement Average Wirelength

In this sub-section, we derive a model for the expected pre-routing wirelength of nets for a circuit implemented on a homogeneous FPGA. In Section 2.3.2, we present the earlier studies that model wirelength as a function of the number of cells in a circuit that is implemented on an ASIC [62, 63, 73]. We observe that these models can be used for FPGA implementations, by treating either Basic Logic Elements (BLEs) or clusters as "cells". (Section 2.1 introduces BLEs and clusters). In this dissertation, we treat the clusters as cells. Clusters loosely represent the Logic Array Blocks (LABs) of Altera FPGA architectures and the Clustered Logic Blocks (CLBs) of Xilinx architectures.

We start with the wirelength model from Davis et al. [62, 63]:

$$l_{avg} = \frac{\frac{p-0.5}{p} - \sqrt{G} - \frac{p-0.5}{6\sqrt{G}(p+0.5)} + \frac{-p-1+4^{(p-0.5)}}{2p(p+0.5)(p-1)} G^{p}}{1 + \frac{-2p-1+2^{(2p-1)}}{2p(p-1)(2p-3)} G^{(p-0.5)} - \frac{p-0.5}{6p\sqrt{G}} - \frac{(p-0.5)\sqrt{G}}{p-1}}, \tag{3.9}$$

where l_{avg} represents the average wirelength for a two-pin connection on any path, G represents the number of cells in the circuit, and p represents the Rent coefficient of the circuit.

There are two types of nets in an FPGA implementation: (a) *intra-cluster nets:* nets that connect the BLEs inside a cluster and (b) *inter-cluster nets:* nets that connect the clusters. Our model for post-placement wirelength considers only the inter-cluster nets. For our applications, this makes sense, since it is the inter-cluster net that determines the amount of routing area that is required in an FPGA as well as the delay of an FPGA implementation.

We observe that the clusters of a post-packing (pre-placement) FPGA implementation are analogous to the cells considered for ASIC implementation in the work of Davis et al. [62, 63]. This observation leads us to use Equation 3.9 to model the average pre-routing wirelength for an FPGA implementation:

$$l_{avg(pin-to-pin)} = \frac{\frac{p-0.5}{p} - \sqrt{n_c} - \frac{p-0.5}{6\sqrt{n_c}(p+0.5)} + \frac{-p-1+4^{(p-0.5)}}{2p(p+0.5)(p-1)} n_c^{p}}{1 + \frac{-2p-1+2^{(2p-1)}}{2p(p-1)(2p-3)} n_c^{(p-0.5)} - \frac{p-0.5}{6p\sqrt{n_c}} - \frac{(p-0.5)\sqrt{n_c}}{p-1}}, \tag{3.10}$$

where $l_{avg(pin-to-pin)}$ represents the average wirelength for a two-pin connection on any path and n_c represents the number of clusters required to implement a circuit on a given FPGA architecture. We use Equation 3.6 to estimate n_c as a function of FPGA architecture parameters. In Section 3.4.2, we discuss the limitations of this wirelength model. In Section 3.4.2.3, we simplify Equation 3.10 without significantly affecting the accuracy.

3.2.1.1 Multi-Terminal Nets

Equation 3.10 models pre-routing wirelength for two-terminal nets. The nets of real circuits typically have more than two terminals. We use a relationship from the earlier work by Davis et al.[62, 63] to approximate l_{avg_placed}, the expected average post-placement wirelength for nets with more than one sink:

$$l_{avg\_placed} = l_{avg(pin-to-pin)} \cdot \frac{4 \cdot f_{avg}}{3 + f_{avg}}, \tag{3.11}$$

where the average fanout f_{avg} is from Equation 3.4.

3.2.1.2 Minimum Rectilinear Spanning Tree (MRST) Length and Minimum Rectilinear Steiner Tree Length

In later sections, we validate our model by comparing the model results to the experimental results from VPR. However, the wirelength models described above estimate the Steiner length for each net. Finding the minimum rectilinear Steiner tree length is an NP-complete problem [101] and heuristic based methods are available to approximate the Steiner tree length. A relationship from Hwang [93] gives us an upper bound for the maximum deviation of a heuristic based Steiner tree length from the minimum Steiner tree length [170]. This relationship shows that the minimum rectilinear Steiner tree length is at least two-third of the length of a MRST:

$$l_{rst} \geq \frac{2}{3} l_{mst}, \tag{3.12}$$

where l_{rst} represents the minimum rectilinear Steiner tree length and l_{mst} represents the MRST length. When validating our model in a later section, we use a technique from Wu and Chao [186] to approximate the Steiner tree length. For completeness, we also present the following relationship from Stroobandt [170] that provides both upper and lower bounds for Steiner tree length:

$$\frac{2}{3} l_{mst} \leq l_{rst} \leq l_{mst}. \tag{3.13}$$

3.2.2 Estimating Area of Implementation

Using the wirelength model derived above in conjunction with the earlier models [70, 104], we now model the area of implementation for a circuit on a given FPGA architecture. Our area metric is the total number of programming bits B_{prog}, required to map an application on the FPGA under investigation.

If n_c clusters are required to implement a circuit on a given FPGA architecture, the number of programming bits can be expressed by:

$$B_{prog} = Number\_of\_Prog\_Bits\_per\_tile * n_c, \tag{3.14}$$

where n_c can be obtained from Equation 3.6.

Number_of_Programming_Bits_per_tile can be found by adding three parameters: (a) programming bits per cluster $B_{cluster}$, (b) programming bits per connection block B_{CB} and (c) programming bits per switch block B_{SB}.

For a cluster with cluster size N and inputs per cluster I, the inputs to the multiplexer will be $(N+I)$. For a LUT size of K, the programming bits required for the cluster can be expressed by:

$$B_{cluster} = 2^K + 1 + \left[N \cdot K \cdot \left(\left\lceil \sqrt{N+I} \right\rceil + \left\lfloor \sqrt{N+I} \right\rfloor \right) \right], \tag{3.15}$$

Inputs to the multiplexers of a connection block will be $W \cdot F_{c_{in}}$ for given architecture parameters. For an architecture with I inputs per cluster, the programming bits required for connection blocks can then be expressed by:

$$B_{CB} = I \cdot \left[\left\lceil \sqrt{W \cdot F_{c\_in}} \right\rceil + \left\lfloor \sqrt{W \cdot F_{c\_in}} \right\rfloor \right], \tag{3.16}$$

Finally, since we assume that the connections are allowed on each side of a cluster in both horizontal and vertical directions (as in the experimental flow VPR [166]). The number of inputs to the switch block multiplexers for connections to the clusters will then be $N/2 \cdot F_{c_{out}}$; the multiplexer inputs for connections to the next switch blocks will be F_s. Total inputs to multiplexer switch blocks will be the summation of these two terms and the programming bits required for switch blocks can be expressed by:

$$B_{SB} = W \cdot \left[\left\lceil \sqrt{\frac{N \cdot F_{c\_out}}{2} + F_s} \right\rceil + \left\lfloor \sqrt{\frac{N \cdot F_{c\_out}}{2} + F_s} \right\rfloor \right]. \tag{3.17}$$

Equation 3.8 gives us the minimum channel width W_{min} required to implement a circuit. We increase W_{min} by 20% ($W = W_{min} * 1.2$) while estimating the area of implementation.

3.3 Modeling Post-Routing Delay

Recall from Chapter 1 that we break up our delay model into stages that are analogous to a typical CAD flow used for FPGA implementation. In this section, we first present two models: (a) a model for the length of the critical path after the technology mapping stage and (b) a model for the length of the critical path after the packing (clustering) stage. We then show how these models can be used to model the critical path delay for an FPGA implementation.

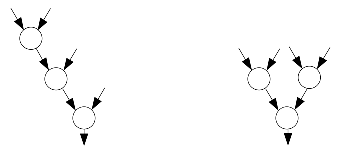


Figure 3.3: Two Possible Mappings for $K = 4$

3.3.1 Modeling Post-Technology Mapping Depth

We first describe a relation between the LUT size of an FPGA architecture, and the expected depth of a circuit after technology mapping. The inputs to this part of the model are the LUT size K and the depth of the circuit before technology mapping d_2. (d_2 is depth of the original circuit). The output is the depth of the circuit after it is mapped into K-input LUTs. This parameter is represented by d_k. We now detail how we derive the model for d_k.

Assume that the portion of an original circuit being implemented consists of 2-input nodes. Most technology mapping algorithms attempt to minimize the depth of the resulting implementation. However, the actual pattern of nodes covered by a single LUT depends on the structure of the original netlist of the circuit. Figure 3.3 shows two possible scenarios in which 2-input nodes are mapped to a 4-input LUT. The depth after the technology mapping stage for these two scenarios will be 3 and 2 respectively. For a K input LUT, the extremes (for reduction in depth with respect to the depth of 2-input nodes) can be generalized as $K-1$ and $log_2(K)$. For a large netlist, we would expect the "average" depth to be between these two extremes.

Recall from Section 3.1.3.1, typically not all K inputs to a K-input LUT are actually used and the expected number of *un*-used inputs in a K-LUT is represented by the parameter γ. Incorporating γ, depth values for the two possible extreme technology mapping solutions similar to ones shown in Figure 3.3 for K=4 can be written as $K-1-\gamma$ and $log_2(K-\gamma)$. We approximate that the average of these two extrema can capture the reduction of depth from d_2 to d_k, giving us the expression for d_k:

$$d_k = \frac{2 \cdot d_2}{K - 1 - \gamma + log_2(K-\gamma)}. \tag{3.18}$$

In Section 3.4.3, we will show that this simple expression matches the experimental results well.

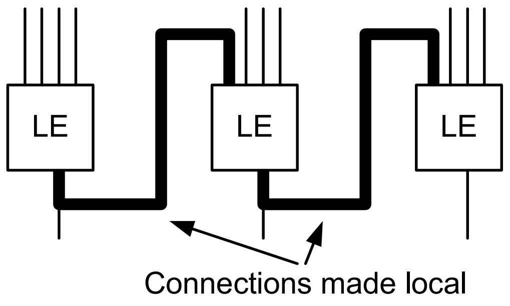


Figure 3.4: Cluster with Three Lookup-Tables

3.3.2 Modeling Post-Clustering Depth

Logic elements (LEs) are usually grouped into tightly connected clusters. Connections within a cluster are faster than connections between the clusters; the clustering (packing) stage attempts to include as many LEs within a cluster as possible. In this sub-section, we derive a relation between the FPGA cluster architecture and the depth of the circuits after they are mapped to clusters.

We derive this relation in two steps. First, we derive the expected proportion of all connections in a circuit that are made local after clustering and denote it as s_{ckt}. Intuitively, as cluster size is increased, more connections can be made local and hence s_{ckt} is increased. Second, we determine the expected proportion of connections *along the critical path* that are made local after clustering, which we will denote by s_{cp}. These two steps allow us to compute the expected number of inter-cluster and intra-cluster connections along the critical path of a given circuit.

Note that each connection in a circuit corresponds to one sink in a multi-terminal net, and represents one input to an LE. Thus, in this thesis, we count connections by counting the number of input pins of a LE, and *not* the output pins. A LE with $K-\gamma$ used inputs and one used output contributes $K-\gamma$ connections to the total connection count.

3.3.2.1 Proportion of Connections Made Local

Most clustering algorithms operate incrementally; that is, they choose a seed and iteratively add related LEs until the cluster is full [21]. Each time an LE is added to the cluster, additional connections are typically made local. These local connections can be one of two types: (1) those that are made local due to the optimization algorithm, and (2) those that are made local "by chance". We separately explain these two types of local connections and derive equations for them.

Consider a cluster consisting of a single LE with $K-\gamma$ used inputs. In such a cluster, the only way a net can be made local (becomes completely absorbed by the cluster) is if the output of the LE feeds directly back to one of its own inputs. Experimentally we have observed that this rarely happens, so we can approximate the number of local connections in this case as 0. Now consider adding additional LEs to the cluster. A timing-driven clustering algorithm would attempt to pack as many LEs along the critical path into a cluster as possible. This often leads to clusterings as shown in Figure 3.4, in which each LE receives an input from an LE that is already in the cluster. Thus, we have the number of connections made local by design $n_{ckt_{design}}$ as:

$$n_{ckt_{design}} = c - 1. \tag{3.19}$$

Recalling from Table 3.1 that the notation c represents the average number of LEs (LUTs) in a cluster n_k/n_c, if there are c LEs in the cluster, then the cluster has a total of $c(K-\gamma)$ connections. Of the remaining $c(K-\gamma)-(c-1)$ connections in the cluster, some will be global and some will be local. We assume that, apart from the $c-1$ connections described above, each connection in the implementation is equally likely to be made local. If there are n_k logic elements in the circuit, and if c of these are packed into each cluster, the probability that the logic elements for each connection is within the same cluster is c/n_k. This construction gives us the expected number of *additional* connections made local as

$$n_{ckt_{chance}} = \frac{c}{n_k}\left[c(K-\gamma)-c+1\right]. \tag{3.20}$$

Combining Equations 3.19 and 3.20 leads to the expected number of local connections n_{ckt} as:

$$n_{ckt} = (c-1) + \frac{c}{n_k}\left[c(K-\gamma)-c+1\right], \tag{3.21}$$

and since there are $c(K-\gamma)$ total connections in each cluster, the expected proportion of the connections made local after clustering s_{ckt} can be expressed by:

$$s_{ckt} = \frac{(c-1) + \frac{c}{n_k}\left[c(K-\gamma)-c+1\right]}{c(K-\gamma)}, \tag{3.22}$$

where $c = n_k/n_c$ and can be written as a function of the architectural parameters N and I and the circuit Rent coefficient p using the following results from [104]:

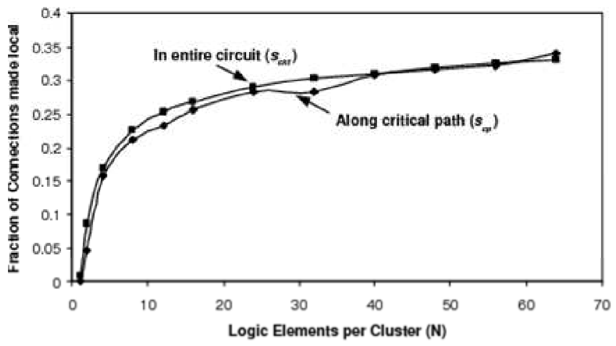


Figure 3.5: Comparison of s_{ckt} and s_{cp} from T-VPACK

$$c = \begin{cases} N & \text{if } I \geq N^p \frac{K+1-\gamma}{1+\frac{1}{f_{avg}}} \\ \sqrt[p]{\frac{I(1+\frac{1}{f_{avg}})}{K+1-\gamma}} & \text{if } I < N^p \frac{K+1-\gamma}{1+\frac{1}{f_{avg}}} \end{cases} \tag{3.23}$$

where the average fanout f_{avg} is given by Equation 3.4.

3.3.2.2 Connections Along the Critical Path

Equation 3.22 gives us the expected number of connections that are made local in a circuit. We now seek s_{cp}, which is the expected number of connections *along the critical path* that are made local after packing (clustering). Intuitively, a timing-driven packing algorithm will attempt to make more paths along the critical path local, compared to other paths, so we would expect $s_{cp} > s_{ckt}$.

We investigated this relation experimentally using two clustering tools: T-VPACK [21] and a replica of iRAC [161]. Both of these are greedy algorithms that pack LUTs into a cluster based on the closeness to the LUTs previously packed. iRAC uses less routing resources while nominally affecting the timing-quality of T-VPACK [190]. As shown in Figure 3.5 (which was obtained using T-VPACK), the values of s_{cp} and s_{ckt} are roughly the same for all values of N. The results from iRAC were similar. Based on these results, our model assumes $s_{cp} = s_{ckt}$.

The results of Figure 3.5 are counter-intuitive. We would expect the clustering algorithm to give preference to paths that are critical. However, as clustering proceeds, the criticality of paths are changed. Even if the criticality of a net is recalculated frequently, the problem of optimizing the wrong path during the early stages of clustering will still exist. This suggests that T-VPACK and iRAC are *not* optimizing the critical path well and are optimizing all paths approximately equally.

This suggests an interesting topic of future work: to find a better way to predict, ahead of time, which paths are actually going to be critical. The clustering stage can then pack the logic blocks along the critical path more efficiently. In such cases, s_{cp} is expected to be higher than s_{ckt}. Modeling the ratio of s_{cp} and s_{ckt} for such packing tools may be an interesting area of future research. Once such a model is available, it can be readily used with our proposed model for d_c.

Our model can be used for evaluating new packing algorithms as well. The equation that we present for s_{ckt} assumes that some connections are made local by chance. If an algorithm can predict the final critical path ahead of time, the value of s_{cp} should be higher than s_{ckt}. In other words, our expression for s_{ckt} can be used as a lower-bound for evaluating new packing algorithms.

3.3.2.3 Overall Model for Post-Clustering Depth

To summarize, the number of connections absorbed in the entire circuit s_{ckt} is equal to the number of connections absorbed along the critical path s_{cp}. Using the expression for s_{ckt} derived above, we can then model the number of clusters on the critical path (the post-clustering inter-cluster depth) d_c as,

$$d_c = d_k \cdot (1 - s_{cp}) = d_k \cdot \left[1 - \frac{(c-1) + \frac{c}{n_k}\left[c(K-\gamma) - c + 1\right]}{c(K-\gamma)} \right], \tag{3.24}$$

where c is given by Equation 3.23.

3.3.3 Modeling Critical Path Delay

Within each cluster, the critical path is expected to pass through d_k/d_c lookup tables. If we have estimates of the intra-cluster delay t_{intra} and the inter-cluster delay t_{inter} then the total critical path delay can be estimated as:

$$t_{c.p.} = d_c \left[t_{inter} + \frac{d_k}{d_c} t_{intra} \right] = d_c \cdot t_{inter} + d_k \cdot t_{intra}. \tag{3.25}$$

We can estimate the values of t_{inter} and t_{intra} in two ways. First, we can use the physical model from the work of Hung et al. [90] to estimate t_{inter} and t_{intra}. In Chapter 5, we will discuss how we use this technique to optimize architectures with respect to critical path delay. Secondly, we can follow the technique detailed in Das et al. [60], which will allow us to use the results from our wirelength model and the first phase of the timing-driven placement stage of a CAD flow (such as, VPR) to estimate t_{inter} and

Table 3.2: MCNC Benchmark Circuits

Circuit Name	n_2	d_2	Inputs	Outputs	Rent Coefficient*
Non pad-constrained circuits					
ex5p	1779	15	8	63	0.738
misex3	2557	13	14	14	0.714
apex4	2196	12	9	19	0.738
alu4	2732	14	14	8	0.662
tseng	1861	43	52	122	0.524
seq	2939	14	41	35	0.721
apex2	3165	17	39	3	0.743
diffeq	2556	39	64	39	0.554
s298	4272	32	4	6	0.560
spla	7438	19	16	46	0.726
frisc	6023	67	20	116	0.644
elliptic	5474	52	131	114	0.593
pdc	8408	19	16	40	0.748
ex1010	8020	17	10	10	0.749
s38584.1	12491	25	39	304	0.632
s38417	13656	25	29	105	0.591
clma	14253	40	383	82	0.726
Pad-constrained circuits					
dsip	2531	10	229	197	0.527
des	2901	14	252	243	0.646
bigkey	2979	10	263	197	0.517

*Rent coefficients shown are representative values (estimated for N=8, K=4)

Table 3.3: QUIP Benchmark Circuits

Circuit Name	n_2	d_2	Inputs	Outputs	Rent Coefficeint*
oc_aes_core_inv	25724	33	260	129	0.670
oc_aes_core	18178	25	259	129	0.673
oc_des_des3perf	78872	16	234	64	0.634
oc_video_compression_systems_jpeg_syn	78245	65	20	27	0.614

*Rent coefficients shown are representative values (estimated for N=8, K=4)

t_{intra}. The first phase of a timing-driven placement is much faster when compared with the total placement time. Since we prefer our model to evaluate delay without going through any stage of the CAD flow, we use the first technique from Hung et al. in estimating t_{inter} and t_{intra}. The interested readers may refer to the work of Das et al. [60] for details on the second technique.

Table 3.4: MCNC Benchmarks, Used to Measure γ for Model Validation

Circuit Name	n_2	Inputs	Outputs
C6288	1820	32	32
C7552	1781	207	107
i10	1668	257	224
apex3	1452	54	50
parker1986	1137	48	8

Table 3.5: γ Values from Five MCNC Benchmarks

K	2	3	4	5	6	7
γ	0.000	0.279	0.427	0.898	1.278	1.648

3.4 Validation of Models

To evaluate the accuracy of different components of our analytical models, in this section, we compare the model predictions to the experimental results, obtained using CAD tools. The first two sub-sections present the evaluation results for area equations and delay equations for twenty large MCNC [192] benchmark circuits listed in Table 3.2. The third sub-section examines the effects of the Rent coefficient on the validation results. The last sub-section presents the validation results for four large QUIP [9] benchmark circuits listed in Table 3.3.

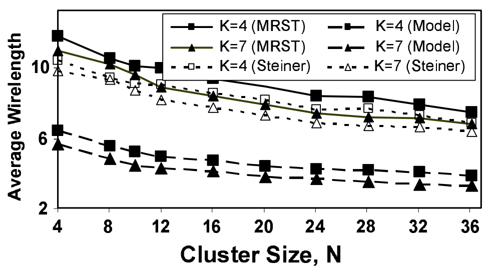
Our model equations require an estimate of Rent parameter. Different techniques have been previously presented for estimating Rent parameters [139]. Pistorius and Hutton [139] compare some of these techniques that uses the netlist after placement for calculation of the Rent parameter, and show that Rent parameters estimated by using different techniques differ in values. We use an inhouse tool, *bcgen*, to estimate the Rent parameters. *bcgen* uses recursive bipartitioning technique to estimate the Rent parameters. To estimate the Rent parameters for a clustered circuit, we use T-VPACK [21] to cluster (pack) the circuit and use the post-clustering netlist as an input to bcgen.

For validation, we also need the values of unused LUT inputs γ, a closed form expression for which is not yet available. We experimentally measure the set of γ values by using five MCNC benchmark circuits that are different from the 24 evaluation circuits listed in Table 3.2 and Table 3.3. These five MCNC benchmark circuits and the estimated values for γ are listed in Table 3.4 and 3.5.

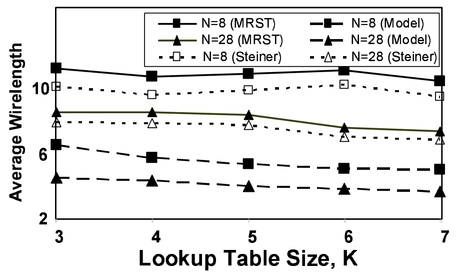
3.4.1 Validation of Wirelength Model

Figure 3.6 presents the validation results for our wirelength model presented in Section 3.2.1. We use our model equations to estimate the post-placement average wirelength. We collect the experimental results for wirelength from the placement stage of an academic CAD tool, VPR [21]. We use the version VPR 5.0 [122] for validation.

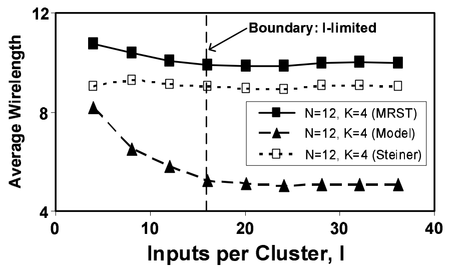
In Figure 3.6, we plot wirelength while sweeping three FPGA architectural parameters: cluster size N, LUT size K and inputs per cluster I. The routing segment length is fixed at 1. In all cases, we find that the model results underestimate the experimental results, but that the trends observed in the model results



(a) N-sweep, I=(K/2)(N+1)



(b) K-sweep, I=(K/2)(N+1)



(c) I-sweep

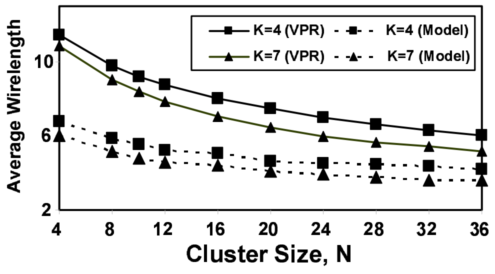
Figure 3.6: Verification of Wirelength Model for 20 MCNC Benchmarks

match the trends observed in the experimental results.

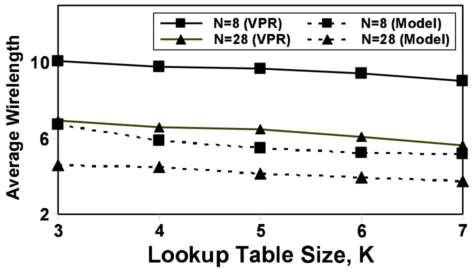
Our experimental setup reports minimum rectilinear spanning tree length (MRST). We use a heuristic-based technique presented by Wu and Chao [186] to approximate the Steiner tree lengths. To adopt this technique for our purpose, we start with the MRST reported by VPR and use the routing resources (such as switch blocks) on the traversal path of MRST as prospective Steiner points. In Figure 3.6, we find that the rectilineal minimum Steiner length is lower than MRST but is higher than the bound defined by Hwang [93] and presented in Equation 3.12. Furthermore, the model results are closer to the lower bounds from Equation 3.12 (results not shown). We make similar observations for all wirelength results. For the remaining sets of results, we only report the MRST length from VPR except for Figure 3.7(c). Due to the effects of pad-constrained circuits that we discuss later, the first point of Figure 3.6(c) for Steiner length is an outlier. Figure 3.7(c) will show that we do not have such an outlier for non pad-constrained circuits.

3.4.1.1 Effects of Pad-Constrained Circuits

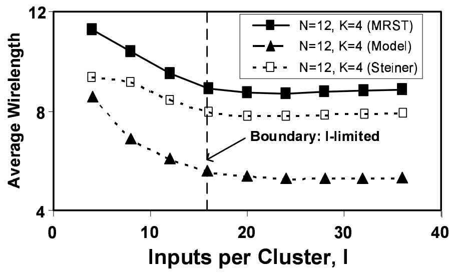
We have observed that the model results for the three pad-constrained circuits *des*, *dsip* and *bigkey*, do not follow the experimental results as closely as for the other circuits. As we observe from Table 3.2, the ratio of input/output pins and logic blocks is higher for the pad-constrained circuits. The device size for these circuits is dictated by the I/O ring [15] and the nature of the I/O ring will affect the wirelength as well.



(a) N-sweep, I=(K/2)(N+1)



(b) K-sweep, I=(K/2)(N+1)

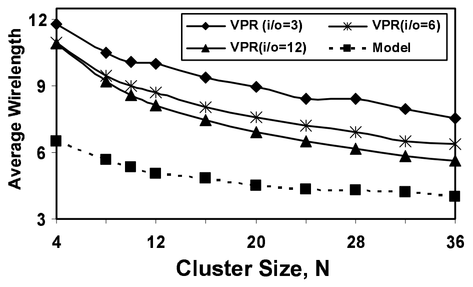


(c) I-sweep

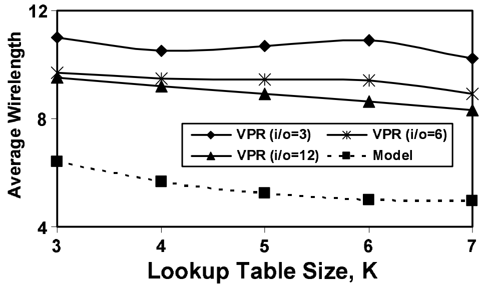
Figure 3.7: Verification of Wirelength Model for 17 Non-Pad-Constrained Benchmarks

Our models do not capture the I/O ring limitations and is therefore not capable of accurately modeling the wirelength for these circuits. Figure 3.7 shows the validation results for the set of seventeen circuits that does not include pad-constrained circuits.

In Figure 3.8, we further investigate the effects of I/O rings on post-placement wirelength. We investigate experimental (VPR) wirelength with 3, 6 and 12 I/Os per I/O pad. We find that with higher i/o capacity, the trends for the experimental results are closer to those for the model results; average wirelength consistently decreases with increasing *N* and *K*. Since our model does not consider the effects of



(a) N-Sweep, K=4, I=(K/2)(N+1)



(b) K-Sweep, N=8, I+(K/2)(N+1)

Figure 3.8: Effects of i/o Capacity on Wirelength for Twenty Benchmarks Including Pad-Constrained Benchmarks; i/o Capacity is the Number of I/Os Contained in the I/O Pads [179]

Table 3.6: Rent Co-Efficients, as a Function of LUT Size *K* (Cluster size, *N*=8)

LUT Size K	Non pad-constrained circuits					Pad-constrained circuits		
	ex5p	misex3	s38584.1	s38417	clma	dsip	des	bigkey
4	0.775	0.692	0.692	0.648	0.737	0.594	0.634	0.659
5	0.736	0.664	0.662	0.626	0.718	0.515	0.643	0.468
6	0.696	0.607	0.651	0.610	0.710	0.620	0.718	0.537
7	0.637	0.582	0.609	0.624	0.688	0.559	0.711	0.612

I/O rings, the plots in Figure 3.8 include only one set of model results.

The nature of pad-constrained benchmarks is further validated by investigating the Rent co-efficients for these circuits. In Table 3.6, as a function of *K*, we present Rent co-efficients for pad-constrained circuits as well as representative non-pad-constrained benchmarks. We find that Rent co-efficients for the pad-constrained benchmarks, in contrast to the others, do not monotonically decrease with increasing *K*. Since the modeled wirelength is a function of Rent coefficient, the wirelength for increasing *K* will not decrease as anticipated by our model.

3.4.1.2 Effects of *N*- and *I*-Limited Architectures

The architectures considered in Figure 3.6(a-b) and in 3.7(a-b) are *N*-limited (as defined in Section 3.1.3.1). Figure 3.6(c) and 3.7(c) include both *N*-limited and *I*-limited architectures. We identify the boundary between *I*-limited and *N*-limited architectures in these latter two figures. From the work of Lam et al. [104], the number of clusters becomes approximately constant beyond this boundary for increasing *I* and fixed *N*. We expect the estimated wirelength to be constant beyond this boundary. This is confirmed by the results in Figure 3.6(c) and in 3.7(c).

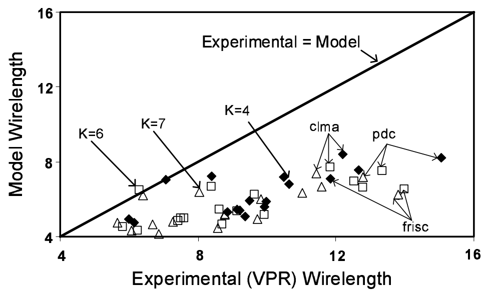


Figure 3.9: Verification of Wirelength for *N*=8 (Circuit by Circuit)

3.4.1.3 Circuit by Circuit Verification

Figure 3.9 shows a plot of the measured versus estimated wirelength for seventeen non pad-constrained benchmarks. For N=8, we have shown three datasets, for K=4, K=6 and K=7. This plot shows that the model results underestimate the experimental results. Furthermore, experimental results monotonically decrease for increasing LUT size, except for *frisc*.

While presenting the results for individual circuits, we note that the goal of our work is to capture the effects of architectural changes on the expected evaluation metrics for a *typical* application. Discrepencies shown by pad-constrained benchmarks (*dsip, des* and *bigkey*) or the outlier of Figure 3.9 (*frisc*) will therefore not affect the applicability of our model in early-stage architecture investigation.

3.4.1.4 Underestimation of Average Wirelength by Our Model

We find from Figure 3.6 and 3.7 that our model underestimates the experimental results in all cases. As we explain in Section 3.2.1, the model equations use Steiner-tree based wirelength estimation. In contrast, VPR estimates the wirelength based on minimum rectilinear spanning tree lengths. (Our Steiner length calculation from experimental results was based on a heuristic based technique.) In the next sub-section, we present a few other reasons that may contribute to the underestimation of average wirelength by our model.

3.4.2 Discussion on Results for Wirelength Model Validation

Earlier studies find that the wirelength models proposed for ASIC domain also typically underestimate experimental results. Lanzerotti et al. [108] find that, when applied to IBM POWER4 chip, the wirelength estimation techniques from Donath [67], Davis et al. [62] and Christie and Stroobandt [44] all underestimate the experimental results. Several earlier publications [54, 74, 108, 169, 193] discuss the reasons behind such discrepancies. We now present the relevant findings from these studies with special focus on Davis' model that we use to derive our model.

3.4.2.1 Issues Related to Estimation of Rent Parameters

Earlier studies find that the estimation of Rent parameters substantially affect the estimated wirelength [55]. Yang et al. [193] observes that when using the Davis model, designers typically estimate the *partition-level* Rent parameters; we also use partition-level Rent parameters to validate our model. For IBM-

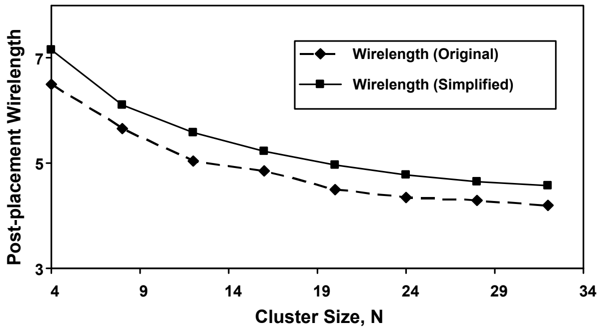
PLACE benchmark suites, they show that the placement-level Rent parameters are larger than the partition-level ones. They further demonstrate that the use of the *placement-level* Rent parameters instead of the partition-level Rent parameters significantly improves the quality of Davis' model, and in some cases, the modeled wirelength overestimates the experimental wirelength. For FPGA architectures, a later study by Pistorius and Hutton [139] finds that even the placement-level Rent parameters may significantly vary when the regioning methods for estimating Rent parameters are varied. ([139] presents three regioning methods to estimate placement-level Rent parameters for the FPGA implementation of circuits).

We use the partition-level Rent parameters to validate our model since our model is designed for early stage architecture design, in which designers will use our model to evaluate a wide range of architectures without going through expensive placement stages. It is interesting to note that the Rent parameters that we use for wirelength estimation is different from the Rent parameters that are used when deriving the model for the number of clusters required to map a circuit n_c, as the latter uses the Rent parameters of the circuit when implemented by 2-input lookup tables before clustering. We note that this discrepancy is an interesting issue that should be addressed for future architectural modeling techniques.

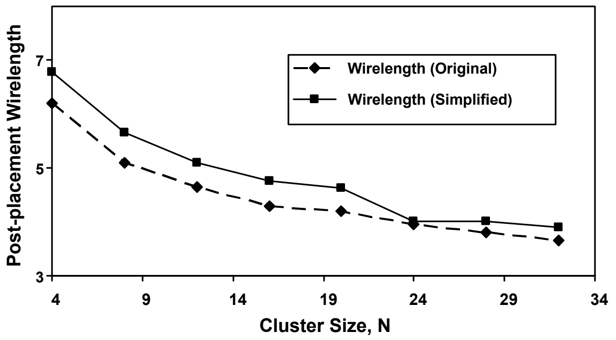
3.4.2.2 Other Issues with Wirelength Estimation Techniques

The earlier studies have found further issues with regards to using Davis' model for wirelength estimation. Lanzerotti et al. [108] and Stroobandt [169] observe that Davis' model is derived on the basis of two-terminal nets, and a linear net model is assumed from source to sink, where each net part is of equal length. The correction factor assuming this linear model has been presented as Equation 3.11 in this thesis. These studies [108, 169] observe that the assumption of a linear net model negatively affects the quality of results from Davis' model. Stroobandt [169] further observes that the use of a 'fan-out distribution' instead of the 'average fan-out for all nets' may improve the quality of Davis' model. Lanzerotti et al. [108] emphasised the necessity of considering rectangular regions along with square regions when deriving wirelength models.

Despite these issues, we observe that the proposed wirelength model can still enable the designers to evaluate the trends of the effects of architectural choices on average wirelength, and can be a valuable tool during early stage architecture investigation.



(a) N-Sweep, K=4, I=(K/2)(N+1)



(b) N-Sweep, K=6, I+(K/2)(N+1)

Figure 3.10: Nominal Effects of Using Simplified Expression for Wirelength Modeling

3.4.2.3 Simplified Form of Wirelength Model

We observe that some terms of Equation 3.10 do not have significant effect on modeled wirelength. Equation 3.26 presents a simplified form of Equation 3.10.

$$l_{avg(pin-to-pin)} = \frac{0.8 * n_c^{p+0.5} - n_c}{\sqrt{n_c} - 1.5 * n_c^p - \frac{(p-0.5)}{p-1} * n_c}, \tag{3.26}$$

In deriving Equation 3.26, we remove insignificant terms. For the remaining terms, we investigate whether a representative value of Rent parameter p can capture the effects of varying Rent parameters for the benchmarks. If that is the case, we use p=0.67 to further simplify the remaining terms. For instance, the value 0.8 in Equation 3.26 has been obtained by using p=0.67 on the expression $\frac{-p-1+4^{(p-0.5)}}{2p(p+0.5)(p-1)}$ in Equation 3.10. Figure 3.10 shows model results for the expected average post-wirelength for nets for multi-terminal nets for two cases: (a) where $l_{avg(pin-to-pin)}$ is estimated by using the original form in Equation 3.10 and (b) where $l_{avg(pin-to-pin)}$ is estimated by using the simplified form in Equation 3.26. We find that the results from these two cases are very close. (The results are averaged over twenty MCNC benchmarks.)

3.4.3 Validation of Delay Models

To evaluate the accuracy of our delay model, we again compare the model predictions to the results that we obtain from academic CAD tools. We separately validate the models for post-technology mapping depth d_k and post-clustering depth d_c. We use twenty large MCNC benchmark circuits, listed in Table 3.2. Although the value of N is typically between 4 and 16 in modern FPGAs, we validate our depth model

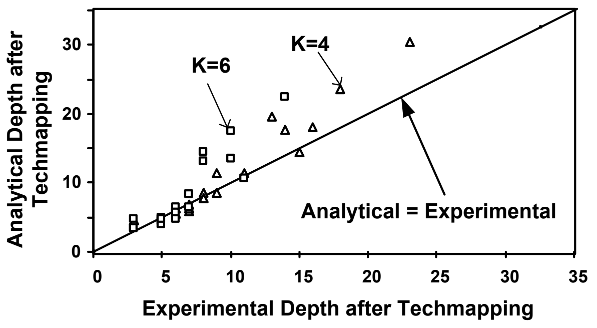


Figure 3.11: Model Verification for d_k (Circuit by Circuit)

Table 3.7: Standard Deviation of Estimating d_k for 20 Circuits

LUT-Size:	3	4	5	6	7
Std. Dev.:	3.45	2.15	2.87	2.57	2.35

for much higher values of N to examine its applicability in future generations of FPGAs.

3.4.3.1 Validation of the Model for Post-Technology Mapping Depth d_k

Measured results for d_k are gathered by recording the maximum depth for each benchmark circuit after it is technology mapped using Flow-Map [50]. Analytical results are obtained by using the measured d_2 and Equation 3.18. The d_2 values are measured from the 2-input netlist of the benchmark circuits and are presented in Table 3.2.

Figure 3.11 shows a plot of the measured versus estimated depth for each of the twenty benchmark circuits. We have shown two representative data sets, for K=4 and for K=6. The solid line with unit slope in Figure 3.11 represents the points where the predicted d_k values are equal to the measured d_k values. (Recall that d_k values are the number of K-LUTs along the critical path after technology mapping.) Due to close proximity of data values, some of the benchmarks overlap with each other both for K=4 and K=6 and data points for all twenty circuits are not visible in this graph. We fit lines to the data-points for K=4 and K=6. The slope for these two lines are 1.4 and 1.6 respectively with r^2 value of 0.94 and 0.83 respectively, where r is the correlation coefficient. Figure 3.11 shows that the prediction loses some accuracy as depth increases. Figure 3.12 plots the post-technology mapping depth along the critical path for different LUT sizes. Each point in this graph represents the arithmetic mean of the depth values across the benchmark suite. As these two graphs show, the analytical results track the experimental results closely.

We also examine the standard deviation of the differences between experimental and estimated values for different values of K, results for which are presented in Table 3.7. The absolute error between

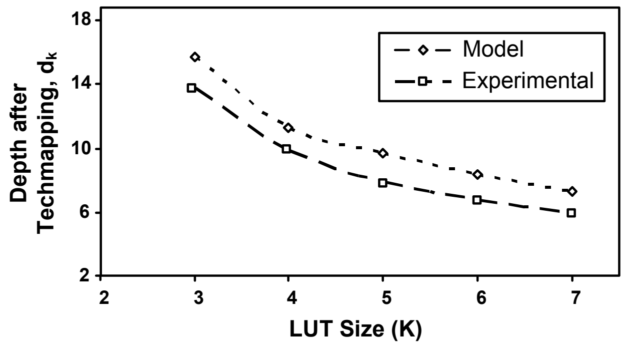


Figure 3.12: Model Verification for d_k (Averaged over 20 Circuits)

Table 3.8: Absolute and % Absolute Difference Between Experimental and Estimated Values of d_k

LUT-Size:	3	4	5	6	7
Absolute Diff.:	2.39	1.83	2.21	2.06	2.15
% Absolute Diff.:	15.24	16.25	22.81	24.68	29.33

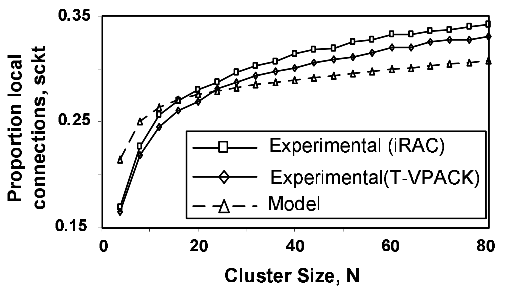
experimental and estimated values (averaged for 20 circuits) are presented in Table 3.8.

3.4.3.2 Validation of Model for Post-Clustering Depth d_c

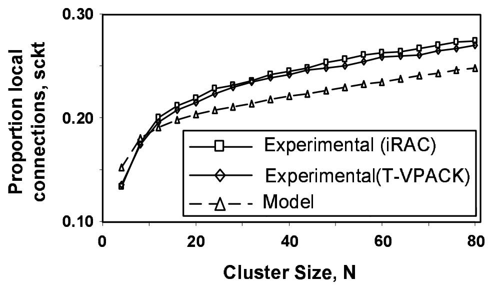
We first present validation results for s_{ckt} that represents the expected proportion of nets made local during the clustering phase. Figure 3.13 illustrates the accuracy of our model in estimating s_{ckt} for three representative values of K. We collected experimental results using two packing tools, T-VPACK [21] and a local implementation of iRAC [161]. As the graphs show, our model captures the experimental trends of both. However, for small clusters, our model overestimates the number of local connections, while for large clusters, our model underestimates. All of the cases in Figure 3.13 are for N-limited clustering, where $I = (K/2) \cdot (N+1)$. Discussion on these discrepancies will follow.

In all cases, however, the slopes for the results from our model are comparable to those for the experimental results, especially for higher values of K. An interesting observation is that for a LUT size of 7, results from T-VPACK almost coincide with the results from our model. This makes us believe that at this LUT-size, T-VPACK makes connections local *by design* and the rest of the local connections are absorbed within clusters *by chance*.

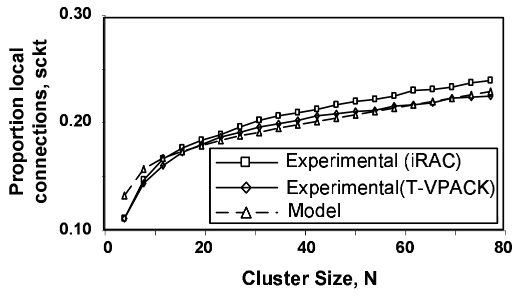
Finally, Figure 3.14 compares our model for post-clustering depth d_c to experimental results obtained using T-VPACK.



(a) K=4



(b) K=6



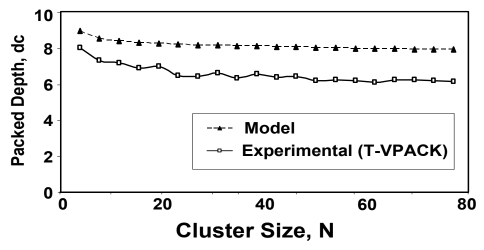
(c) K=7

Figure 3.13: Verification of Equation for s_{ckt}

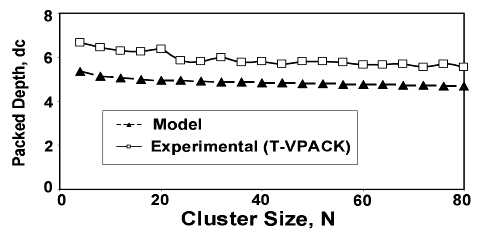
3.4.4 Discussion on Results for Delay Models Validation

The graphs of Figure 3.13 show that, for small clusters, our model overestimates the local connections, while for large clusters, our model underestimates. These discrepancies can be partially explained as follows. First, consider a small cluster with $N = 2$, $K = 4$ and $I = 6$. As shown in Figure 3.4, our model assumes that the clustering algorithm will always find a second LE that can use the output of the first LE. If the clustering algorithm chooses an LE with four inputs as the seed, the second LE will use the output from the seed and at most two more unique inputs. It seems likely that, often, the clustering algorithm will be unable to find such an LE, so it would instead choose an LE that shared the appropriate number of inputs, but not the output from the first LE. Our model then overestimates the local connections.

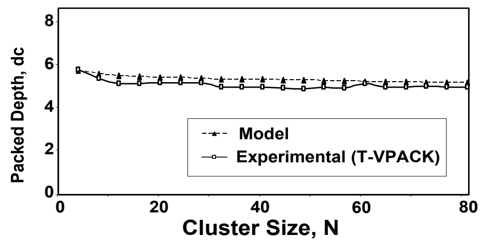
For large clusters, the situation is different. In such cases, it is possible that LEs may receive more than one input from a local LE (so adding a LE creates more than one new local connection). As a result, the number of connections made local by design will be more than $c-1$ that is assumed in Equation 3.19 and the experimental results will be higher than the predicted values.



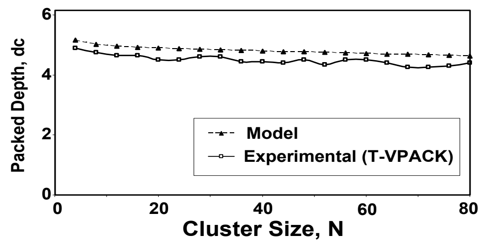
(a) K=4



(b) K=5



(c) K=6



(d) K=7

Figure 3.14: Verification of Equation for d_c

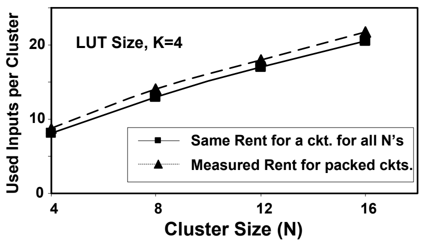
3.4.5 Effects of Rent Coefficient

This thesis also investigates the effects of Rent coefficient on the results from our models as well as the area models of Lam et al. [104]. We have examined the impact of the Rent coefficient's effect by using two sets of Rent coefficients for each circuit:

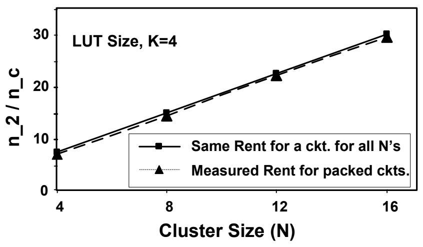
1. In the first case, for each circuit, we have used a fixed value of Rent coefficient p. We do not change this value with the changes in architectural parameters, such as N, K and I. We have listed these values of p for the MCNC and the QUIP benchmark circuits in Tables 3.2 and 3.3 respectively. This set of Rent coefficients have also been used to validate our model outputs.

2. In the second case, we measures the Rent coefficients for the packed circuits from T-VPACK by using our inhouse tool *bcgen* that uses a recursive bi-partitioning method. These measured values of p may change with the architectural parameters.

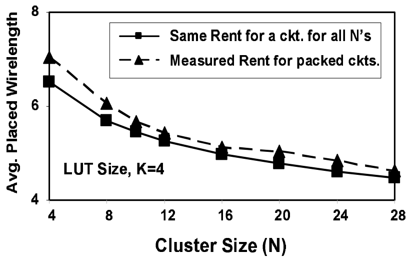
The differences between the maximum and the minimum values of the Rent coefficients for the latter range between 3% to 15% for all circuits except for *s298*, *dsip* and *bigkey*. We generate two sets of model outputs by using the above two sets of Rent coefficients separately. Figure 3.15 shows the results. Figure 3.15(a) to (c) show the effects of estimated Rent coefficients on the area models. In all cases, we find neglible differences between these two sets of results. Our delay equations depend on the Rent



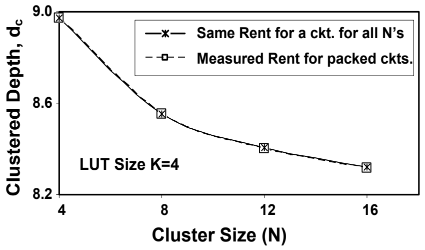
a) Used inputs per cluster vs. *N*



b) Logic packed per cluster vs. *N*



c) Average post-placement wirelength vs. *N*



d) Post-clustering depth vs. *N*

Figure 3.15: Effects of Rent Coefficient on Area and Delay Equations

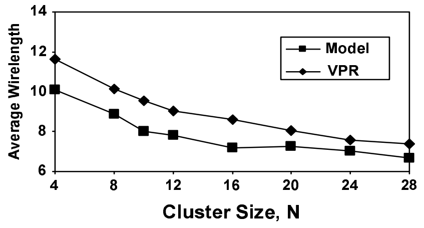
coefficient through the area parameters, n_c and i. Consequently, the changes in Rent coefficients will not have significant effects on on the estimated depth values. This argument is validated by the results presented in Figure 3.15 (d).

As we discuss in Section 3.4.2, Rent coefficients may still have prominent effect based on whether they are estimated from post-packing or post-placement netlist. Figure 3.15 however illustrates that the effects of Rent coefficients on the model outputs can be assumed to be negligible for early stage architecture evaluation. More accurate Rent coefficients may be estimated to further evaluate the architectures that are short-listed by early-stage evaluation.

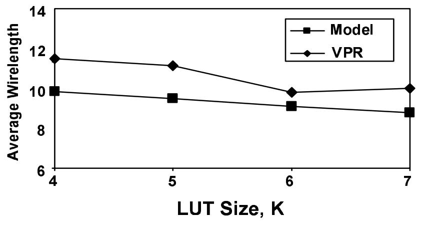
3.4.6 Validation for Large QUIP Benchmark Circuits

We also validate our wirelength and delay models using four additional QUIP benchmark circuits that are much larger than the MCNC benchmark circuits. The QUIP benchmark circuits are written in VHDL and Verilog and are provided by Altera Corporation [9]. Table 3.3 lists the four circuits that we use, which are the larger circuits in the QUIP suite.

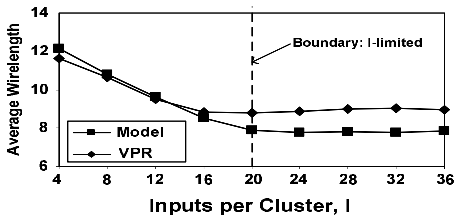
Figure 3.16 illustrates the accuracy of our wirelength model for those larger circuits. Results represent the average of the results for the QUIP circuits. Figure 3.16 corresponds to Figure 3.6 and Figure 3.7 that present wirelength results for the MCNC benchmarks. Figure 3.17 and 3.18 illustrate the accuracies of the



(a) N-sweep, K=4, I=(K/2)(N+1)



(b) K-sweep, N=8, I=(K/2)(N+1)



(c) I-sweep, N=12, K=4

Figure 3.16: Verification of Wirelength Model for QUIP Benchmarks

post-technology mapping and post-clustering depth models; and correspond to Figure 3.12 and 3.14(a) that present results for MCNC circuits.

For these large benchmarks, we find that the model results follow the trends of the experimental results fairly well. Furthermore, the conclusions that we can draw for the QUIP benchmarks agree with the ones drawn for the MCNC benchmarks. For instance, Figure 3.16(c) presents average post-placement wirelength with respect to inputs per cluster *I* for the QUIP circuits. Figure 3.6(c) and 3.7(c) present similar plots for 20 MCNC circuits and 17 non-pad-constrained MCNC circuits, respectively. In all of these cases, the trends of the model results are similar. The boundary between *I*- and *N*-limited regions is very similar for both sets of benchmarks. This demonstrates the depth model's applicability for larger benchmarks.

In Table 3.9, we present numerical comparison of accuracy for our depth models. Since the work of

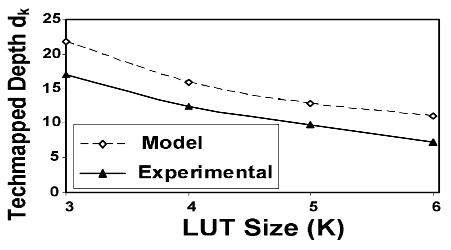


Figure 3.17: QUIP Benchmark Circuits: Model Verification for d_k

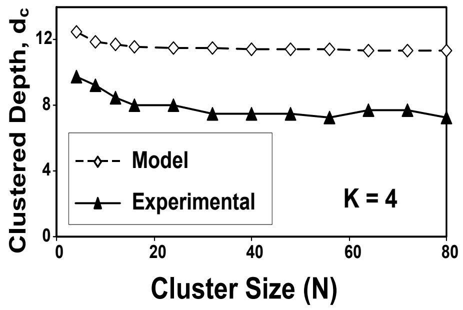


Figure 3.18: QUIP Benchmark Circuits: Model verification for d_c

Table 3.9: Comparison of Accuracy

	QUIP		MCNC		QUIP + MCNC	
	Model	Exp.	Model	Exp.	Model	Exp.
Cluster Size N: 8; LUT Size K: 4, Inputs per Cluster I: 18						
n_2/n_c	15.4	19.0	14.8	14.2	14.9	15.0
i	13.2	13.4	14.2	13.6	14.0	13.5
l_{avg_placed}	8.88	10.13	6.06	10.51	6.53	10.45
d_c	11.9	9.3	8.5	7.4	9.0	7.7
Cluster Size N: 24; LUT Size K: 6, Inputs per Cluster I: 75						
n_2/n_c	86.9	133.6	60.3	56.9	64.8	69.7
i	40.5	36.4	35.3	38.5	36.2	38.1
l_{avg_placed}	6.22	8.94	4.55	7.65	4.83	7.87
d_c	9.3	5.8	8.0	8.0	8.3	5.1

Lam et al. [104] has been used to derive our depth models, we also present the comparison results for the models from that work. Table 3.9 presents the numerical comparison for (a) QUIP benchmark circuits, (b) MCNC benchmark circuits and (c) the combination of QUIP and MCNC benchmark circuits, for two representative sets of values for $\{N, K, I\}$. We present the results for the total logic packed per cluster (n_2/n_c), the total used inputs per cluster (i), the average post-placement wirelength (l_{avg_placed}) and the depth after clustering (d_c). The values are averaged over the corresponding number of circuits.

3.5 Summary

This chapter described *analytical models that relate architectural choices to implementation area and critical path delay* for FPGA implementations. The major challenges tackled in deriving the equations were balancing accuracy with simplicity, and to understand the impact of the parameters used by the model, such as Rent coefficients. The models were validated against the commonly used academic CAD tool, VPR. Despite making simplifying assumptions when deriving the models, validation results show that the models can capture the trends of the experimental results.

The models from this chapter are used in later contributions of this dissertation presented in Chapter 4 and 5. We use the wirelength model as part of our routability model presented in Chapter 4. We use the expressions presented in Section 3.2.2 for area of implementation and in Section 3.3.3 for critical path delay in Chapter 5. We use these expressions to investigate the capabilities of the analytical model-based design approach.

Chapter 4

Analytical Model Relating Architecture and Routability

This chapter presents an analytical model to relate an FPGA architecture to the routability of applications mapped on that FPGA. While our model is focused on relating the FPGA routing fabric to routability, it is also capable of capturing the effects of the FPGA logic fabric on routability. We define routability as the expected proportion of the nets that can be successfully routed on a given FPGA architecture.

The implementation area and critical path delay that we model in Chapter 3 are dependent on both logic fabric parameters and routing fabric parameters. The models presented in Chapter 3 for area and delay assume that the circuits are routable for a given set of parameters, and estimate area and delay based on this assumption. However, a poorly designed FPGA routing fabric may not be flexible enough to route some circuits. The focus of this chapter is to investigate the routability of applications on a given FPGA architecture. Once an FPGA architecture is found to present sufficient routability for a typical application, the designers may use the models in Chapter 3 to investigate area and delay for that architecture. This argument is in line with the flows for model-based architecture design that we present in Figure 3.1 and 3.2 of Chapter 3.

In a typical experimental approach for designing a new routing fabric, FPGA architects begin with a large set of promising routing fabrics. They then change the routing parameters and run many iterations of expensive experiments to investigate the effects of such changes. The experimental results are evaluated empirically to fine-tune the routing parameters for the next iteration. Such empirical evaluation demands a high number of experimental runs. Each of these experimental runs incur substantial CAD run-time. It has been observed that as the number of circuit elements increases, the order of the run-time for the placement stage grows faster than linear [29, 40]. This issue is further aggrevated if the architecture designers are also interested in investigating the effects of logic fabric (such as, LUT size, cluster size etc.) on routability. In contrast to such an experimental approach, our work models the effects of a wide range of architectural parameters on the expected routability for a typical user application (we also show that our work can estimate the routability of individual applications fairly well). When used with the analytical models discussed in Chapter 3, our model can estimate the effects of architectural choices on routability, without going through the CAD flow. During early-stage architecture evaluation, this model thus allows the architecture designers to quickly evaluate a wide range of promising logic and routing fabrics.

The remainder of this chapter is organized as follows. Section 4.1 presents background for our model, assumptions that we make in developing the model and a brief summary of the stages that we use for model development. Section 4.2 details the development of our routability model. Section 4.3 discusses how some approximations that we use in developing the model can be relaxed. We validate our model results against the experimental results obtained from VPR 5.0 in Section 4.4. Finally, Section 4.5 summarizes the work presented in this chapter.

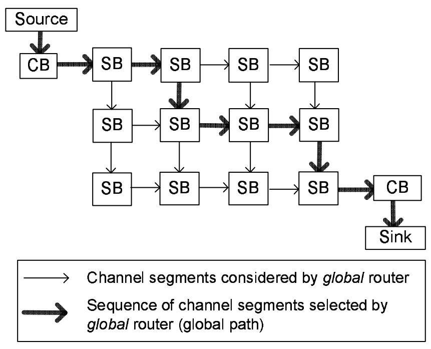
Parts of this chapter has been published in [58].

4.1 Overview of Routability Model

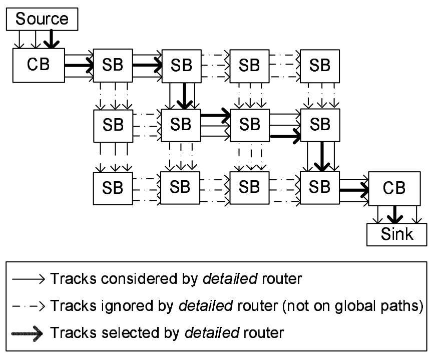
In this section, we first give a brief overview of algorithms that are typically used to route circuits on a given FPGA. We next put our routability model into context of the other studies on FPGA routability. We also present the approximations and assumptions made in deriving the routability model and finally give a brief overview of the three stages of our routability model.

4.1.1 FPGA Routing: Two-Step and Combined Single-Step Routers

As we explain in Section 2.2 of Chapter 2, once the clusters are placed on a given FPGA architecture by the placement stage of a Computer-Aided Design (CAD) flow, the nets are routed between these clusters during the routing stage of the CAD flow. FPGA routers that route the nets may be categorized into two types: two-step routers and single-step combined global-detailed routers [21]. We elaborate on these two


Source
CB
SB
Sink
Channel segments considered by *global* router
Sequence of channel segments selected by *global* router (global path)

(a) Step - 1: Global routing


Source
CB
SB
Sink
Tracks considered by *detailed* router
Tracks ignored by *detailed* router (not on global paths)
Tracks selected by *detailed* router

(b) Step - 2: Detailed routing

Figure 4.1: Behavior of a Typical Two-Step FPGA Routing (with Channel-Width W=3). CBs and SBs Respectively Represent Connection Blocks and Switch Blocks

types of routers below.

4.1.1.1 Two-Step Router

A two-step router works in two phases: *global* and *detailed*. Figure 4.1 illustrates these two phases. In this example, a two-terminal net (connection) is routed from the source logic block to the sink logic block through Connection Blocks (CBs), Switch Blocks (SBs) and routing channels. For brevity, we constrain the available routing paths by the minimum bounding box, defined by the locations of source and sink clusters. The global phase of a two-step router will assign a sequence of channel segments to route the net under consideration. In Figure 4.1(a), we assume that the global phase chooses the sequence of channels that are marked by bold arrows.

The detailed phase considers only the tracks within the pre-defined sequence of channel segments. Assuming the channel-width to be three, the routing tracks that a detailed router considers for our example net are presented by solid (both bold-solid and light-solid) arrows in Figure 4.1(b). Dash-dotted arrows in Figure 4.1(b) represent the routing tracks that are ignored by the detailed router even though they fall within the minimum bounding-box. For completeness, Figure 4.1(b) also shows the example final routing solution from the detailed router (marked by bold-solid arrows).

4.1.1.2 One-Step Combined Router

Figure 4.2 illustrates a one-step router. This figure uses the same example net from Figure 4.1. In contrast to the detailed phase of a two-step router, the combined single-step router of Figure 4.2 finds the complete routing path in one step by considering all available tracks.

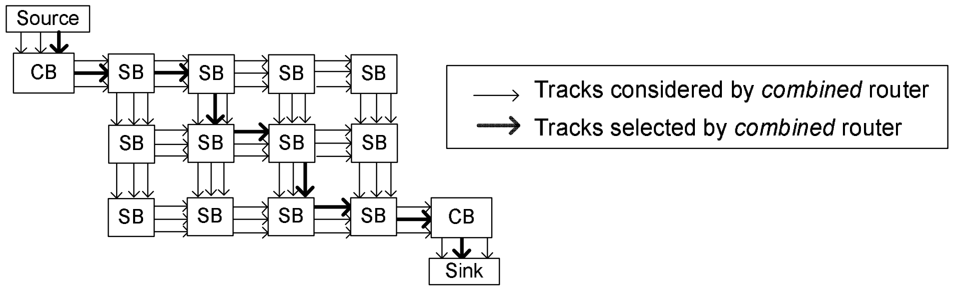

Source
CB
SB
CB
Sink
Tracks considered by *combined* router
Tracks selected by *combined* router

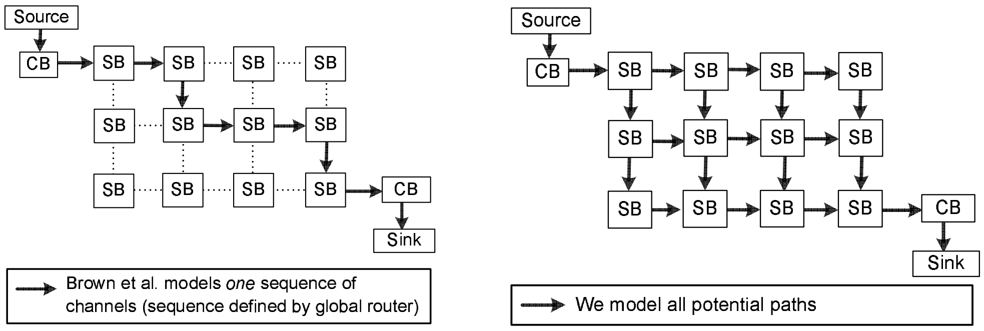
Figure 4.2: Behavior of a Typical One-Step Combined Global/Detailed FPGA Router (with Channel-Width W=3). CBs and SBs Respectively Represent Connection Blocks and Switch Blocks.

4.1.2 Context of our Routability Model

We now put our routability model for a combined router into the context of existing studies.

A number of publications present algorithms for two-step routers and one-step routers and several studies compare the performance of these two types of routers [21, 110, 114, 183, 187]. The majority of these studies find that the detailed phase of a two-step router is highly constrained by the pre-defined paths from the global phase, and that the required channel width for such a router is higher when compared to a combined router. Wu and Marek-Sadowska [187] illustrates how the detailed phase of a two-step router is forced to increase the required channel width and introduces the term *mapping anamoly* to represent this phenomenon. Furthermore, the global router does not have the details of routing obstacles or pre-routed nets, an issue that is more serious in FPGA routing since the types of routing resources may greatly affect a detailed router's performance [32]. Due to these reasons, combined routers are typically used to map circuits on modern FPGAs, both in academia and in industry.

From the above discussion, the routability of a circuit will be higher when a single-step router is used. The only previous model that relates architecture parameters to routability is from [25], which assumes a two-step router. In contrast, our model assumes a modern single-stage router. As we will show, this changes the formulation of the model significantly. In Figure 4.3(a), Brown et al. models the routability using only the sequence of channels that is pre-determined by the global step, such as the 'dark-solid' channel segments. The detailed router and hence Brown's model ignores the other potential channel segments, such as the 'light-dotted' ones. To capture the properties of a combined router, we consider all available paths as shown by dark-solid lines in Figure 4.3(b). As we will show, this changes the formulation of the model significantly.


Source
CB
SB
CB
Sink
Brown et al. models *one* sequence of channels (sequence defined by global router)
We model all potential paths

a) Work of [25]: detailed router b) Our work: combined router

Figure 4.3: Brown's Model and Our Work

4.1.3 Approximations and Assumptions

We observe that our problem is related to the problem of estimating the reliability of a multi-terminal stochastic network. For given network constraints, the reliability of such a network is measured by the existence of at least one useful communication path between the terminals (out of many possible paths). This is analogous to our problem, which is to model routability with given FPGA architecture constraints while assuming that the routing of the nets may use any of the many possible routing paths.

We make an important simplification by assuming that each net has only one sink. Clearly, real nets often have more than one sink, and the extension of our model to explicitly handle multi-sink nets is an interesting avenue for future work. As we will show in the results, even with this simplification, we get results good enough to short-list a set of interesting routing fabrics during early stage architecture investigation. To simplify our discussion we also assume that all nets are routed using their shortest path. We define the shortest path to be within the minimum bounding box found from the terminals of a two-terminal net. In Section 4.3.1, we discuss how this constraint can be relaxed. We further assume an island-style FPGA architecture that consists of an array of clusters and ignore the existence of embedded blocks.

4.1.4 Stages of the Routability Model

In this section, we give an intuitive introduction to the model; a detailed derivation is deferred to Section 4.2.

The inputs and outputs of the model are listed in Table 4.1. Inputs W, $F_{c_{out}}$, $F_{c_{in}}$, F_s, N_x and N_y describe

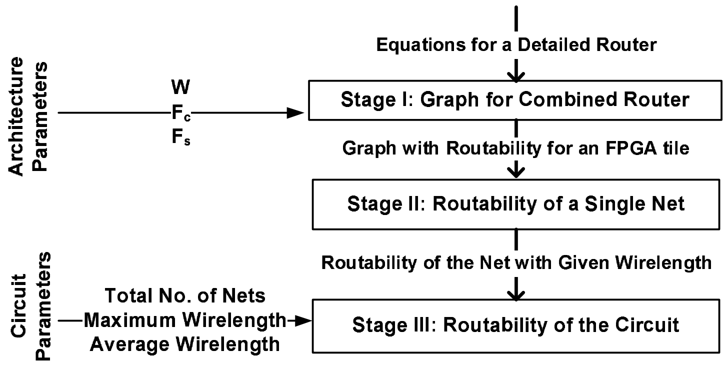

Equations for a Detailed Router
Architecture Parameters
W
F_c
F_s
Stage I: Graph for Combined Router
Graph with Routability for an FPGA tile
Stage II: Routability of a Single Net
Routability of the Net with Given Wirelength
Circuit Parameters
Total No. of Nets
Maximum Wirelength
Average Wirelength
Stage III: Routability of the Circuit

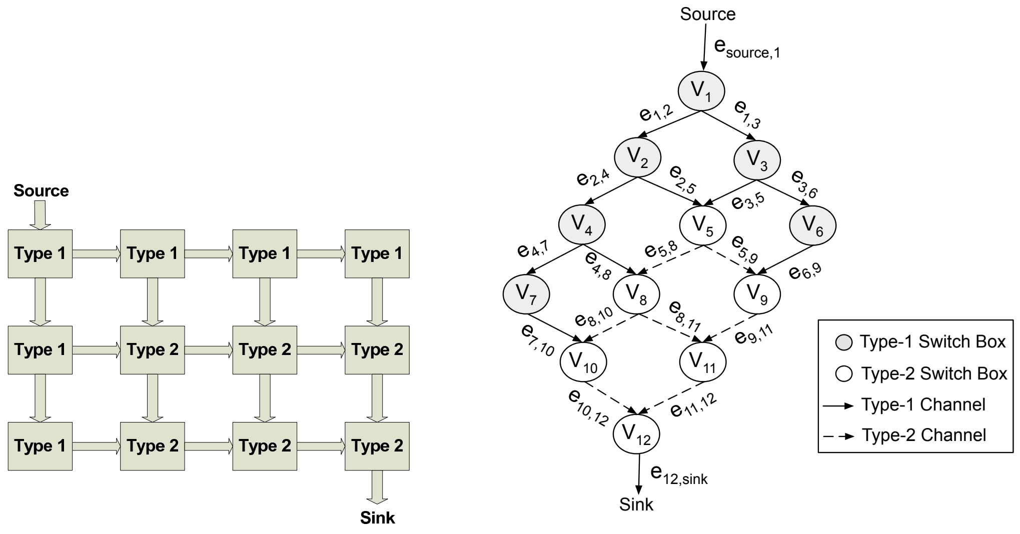
Figure 4.4: Overview of Routability Model

the architecture of the FPGA. Inputs $|\psi|$, l_{avg} and l_{max} describe the circuit to be implemented on the FPGA, and output $Pr[R_{ckt|comb}]$ is the overall routability of the FPGA when routing the circuit.

Our model consists of three stages, as shown in Figure 4.4. In the first stage, we construct a set of graphs G; each graph $G_l(V,E) \in G$ describes the routability of an FPGA when routing a net with wirelength l. Each node in the graph corresponds to a switch block or a connection block, and each edge corresponds to a routing channel. Each edge (directed) is associated with a weight which represents the probability that a net can be routed along the associated channel, given that it was successfully routed to the preceding switch block. The key challenge in this stage is estimating the weights. As described in Section 4.2, our estimations are based on the work by [25], with suitable modifications to account for the fact that in a two-dimensional grid, there is more than one way a net can arrive at a given channel.

In the second stage of our model, we estimate the overall routability for a net through the two-dimensional grid. We observe that our problem is related to the problem of estimating the reliability of a multi-terminal stochastic network. For given network constraints, the reliability of such a network is measured by the existence of at least one useful communication path between the terminals (out of many possible paths). This is analogous to our problem, which is to model routability with given FPGA architecture constraints while assuming that the routing of the nets may use any of the many possible routing paths.

In [16], Ball shows that finding the exact solution for the reliability of a stochastic network is an *NP*-hard problem, and suggests to look for approximate answers, such as reliability bounds. Several studies on the reliability problem use graph-theoric techniques to bound the reliability of communication between the terminals of the network [49, 69, 142, 155, 156]. As described in Section 4.2.2, we adapt the



(a) Two-dimensional routing problem (b) Graph representation of the problem

Figure 4.5: Two-dimensional Routing Problem and the Graph Representation.

technique from [155, 156] that uses the consecutive minimal cutsets of a stochastic network for bounding the reliability of systems.

Finally, the third stage of our model estimates the routability of the entire circuit. As described in Section 4.3, we do so by assuming that wirelengths in a circuit follow a geometric distribution.

Table 4.1: Model Parameters

Model Inputs - Architectural and Circuit Parameters	
W	Channel width (tracks per a routing channel)
$F_{c_{out}}$ ($F_{c_{in}}$)	Source (Sink) connection box flexibility
F_s	Switch box flexibility
N_x (N_y)	FPGA grids in columns (rows); total grids=$N_x.N_y$
$\|\psi\|$	Number of two-terminal nets (interconnects), ψ_is
l_{avg}[a]	Average post-placement wirelength of a circuit
l_{max}	Maximum post-placement wirelength of a circuit
Model Outputs - Implementation Parameters	
$Pr[R_{ckt\|comb}]$	Expected routability of a circuit mapped on FPGA using a combined router

[a]It may be noted that our wirelength model estimates pre-routing wirelength. It therefore does not need to consider the effects of the routing fabric, i.e. routability of the given FPGA architecture.

4.2 Model Formulation

4.2.1 Stage 1: Constructing the Routing Graphs

The first stage in our model is constructing a set of routing graphs. This stage is described in this subsection.

4.2.1.1 Graph Topology

We first construct the graph $G(V,E)$ to represent the FPGA routing fabric that lies between the source and sink of a given net. Figure 4.5(a) shows a region of the FPGA fabric upon which a net could be routed, and Figure 4.5(b) shows a graph that corresponds to this region of the FPGA fabric (the distinction between *Type-1* and *Type-2* switch blocks will be explained later). Each node $v_i \in V$ represents a switch block in the architecture. Source and sink nodes are also included to represent the source and sink connection blocks of the net. Each directed edge in $e_{i,j} \in E$ represents a routing channel that connects the switch blocks represented by v_i and v_j. Each edge has an associated weight that will be described in the subsequent subsections.

Note that in constructing the graph, we have assumed a particular wire length for the net (by assuming a particular start and end point on the grid). In the discussion in this section, we construct a graph for all possible wire lengths l for $1 \leq l \leq l_{max}$. We will later show that it is possible to calculate routability using a smaller number of graphs. We also assume that each net is routed from top-left to bottom-right; due to the symmetry in FPGAs, this will not affect our overall routability estimation. Finally, we assume that each net is routed using its shortest path and within its bounding box; in a later section we will relax this assumption.

Each edge $e_{i,j}$ in the graph is assigned a weight $p_{i,j}$ which indicates the routability of the routing channel between switch blocks i and j. More precisely, the weight for edge $e_{i,j}$ is the probability that a net traversing the channel $e_{i,j}$ would find at least one free track in the channel, given that it could be successfully routed to the input of the preceding switch block (vertex v_i). The weight for the edge $e_{\text{source},1}$ (which is the edge between the source node and the first switch block) is the probability that the net exiting the source logic block can find an available track in the first channel, and the weight for edge $e_{n,\text{sink}}$ is the probability that the net can be connected to the sink logic block input pin given that it has been successfully routed to the input of the last switch block (v_{12} in the example of Figure 4.5).

These probabilities depend on the architecture parameters of the routing fabric. Intuitively, the probability of finding a successful connection through a channel is low in an FPGA with a small amount of routing flexibility (small values of F_s and F_c); as the amount of routing flexibility increases, the probability of finding a successful connection increases. The probability of a connection also depends on the number of other nets that are routed on the fabric; if there are many nets routed, it is possible that even if a track is available, another net is using it. In the following discussion, we show how we estimate $p_{i,j}$ for all edges in the graph as a function of F_s, F_c, and the distribution of the excepted occupancy of each track in a channel.

4.2.1.2 Weight Estimation: Relevant Terminology

As described above, each weight of each edge corresponds to the probability that a connection can be made between successive switch blocks. Consider routing a net ψ_i with length $n+1$ along one possible routing path through switch blocks SB_1, SB_2, $\cdots$, SB_{m-1}, SB_m, $\cdots$, SB_{n-1}, SB_n. The channel on the outgoing side of SB_m will contain W routing tracks (W is termed the channel width). Successfully routing through SB_m is a *conditional* event that the net will find at least one free track on the outgoing channel of SB_m. This event is *conditioned* by the net's successful traversal through the source connection block as well as the preceding switch boxes SB_1, SB_2, $\cdots$, SB_{m-1}. The probability of this conditional event can then be represented by $Pr[S_m|S_{m-1} \cap \cdots \cap S_1 \cap X_1]$.

When routing through a switch block, not all of the W outgoing tracks may be available for two reasons. First, some of the outgoing tracks of SB_{m-1} may have been used by previously routed nets. Secondly, the construction of routing resources may not allow the routing of the net using any of the W tracks. For instance, if the connection block flexibility is 0.5, no more than 50% of tracks can be used to route a net through the source connection block and the subsequent switch blocks (assuming that a value of F_s=3 is used).

As in [25], we use three quantities (a, d and k) to capture these effects when estimating the routability through switch block SB_m (Figure 4.6). The quantity a is the number of outgoing tracks of the switch block SB_{m-1} that can be used to route the net ψ_i. In other words, any of a tracks can be used to route the net from SB_{m-1} to SB_m. The quantity d represents the outgoing tracks of SB_m that have been used by the previously routed nets $1, 2, \cdots, \psi_{i-1}$. The router can not use these tracks to route the net ψ_i through SB_m. Finally, the quantity k represents the number of outgoing tracks of SB_m that the router can use to route the

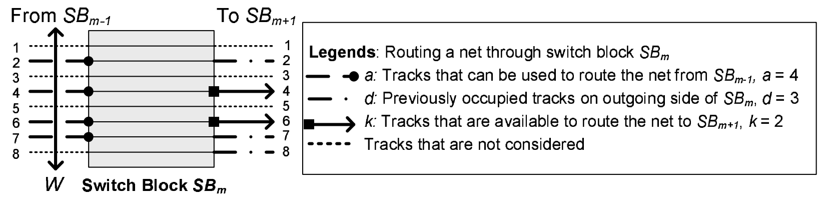

From SB_{m-1}
To SB_{m+1}
1
2
3
4
5
6
7
8
W
Switch Block SB_m
Legends: Routing a net through switch block SB_m
a: Tracks that can be used to route the net from SB_{m-1}, a = 4
d: Previously occupied tracks on outgoing side of SB_m, d = 3
k: Tracks that are available to route the net to SB_{m+1}, k = 2
Tracks that are not considered

Figure 4.6: Parameters Related to Routing a Net ψ_i Through Switch Block SB_m

net ψ_i through SB_m towards SB_{m+1}. From the definition of a above, when estimating routability through SB_{m+1}, k for the switch block SB_m will be equal to a for the switch block SB_{m+1}.

Figure 4.6 illustrates these quantities for routing a net ψ_i through switch block SB_m. Incident tracks to SB_m are from switch block SB_{m-1} on the left and the outgoing tracks from SB_m are to switch block SB_{m+1} on the right. The channel width W is 8 for this example and the tracks are numbered from 1 to 8. We consider the switch box with $F_s = 3$ constructed such that a track on the left is connected to a track with the same numbering on the right (1 to 1, 2 to 2 etc.)

In this example, any of tracks 2, 4, 6, 7 on the left can be used to route the net from SB_{m-1} to SB_m. We also find that three tracks on the outgoing side are used by previously routed nets. Out of these previously used tracks, the current net ψ_i could have used 2 and 7. From the definitions of a and d above, we have $a = 4$ and $d = 2$ for switch block SB_m. For the example in Figure 4.6, we have only two tracks to route the net through SB_m to SB_{m+1} (tracks 4 and 6) . Therefore $k = 2$ for switch block SB_m. Since a for SB_{m+1} equals to k for SB_m, we know that $a = 2$ for switch block SB_{m+1}.

4.2.1.3 Weight Estimation: Source to First Switch Block Weight

Using an equation from the work of Brown et al. [25], we estimate the probability of a successful connection through the source connection box by:

$$Pr[X_1] = \sum_{a=1}^{F_{c\_out}} \sum_{d=0}^{W} p\left(\lambda_g, d\right) \cdot \frac{{}^{d}C_{F_{c\_out}-a} \cdot {}^{W-d}C_a}{{}^{W}C_{F_{c\_out}-a} \cdot {}^{W-(F_{c\_out}-a)}C_a} \tag{4.1}$$

In this equation, ${}^{n}C_r = \frac{(n)!}{(r)!.(n-r)!}$ and the architectural parameters are as listed in Table 4.1. The term inside the summations represents that the probability that exactly a different connections can be made from the output pin of a logic block to a routing track in the first channel, given that d tracks in that channel have already been used. The summations sum over all legal values of a and d, giving the overall probability that a connection between the source pin and a track in the first routing channel can be made. For a detailed derivation of this equation, see [25].

In Equation 4.1, d represents the number of tracks in the first channel that have been used by the previously routed nets. The Poisson distribution $p(\lambda_g, d)$ models the effects of the number of tracks used by previously routed nets, on routing the current net. Since the number of tracks d, used by previously routed nets, changes after each net is routed, the parameter λ_g for $p(\lambda_g, d)$ may be updated for the current net j as [25]:

$$\lambda_{\psi_j} = \frac{1}{N_x.N_y} \sum_{k=1}^{j-1} Pr[\psi_k] \tag{4.2}$$

and, λ_g=$(\lambda_{\psi_j} \cdot l_{avg})/2$ with λ_{ψ_0} =$1/(N_x.N_y)$.

4.2.1.4 Weight Estimation: Switch Block to Switch Block Weights

We first make the distinction between *Type-1* and *Type-2* switch blocks. As shown in Figure 4.5(a), all switch blocks on the top and left side of the grid will be referred to as *Type 1* switch blocks, and the other switch blocks will be referred to as *Type 2* switch blocks. Type 1 switch blocks have the property that if they are used in the path, the net can enter the switch block from only one side (either the top or the left side, depending on its location in the grid). For Type 2 switch blocks, the net may enter the switch block from either of the *two* sides, the top and the left. The corresponding outgoing channels from the switch blocks are referred to as Type 1 and Type 2 channels. We calculate the routability of each type of switch block differently.

Type 1: For Type 1 switch blocks, the net will arrive from only one direction. In other words, there will be one set of incident tracks. $Pr[R_{Ch_{m|Type-1}}]$ represents the conditional probability for the successful routing along a Type 1 channel Ch_m. Since there is only one set of incident tracks on a Type 1 switch block, it is analogous to a switch block along a detailed routing path, as considered in [25]. Thus, we can use an equation from Brown to estimate the routability of a channel at a distance m from the source connection block:

$$\begin{aligned} Pr[R_{Ch_{m|Type-1}}] &= Pr[R_{Ch_m} | (R_{Ch_{m-1}} \cap \cdots \cap R_{Ch_1})] \\ &= \sum_{k=1}^{W} \sum_{a=1}^{W} \frac{Pr[A_a^{Ch_{m-1}}]}{\sum_{j=1}^{W} Pr[A_j^{Ch_{m-1}}]} \cdot \sum_{d=0}^{W} p(\lambda_g, d) \cdot \frac{{}^{d}C_{\alpha a-k} \cdot {}^{W-d}C_k}{{}^{W}C_{\alpha a-k} \cdot {}^{W-(\alpha a-k)}C_k} \cdot {}^{\alpha a}C_k \end{aligned} \tag{4.3}$$

where, routing along $Ch_{m|Type-1}$ is possible with exactly k tracks $(1 \leq k \leq W)$ and a is the number of tracks

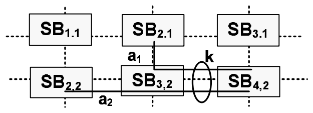


Figure 4.7: A Type 2 Channel for Combined Router

incident to the preceding switch box S_m. Equation 4.3 gives us the weights $p_{i,j}$'s for Type 1 channels, such as $e_{1,2}$ and $e_{1,3}$ in Figure 4.5. For Ch_1, a and j will range from 1 to $F_{c_{out}}$.

Type 2: For a Type 2 switch block, the net may arrive on either the top or the left side of the switch block. For the switch box $SB_{3,2}$ of Figure 4.7, there will be incident tracks from the switch boxes $SB_{2,1}$ and $SB_{2,2}$. To model the probability of successful outgoing connections along the channel connecting $SB_{3,2}$ and $SB_{4,2}$ (marked by the ellipse), we need to consider incoming tracks on $SB_{3,2}$ from both $SB_{2,1}$ and $SB_{2,2}$.

For an arbitrary Type 2 switch block, consider the case when the net traverses through the two preceding switch blocks with a_1 and a_2 available tracks respectively. We modify Equation 4.3 to model the routability through a Type 2 channel:

$$\begin{aligned} Pr[R_{Ch_m|Type-2}] &= Pr[R_{Ch_m}|(R_{Ch_{m-1}} \cap \cdots \cap R_{Ch_1})] \\ &= \sum_{k=1}^{W} \sum_{a=1}^{W} \frac{Pr[A_a^{Ch_{m-1}}]}{\sum_{j=1}^{W} Pr[A_j^{Ch_{m-1}}]} \cdot \sum_{d=0}^{W} p(\lambda_g, d) \cdot \frac{{}^{d}C_{\alpha a'-k} \cdot {}^{W-d}C_k}{{}^{W}C_{\alpha a'-k} \cdot {}^{W-(\alpha a'-k)}C_k} \cdot {}^{\alpha a}C_k \end{aligned} \tag{4.4}$$

where, $a' = a_1 + a_2$ if $a_1 + a_2 \leq W$ and $a' = W$ otherwise. (The total number of incident tracks on a switch block can not exceed the channel width W). We use Equation 4.4 to calculate the values of $p_{i,j}$ for Type 2 channels, such as $e_{5,8}$ and $e_{5,9}$ in Figure 4.5.

We make an approximation in deriving this equation. We assume that if a_1 and a_2 tracks are incident from either side, they will combinedly attempt to connect to $a_1 + a_2$ number of tracks on the outgoing side. This may not be the case since some incident tracks from the horizontal and the vertical directions may be connected to the same outgoing track of the switch box. Thus, our model may provide optimistic values for routability. The other approximation that we make is that the conditional events of success for the channels incident to a switch box are independent. The relaxation of these approximations may be an interesting topic for future research.

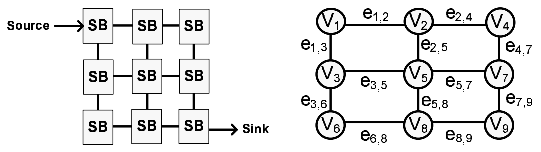


Figure 4.8: Example Two-Dimensional Routing Problem and Corresponding Graph

4.2.1.5 Weight Estimation: Final Switch Block to Sink Weight

We estimate the probability of successful connection through the sink connection box using:

$$Pr[X_2|S_n \cap S_{n-1} \cap \cdots \cap S_1 \cap X_1] = \sum_{k=1}^{W} \sum_{a=1}^{W} \frac{Pr[A_a^{S_n}]}{\sum_{j=1}^{W} Pr[A_j^{S_n}]} \cdot \frac{{}^{W-F_{c\_in}}C_{a-k} \cdot {}^{F_{c\_in}}C_k}{{}^{W}C_{a-k} \cdot {}^{W-(a-k)}C_k} \cdot {}^{a}C_k \tag{4.5}$$

Equation 4.5 does not contain d related term since the effect of d (the previously routed nets) for the channel connecting SB_n and the sink connection box is considered when we estimate the probability of successful connection through SB_n.

4.2.2 Stage 2: Overall Routability of a Single Net

The previous subsection constructs the routing graph $G(V,E)$ describing the potential multiple routing paths of a net and assigns weights to the edges of the graph. The next step is to use this graph to estimate the overall routability of the net, *while assuming a given length*. If there was only one path from source to sink, we could multiply all the weights (probabilities) along the path:

$$\begin{aligned} Pr[R_{\psi_i|l}] &= Pr[X_1 \cap S_1 \cap S_2 \cap \cdots \cap S_n \cap X_2] \\ &= Pr[X_1] \cdot Pr[S_1|X_1] \cdots Pr[S_n|S_{n-1} \cap \cdot \cdot \cap X_1] \cdot Pr[X_2|S_n \cap S_{n-1} \cap \cdots \cap S_1 \cap X_1] \end{aligned}$$

Our case is more complicated since there are many potential paths from source to sink, and we want to compute the probability when the net can use *at least one* of these paths. Our approach is to use methods from network reliability theory that bounds the reliability of a multipath network. Such techniques find the probability of having at least one successful path between the source terminal and the sink terminal, given that the failure rates of the links (edges) along the path are pre-determined. Our problem is similar if we consider a “failed link” along a potential routing path to consist of at least one channel segment that is too congested to route a net. With this construction, the failure rates of the edges can be obtained from the weights $p_{i,j}$’s that we have derived earlier.

4.2.2.1 Supplementary Definitions

In the discussion that follows, we will consider the simple routing problem and the corresponding graph, shown in Figure 4.8.

Probabilistic graph: The input to the network reliability theory is a probabilistic graph that represents the communication links (edges) between the source and the sink nodes as well as the probability of operation associated with each edge or node [48]. In our case, the graph that we form in Stage 1 will be the input probabilistic graph, with the probabilities of operation being associated with edges.

Pathset: A set of paths between the source and the sink nodes of a network (graph). In Figure 4.8, the pathset will contain the set of all potential paths for routing, such as, $\{V_1 - V_2 - V_4 - V_7 - V_9\}$, $\{V_1 - V_2 - V_5 - V_7 - V_9\}$, $\{V_1 - V_3 - V_5 - V_8 - V_9\}$ etc. By definition, the routing is successful when at least one of the paths functions.

Cutset: A set of cuts between the source and the sink nodes of a graph. Each cut is a set of directed edges. The removal of all edges of such a cutset will disconnect the source and the sink nodes. An example cutset for the graph in Figure 4.8 is $\{e_{2,4}, e_{2,5}, e_{3,5}, e_{3,6}\}$. If we remove all four of these edges, the source will be disconnected from the sink. In other words, the net becomes unroutable when all edges of *any* cut fails.

Minimal cutset: A set of minimal cuts for a graph. If any edge of a minimal cutset is removed, the remaining edges no longer form a cut. The example cutset $\{e_{2,4}, e_{2,5}, e_{3,5}, e_{3,6}\}$ above is a minimal cutset. For instance, if we remove $e_{2,4}$ from this cut, the resulting sequence of channels $\{e_{2,5}, e_{3,5}, e_{3,6}\}$ is no longer a cut. This is because even when the edges $e_{2,5}$, $e_{3,5}$ and $e_{3,6}$ of this cutset concurrently fail, we may still have successful routing through $\{e_{1,2}, e_{2,4}, e_{4,7}, e_{7,9}\}$. We may make similar observations for the remaining edges of this cutset; this is consequently a 'minimal' cutset for the example graph.

Set of consecutive cutsets: An ordered set of cuts, for which any edge that belongs to cutsets i and k ($i \leq k$) also belongs to cutset j, for all $i < j < k$ [156].

System with consecutive minimal cutsets: Any system for which the minimal cutsets can be ordered to form a set of consecutive cutsets [156]. The detailed routing network of an island-style FPGA is such a system.

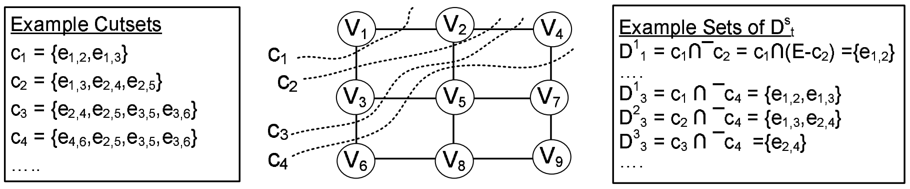

Example Cutsets
$c_1 = \{e_{1,2}, e_{1,3}\}$
$c_2 = \{e_{1,3}, e_{2,4}, e_{2,5}\}$
$c_3 = \{e_{2,4}, e_{2,5}, e_{3,5}, e_{3,6}\}$
$c_4 = \{e_{4,6}, e_{2,5}, e_{3,5}, e_{3,6}\}$
…..
c_1
c_2
c_3
c_4
V_1
V_2
V_4
V_3
V_5
V_7
V_6
V_8
V_9
Example Sets of D^s_t
$D^1_1 = c_1 \cap \overline{c}_2 = c_1 \cap (E - c_2) = \{e_{1,2}\}$
….
$D^1_3 = c_1 \cap \overline{c}_4 = \{e_{1,2}, e_{1,3}\}$
$D^2_3 = c_2 \cap \overline{c}_4 = \{e_{1,3}, e_{2,4}\}$
$D^3_3 = c_3 \cap \overline{c}_4 = \{e_{2,4}\}$
….

Figure 4.9: Examples of Consecutive Cutsets

4.2.2.2 Upper Estimate: Consecutive Minimal Cutsets of the Routing Graph

There are several ways to use cutsets to estimate the FPGA routability. Clearly if all edges in a cutset are not routable, then the network itself is not routable. Therefore, one approach to estimating routability would be to calculate a set of edge-disjoint cutsets, and determine the probability that all edges in any of these cutsets fail. This would, however, significantly over-estimate the real routability. To understand why this is so, consider Figure 4.8, in which $\{e_{1,2}, e_{1,3}\}$ and $\{e_{2,4}, e_{2,5}, e_{3,5}, e_{3,6}\}$ are two edge-disjoint cutsets. If edges $e_{1,2}$, $e_{2,4}$ and $e_{2,5}$ fail, our simple approach would conclude that the network is routable. However, from the figure, it can be seen that there is not a path from source to sink.

To achieve a better estimate, we use a technique from Shantikumar [155, 156] which provides an upper estimate for the routability of a net. To use this technique, we first determine the ordered set of r minimal cutsets C ($C \equiv c_1, c_2, \cdots, c_r$) for $G(V,E)$. The net is unroutable only when all the edges (FPGA routing channels) in at least one of the consecutive minimal cutsets fail. We explain the technique below, using Figure 4.9 as an example.

We first determine cut sets c_i for $0 \leq i < N$ where N is the number of nodes in the graph, excluding the source and sink nodes. The cut set c_i represents the set of edges from nodes $\{v_1, v_2, .., v_i\}$ to nodes $\{v_{i+1}, v_{i+2}, .., v_n\}$, where v_1 and v_n are connected to the source and the sink connection blocks. In Figure 4.9, n=9 and $c_4 = \{e_{i,j}$; where $i \in \{v_1, v_2, v_3, v_4\}$; and $j \in \{v_5, v_6, v_7, v_8, v_9\}\}$. The cut set c_4 thus represents the possible edges between $\{v_1, v_2, v_3, v_4\}$ and $\{v_5, v_6, v_7, v_8, v_9\}$; yielding $c_4 = \{e_{2,5}$, $e_{3,5}$, $e_{3,6}$, $e_{4,7}\}$ where $e_{i,j} \in E$.

We then compute D^s_t which is the set of edges connecting the nodes $\{1, \cdots, s\}$ and the nodes $\{s+1, \cdots, t+1\}$. This set is derived from the set of minimal cutsets c_i:

$$D^s_t = c_s \cap \overline{c}_{t+1}, s = 1, \cdots, t; \quad t = 1, \cdots, n-2 \tag{4.6}$$

where, $\bar{c}$ is the set of edges that belong to the set E, but not to c; $\bar{c} = E - c$.

Figure 4.9 lists some members of D_t^s for our example routing problem. For instance, from the definition above, we can define the sets of edges c_1 and c_4 as $c_1 = \{e_{1,2}, e_{1,3}\}$ and $c_4 = \{e_{4,6}, e_{2,5}, e_{3,5}, e_{3,6}\}$. Then, $\bar{c}_4 = \{e_{1,2}, e_{1,3}, e_{2,4}, e_{5,7}, e_{5,8}, e_{6,8}, e_{7,8}, e_{8,9}\}$, and

$$\begin{aligned} D_3^1 &= c_1 \cap \bar{c}_{(3+1)} = c_1 \cap \bar{c}_4 \\ &= \{e_{1,2}, e_{1,3}\} \cap \{e_{1,2}, e_{1,3}, e_{2,4}, e_{5,7}, e_{5,8}, e_{6,8}, e_{7,8}, e_{8,9}\} = \{e_{1,2}, e_{1,3}\} \end{aligned} \tag{4.7}$$

For a set of edges in D_t^s, we then find E_t^s, which represents the events that the edges in D_t^s will not function, implying that the corresponding FPGA channels will not successfully route the net. The probability of these failures is represented by β_t^s; so $\beta_t^s = Pr[E_t^s]$.

In our weighted routing graph $G(V,E)$, recall that a weight $p_{i,j}$ represents the probability of successful routing through a channel segment that is represented by the edge $e_{i,j}$. We can therefore write the failure rate of this channel, $q_{i,j}$, as $q_{i,j} = 1 - p_{i,j}$ For a set of edges D_t^s, β_t^s will be the product of the failure rates for all edges that are member of D_t^s:

$$\beta_t^s = \prod_{(i,j) \in D_t^s} q_{i,j} \tag{4.8}$$

Finally, we compute Q_t which accumulates the effects of failure events β_t^s. For a set of consecutive minimal cutsets $C(C \equiv c_1, c_2, ..., c_t)$, Q_t captures the event when all edges of at least one of these minimal cutsets fail. Since, β_t^s estimates the probability of the failure of all edges in cutsets, Q_t can be calculated by using the value of β_t^s from Equation 4.8:

$$Q_t = \sum_{s=1}^{t} (1 - Q_{s-1}) \beta_t^s; t = 1, ..., n-1 \quad and \quad Q_0 = 0 \tag{4.9}$$

Details of this formulation may be found in [155].

Similar to the technique presented by [156], we now use the expressions listed above, on our routing graph $G(V,E)$ to upper-bound the routability for our FPGA. A net ψ_i with length l will require $l-1$ switch blocks for routing along the shortest path. To calculate an upper estimate for the routability for this net, we first consider the successful routing along these $l-1$ switch blocks. This probability $Pr[R_{\psi_i|l-1}]$ can be expressed by:

$$Pr[R_{\psi_i|l-1}] \leq 1 - Q_{n-1} \tag{4.10}$$

where n is the total number of switch blocks along *all* of the possible paths for the net and Q_{n-1} can be estimated from Equation 4.9. (n is 9 in Figure 4.9).

Since we ignore the source and sink connection blocks while forming $G(V,E)$, the length in the subscript of the left side expression of Equation 4.10 is denoted by $l-1$, rather than by l. We now incorporate the probabilities of successful connections through these two connection blocks to model $Pr[R_{\psi_i|l}]$:

$$Pr[R_{\psi_i|l|comb}] = Pr[X_1] \cdot Pr[X_2] \cdot Pr[R_{\psi_i|l-1}] \leq Pr[X_1] \cdot Pr[X_2] \cdot (1 - Q_{n-1}) \tag{4.11}$$

where Equations 4.1 and 4.5 give us $Pr[X_1]$ and $Pr[X_2]$ respectively, representing a successful routing through source and sink connection blocks. Q_{n-1} is from Equation 4.9.

The output of the second stage of our model is $Pr[R_{\psi_i|l}]$ that represents the upper bound of the probability of successfully routing a net with given length l.

4.2.3 Stage 3: Routability of a Circuit with Many Nets on a Given FPGA

We now have the routability for a single net with given wirelength. Stage 3 uses this information to find the upper estimate of the routability for a circuit with many nets.

We first determine the routability for a net ψ that may assume any length within the range (1,l_{max}), with l_{max} being the maximum possible wirelength for a net of the circuit being mapped on FPGA. Similar to [25], we have:

$$Pr[R_{\psi_i}] = \sum_{l=0}^{l_{max}} Pr[\psi_i|l] \cdot Pr[R_{\psi_i|l|comb}] \tag{4.12}$$

where, $Pr[\psi_i|l] = p \cdot q^{l-1}$, with $p = 1/l_{avg}$ and $q = 1-p$. We can substitue $Pr[R_{\psi_i|l}]$ of Equation 4.12 by the upper estimates from Equations 4.11 to determine the routability of a net that does not have a wirelength constraint.

Earlier studies observe that wirelengths typically follow a geometric distribution [160, 197]. In our model, we also use a geometric distribution to estimate $Pr[L_l]$. In contrast, several studies use Rent's rule based techniques for wirelength distribution models [46]. We compare the values of $Pr[L_l]$ obtained by using these two approaches: (a) geometric wirelength distribution model that we use in our routability model and (b) Davis' model [62] that is a Rent's rule based wirelength distribution model . We compare the results from these two models and find that the values are comparable. We further find that $Pr[L_l]$

Table 4.2: Comparison of $Pr[\psi_i|l]$ Values for the Largest MCNC Benchmark: *clma*

Length l	VPR Post-Placement	Geometric Distribution	Davis' Model
1	0.091	0.114	0.672
2	0.283	0.101	0.226
3	0.182	0.089	0.118
4	0.120	0.079	0.073
5	0.079	0.070	0.050
6	0.056	0.062	0.037
7	0.040	0.055	0.028
8	0.027	0.049	0.022
9	0.023	0.043	0.018
10	0.017	0.038	0.015

Table 4.3: Values of $Pr[\psi_i|L]$ for a Typical Circuit

l_{avg}=4, p_ψ=0.25, q_ψ=0.75

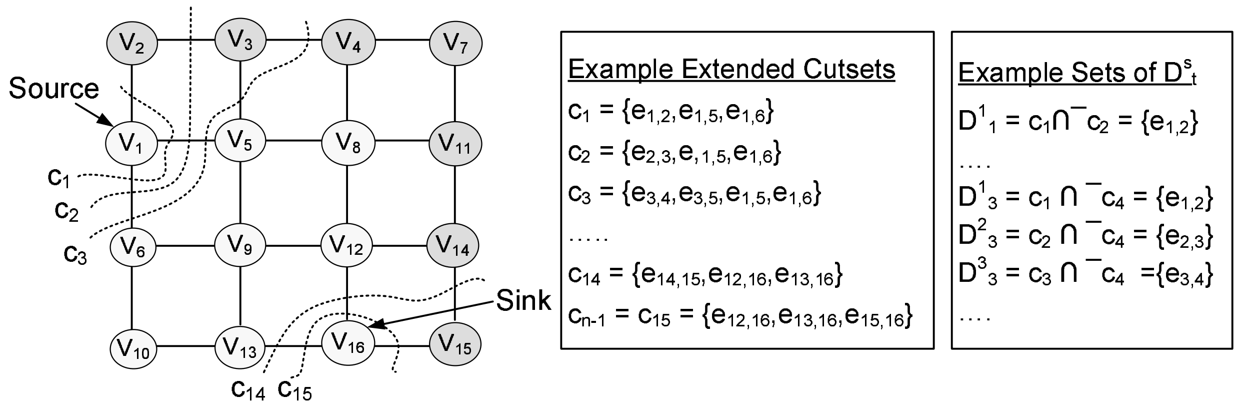
Length, l	1	4	10	15	50	100
$Pr[\psi_i\|L]$	0.25	0.105	0.004	2.5E-04	1.9E-07	1.1E-13

follows the post-placement wirelength distribution values from VPR 5.0. Table 4.2 presents comparison results for the largest MCNC benchmark *clma* for the first ten values of l. That being said, as explained in Chapter 3, accurate wirelength modeling is notoriously difficult; the goal of finding better wirelength models continues to be an interesting area of future research.

We make a further observation here. Due to the geometric nature of the wirelength distribution, the probability term $Pr[\psi_i|l]$ diminishes with increasing l. Subsequently, in Equation 4.12, the impact of the product $Pr[\psi_i|l] \cdot Pr[R_{\psi_i|l}]$ will be negligible for higher values of l. We can then ignore the higher values of l when calculating $Pr[\psi_i|l]$ from Equation 4.12. This observation allows the designers to use a lower value of l_{max} to speed up the routability estimation. For a typical circuit with $l_{avg} = 4$, Table 4.3 shows the diminishing nature of $Pr[\psi_i|l]$ as the length increases.

Finally, we model the routability of a circuit that contains many nets. We express the routability of a circuit with $|\psi|$ number of two-terminal nets ψ_i by $Pr[R_{ckt|comb}]$:.

$$Pr[R_{ckt|comb}] = \frac{1}{|\psi|} \cdot \sum_{i=1}^{|\psi|} Pr[R_{\psi_i}] \tag{4.13}$$


Source
Sink
V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8 V_9 V_{10} V_{11} V_{12} V_{13} V_{14} V_{15} V_{16}
c_1 c_2 c_3 c_{14} c_{15}

Example Extended Cutsets

$c_1 = \{e_{1,2}, e_{1,5}, e_{1,6}\}$

$c_2 = \{e_{2,3}, e_{1,5}, e_{1,6}\}$

$c_3 = \{e_{3,4}, e_{3,5}, e_{1,5}, e_{1,6}\}$

.....

$c_{14} = \{e_{14,15}, e_{12,16}, e_{13,16}\}$

$c_{n-1} = c_{15} = \{e_{12,16}, e_{13,16}, e_{15,16}\}$

Example Sets of D^s_t

$D^1{}_1 = c_1 \cap \bar{c}_2 = \{e_{1,2}\}$

....

$D^1{}_3 = c_1 \cap \bar{c}_4 = \{e_{1,2}\}$

$D^2{}_3 = c_2 \cap \bar{c}_4 = \{e_{2,3}\}$

$D^3{}_3 = c_3 \cap \bar{c}_4 = \{e_{3,4}\}$

....

Figure 4.10: Consecutive Cutsets for the Extended Graph. Shaded Nodes Represent Additional Nodes (Switch Blocks) When Shortest-Path Constraint is Removed.

4.2.4 Summary

For a net ψ_i with given length l, we form the routing graph $G(V,E)$ and assign weights $p_{i,j}$'s to the edges by using Equations 4.3 and 4.4. For $G(V,E)$, we follow the techniques presented in Section 4.2.2.2 to form the sets c_s and D^s_t; and the events E^s_t. Using these sets and events, we calculate β^s_t and Q_t from Equations 4.8 and 4.9 respectively. Equation 4.11 gives us the probability of successfully routing a net with given length. Equation 4.12 models the routability of a net without imposing any length constraint. Finally, we use Equation 4.13 to model the routability of the circuit.

4.3 Plausible Extensions of Our Model

This section illustrates two examples of how our routability model can be extended to relax the assumptions that we make during model development. Further extensions to our model are left for future research.

4.3.1 Relaxation of the Shortest Path Constraint

When deriving the model, we have constrained the routing paths for a net by the minimum bounding box. This constraint can be relaxed by extending the routing graph $G(V,E)$. Figure 4.10 shows an example of relaxing this constraint, where s and t represent the switch boxes connected to the source and the sink connection boxes respectively. The vertices with white background are within the minimum bounding box. Removal of the constraints will give us additional vertices, shown by the shaded nodes. Compared to Figure 4.9, we have additional and/or extended cuts for the routing graph in Figure 4.10. To model the routability, we need to consider these additional/extended cuts while using Equations 4.6 to 4.10. Figure 4.10 shows some examples of additional / extended cutsets. Also shown are example members of the set of edges D_t^s that have been determined using Equation 4.6.

It may be possible to relax the shortest-path constraint in stages. In such a case, we allow the model to use m units of length outside the minimum bounding box, and increment the value of m in stages. This will help evaluate to what extent the shortest path constraint must be relaxed in order to route a circuit that is unroutable with the shortest-path constraint.

4.3.2 Non-Island Style Architectures

In Section 4.2, we use an island-style architecture to present our routability model. However, the graph-theoric technique that we use to bound routability is not limited to island-style archtectures. Once a routing fabric is represented by a graph $G(V,E)$, our model can be used to model the consequent routability. In other words, to model the routability of an innovative routing fabric, we will need to (a) reconstruct the routing graph $G(V,E)$ to represent the fabric and (b) apply the graph-theoric techniques on this new graph.

4.3.3 Effects of Faulty Tracks

While designing a new FPGA routing fabric, it may be desirable to reserve some of the available tracks for use by embedded blocks. An application not requiring embedded blocks may not use these tracks. We further note that modern process technologies make the FPGA devices more prone to failure than prior technologies, and some of the tracks may not be available on a field-deployed FPGA device. These faulty tracks will also be unavailable when routing a circuit. The architecture designers may need to investigate the effects of such reserved and/or faulty tracks on routability; we suggest that our model can capture these effects.

Since reserved or faulty tracks may not appear uniformly across the FPGA array, simply running experiments with reduced channel width will not mimic the desired effects. However, we can readily extend our model to incorporate the information about reserved or faulty tracks. We do this by redefining d when using Equations 4.1, 4.3 and 4.4 (d is the number of tracks probabilistically used by the earlier routed nets). For instance, we can rewrite Equation 4.3 as:

$$Pr[R_{Ch_m|Type-1}] = \sum_{k=1}^{W}\sum_{a=1}^{W}\frac{Pr[A_a^{Ch_{m-1}}]}{\sum_{j=1}^{W} Pr[A_j^{Ch_{m-1}}]}\sum_{d_1=0}^{W} p(\lambda_g, d_1)\frac{{}^{d_2}C_{\alpha a-k}\cdot{}^{W-d_2}C_k}{{}^{W}C_{\alpha a-k}\cdot{}^{W-(\alpha a-k)}C_k}, \qquad (4.14)$$

where d_1 accounts for the track usage by the current circuit and d_2 is the sum of two values: (a) the tracks used by the previously routed nets of the current application d_1, and (b) the anticipated number of faulty tracks.

4.4 Validation

In this section, we investigate the performace of our routability model with respect to an academic CAD tool, VPR 5.0 [21, 122]. VPR uses the Pathfinder algorithm [130] for routing the nets. We use -*route_type* switch of VPR to ensure it uses combined global/detailed router.

We first use MCNC benchmarks [192] to compare the predictions from our model to the experimental results, obtained using T-VPACK and VPR 5.0. We also compare the predictions from Brown's model [25] to the experimental results for the combined router. We investigate the effects of varying routing and logic architecture parameters on both model results and experimental results. We also validate our model against much larger QUIP benchmarks [9]. We finally present results to illustrate the impact of relaxing the shortest path constraint.

4.4.1 Validation Methodology

This section presents the methodology that we follow to collect the experimental results and the model results.

4.4.1.1 Obtaining Model Results Without Going Through any Experimental CAD Stage

Table 4.1 lists the input parameters required for our model. Rather than using experimentally values for the input parameters N_x, N_y, l_{avg}, and l_{max}, we use previously published analytical models. This allows us to use model without going through *any* stage of a CAD flow and to compare purely analytical results to experimental results.

Specifically, we assume the grid-size $N_{xy} = \sqrt{n_c}$, where the number of clusters n_c is from [104]. We use the work of [168] to estimate average wirelength l_{avg}. Finally, we use an approximation for the maximum wirelength using the work of [143], which models the critical path wirelength as $2 \cdot N_{x,y} + (d_c -$

Table 4.4: Model Inputs for *spla*

Parameters from the post-placement stage of VPR	
Parameter	Value
Grid size N_{xy} (=N_x=N_y)	22
Avg. post-placemet wirelength l_{avg}	3.80
Max. post-placement wirelength l_{max}	60

Input parameters obtained from earlier analytical models	
Parameter	Value
Grid size N_{xy}=$\sqrt{n_c}$ [61]	22.9
Avg. post-placemet wirelength l_{avg} [168]	3.69
Max. post-placement wirelength l_{max} [60] [143]	66.2

$1) \cdot l_{avg}$, where d_c is the post-packing critical path depth for a circuit and can be obtained from [60, 61]. From Section 4.2.3, approximation of l_{max} will weakly affect the routability. In Table 4.4, we use these model-based techniques to estimate model inputs (for the benchmark *spla*), and compare them with the results from VPR. We find that these two sets of values are fairly close.

To obtain the results for Brown's model, we directly use that work's equations.

4.4.1.2 Obtaining Experimental Results

We collect the experimental results in the following manner. We first attempt to route the circuit in VPR using 50 iterations, and impose the minimum-path constraint in VPR by setting the *bb_factor* flag to 0. If, after 50 iterations, some nets are unroutable, we break up the multi-terminal nets into two-terminal nets. We then iterate through these two-terminal nets to investigate the resources that they use. If the 'occupancy' of any resource used by a net is higher than the 'capacity' of this resource, the corresponding net is marked as unroutable. After iterating through all of the two-terminal nets, we calculate the proportion (in %) of the nets that are routable for the given architecture.

This technique has some limitations. The ordering of the nets used to investigate the resource usages may affect the routability value. The limited number of routing iterations will also affect the routability values. Furthermore, the Pathfinder algorithm with VPR rips up all of the nets following an unsuccessful attempt (iteration). The results that we collect from the last unsuccessful iteration may not be the best one in terms of routability. For these reasons, our model will tend to overestimate VPR results. Despite these limitations, we believe that this definition of routability is sufficient for validating our model's predictions. While collecting experimental results, we set the placement seed of VPR at 1 and use a maximum of 50 iterations. Through experimentation, we have determined that these settings do not affect our overall conclusions (results not shown).

Table 4.5: Benchmark Circuits Used for Routability Model Validation

MCNC Benchmarks					
#	Circuit	Two-Terminal Nets	#	Circuit	Two-Terminal Nets
1	ex5p	2,142	11	s298	3,237
2	misex3	2,688	12	bigkey	2,468
3	apex4	2,460	13	spla	7,233
4	alu4	2,730	14	frisc	6,107
5	tseng	1,638	15	elliptic	5,737
6	seq	3,373	16	pdc	9,292
7	apex2	3,755	17	ex1010	9,139
8	diffeq	2,335	18	s38584.1	8,461
9	dsip	2,346	19	s38417	9,058
10	des	3,090	20	clma	14,673
QUIP Benchmarks					
#	Circuit	Two-Terminal Nets	#	Circuit	Two-Terminal Nets
1	oc_aes_core_inv	21712	3	oc_des_des3perf	55,683
2	oc_aes_core	16,681	4	oc_video_compression_systems_jpeg	35,994

Both experimental and model results are averaged over the set of benchmarks.

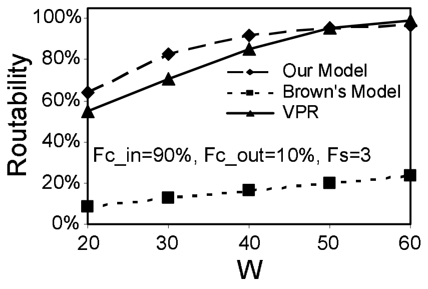
The results from VPR's placement algorithm may depend on some experimental settings such as the placement seed [179] and maximum number of iterations [179]. While collecting experimental results, we set the placement seed of VPR at 1 and use a maximum of 50 iterations. Through experimentation, we have determined that these settings do not affect our overall conclusions (results not shown).

4.4.2 Validation Results - MCNC Benchmark Circuits

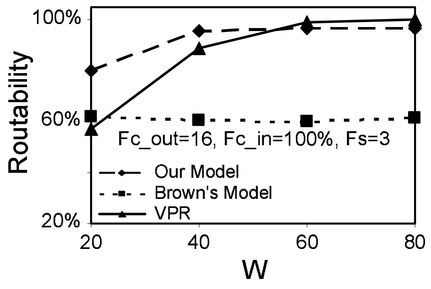
We first use twenty large MCNC benchmark circuits to validate our routability model. The circuits that we use are listed in Table 4.5. Figure 4.11 presents the validation results for LUT size K=4, cluster size N=8 and inputs per cluster I=18. For I, we use the equation $I = (K/2).(N+1)$, which gives the lowest value for I for 98% utilization of logic blocks [6]. The segment length L is fixed at 1.

Figure 4.11 shows that our model predictions follow the trends of experimental predictions with respect to routing parameters quite closely. We believe that this characteristic will help the FPGA architects to quickly investigate the absolute and/or relative effects of each routing parameter (and/or the combinations of multiple routing parameters), on the consequent routability. Figure 4.11 further shows that our model is more accurate than the earlier model by Brown et al. [25] in most cases, especially for the highly constrained architectures. We also find that the earlier model cannot properly capture the trends of the experimental results with respect to the changes in routing fabric.

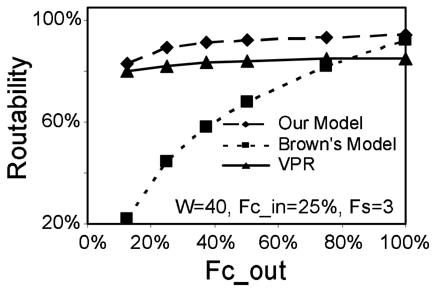
While validating the results with respect to one routing parameter (such as channel width W), we fix



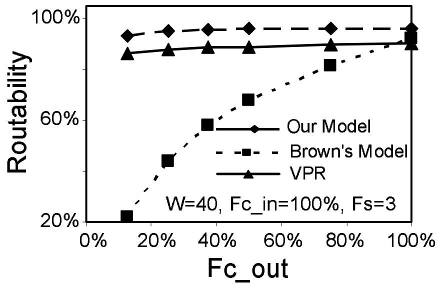
(a) Validation for *W*



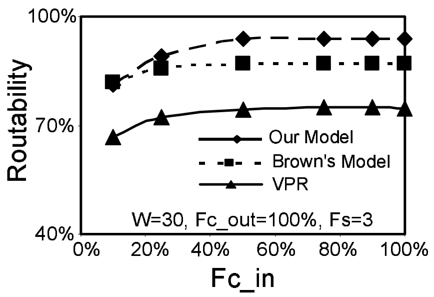
(b) Validation for *W*



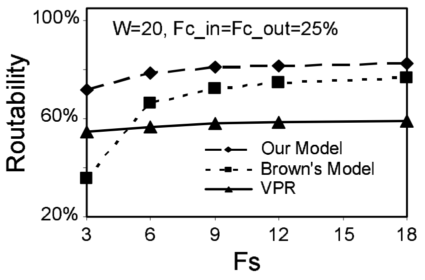
(c) Validation for $F_{c_{out}}$



(d) Validation for $F_{c_{out}}$



(e) Validation for $F_{c_{in}}$

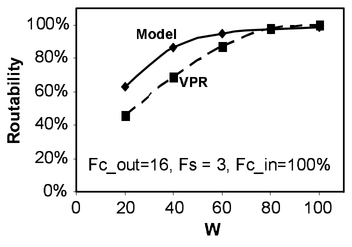


(f) Validation for F_s

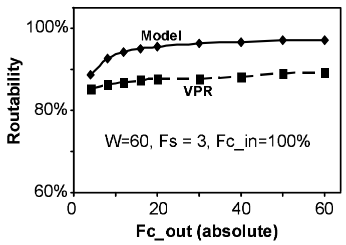
Figure 4.11: Validation for 20 MCNC Benchmark Circuits with K=4, N=8 and I=18

the values of the other parameters. We investigate whether our model results are affected when different sets of values are used to fix the parameters. For instance, both Figure 4.11(a) and Figure 4.11(b) present routability results while sweeping W, albeit for different sets of values for $F_{c_{in}}$, $F_{c_{out}}$ and F_s. We find that our model can capture the trends of the experimental results in both cases. We make similar observations when we compare the results in Figure 4.11(c) and 4.11(d). This further demonstrates our model's ability to follow the routability trends for different combinations of routing parameters.

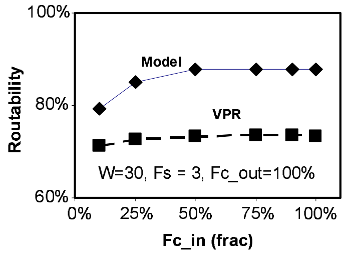
From Figure 4.11, it is clear that our model overestimates the experimental results, especially for the resource constrained architectures. From our discussion in Section 4.2 and 4.4.1, we identify three



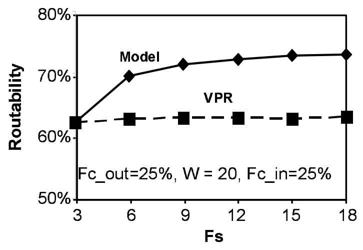
(a) Validation for W



(b) Validation for $F_{c_{out}}$



(c) Validation for $F_{c_{in}}$



(d) Validation for F_s

Figure 4.12: Impact of Logic Fabric: Results with K=6, N=16 and I=51

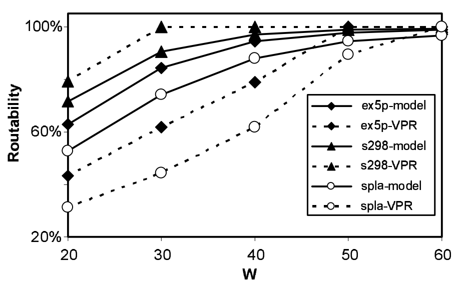
reasons for such over-estimation. First, since we use the upper bounds of the routing graph $G(V,E)$ to estimate the routability, the model is expected to over-estimate the experimental results. Secondly, while deriving equations for Type 2 switch blocks, we assume that the switch box construction is such that two sets of incident tracks will connect to the separate sets of tracks on the outgoing side of the switch block. Finally, as we explain in Section 4.4.1.2, the last unsucessful iteration of Pathfinder that we use to collect experimental results contributes to the over-estimation by the model results.

4.4.2.1 Impact of Logic Fabric on Routability

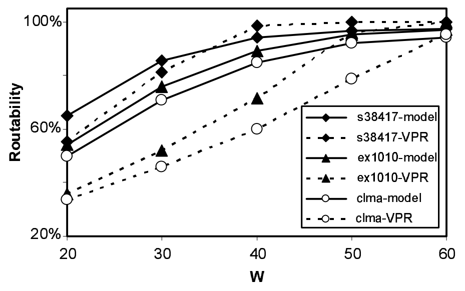
In Figure 4.12, we investigate to what extent our model can capture the impact of the logic fabric on routability. The logic fabric parameters that we consider are LUT size K, cluster size N, and inputs per cluster I. For this investigation, we set N, K and I respectively at 16, 6 and 51[1]. The corresponding results are presented in Figure 4.12. We compare the results presented in Figure 4.12 with the results in Figure 4.11; results in Figure 4.11 are for N=8, K=4 and I=18.

We find that model results can capture the effects of changes in logic fabric parameters; and can provide useful architectural conclusions with respect to these parameters. For instance, with higher values of N, K and I, we expect each cluster to contain more logic blocks, requiring more routing tracks per

[1]We use N=16 to investigate the effects of using a cluster size that is higher than the ones typically investigated by academic studies. K=6 is a representative value for LUT-size and the formula $I = (K/2)\cdot(N+1)$ [6] is used to set I at 51.



(a) Three smaller circuits



(b) Three larger circuits

Figure 4.13: Routability for Individual Circuits

Table 4.6: The Absolute Difference and the Standard Deviation of the Differences between the % Model and the % VPR Routability (Based on Circuit-By-Circuit Results for MCNC Suite)

W-sweep [$F_s = 3, Fc_{in} = 90\%, Fc_{out} = 10\%$]					
W:	20	30	40	50	60
Std. Dev.:	9.50	16.41	15.65	11.21	5.26
Abs. Diff.:	15.00	18.39	15.69	6.34	3.13

Fc_{out}-sweep [$W = 40, F_s = 3, Fc_{in} = 25\%$]					
% Fc_{out}:	13	25	50	75	100
Std. Dev. (in %):	15.62	15.74	16.77	17.00	16.89
Abs. Diff. (in %):	15.16	15.94	17.17	16.92	16.78

channel segment (W) to route the nets between the clusters. Comparison between the sets of results presented in Figure 4.11(a) and 4.12(a) show that for larger N and K, VPR requires more routing tracks per channel segment (W) to approach 100% routability. More importantly, we observe from these two figures that our model can correctly predict this increased routing demand. This demonstrates our model's capability in capturing the effects of the FPGA logic fabric on the consequent routability.

4.4.2.2 Variation across Individual Circuits

Figure 4.13 compares our model results to the experimental results for selected individual circuits. For brevity, we present results for six MCNC benchmark circuits only. Figure 4.13 demonstrates that the model can capture routability trends for individual circuits in most of the cases. Out of the twenty MCNC circuits, the model results over-estimate the experimental results in all cases except for only one circuit *s298*, the results for which have been presented in Figure 4.13(a). For completeness, Table 4.6 presents (a) the standard deviation of the differences and (b) the absolute difference, between the experimental and the model results; for W-sweep and $F_{c_{out}}$-sweep. Results in Table 4.6 are averaged for 20 MCNC benchmarks.

Table 4.7: Range of Values for Routability and Corresponding Number of Circuits (Out of Twenty MCNC Benchmarks), for Channel Width W-Sweep

N=8, K=4, $F_{c_{in}}$=90%, $F_{c_{out}}$=10%, F_s=3

W	Model Routability				Experimental (VPR) Routability			
	$<$ 80%	80% - 90%	90%-100%	100%	$<$ 80%	80% - 90%	90%-100%	100%
20	16	4	0	0	16	2	2	2
30	8	7	5	0	12	2	6	6
40	0	6	14	0	7	1	12	8
50	0	1	19	0	2	1	17	15
60	0	1	19	0	0	1	19	18

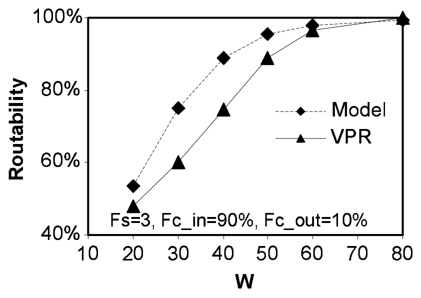
4.4.2.3 Routability for a Specific Routing Fabric

In the above sections, we have presented results pertaining to the goal of this project: investigation on how the changes in routing fabrics affect the routability of typical applications. Accordingly, we have focused on the capability of our model in capturing the trends of the effects of changing routing parameters. However, architects may ask whether our model can be used to investigate the capability of a *specific* routing fabric in successfully routing all or most of the nets of typical applications. In Table 4.7, we attempt to answer this question. In this table, we sweep channel-width W while keeping the other parameters fixed. For each of the values for W, we bin the circuits according to their routability. We present both model and experimental results. Table 4.7 demonstrates that both sets of results report similar number of circuits within each routability bin. (The model results do not reach 100% due to their probabilistic nature.) However, some discrepancies are found for individual circuits. For instance, VPR can route the benchmark *bigkey* even with W=20; this contradicts the model, which reports 86.3% routability for this benchmark at W=20. Whereas, the model reports 95% routability for the benchmark *pdc* at W=60 in contrast to the experimental routability of 81%. Investigation of discrepancies for these outlier benchmarks may be an interesting area of future research.

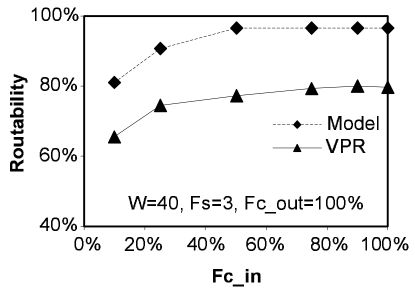
We omit further results for brevity.

4.4.3 Validation Results - QUIP Benchmark Crcuits

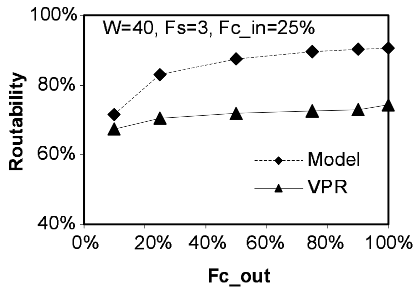
To investigate whether our model can capture the routability trends of large circuits, we validate our model using four QUIP benchmarks [9] that are much larger than the MCNC benchmark circuits. Table 4.5 lists the QUIP benchmarks that we use.



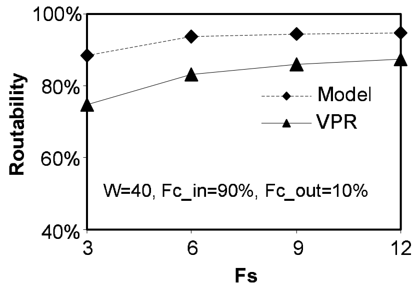
(a) Validation for W



(b) Validation for $F_{c_{in}}$



(c) Validation for $F_{c_{out}}$



(d) Validation for F_s

Figure 4.14: Validation for QUIP Benchmark Circuits with LUT-Size K=4, Cluster-Size N=8 and Inputs per Cluster I=18

Table 4.8: Time Required (in Minutes) for QUIP Benchmarks

N=8, K=4, I=18, W=30, F_{c_in}=0.9, F_{c_out}=0.1
Net-Skip=100 for Model-Regular and Net-Skip=4000 for Model-Fast

BM	Model-Regular		VPR-Regular		Model-Fast		VPR-Fast	
	Time (Mins.)	Routability	Time (Mins.)	Routability	Time (Mins.)	Routability	Time (Mins.)	Routability
quip1	63.2	75%	97.5	76%	5.4	80%	12.4	63%
quip2	28.7	75%	84.2	71%	5.5	82%	7.3	56%
quip3	127.5	80%	557.2	50%	20.1	85%	164.5	38%
quip4	103.2	82%	524.4	65%	11.6	88%	114.5	50%

Figure 4.14 presents the validation results for the QUIP benchmark circuits. We again find that the model results follow the trends of the experimental results. Comparing the results of Figure 4.11 and 4.14, we find that for both MCNC and QUIP benchmarks, the model exhibits similar routability trends when we vary the routing parameters. The above findings illustrate our model's usefulness over a wide range of benchmarks.

4.4.3.1 Computation Time

Table 4.8 compares the computation time required to collect model results and VPR results using the QUIP benchmarks. We present results for two modes of VPR 5.0 as well as our model: *regular*-mode and *fast*-mode.

The architects can make some simplifications to accelerate the use of our model, especially during early stage architecture evaluation. Most notably, Equation 4.2 requires us to update the value of λ_g after finding the routability for each net, whereas for a large circuit with thousands of nets, the changes in λ_g after routing each net are negligible. In other words, the routability of the net# 101 of a large circuit will be very close to the routability of the net# 200. This observation allows the architects to use the routability value of net# 101 for the nets lying within the range of net# 101 and net# 200 while maintaining acceptable level of quality. We use the term *net-skip* to represent this range (*net-skip*=100 for the above discussion). The results in Table 4.9 for the largest MCNC benchmark circuit *clma* verifies this argument for even higher values of *net-skip*. The model results for the two modes (regular and fast) in Table 4.8 use different values for *net-skip*.

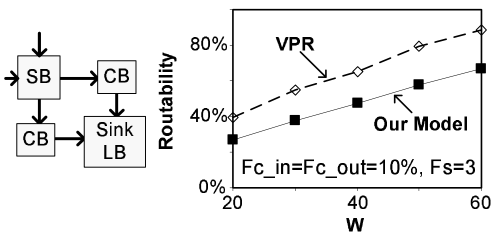
We find that model estimation is faster than VPR, especially for the larger circuits, such as, quip3 (*oc_des_des3perf*), which contains 35,994 nets. We also investigate how close the routability values from the regular and the fast modes are. We find that the differences of routability from these two modes are much smaller for our model in comparison with the results from the two modes of VPR. In other words, either mode of our model can provide useful results much faster than VPR.

4.4.4 Relaxation of Shortest-Path Constraint

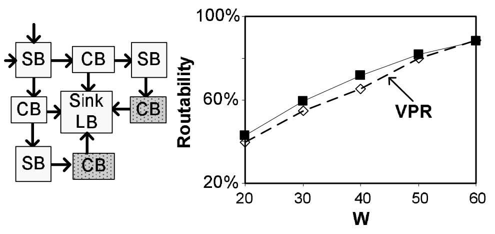
Figure 4.15 highlights an interesting observation. In this case, we have used a highly constrained architecture, in which $F_{c_{in}}$ and $F_{c_{out}}$ are set at 10%. Figure 4.15(a) shows the routability values when we strictly constrain the nets by the minimum bounding box. Unlike the previous results, in this case, our model *underestimates* the VPR results. The explanation for this is as follows. In collecting the model

Table 4.9: Effects of Skipping Nets to Accelerate Estimation

clma (2-Terminal Nets: 14673); W=20, f_s=3, f_{c_in}=90%, f_{c_out}=10%							
Skipped Nets	1	50	100	500	1000	2000	4000
% Routability	47.7	47.8	47.8	48.2	48.6	49.6	51.6



(a) Strictly shortest path



(b) Additional switch block (SB) allowed

Figure 4.15: Routability for Highly Constrained Architecture (Averaged over 20 MCNC Benchmarks)

results, we assume that all nets are routed using the shortest path from source to sink. In the left-side of Figure 4.15(a), this means that a net originating from above and to the left of the sink LB will connect to the sink LB through one of the two shaded connection blocks.

For architectures in which the connection block flexibility is low, however, it is conceivable that no such connection is possible and a router will attempt to use one of the connection blocks on the “far” side of the sink LB, represented by the dotted boxes in Figure 4.15(b). For constrained architectures, such connections occur often. Since the generic VPR router allows such connections (even when the bb_factor flag is set to 0), model results in Figure 4.15(a) underestimates the VPR results.

We now investigate the effects of allowing these connections while collecting model results. We slightly relax the minimum-path constraint to model such effects by following the technique presented in Section 4.3.1. Doing so leads to the results in Figure 4.15(b). Clearly, this provides much more accurate estimates for limited-flexibility architectures.

4.4.5 Effects of Faulty Tracks

For the QUIP benchmark circuit *oc_aes_core*, Figure 4.16 compares the model results with the experimental results when some of the routing tracks are reserved or faulty. We generate the model and the VPR results as in Section 4.4.1. However, we use Equation 4.14 when collecting the model results. In collecting the experimental results, we modify the VPR module that initializes the routing resource graph to randomly mark some of the routing tracks as faulty (‘used’), as a percentage of channel width W.

From Figure 4.16, we find that although the absolute values for model and experimental results differ, they follow the same trend.

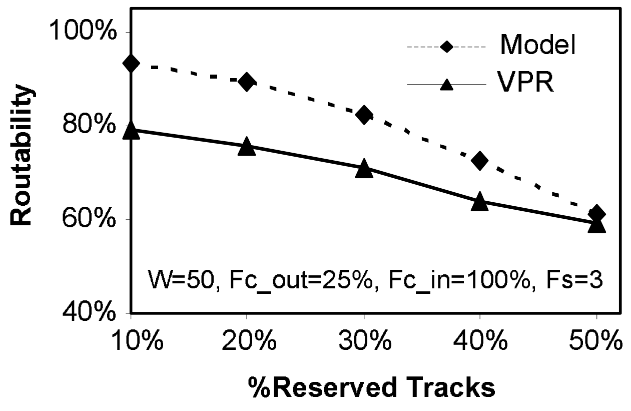


Figure 4.16: Investigation of Effects of Faulty Routing Tracks

4.4.6 Model Summary and Discussion on Results

4.4.6.1 Model Summary

We list here the major results of our model and the corresponding sections that present detailed derivation.

1. Section 4.2.2.2. The upper estimate of the routability of the i^{th} net ψ_i with a given length l is $Pr[R_{\psi_i|l}]$ and is given by Equation 4.11:

$$Pr[R_{\psi_i|l}] \leq Pr[X_1] \cdot Pr[X_2] \cdot (1 - Q_x)\,,$$

   where, the parameter Q_x is calculated by using the consecutive cutsets of the routing graph $G(V,E)$. $Pr[X_1]$ and $Pr[X_2]$ are the routability through the source and the sink connection boxes, respectively.

2. Section 4.2.3. $Pr[R_{\psi_i}]$ represents the probability of a net that may assume any length between 1 and l_{max}, and is given by Equation 4.12:

$$Pr[R_{\psi_i}] = \sum_{l=0}^{l_{max}} Pr[\psi_i|l] \cdot Pr[R_{\psi_i|l}],$$

   where, $Pr[\psi_i|l] = p.q^{l-1}$ represents the probability that a net i has the length l with p=1/l_{avg} and $q = 1 - p$. $Pr[R_{\psi_i|l}]$ is obtained from Section 4.2.2.2.

3. Section 4.2.3. $Pr[R_{ckt|comb}]$ represents the routability of a circuit that contains $|\psi|$ nets ψ_i's and that

is mapped on an FPGA using a combined router. From Equation 4.13:

$$Pr[R_{ckt|comb}] = \frac{1}{|\psi|} \cdot \sum_{i=1}^{|\psi|} Pr[\psi_i].$$

4.4.6.2 Discussion on Results

We have presented validation results averaged over a set of applications as well as for individual applications. The results demonstrate that our model captures the effects of the changing routing fabrics, on routing an average application as well as individual applications. We have made this observation for two sets of benchmarks with a wide range of sizes. When investigating the effects of one routing parameter, our model gives reliable estimates even when the values of the other parameters are changed.

4.5 Summary

This chapter described an *analytical model that relates architectural choices to routability* for FPGA implementations. The major challenge tackled in deriving the model was related to using a modern single-step combined router; use of such a router allows the nets to take any of the many possible routing-paths that are not independent. We use a graph-theoric technique to tackle this challenge. Since placement and routing are the most time consuming stages of a CAD flow, designers will benefit from using our model to quickly evaluate a wide range of routing fabrics with respect to routability. We validate our models against the commonly used academic CAD tool, VPR. We show that despite the assumptions made in deriving our model, it can effectively capture the trends of the effects of architectural choices on consequent routability.

Chapter 5

Applications of Analytical Models: Capabilities and Limitations

The focus of this chapter is to investigate the capabilities and limitations of analytical models when evaluating new FPGA architectures. Despite the promises of significant reduction in design time, analytical models are yet to be extensively used in the design flows used for commercial FPGAs. To make the model-based design technique attractive to a wide audience, in-depth studies are required to investigate whether the analytical models can effectively and correctly answer the questions that FPGA architects ask during architecture development. Feedback from industry also motivated us to identify the categories of design-questions that can *not* be correctly answered by analaytical models. This chapter provides such an understanding. In this chapter, we use the term *model-based design technique* to represent a design process that uses analytical models to evaluate new FPGA architectures.

Our investigation presented in this chapter consists of two parts. First, we investigate whether during the very initial "back of the envelope" design, the analytical models can provide intuition about which architectural parameters have a more significant impact on the FPGA density and speed. Later stages of architectural development involve "parameter sweeps" which involve estimating the area and delay of an architecture for various values of architectural parameters. The second part of this chapter evaluates the effectiveness of the model-based design technique in reaching useful conclusions during parameter sweeps. We use two architecture design questions to illustrate the capabilities and limitations of the model-based design technique. Table 5.1 presents the parameters that we use for our investigation.

Table 5.1: Parameters Used to Optimize an FPGA Architecture

Inputs:	
Architectural Parameters (Logic Fabric):	
N	Cluster size
K	Lookup Table (LUT) Size
I	Inputs per cluster
Architectural Parameters (Routing Fabric):	
W	Channel width (tracks per a routing channel)
$F_{c_{out}}$	Source Connection Block (CB) flexibility
$F_{c_{in}}$	Sink CB flexibility
F_s	Switch Block (SB) flexibility
N_{xy}	FPGA grid size with number of logic objects being N_{xy}^2
Circuit Parameters:	
n_2	Number of logic blocks for 2-LUT implementation
d_2	Circuit depth for 2-LUT implementation
l_{max}	Maximum post-placement wirelength of a circuit
p	Rent parameter of a circuit
Outputs	
n_k	Expected number of LUTs required for a circuit
n_c	Expected number of clusters required for a circuit
d_k	Expected maximum post technology mapping depth for a circuit
d_c	Expected maximum post-clustering depth for a circuit
B_{prog}	Expected programming bits required to implement a circuit
T_{cp}	Expected critical path delay for a circuit

This chapter is organized as follows. Section 5.1 presents our work related to the use of analytical models during very initial stage of architecture development. Section 5.2 presents an overview of using model-based design technique during parameteric sweep, which is the Part 2 of our investigation; this section also introduces the design questions. Section 5.3 and 5.4 present our work related to Part 2; these two sections illustrate the capabilities and limitations of model-based design technique. Finally, Section 5.5 summarizes this chapter.

Parts of this chapter were published in [58, 59, 61].

5.1 Impact of Architectural and Circuit Parameters

During the very initial "back of the envelope" design, designers aim to identify the interesting parameters that will have significant impact on the device performance. This could be done using experimental techniques, however, experimental results using benchmark circuits often display experimental "noise", caused by second and third order effects, and often are a result of pathological mapping results of the benchmark circuits [149, 191]. As an alternative approach, we investigate whether analytical models can be effectively used to identify interesting architectural parameters.

The closed-form nature of our area and delay models allows us to calculate the derivative of the area and delay characteristics of an FPGA architecture with respect to architectural parameters, and use this to understand the impact of those parameters. This section evaluates this idea for our delay models as well as the area models from Lam et al. [104].

5.1.1 Derivatives of Analytical Models

To illustrate the use of derivatives to identify the impacts of architectural parameters, consider the impact of the LUT size on the values of n_k/n_2 and d_k/d_2. Clearly, both quantities decrease as K increases, but the rate at which each decreases as well as the impact of other parameters on this decrease are not clear. Differentiating Equations 3.1 and 3.18 with respect to K, we get:

$$\frac{\partial(n_k/n_2)}{\partial K} = -\frac{\sqrt[p]{4}}{p(K+2)^{\frac{1+p}{p}}}, \tag{5.1}$$

and

$$\frac{\partial(d_k/d_2)}{\partial K} = -\frac{3}{2}\left[\frac{1+\frac{4}{(\ln 2)(3K+2))}}{\left[\frac{3K}{4}-\frac{1}{2}+\log_2\left(\frac{3K}{4}+\frac{1}{2}\right)\right]^2}\right], \tag{5.2}$$

where the parameters are from Table 5.1. We omit the derivation of these expressions for brevity. When deriving these expressions, we replaced γ with a linear approximation ($\gamma = \frac{1}{4}K - \frac{1}{2}$). We can calculate the relative change in d_k/d_2 due to a change ΔK around point K as follows:

$$\frac{f'(K)}{f(K)} \cdot \Delta K, \tag{5.3}$$

where $f(K) = d_k/d_2$ and $f'(K) = \partial(d_k/d_2)/\partial K$. Similarly, we can find the relative change in n_k/n_2.

The impact of the parameter N can be obtained similarly. For N-limited clustering, by differentiating Equations 3.6 and 3.24 with respect to N, we obtain:

$$\frac{\partial(d_c/d_k)}{\partial N} = -\frac{1}{K-\gamma}\left[\frac{1}{N^2}+\frac{K-\gamma-1}{n_k}\right], \tag{5.4}$$

and

$$\frac{\partial(n_c/n_k)}{\partial N} = -\frac{1}{N^2}. \tag{5.5}$$

We can use the last two equations to calculate the relative change in n_c/n_k and in d_c/d_k due to a

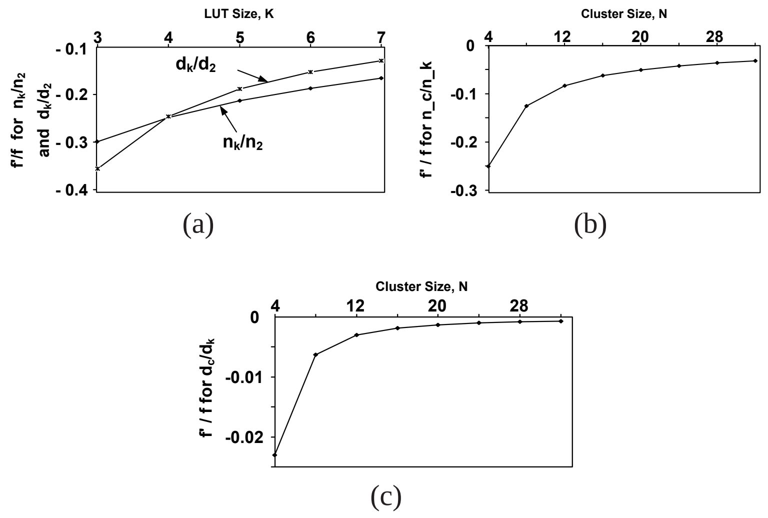


Figure 5.1: Impact of Architectural and Circuit Parameters. (a) Impact of LUT Size on the Number of Logic Blocks and the Post-Technology Mapping Depth. (b) Impact of Cluster Size on the Number of Clusters. (c) Impact of Cluster Size on the Post-Clustering Depth.

change of ΔN around point N:

$$\frac{f'(N)}{f(N)} \cdot \Delta N, \tag{5.6}$$

where $f(N) = d_c/d_k$ and $f'(N) = \partial(d_c/d_k)/\partial N$. Similarly we can find $f'(N)/f(N) \cdot \Delta N$ for $f(N) = n_c/nk$.

5.1.2 Results and Discussions

Figure 5.1(a) shows f'/f for both n_k/n_2 and d_k/d_2, as function of LUT size K. The Rent parameter p is fixed at 0.67. The graph shows that increasing K by ΔK causes a $0.3\Delta K$ reduction in n_k/n_2 for $K = 4$ and the reduction in n_k/n_2 slightly changes for higher values of K. Figure 5.1(a) also shows that for $K = 4$, increasing K by ΔK causes a $0.25\Delta K$ reduction in d_k/d_2. For larger values of K, the impact on d_k/d_2 is smaller. Taken together, these graphs show that for the values of $K > 4$, the parameter K has a much stronger impact on the total number of LUTs required to implement a circuit than the number of LUTs along the circuit's critical path. In other words, incresing K beyond $K = 4$ is expected to have more significant impact on area than on critical path delay.

Figure 5.1(b) and (c) plot f'/f for n_c/n_k and d_c/d_k as a function of cluster size N for p=0.67. These two plots represent the effects of clustering phase of CAD flow on the reduction in the amount of logic and speed of logic. In Figure 5.1(c), we assume n_k to be 3,000; we observe that d_c/d_k is a weak function of n_k. We also observe that, for smaller values of N, the ΔN change will have a significant impact on both n_c/n_k and d_c/d_k. However for higher values of N, the change in N, ΔN will not have a significant impact on the number of clusters n_c or the post-clustering depth d_c. Figure 5.1(b) and (c) also show the relative impact of the cluster size N on the density and the speed, where ΔN affects density (function of n_c/n_k) more than speed (function of d_c/d_k) by an order of magnitude.

The results illustrate that the derivatives of model equations can provide FPGA architects with insights on which parameters will have significant impact on the amount and speed of logic implemented on FPGAs. Such insight will help the architects to identify the parameters that need to be focused on during parameteric sweeps.

5.2 Effectiveness of Analytical Models during Parameter Sweeps

In a typical experimental design flow, the very initial back of the envelope design is followed by parametric sweeps. During a parameteric sweep, FPGA architects sweep architectural parameters to investigate consequent effects on evaluation metrics, with the goal of identifying the best architectures. Table 5.1 lists examples of architectural parameters that are typically swept by architects.

As described in Section 1.2.3, we envisage that analytical models can be used to accelerate this process by short-listing a small number of architectures for further experimental evaluation. Intuitively, pruning potential architectures using our models should be effective, because we have shown that our models, individually, correctly capture the impact of the relevant architectural parameters. However, there are two potential issues that must be investigated:

1. *Assumptions made in deriving the analytical models:* In the derivations in the previous chapters, several simplifying assumptions were made. In particular, each derivation focused on a small number of parameters, assuming the other effects of the other parameters were negligible. As shown in Yan et al. [191], this assumption may not be correct. It is conceivable that when combining the models, these assumptions may lead to misleading conclusions.

2. *Continuous nature of analytical models:* The models in the previous chapters are continuous. However, most parameters are limited to discrete values. As an example, consider optimizing the lookup-table size for an FPGA tuned to implement multiplexer-intensive circuits. An architecture consisting of 6-input lookup tables can effectively implement 4-input multiplexers, since such

a multiplexer would have 4 inputs and 2 select line inputs. Increasing the size of each lookup table to 7 inputs, however, provides little additional benefit when implementing such a circuit. Our model would not capture this effect.

To investigate whether these issues impact the utility of our model-based design flow, we perform two sets of experiments. We first consider the optimization of a general-purpose FPGA and determine whether the architectural conclusions predicted by our model are in alignment with the architectural conclusions presented by an experimental flow. These sets of experiments are presented in Section 5.3. Second, in Section 5.4, we consider the optimization of an FPGA tuned to implement crossbar-intensive circuits that contain a large number of multiplexers and potentially suffer from discrete effects that we describe above.

5.3 Design Question 1: Optimization of General Purpose FPGA Architecture

In this section we optimize a general-purpose FPGA architecture. We consider five architecture parameters from Table 5.1: N, K, I, F_{c_in} and F_{c_out}. In the experimental technique shown in Figure 1.1, architecture parameters are swept to identify the best values of those parameters for a range of benchmarks. An exhaustive sweep that considers all possible combinations of parameter values requires too many invokations of the CAD tool to make this approach feasible. Consider the case where designers want to investigate five architectural parameters $N, K, I, F_{c\_in}, F_{c\_out}$, with these parameters respectively having 25, 5, 20, 10 and 10 possible values (levels). Designers will require a quarter of a million runs to perform this investigation by exhaustively sweeping the parameters.

Rather than using such an expensive exhaustive-sweep based approach, we use two alternative experimental optimization approaches. We compare the conclusions from these two experimental approaches to those obtained from an analytical model-based design flow. Each of these three design flows optimize the architecture parameters separately with respect to two evaluation metrics: the area of implementation (captured by the number of programming bits) and the critical path delay. These three design flows are introduced below:

1. *Design flow 1:* In the first design flow, we run experiments to sequentially optimize the five architectural parameters that we investigate. When optimizing a parameter in this flow, we fix the values for the other parameters; this design flow mimics the traditional design flow often adopted for new

Table 5.2: Design Space for Exploration

	Architectural Parameters				
	N	K	I	F_{c_in}	F_{c_out}
Minimum Values for the Parameters	4	4	8	0.05	0.05
Maximum Values for the Parameters	20	7	64	0.55	0.55
Incremental Values for the Parameters	2	1	4	0.10	0.10
Starting Values for the Parameters	8	6	28	0.25	0.25
Total possible architecture configurations: 19,440					

architecture design. Extensive designers' intuition is required in various stages of this flow such as deciding the sequence that will be used to optimize the parameters. Section 5.3.1 details this flow.

2. *Design flow 2:* The second design flow uses a Design of Experiments (DOE) based approach. A DOE-based flow is expected to significantly reduce the number of experiments when compared to the exhaustive-sweep based approach, while reducing the dependence on designers' intuition during the optimization process. This approach considers the interactions between the architectural parameters and is expected to provide better quality when compared to a sequential optimization flow. Section 5.3.2 details the DOE-based design flow.

3. *Design flow 3:* The third design flow will use our model equations for optimization. This flow is purely analytical and will optimize the architecture parameters only by using the models presented in Chapter 3 and 4 of this thesis, the channel-width model presented by Fang and Rose [70] and the physical delay model presented by Hung et al. [90]. Section 5.3.3 details this design flow.

When optimizing the architectural parameters using these flows, we use the results for area and delay averaged over twenty MCNC benchmarks [192]: *ex5p, misex3, apex4, alu4, tseng, seq, apex2, diffeq, dsip, des, s298, bigkey, spla, frisc, elliptic, pdc, ex1010, s38584.1, s38417* and *clma*.

Each of these three flows explores an identical design space [1]. The design space that we use is given in Table 5.2. The maximum values and the minimum values bound the design space that we investigate. We use the incremental values to investigate the values of the parameters lying within the design space. For instance, while optimizing N, we investigate the results for $N = 4, 6, 8, \ldots, 20$. A total of 19,440 architecture configurations may be formed using the maximum, the minimum and the incremental values from Table 5.2 for the five architectural parameters that we investigate.

[1] We define *design space* by the range of values that we use for the parameters, i.e. the maximum and the minimum values as well as the increments used to explore the values within the range.

As we will explain in Section 5.3.1 and 5.3.2, *starting values* in Table 5.2 play a significant role in both design flows 1 and 2. We use the same set of starting values for both area and delay optimization. In defining the starting values for architectural parameters, we use the values and thumb-rules that earlier studies have found to be “good” for earlier generations of process technology and CAD tools. This allows us to investigate how these conclusions are affected by the changes in process technology and CAD tools.

We now present the findings from earlier studies that we use to define the starting values. The work by Betz et al. [21] in 1999 finds the optimal value of cluster size N to be 6-8 with respect to area-delay product (Figure 6.12, page 145 in [21]) and the optimal value of connection box flexibility F_c to be 0.25 with respect to area when the *Wilton* switch block is used (Figure 7.2, page 161 in [21]). It may be noted that Betz et al. [21] uses $F_c = 0.25$ while setting $N = 4$, LUT size $K = 4$ and inputs per cluster $I = 10$. However, the work by Lemieux and Lewis [111] finds that critical path delay decreases significantly when K is changed from $K = 4$ to $K = 6$ (Figure 5.7a, page 96 in [111]); the decrease in critical path delay is less significant when K is changed from 6 to 7. We use $K = 6$ as the typical value for LUT size. Using the rule of thumb for the optimal value of inputs per cluster I from the work of Ahmed and Rose [6], we have $I = (K/2) \cdot (N+1) = 27$; we use $I = 28$ as the starting value for inputs per cluster.

5.3.1 Design Flow 1: Sequential Optimization Through Experimentations

In an experimental approach, FPGA architectures are typically designed by sequentially optimizing the parameters under investigation. In this sub-section, we detail how we use such an approach to optimize the five architectural parameters that we consider. We first detail the stages involved with this design flow and then present results from this flow.

5.3.1.1 Stages of Design Flow 1

The first issue that we tackle is defining the order of the architectural parameters for optimization. We need to identify the parameter that we will investigate first, the parameter that we will investigate next and so on. We do not know of any previous study that recommends an ordering for optimizing FPGA architectural parameters. In ordering parameters, we use insights from earlier studies. These studies investigate the effects of a single architectural parameter (or a set of architectural parameters). In the cases where we have not found any study to provide such insight, we have used our intuition as well as preliminary experimental results to order the parameters.

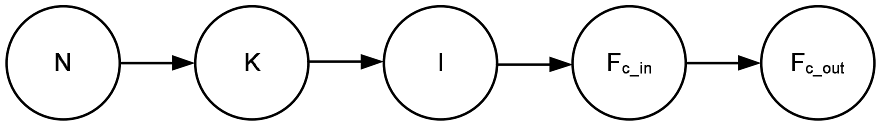


Figure 5.2: Orders of Optimization (Both Area and Delay) for the Architectural Parameters

The ordering that we use for both area and delay optimizations is shown in Figure 5.2. We use N as the first parameter to optimize based on the experimental results from Betz et al. [21] that shows significant reduction in delay for varying cluster sizes (Figure 6.11, page 143). For ordering the remaining parameters, we use the insights from the work of Lemieux and Lewis [111]. That work finds that the delay is more dependent on LUT-size K as compared to the connection box flexiblities F_c. We therefore give preference to K over F_c in Figure 5.2.

After setting the order of parameters, we sequentially optimize these parameters. To optimize a parameter with respect to an evaluation metric, we sweep the parameter within the design space defined in Table 5.2. We use fixed values for the other parameters during this process. If any of the other parameters has been previously optimized, we use the optimized value for that parameter; we use the starting values from Table 5.2 otherwise. For instance, to optimize N for delay, we sweep N within the space $N = 4, 6, \ldots, 20$ while using $\{K, I, F_{c\_in}, F_{c\_out}\} = \{6, 28, 0.25, 0.25\}$. We find that $N = 8$ is the most promising value for N with respect to delay. While sweeping K, we will then set $N = 8$ and $\{I, F_{c\_in}, F_{c\_out}\} = \{28, 0.25, 0.25\}$. In all cases, we fix the switch box flexibility F_s to 3 and the segment length L to 1.

5.3.1.2 Methodology

In our experiments, we start with the netlist of the circuits described in *.blif* format and use T-VPACK [21] to pack (cluster) these circuits while maintaining architectural constraints. We then run the VPR 5.0 [21, 122] place-and-route tool to place and route the circuits; we use timing-driven mode of VPR for our experiments. VPR's 'binary_search_place_and_route' routine has been used in experiments to estimate the minimum channel-width required to route a circuit. We increase minimum channel-width by 20% to mitigate the noise in VPR flow and to ensure 'low-stress' routing [21]; we use this increased channel-width to route the circuits. The results for area and delay are collected after the circuits are placed and routed.

Result collection for area We capture the area of implementation by the number of programming bits required to map a circuit on a given FPGA. We use the following expression to capture the the total number of programming bits B_{prog}:

$$B_{prog} = Number\_of\_Prog\_Bits\_per\_tile * n_c, \qquad (5.7)$$

where, n_c represents the total number of clusters. For our experiments, we obtain the values of n_c from the output of T-VPACK.

From Section 3.2.2, *Number_of_Programming_Bits_per_tile* can be found by adding three parameters: (a) programming bits per cluster $B_{cluster}$, (b) programming bits per connection block B_{CB} and (c) programming bits per switch block B_{SB}. These three parameters can be expressed by Equations 3.15 to 3.17. In Equations 3.16 to 3.17, we set the channel width W at $W = W_{min} * 1.2$, where W_{min} is obtained from the binary-search routine of VPR.

For each sweep, the parameter value leading to the best area results is selected. Results are averaged over twenty MCNC benchmarks that we use.

Result collection for critical path delay To optimize the parameters with respect to delay, we collect the critical path delay from VPR. We again identify the minimum channel-width from VPR's binary-search-place-and-route routine and increase minimum channel width by 20% to route the circuits.

The critical delay values that we have collected from VPR are process-technology dependent values; we use *Predictive Technology Model (PTM) 45nm* technology. We now explain how we use technology dependent delay values in VPR. We use the process-dependent parameters (such as, capacitance of minimum-size transistor) with the physical delay model from Hung et al. [90]. The work of [90] allows us to estimate different delay parameters, such as, delay between cluster input pin and LUT input pin, delay from one switch block to the next switch block and the delay between switch block to input connection block. We incorporate values for these delay components into the VPR architecture files. We have also made necessary modifications to VPR to ensure that VPR uses these values to calculate the critical path delay.

Table 5.3: Design Flow 1 Results : Sequential Optimization for the Number of Programming Bits

Parameter being optimized	Best optimization		Second best optimization	
	Parameter value	Prog. Bits	Parameter value	Prog. Bits
N	14	2.65E+05	6	2.66E+05
K	6	2.65E+05	7	2.71E+05
I	32	2.60E+05	28	2.65E+05
F_{c_in}	0.05	2.55E+05	0.15	2.56E+05
F_{c_out}	0.05	2.33E+05	0.45	2.54E+05
No. of architectures explored: 40 (Each exploration requires running 20 benchmarks through VPR)				

5.3.1.3 Results from Design Flow 1

Area results Table 5.3 presents the results related to the the sequential optimization of the five architectural parameters while following the optimization order shown in Figure 5.2. For each step, Table 5.3 presents the optimized value for the parameter that is investigated at that step. For each step, we also present the value of the corresponding parameter that is the second best with respect to area. We report these values to facilitate the discussion on results in a later sub-section; we do not use these values otherwise. Figure 5.3 presents the same set of results and illustrate how the area requirement is changed while we proceed through the optimization stages.

From Table 5.3 and Figure 5.3, the best architecture with respect to the number of programming bits is identified as $\{N, K, I, F_{c\_in}, F_{c\_out}\} = \{14, 6, 32, 0.05, 0.05\}$ that requires $2.33X10^5$ programming bits on average.

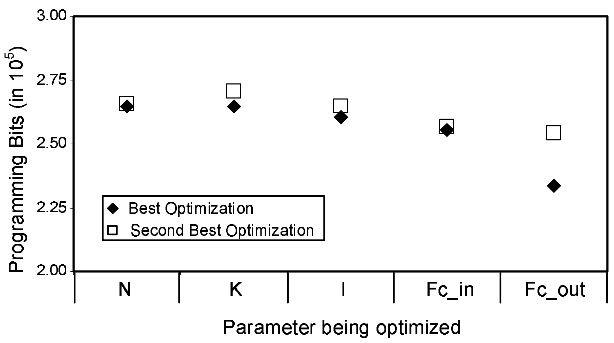


Figure 5.3: Design Flow 1: Number of programming Bits at the End of Each Optimization Stage

Table 5.4: Design Flow 1 Results: Sequential Optimization for the Critical Path Delay

Parameter being optimized	Best optimization		Second best optimization	
	Parameter value	Delay	Parameter value	Delay
N	8	6.15E-09	10	6.31E-09
K	6	6.13E-09	7	6.14E-09
I	28	6.13E-09	20	6.15E-09
F_{c_in}	0.35	5.94E-09	0.25	6.13E-09
F_{c_out}	0.05	5.22E-09	0.15	5.76E-09
No. of architectures explored: 40 (Each exploration requires running 20 benchmarks through VPR)				

Delay results Table 5.4 presents the results related to the sequential optimization of five parameters with respect to critical path delay. We again present the second-best values for the parameters being optimized. Figure 5.4 illustrates the change in critical path delay while we sequentially optimize the architectural parameters.

The best architecture with respect to critical path delay is found to be $\{N, K, I, F_{c\_in}, F_{c\_out}\} = \{8, 6, 28, 0.35, 0.05\}$, which results in an average critical path delay of 5.22*ns*.

5.3.2 Design Flow 2: DOE-Based Experimental Technique

We next use a DOE-based experimental technique to optimize the five parameters that we consider. More specifically, we use the DOE-based Pareto-point generation (DPG) technique, proposed by Sheldon and Vahid [157]. That work's objective is to optimize the configurable components of an existing FPGA device, which is different from our objective of designing new FPGA architectures. We make several modifications to the DPG technique to make it suitable for our purpose. In this sub-section, we first detail

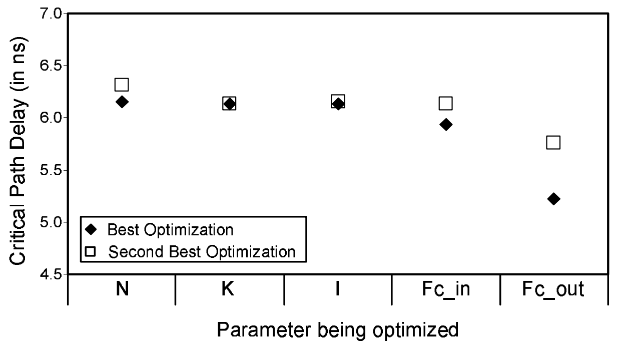


Figure 5.4: Design Flow 1: Critical Path Delay at the End of Each Optimization Stage

the stages of DOE-based DPG technique that we follow and then present the results from this approach. We use the VPR framework to run experiments as required by this design flow. For collecting results from T-VPACK and VPR, we follow the steps detailed in Section 5.3.1.2.

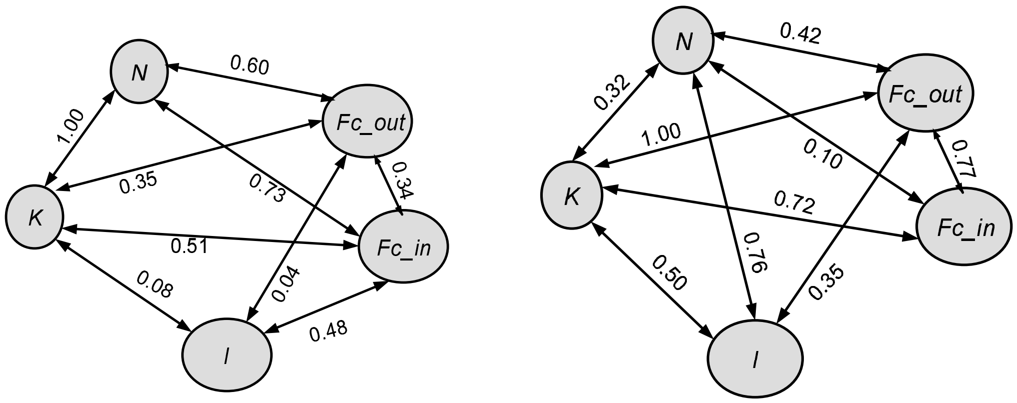
Explored design space DOE-based techniques typically use two or three levels (values) for each input parameter, and these levels are traditionally identified by +1 (maximum), 0 (median, if any) and -1 (minimum). Use of a DOE-based technique therefore allows us to use a maximum of three values for each architectural parameter. We use the values from Table 5.2 to represent these three levels. More specifically, the maximum values, the minimum values and the typical values of Table 5.2 respectively represent level +1, level -1 and level 0 in our experimentations. We later explain how the DPG technique can explore a larger range of values by using a *fill-in* stage.

5.3.2.1 Stages of Design Flow 2

Phases of DPG The DPG technique has three phases. First, a minimal number of experiments are used to generate a parameter interdependency graph [79, 157]. For a given evaluation metric (such as area), this graph presents the interdependency between any pair of architectural parameters. For example, this graph will tell us how much effect the designers will have on area, when they simultaneously change the values of a pair of parameters (such as K and N) from minimum to maximum. In the second phase, based on the information from the parameter interdependency graph, pairs of parameters are optimized sequentially, starting with the pair that is expected to have the most significant effect on evaluation metrics. These first two phases use three levels for inputs. Typical architectural parameters however will have many more possible values. To address this, Sheldon et al. [157] uses the third phase (*fill-in* phase) to investigate other promising values, after the first two phases are completed. In our work, while optimizing the architectural parameters for a new FPGA architecture, we find that it is beneficial to perform the second and third phases simultaneously.

We now detail each of these three phases.

Generation of the interdependency graph Since we optimize the architectures for area and delay separately, we form two parameter interdependency graphs for these two evaluation metrics. Figure 5.5 shows these two graphs, details on generation of which will follow. Each of these graphs will have five nodes



(a) Number of programming bits (b) Critical path delay

Figure 5.5: Parameter Interdependency Graphs (Results Averaged over 20 MCNC Benchmarks)

representing the five architectural parameters that we consider. The edge between a pair of nodes is assigned with a weight that represents the interdependence between these two parameters with respect to area or delay.

Parameter interdependency graphs could be formed by exhaustively sweeping the architecture parameters and observing the consequent effects on area or delay. Bounding the architecture parameters by the maximum (+1) and minimum (-1) values only, we would then require $2^5 = 32$ runs to form the graph. To reduce the number of required runs, we use a technique presented by Plackett and Burmann [140]. This technique uses a set of limited number of experiments, known as Plackett-Burmann (PB) set of experiments. Based on the results obtained from PB set of experiments, we first calculate the *independent* effect of each architecture parameter on an evaluation metric, while neglecting the interactions between the remaining parameters. The estimated results for independent effects are then used to calculate the *interdependent* effects of a pair of parameters. We use the technique from Sheldon et al. [157] for this last task. We now discuss these steps in more details.

Formation of the PB set of experiments For the five parameters that we investigate, the PB set of experiments is formed using only the maximum and the minimum values. Several studies discuss the formation of PB sets of experiments to properly capture the independent effects of the inputs on the outputs. These studies propose a varying number of required experimental runs. At the minimum, $(F+1)/(L-1)$ runs can be used to measure the effects of F factors (inputs) on a response (output), with each factor having L levels. Six experiments can therefore be used to measure the independent

Table 5.5: Plackett-Burman (PB) Matrix for Generating Interdependency Graphs

	Levels for parameters					Values for parameters				
PB Runs	N	K	I	Fc_{in}	Fc_{out}	N	K	I	Fc_{in}	Fc_{out}
1	1	-1	1	-1	-1	20	4	64	0.05	0.05
2	1	1	-1	1	-1	20	7	8	0.55	0.05
3	-1	1	1	-1	1	4	7	64	0.05	0.55
4	1	-1	1	1	-1	20	4	64	0.55	0.05
5	1	1	-1	1	1	20	7	8	0.55	0.55
6	1	1	1	-1	1	20	7	64	0.05	0.55
7	-1	1	1	1	-1	4	7	64	0.55	0.05
8	-1	-1	1	1	1	4	4	64	0.55	0.55
9	-1	-1	-1	1	1	4	4	8	0.55	0.55
10	1	-1	-1	-1	1	20	4	8	0.05	0.55
11	-1	1	-1	-1	-1	4	7	8	0.05	0.05
12	-1	-1	-1	-1	-1	4	4	8	0.05	0.05

effects of five architectural parameters (F=5) on area or delay, with each parameter having two levels (L=2). However, a study by Heyden et al. [86] shows that the effects of inputs are significantly aliased (confounded)[2] when using a very low number of experiments. We therefore use a twelve-run PB matrix instead of using a lower number of experiments. Details of forming the PB matrix can be found in several publications [86, 118, 140]. Table 5.5 presents the PB matrix of runs that we use to generate interdependency graphs.

Estimation of independent effects of the parameters while neglecting interactions For each of the runs in Table 5.5, we use the corresponding values of architectural parameters with T-VPACK and VPR and collect the results for area (total number of programming bits) and critical path delay. These results are used to estimate the effect that an architectural parameter will have on area or delay, independent of the other architectural parameters. To estimate these independent effects, we use techniques similar to the ones presented in earlier studies by Plackett and Burman [140] and Heyden et al. [86]. As an example of using this technique, consider the estimation of the effect of cluster size N. From Table 5.5, we find that the maximum value (+1) for N appears in rows 1-2, 4-6 and 10, and the minimum value (-1) appears in rows 3, 7-9 and 11-12. Representing the experimental results (from VPR) for area for the i^{th} row as A_i,

[2]Aliasing: When the estimate of the effect of changing one factor (architectural parameter) is influenced by the effects of other factors (usually by the higher order interactions between other factors).

the independent effect of N on area can be estimated as:

$$E_{N_{area}} = \frac{A_1 + A_2 + A_4 + A_5 + A_6 + A_{10} - A_3 - A_7 - A_8 - A_9 - A_{11} - A_{12}}{12}. \tag{5.8}$$

In a similar fashion, representing the experimental results for delay for the i^{th} row as D_i, the independent effect of N on area can be estimated as:

$$E_{N_{delay}} = \frac{D_1 + D_2 + D_4 + D_5 + D_6 + D_{10} - D_3 - D_7 - D_8 - D_9 - D_{11} - D_{12}}{12}. \tag{5.9}$$

Estimation of edge-weights (interdependence) The DPG technique uses the estimated independent effects from above in assigning edge-weights. We now give an example to illustrate how we follow the guidelines from Sheldon and Vahid [157] to estimate the weight of the edge I-F_{c_in} in the interdependency graph for critical path delay. The independent effects of I and F_{c_in} on delay are estimated as -1.6*ns* and -0.3*ns* respectively (from the PB set of experiments). This implies that changing the value of I from 8 (level '-1') to 64 ('+1') and the value of F_{c_in} from 0.05 ('-1') to 0.55 ('+1') will reduce delay by 1.6*ns* and 0.3*ns* respectively, when the other parameters are set at fixed values. If I and F_{c_in} were truly independent; changing I and F_{c_in} together would have reduced the critical path delay by 1.9*ns*. To investigate whether there is any interaction between I and F_{c_in}, we run two experiments, where the first experiment uses minimum values for I and F_{c_in}, and the second one uses maximum values. The other four architecture parameters are fixed. We run experiments using $\{N, K, I, F_{c\_in}, F_{c\_out}\}$ = $\{4, 4, 8, 0.05, 0.05\}$ and $\{N, K, I, F_{c\_in}, F_{c\_out}\}$ = $\{4, 4, 64, 0.55, 0.05\}$ and find the critical path delay to be 8.2*ns* and 6.8*ns*, respectively. In other words, changing I and F_{c_in} simultaneously reduces the critical path delay by $8.2ns - 6.8ns = 1.4ns$, which is lower than 1.9*ns*. This implies that these two parameters are inter-dependent and we represent this inter-dependence by assigning a weight to the I-F_{c_in} edge; the weight will be equivalent to $1.9ns - 1.4ns = 0.5ns$ (which is 0 when normalized).

We follow this technique to measure the weights for each edge in parameter interdepency graphs. If two parameters are independent of each other (if the edge weight is negligible), there will be no edge between the corresponding nodes, such as the I-F_{c_in} edge from the above example. A higher weight for an edge will represent higher interdependency between the nodes connected by that edge. Figure 5.5 shows the parameter interdepency graphs for area and delay.

Optimizing the architectural parameters The DPG technique starts architecture optimization by optimizing the pair of parameters connected by the highest weighted edge. In Figure 5.5(a), the edge between K and N has the normalized weight of 1.00. We therefore optimize these two parameters first for area optimization. While optimizing a pair of parameters, we allow the parameters to have three values: -1, 0 and $+1$. These values correspond to the maximum, the typical and the minimum values in Table 5.2. We need a maximum of nine runs for any pair of parameters, when the other parameters fixed.

After optimizing a pair of parameters, the edge between the parameters is removed from the graph and the two nodes are merged into one. For instance, we find that the optimized values of K and N to be 6 and 8. To investigate the remaining edges, we remove the edge K-N, and use $K = 6$ and $N = 8$ for the remaining experiments. If an optimized parameter is a member node of the edge that we investigate at a later stage, we use a fixed value for this parameter and the required runs to investigate this latter edge is reduced from nine to three. Furthermore, we ignore an edge if both of the edge-nodes (architectural parameters) have been investigated earlier.

Fill-in phase While optimizing a pair of parameters, we may also use the *fill-in* stage, whenever required. For example, if delay for $N = 20$ differs significantly from $N = 8$, we may either explore the intermediate values for N such as $N = 14, 16, 18$ etc. or larger values for N such as $N = 24, 28$ etc. To keep the design space consistent across all design flow, during fill-in phase, we only explore the values lying between the maximum and the minimum values.

Table 5.6: DPG Stages: Optimizing an Architecture for the Area of Implementation (No. of Prog. Bits). Sequence of the Pairs is Determined from Figure 5.5a); F_s=3 in all cases

		Optimized Parameters		Minimum No. of Prog.
Stage No.	Pair of Parameters	1st Parameter	2nd Parameter	Bits after this Stage
1	PB-Stage	–	–	$3.08X10^5$
2	K, N	K=6	N=8	$2.72X10^5$
3	N, F_{c_in}	N=8	F_{c_in}=0.05	$2.60X10^5$
4	N, F_{c_out}	N=8	F_{c_out}=0.05	$2.55X10^5$
5	I, F_{c_in}	I=28	F_{c_in}=0.05	$2.55X10^5$
6	Fill-in: N, K, I	$N = 10$, $K = 6$, $I = 36$		$2.52X10^5$

Optimized architecture for area: $\{N, K, I, F_{c\_in}, F_{c\_out}\}$ = {10,6,36,0.05,0.05}
No. of architectures explored: 47 (Each exploration requires running 20 benchmarks through VPR)

5.3.2.2 Results from Design Flow 2

Area results In Table 5.6, we present results collected from different stages of the DOE-based approach when the architecture is being optimized for area; as before, we capture the area by the number of programming bits required to implement a circuit. The first data-row represents the results from the experiments using Plackett-Burman matrix. The remaining data-rows present data corresponding to the pair-wise optimization stages. The second column of this table presents the pairs being investigated, starting with the pair having the highest interdependency weight in Figure 5.5(a). The next two columns present the optimal values for the corresponding parameters. Finally, the rightmost column presents the minimum number of programming bits as reported upon completion of the corresponding stage. We find from Table 5.6 that the required number of programming bits decreases consistently while we progress through pair-wise optimization stages. For area optimization, we had to explore 47 architecture configurations in DOE-based experimental flow.

Delay Results Table 5.7 presents the results from DPG-based design flow when the architecture is being optimized for critical path delay. The construction of this table is similar to that of Table 5.6. From the rightmost column of Table 5.7, we find that as we progress through DPG stages, the critical path delay conistenty goes down. We explore 49 architecture configurations to optimize architectural parameters with respect to critical path delay.

Table 5.7: DPG Stages: Optimizing an Architecture for Critical Path Delay. Sequence of the Pairs is Determined from Figure 5.5(b); F_s=3 in All Cases

		Optimized Parameters		Minimum Crit. Path
Stage No.	Pair of Parameters	1st Parameter	2nd Parameter	Delay after this Stage
1	PB-Stage	–	–	5.23 ns
2	K, F_{c_out}	K=7	F_{c_out}=0.05	4.83ns
3	F_{c_in}, F_{c_out}	F_{c_in}=0.55	F_{c_out}=0.05	4.73ns
4	N, I	N=20	I=64	4.37ns
5	Fill-in: N, I, F_{c_in}	$N = 20$, $I = 64$, $F_{c\_in} = 0.25$		4.24ns

Optimized architecture for critical-path delay: $\{N, K, I, F_{c\_in}, F_{c\_out}\}$ = {20,7,64,0.25,0.05}
No. of architectures explored: 49 (Each exploration requires running 20 benchmarks through VPR)

5.3.3 Design Flow 3: Model-Based Optimization

In this section, we use analytical model-based approach to optimize the five architectural parameters with respect to area of implementation and critical path delay. This design flow is purely analytical. We first detail this design flow and then present the results. We again use the design space presented in Table 5.2. An architecture can be very quickly optimized using the analytical models (fraction of a second). This allows us to exhaustively explore 19,440 possible architecture configurations from the design space.

5.3.3.1 Stages of Design Flow 3

In this sub-section, we detail how we use the analytical models to optimize the architectural parameters. We first detail how we collect results for area and critical path delay.

Collecting results for area of implementation We use Equations 5.7 to 3.17 to estimate area (number of programming bits). In addition to the five architectural parameters that we investigate, these equations require (a) the estimates of the number of clusters required to map a circuit n_c, (b) the switch box flexibility of an architecture F_s and (c) the channel-width used to map a circuit W. For a given benchmark implemented on a given FPGA architecture, we can estimate n_c using the relation studied by an earlier work [104] and presented in Equation 3.6 of this dissertation. Similar to the first two design flows, we set the value of F_s at 3. We use the model from the work of Fang and Rose [70] to estimate W. The model for minimum channel width W_{min_model} from [70] has been presented in Equation 3.8. We increase this minimum channel-width by 20%, which gives us $W = 1.2W_{min\_model}$. Estimation of W_{min_model} requires the estimate of the average wirelength l_{avg}. We use our wirelength model for this purpose.

Collecting results for critical path delay To analytically model the critical path delay T_{cp}, we use Equation 3.25 from Section 3.3:

$$T_{cp} = d_c \left[t_{inter} + \frac{d_k}{d_c} t_{intra} \right] = d_c \cdot t_{inter} + d_k \cdot t_{intra}. \tag{5.10}$$

We use our delay models from Chapter 3 to estimate d_k and d_c. We now detail how we estimate t_{intra} and t_{inter} using our wirelength model and the physical delay model from the work of Hung et al. [90].

Estimation of t_{intra} t_{intra} represents the delay incurred by a net while traversing through the logic elements inside the clusters. This parameter consists of two components: (a) delay incurred by the logic elements and (b) delay incurred by the interconnect network within a cluster. We use the model of [90] to estimate these two components and add them to estimate the corresponding values for t_{intra}.

Estimation of t_{inter} Hung et al. [90] provides the delay components for traversing a net through an FPGA tile with given architectural parameters. Estimation of t_{inter} further requires the average wirelength for a circuit along the post-routing critical path. Our wirelength model gives us the average post-placement wirelength l_{avg_placed}. To estimate t_{inter}, we do not directly use this l_{avg_placed} for two reasons. First, the post-placement wirelength may be inflated during routing phase due to congestion. Increasing the channel width by 20% as discussed above will mitigate this issue to some extent. Second, we observe that wires along the critical path are typically longer than the "average wirelength". This appears counter-intuitive. We would expect a timing-driven placement algorithm to place cells so that wires along the critical path are shorter than the average. However, the critical path after placement often is not the same path as the one before placement. In fact, those nets that were deemed "not critical" before placement tend to be longer than average, and paths using these longer segments are more likely to become critical. To account for this, we assume that the wires along the critical path are a factor of β slower than the average wire. Experimentally, we find that $\beta = 2$ works well, and we use this scaling factor to compute t_{inter}. In other words, we use $l_{avg\_routed} = 2 \cdot l_{avg\_placed}$ and $W = 1.2 \cdot W_{min\_model}$ when estimating t_{inter}.

Optimization of architectural parameters The two experimentation-based design flows that we present earlier are constructed in a way such that the designers do not need to exhaustively search the design space. However, evaluating an architecture using analytical models is very fast, so in Design Flow 3, we exhaustively search the entire design space. For exhaustive sweep, we use Table 5.2 to define maximum, the minimum and the incremental values for each parameter. We use an inhouse C-program to exhaustively sweep architectural parameters using model equations. This inhouse program uses the techniques presented above and generate the area and delay results.

Table 5.8: Optimization for Area

1	2	3	4	5	6	7	8	9
		Architectural Parameters					No. of Prog. Bits (in 10^5)	
No.	Rank	N:	K:	I:	Fc_in	Fc_out	Model Results	VPR Results
1	1	12	7	36	0.05	0.15	2.260	2.385
2	1	12	7	36	0.05	0.05	2.260	2.393
3	2	10	6	24	0.05	0.05	2.290	2.311
4	2	10	6	24	0.05	0.15	2.290	2.270
5	3	8	6	24	0.05	0.05	2.310	2.467
6	3	8	6	24	0.05	0.15	2.310	2.467
7	4	10	5	24	0.05	0.15	2.320	2.364
8	4	10	5	24	0.05	0.05	2.320	2.473
9	5	14	6	32	0.05	0.05	2.330	2.335
10	6	8	7	24	0.05	0.15	2.330	2.411
No. of architectures explored: 19,440 (Each exploration requires running 20 benchmarks through model-equations)								

5.3.3.2 Results from Design Flow 3

Optimization of architectures for area Table 5.8 presents the optimized architectures with respect to area that we find by using purely model-based design flow. Column 2 of the table represents the rank that we assign based on the position within sorted results (the model-based flow may report the same area values for multiple architectures). Columns 3 to 7 present the values of the architectural parameters that we investigate. Column 8 presents the model results. We collect experimental results for each of the ten best architectures in Table 5.8 and present them in the rightmost column. Based on the experimental validation results, we find the architecture with $\{N,K,I,F_{c\_in},F_{c\_out}\}$ = $\{10,6,24,0.05,0.15\}$ to be the most promising one with the requirement of $2.27X10^5$ programming bits on average.

Optimization of the architectures for the critical path delay Table 5.9 presents the optimized architectures with respect to critical path delay that we find by using purely model-based design flow. Column 1 of the table represents the rank that we assign based on the position within sorted results. Columns 2 to 6 present the values of the architectural parameters that we investigate. Column 7 presents the model results for critical path delay. Finally, we collect the results for critical path delay from the VPR framework for the architectures found to be optimal by model-based design flow and present these results in the rightmost

Table 5.9: Optimization for Critical Path Delay

1	2	3	4	5	6	7	8	9
No.	Rank	Architectural Parameters					Critical Path Delay (in ns)	
		N:	K:	I:	Fc_in	Fc_out	Model Results	VPR Results
1	1	16	7	64	0.05	0.05	4.600	4.500
2	2	16	7	56	0.05	0.05	4.610	4.647
3	3	16	7	64	0.15	0.05	4.610	4.431
4	4	16	7	60	0.05	0.05	4.610	4.381
5	5	16	7	56	0.15	0.05	4.620	4.409
6	5	18	7	64	0.05	0.05	4.620	4.447
7	6	16	7	64	0.25	0.05	4.620	4.439
8	7	18	7	60	0.05	0.05	4.630	4.201
9	8	16	7	60	0.15	0.05	4.630	4.237
10	8	16	7	52	0.05	0.05	4.630	4.595
No. of architectures explored: 19,440 (Each exploration requires running 20 benchmarks through model-equations)								

column. Based on the results from VPR, the architecture with $\{N, K, I, F_{c\_in}, F_{c\_out}\}$ = $\{18,7,60,0.05,0.05\}$ is the best architecture with respect to critical path delay.

Discussion on results from design flow 3 We find discrepencies when comparing model results with experimental results both in Table 5.8 and 5.9. For instance, in Table 5.9, the ranked-7 architecture (row-8) is found to be optimal by VPR when experiments are conducted on the ten short-listed architectures. This supports our earlier argument that the short-listed architectures from model-based design flow needs to be validated and fine-tuned using experimental results. We further note that the discrepancies that we find in Table 5.8 and 5.9 are not significant.

5.3.4 Comparison of Results from Three Design Flows and Discussion

In the preceding three sub-sections, we optimize five architectural parameters using a sequential optimization technique, a DOE-based optimization technique and a model-based optimization technique. In this sub-section, we compare the optimization results generated by these three techniques.

Area of implementation We compare the area-optimization results presented in Table 5.3, 5.6 and 5.8 for Design Flows 1, 2 and 3 respectively. Table 5.10 presents the best architectures identified by each of

Table 5.10: Comparison of Design Flows: Area of Implementation

1	2	3	4	5	6	7	8	9
	Optimized Architecture					Prog. Bits	Architectures	Exploration
Design Flow	N:	K:	I:	Fc_in	Fc_out	(Experimental)	Explored	Time (hours)
1: Sequential	14	6	32	0.05	0.05	$2.335 \text{ X } 10^5$	40	30.00
2: DOE-based	10	6	36	0.05	0.05	$2.515 \text{ X } 10^5$	47	35.25
3: Model-based	10	6	24	0.05	0.15	$2.270 \text{ X } 10^5$	19440	7.25^{*+}
* Includes approx. 10 experimental runs to validate and fine-tune optimized architectures (7.0 hours) $^+$ Exploration of 19,440 configurations requires 13 minutes in model-based flow								

these design flows, with respect to area. Columns 2 to 6 present the optimized architectures and column 7 shows the corresponding results for area from VPR. Column 8 shows the total number of architectures that we explore for each flow. Finally, the time required to explore these architectures is presented in the right-most column. The exploration of each architecture requires running twenty MCNC benchmarks through the VPR flow (for Design Flows 1 and 2) or through the model-based framework consisting of analytical equations (for Design Flow 3). In our machine with 2.5GHz quad-core processor and 24GB memory, we require 45 minutes on average to explore one set of architectural parameters in Flows 1 and 2. In a model-based flow, our inhouse C program requires nominal amount of time to explore a potential architecture (0.04 seconds). Model-based flow require experimentations only to identify the best architecture from the short-listed architectures.

From Table 5.10, we find that the analytical model-based design flow finds a better architecture than is found by the other two flows. This is because this flow allows us to quickly sweep the entire design space to identify a set of promising architectures.

Critical path delay Table 5.11 compares the results for optimized architectures with respect to critical path delay. We again find that the model-based approach gives a better architecture than the other two flows while requiring significantly less design-effort.

Table 5.11 shows that the sequential optimization flow reports the best value of N to be 8 in contrast to $N = 18$ reported by model equations. This illustrates a limitation of the sequential optimization approach. When optimizing N for delay using sequential optimization approach, we set $I = 28$ and $K = 6$ (typical values) and given these values, we find $N = 8$ to be optimal value for cluster size. When exploring K and I at later stages, the value of N is fixed at 8. In other words, we never explore the architectures, for which

Table 5.11: Comparison of Design Flows: Critical Path Delay

Design Flow	Optimized Architecture					Crit. Path Delay	Architectures	Exploration
	N	K	I	Fc_in	Fc_out	(Experimental)	Explored	Time (hours)
1: Sequential	8	6	28	0.35	0.05	5.221ns	40	30.00
2: DOE-based	20	7	64	0.25	0.05	4.239ns	49	36.75
3: Model-based	18	7	60	0.05	0.05	4.201ns	19440	7.25^{*+}

* Includes approx. 10 experimental runs to validate and fine-tune optimized architectures (7.0 hours)
$^{+}$ Exploration of 19,440 configurations requires 13 minutes in model-based flow

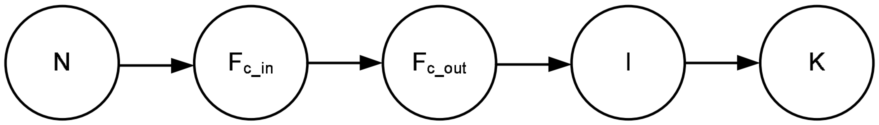


Figure 5.6: Area Optimization with Different Ordering of Parameters

$N > 8$ and $K > 6$ and $I > 28$.

Further discussion As we explained earlier, the designers' experience and intuition are required in all stages of the experimental design flows. In sequential optimization, we need designers' experience in setting the ordering of parameters for optimization; different ordering of parameters may yield different values for architectural parameters. To demonstrate this, we use the ordering of parameters shown in Figure 5.6, for area optimization. This ordering is different from the one that we have used for area optimization in Section 5.3.1. Table 5.12 presents the optimization results for this new ordering. Comparing the results from Table 5.12 and Table 5.3, we find that the new ordering results into a degraded architecture.

Furthermore, Figure 5.3 and 5.4 illustrate that when optimizing certain parameters (such as, K and I in Figure 5.4), multiple architecture configurations may give very close area or delay results. A wrong choice may have a negative impact on the optimization of the remaining parameters. We also observe similar issues in DOE-based Design Flow 2.

Since the DOE-based flow considers the interactions between parameters and reduces some dependence on designers' intuition, such a flow is expected to perform better than sequential optimization. This expectation is confirmed by the minimum critical path delay found by Design Flow 3 (4.24ns) that is significantly lower than the one found by Design Flow 1 (5.22ns). However, during area optimization, the

Table 5.12: Sequential Optimization for the Number of Programming Bits with Different Ordering of Parameters as Compared to Table 5.3

Parameter being optimized	Best optimization		Second best optimization	
	Parameter value	Prog. Bits	Parameter value	Prog. Bits
N	14	2.65E+05	6	2.66E+05
$F_{c_{in}}$	0.15	2.60E+05	0.05	2.62E+05
$F_{c_{out}}$	0.05	2.43E+05	0.25	2.60E+05
I	28	2.43E+05	32	2.45E+05
K	6	2.43E+05	7	2.52E+05
Optimized $\{N,K,I,F_{c_{in}},F_{c_{out}}\}$ from Table 5.3={14,6,32,0.05,0.05}; Bits: 2.33E+05 Optimized $\{N,K,I,F_{c_{in}},F_{c_{out}}\}$ with new ordering={14,6,28,0.15,0.05}; Bits: 2.43E+05				

DOE-based flow results in a sub-optimal architecture when compared to the sequential optimization. A fill-in stage after optimizing the K-N pair (Table 5.7) might improve DOE's performance at the expense of increased experimental runs.

Furthermore, due to the extensive design effort that we require for experimental flows, we conservatively choose the design space. For instance, we limit the maximum and the incremental values for F_{c_in} to 0.55 and 0.10; whereas, setting these values at 1.00 and 0.05 might yield better architectures.

As we argue earlier in this dissertation, the model-based results should only be used to quickly shortlist a set of architectures that need to be further investigated using experimentation. The results that we present in this section demonstrate that the model-based flow successfully performs this task and even finds better architectures in comparison with the architectures experimentally optimized. The differences in the ranks between model results and VPR results in Table 5.8 and Table 5.9 also highlights the significance of our statement that 'the model-based flow should be used as a supplement to the experimental flow, and not as the replacement'.

The above comparison results [3] and discussion lead us to conclude that the model-based technique can effectively optimize a general-purpose FPGA architecture while using a significantly lower number of experimental runs.

[3]The comparison of results from three design flows would be more robust if we had a golden set of experimentally obtained results for the 19,400 architectures that the model-based design flow investigates. However, we note that a single run through the experimental CAD flow requires almost an hour and the investigation of 19,440 architectures will require more than two years of computational time on a single processor machine.

5.4 Design Question-2: Optimization of Application-Specific FPGA Architecture (ASFPGA)

As described in Section 5.2, due to the continuous nature of our model, we anticipate our models may not capture the “discrete effects” exhibited by some architectures and user circuits. In this section, we show an example in which this is so, and hence illustrate an important limitation of the application of our models.

5.4.1 Methodology

We consider the design of an Application-Specific FPGA architecture (ASFPGA) which is optimized to implement crossbar-intensive circuits. Crossbar structures are common in communication switching applications; such applications make up a significant share of the FPGA market. A crossbar switch has several input and output buses each output bus can be connected to any input bus. Such circuits contain large multiplexers, and as described in Section 5.2, these types of circuits may not match our model predictions closely. To illustrate the failure of our model, we consider the optimization of LUT size K for such an ASFPGA.

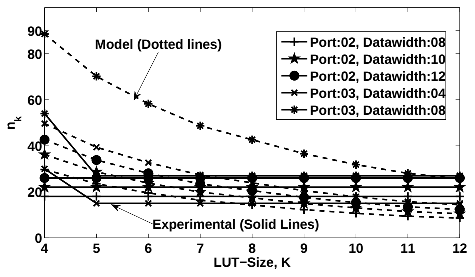
In order to optimize such an architecture, we would use circuits from the target domain rather than the more “general-purpose” MCNC circuits. Since we do not have such circuits available, we generate representative crossbar circuits using a custom-written script. As shown in Table 5.13, we generate circuits with different combinations of two parameters: (a) the number of ports, and (b) the data width of each bus., These circuits are converted to BLIF format using Altera’s *quartus_map* tool, and then processed by SIS [134] using guidelines provided by the University of Wisconsin [180].

5.4.2 Results and Discussion

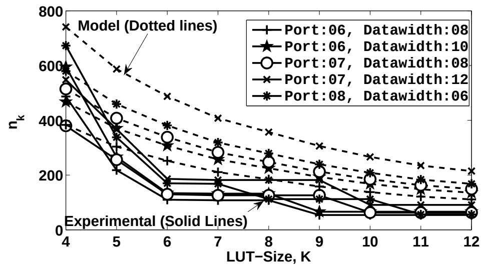
Using both the experimental and analytical techniques, we sweep the lookup-table size, K, from 4 to 12, and find the number of LUTs required to implement the circuit, n_k. The experimental and analytical

Table 5.13: Parameters of the Investigated Crossbar Switches

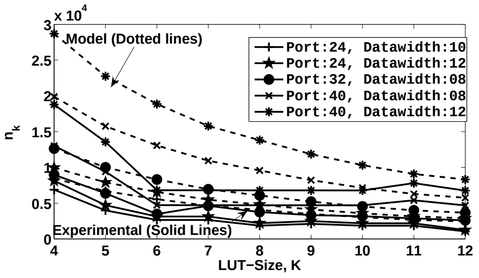
Parameter	Values used	Total number of values
Number of Ports	2, 3, 4, 5, 6, ... 16, 20, 24, 32, 40, 48	20
Data-width	1, 2, 4, 6, 8, 10, 12	7



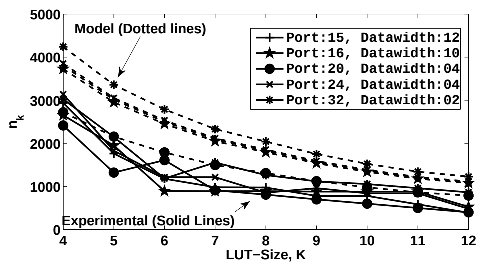
(a) First set of crossbar switches



(b) Second set of crossbar switches



(c) Third set of crossbar switches



(d) Fourth set of crossbar switches

Figure 5.7: Results to Investigate the Limitation of Model-Based Approach in Capturing the Required Number of LUTs, n_k for Crossbar Switches

results are shown in Figure 5.7. For clarity, the results are separated into four groups, since the magnitude of n_k varies considerably among the considered circuits. In all cases, the model results do not closely match the experimental results. As expected, the model results are continuous, while the experimental results show "breaks" at various values of K. As an example, consider the crossbar switch with 7 ports and a data width of 8 in Figure 5.7(b) represented by round-white markers. For this crossbar, the experimental results show that n_k changes abruptly when K is changed from 4 to 6 and then remains almost constant until K=9. In contrast, the model predicts a smooth change in this region. The impact of these "breaks" would be important for an FPGA architecture to understand, yet our model does not capture them at all. Model estimates for the other implementation parameters, such as n_c and d_c, depend on the values of n_k. We would expect that our model can not adequately capture the effects of K on these parameters either.

The conclusions from these results is that our models have limitations when the underlying architecture or circuits exhibits "discrete effects". However, it does not mean our model can not be used in such cases. It is still possible to use our model to prune out portions of the architecture space that are clearly poor. Experimental results would then be required to investigate architectures within the region not pruned out. Thus, we expect that, in general, a combination of analytical models to prune the space, and experimental techniques, to select architectures within this space, may lead to the best overall solution.

5.5 Summary

This chapter analyzes the capabilitites and limitations of model-based architecture design technique. We present the capabilities of using analytical models: (a) during very initial back-of-the-envelope stage of a design flow and (b) during later stages of the design flow that require parameteric sweeps. We find that the model-based design technique is effective in optimizing general-purpose FPGA architectures. We use sequential optimization and DOE-based optimization techniques to compare the effectiveness of analytical models with. We further show that the continuous nature of analytical models makes the model-based technique ineffective in designing application-specific FPGA architectures for certain application domains. We use the examples of applications having discrete effects for this purpose.

Chapter 6

Conclusion

In this chapter, we first summarize the contributions presented in this thesis. We then discuss the limitations of the research and describe future work required to address these limitations. Finally, we present directions for long-term future research.

6.1 Dissertation Summary

The target audience of this thesis consists of FPGA architects and/or FPGA vendors. In recent years, there have been significant improvements in FPGA architecture, and FPGA devices now provide higher density, lower power consumption and faster circuit implementations. An experimental approach is typically followed in designing new FPGA architectures. In an experimental approach, significant design-time is incurred by the collection of a range of representative benchmark circuits, making required changes in the existing CAD tools and running these benchmark circuits through the CAD flow, for each architecture under investigation.

This thesis addresses these issues by presenting a body of theory that the architects can use during design space exploration. The analytical models presented in this thesis take architecture parameters as inputs and generate the evaluation metrics, such as area, delay and routability. We envisage the analytical models to be used in early stage design space exploration, when designers do not have the luxury of modifying CAD tools and/or collecting benchmark circuits for each of the combinations of the architecture parameters that they want to investigate. During early stages of architecture investigation, architects may use our models to quickly short-list a set of interesting architectures; only these architectures need to be experimentally evaluated. This dissertation makes three contributions towards building a model-based framework for designing new FPGA architectures.

Chapter 3 presents our first contribution. In that chapter, we present analytical models for the area and delay of an FPGA implementation. We first present the wirelength model that relates the FPGA architecture parameters and a few circuit parameters to the average post-placement wirelength. Since the interconnect fabric is responsible for highest proportion of area in an FPGA device, the architects need to evaluate whether the architecture under investigation can support the wirelength requirement of a wide range of user circuits. Studies exist that estimate the wirelength of FPGA implementation [15, 131, 131]. The novelty of our work is that it identifies the effects of the architecture parameters on wirelength, rather than estimating wirelength for a given circuit on a given FPGA. Insights from our model will be helpful to the FPGA architects in understanding the effects of architecture choices on wirelength. Combined with the works from Lam et al. [104] and Fang et al. [70], our model can quickly investigate such effects. Our wirelength model has been published in [168].

In Chapter 3, we also present our work on modeling critical path delay. Specifically, we present the models that relate architectural choices to the post-technology mapping depth and the post-clustering depth. We show how these models can be used to quickly estimate the critical path delay of FPGA implementations. In modeling critical path delay, we require knowledge of the intra-cluster and inter-cluster delays. We obtain this information either from the analytical model presented by Hung et al. [90] or from the early phase of a CAD tool's placement algorithm [60]. Our depth models have been published in [60, 61].

Our models in Chapter 3 were validated against the experimental results from an academic CAD flow, VPR [21]. Through the validation results, we show that our models can effectively capture the effects of architecture choices on wirelength and critical path depths.

In Chapter 4, we present a model for the routability of an FPGA routing fabric assuming that a combined global/detailed router is used. Several earlier works proposed techniques to estimate routability for FPGA and ASIC implementations [27, 35, 99, 120, 175, 194]. The works from ASIC domain can also be used in FPGA domain. The focus of these earlier works was not to relate the architecture choices to routability, a contrast to the objective of this thesis. The only work [24] that relates FPGA architecture choices to routability models the routability for the detailed step of a two-step global-detailed router. In such a router, the global step allocates channels segments to the nets and the detailed step assigns individual wires within these pre-defined sequence of channels to each net. Our routability model is more representative of the algorithms that the modern routers use. To capture the existence of many possible paths for routing a net, we use graph-theoric technique to model an upper-bound for the routability.

We validate our routability model against the experimental results from VPR. We find that our model can capture the trend of the effects of logic fabric and routing fabric parameters on FPGA routability. Our routability model has been published in [58].

In Chapter 5, we investigate the capabilities and limitations of our analytical models. We investigate two stages of new architecture design: (a) the very initial back-of-the envelope design stage and (b) the parameter sweep stage.

We show that analytical models can provide useful insights for the very initial back-of-the envelope stage of new architecture design. For the parameter sweep stage, we specifically investigate two issues. First, our derivations make several assumptions including architectural assumptions. These assumptions may affect the quality of architectural conclusions when a range of analytical models are combinedly used for new architecture design. Secondly, the continuous nature of analytical models may not adequately capture the behavior of applications having ‘discrete effects’.

Chapter 5 uses two design questions to investigate these two issues relevant to the second stage of new architecture design. *First,* we investigate whether results from more than one model can be combined to optimize a general-purpose FPGA architecture with respect to area and critical-path delay. We compare the results from the model-based approach with the ones from two different experimental approaches: sequential optimization and DOE-based optimization. While the first one of these two approaches is often used for FPGA architecture optimization, the second one is used in a wide range of domains (such as automotive and agriculture) for optimization purpose. We find that the architectural conclusions drawn by the model-based approach are better than the ones drawn by these much more expensive experimental approaches; the run-time for the model-based approach is 75% to 80% faster than the experimental approaches. *Secondly,* using the example of crossbar switches, we demonstrate that the analytical models may not be capable in drawing useful conclusions for applications having discrete effects. Our work related to this contribution has been published in [59].

6.2 Limitations and Short Term Future Work

This section summarizes the limitations of the three contributions described in this dissertation and identifies the possible avenues of future work to address these limitations.

6.2.1 Analytical Models Relating Architecture to Area and Delay

While presenting wirelength and depth models in Chapter 3, we assume a homogeneous FPGA architecture. Smith et al. [166] shows that the wirelength model presented for homogeneous architectures can be used to model average wirelength for heterogeneous architectures that contain different types of embedded blocks. Further study is required to enable our models to capture the effects of heterogeneous architectures on area and delay. Some other limitations of our area and delay models are listed below.

- In deriving our models, we fix segment length of the routing fabric to 1. Generalization of our model is required to address this limitation.
- Our wirelength model does not work well for pad-constrained benchmarks. To address this issue, our model needs to incorporate the I/O characteristics of an architecture. Due to the constrained number of I/O ports, some of the blocks in an FPGA may remain unused when mapping pad-constrained circuits. This effect can be captured by using a technique presented in [168].
- Our depth model cannot accurately predict the proportion of local connections for higher cluster sizes. This is primarily due to the simplified assumptions regarding the number of connections shared during clustering. Our depth model further overestimates the post-technology mapping depth values for high values of depth. One possible way to resolve these issues is to derive more complicated expressions for the number of connections (along critical path) absorbed within clusters by using the interconnect distribution model from Davis et al. [62, 63]. However, since simple equations may provide designers with more insights about the effects of architectural trade-offs, such detailed modeling should carefully balance complexity and accuracy.

6.2.2 Analytical Model Relating Architecture and Routability

Our routability model assumes a homogeneous architecture. In Section **??**, we discussed how we can extend our model to capture the effects of embedded blocks on the routability of the circuits that do not use these blocks. Detailed model needs to be developed to capture the inherent characteristics of the embedded blocks and the consequent demand that they place on routing fabric. Our routability model poses some more limitations as listed below.

- We consider two-terminal nets for modeling routability. Future work may extend the model to multi-fanout nets by modifying the graph formation technique (Stage I).

- We approximate a switch box construction in that we assume that incoming tracks from different directions can connect to a separate set of outgoing tracks. We also assume that the events describing the number of available tracks to each channel incident to a switch block are statistically independent. Further work is required to investigate the effects of these assumptions on the estimated routability.

- Finally, we assume a segment length of 1 to make the generated routing graph manageable. Extending the graph formation component of our work (Stage I) may address this issue.

Short-term work may also extend our model to capture the behavior of modern routing fabrics. For instance, in the routing fabric that we consider, the connections from the switch boxes to the sink logic blocks (clusters) are made through sink connection boxes. In contrast, modern FPGAs allows the switch blocks to directly make connections to the sink logic blocks. To capture this behavior, we need to extend the graph that we form in Section 4.2.1.

6.2.3 Applications of Analytical Models: Capabilities and Limitations

While investigating the capabilities and limitations of model-based design technique in Chapter 5, we compare the architecture conclusions from the model-based design approach and two experimental approaches. We consider identical design space for all of these approaches, and assume that the experimental approach correctly identifies the optimized architectures. Further research needs to investigate whether analytical models can identify optimized architectures that are not within the initial design space. For this purpose, an additional stage may be used with model-based design technique. This additional stage may follow the concept of the fill-in stage from [157] with the goal of extending the design-space. To demonstrate the limitations of analytical models, we investigate application-specific FPGAs only for the applications that have discrete effects. The model-based design approach needs to be investigated for a wide range of application domains to identify the applications for which models can (and can not) effectively draw correct conclusions. Furthermore, we compare the results from two experimental techniques when evaluating the capabilities of the analytical model-based design flow. It would also be possible to evaluate the model-based design approach against other experimental techniques, such as the regression-based techniques [96, 109, 136] as discussed in Section 2.3.1.4.

6.3 Long-Term Directions for Future Work

The previous section presented short-term future work for addressing the limitations of the research presented in this thesis. This section presents long-term future research directions.

6.3.1 Analytical Models for Other Evaluation Metrics

The models presented in this dissertation target architecture optimization with respect to key evaluation metrics: area, critical path delay and routability. A recent study by Rajavel and Akoglu [144] uses our models to analytically relate architecture parameters to energy consumption. Further works may extend our models and/or develop new models for more evaluation metrics, such as reliability of FPGA devices.

6.3.2 Analytical Models for Embedded Blocks

We have identified the limitations of our models in capturing the effects of heterogeneous architectures. Since the current generation FPGAs include embedded blocks without exception, it is important to extend our models for embedded blocks. We have mentioned the work of Smith et al. [168] that extends our wirelength model for heterogeneous architectures. Future research may extend other components of area and delay models for heterogeneous architectures.

While extending the other models for heterogeneous architectures, researchers may use the concept of Virtual Embedded Blocks (VEB) from Ho et al. [87] that captures the effects of embedded blocks. These VEB blocks are placed on a commercial FPGA as black-boxes in such a way as to match the intended locations of real embedded blocks. This technique then estimates the required logic resources of the target commercial FPGAs when the embedded blocks represented by the VEBs are incorporated. For this purpose, estimates of area and of delay for ASIC implementations of VEBs are used. In a similar fashion, when capturing the effects of embedded blocks on our model, area and delay of embedded blocks can be first modeled separately. These models then can be incorporated into the models developed for homogeneous architectures.

Studies may also find that due to high interactions between numerous FPGA architecture parameters, it may not be possible to model heterogeneous architectures using closed-form expressions. In such a case, a boundary needs to be defined, beyond which empirical expressions will be required to supplement the analytically derived closed-form expressions. Future work may also investigate the use of regression-based methods in exploring design space beyond this boundary.

6.3.3 Effects of Fabrication Issues

Our models use the physical area and delay models from Fang et al. [70] and Hung et al. [90]. These earlier works can be further extended to investigate the effects of the transistor level choices on evaluation metrics such as area and delay. To make the analytical models attractive to a wider audience, the issues related to sub-lithographic variations may be incorporated into our models. A study from Wong et al. [184] may be a good starting point for such an extension. While modeling chip level leakage and timing variations, Wong et al. [184] considers variations in channel length, threshold voltage and gate-oxide thickess. Models for leakage and timing variations are then used to model the yield of the FPGA chips. It may be interesting to determine whether the results from this dissertation can be used with Wong's models directly or through extensions. For instance, Wong et al. used the work from Cheng et al. [36, 37] for modeling timing variations. Cheng et al. [36, 37] modeled delay by using experimentally-obtained near-critical paths for target applications. Future research may investigate whether our depth models (or their variants) can be used to define such near-critical paths. In such case, combining our models with the works from Cheng et al. [36, 37] and Wong et al. [184] can capture the effects of architecture choices on chip-level variations.

6.3.4 Optimization by Using Convex Programming Tools

Subsequent to the publication of our models, related work [164, 166, 167] have used our area and depth models in a geometric programming framework to concurrently optimize FPGA architectures for area and critical path delay. In a similar fashion, our routability model and any other model proposed by future studies can be used in geometric programming framework.

6.3.5 Optimization of CAD Tools

Future research may investigate whether the insights from analytical models can be used to optimize or enhance existing CAD tools. For instance, in Chapter 3, we found that the clustering stage optimize all paths equally. Further work may find a way to predict the paths that are actually going to be critical. The clustering stage can then pack the logic blocks along the critical path more efficiently.

6.3.6 Investigation of Radically Different Architectures

The models presented in this thesis are for island-style architectures. Further investigations are required to investigate whether these models can be used for other types of FPGA architectures. Further work is also required to determine how to use analytical models in developing architectures that are different from the typical FPGA architectures that are currently used in academia or industry.

Bibliography

[1] Achronix Semiconductor Corporation. Introduction to Achronix FPGAs, 2011. URL http://www.achronix.com/?s=introduction+to+picopipe. [Online; accessed 15-May-2012]. → pages 13, 16

[2] Actel Corp. (subsequently, Microsemi Corp.). Product catalogue, May, 2009. URL http://www.actel.com/documents/ProdCat_PIB.pdf. [Online; accessed 15-May-2012]. → pages 13

[3] A. Aggarwal and D. Lewis. Routing architectures for hierarchical Field Programmable Gate Arrays. In *Computer Design: VLSI in Computers and Processors, 1994. ICCD '94. Proceedings., IEEE International Conference on*, pages 475 –478, oct 1994. doi:10.1109/ICCD.1994.331954. → pages 17

[4] P. Agrawal. On the probability of success in a routing process. *Proceedings of the IEEE*, 64(11): 1624 – 1625, nov. 1976. ISSN 0018-9219. doi:10.1109/PROC.1976.10385. → pages 24

[5] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. In *FPGA '00: Proc. of the ACM/SIGDA Int'l symposium on Field programmable gate arrays*, pages 3–12, New York, NY, USA, 2000. ACM. ISBN 1-58113-193-3. doi:http://doi.acm.org/10.1145/329166.329171. → pages 13

[6] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *VLSI, IEEE Trans. on*, 12(3):288 –298, march 2004. ISSN 1063-8210. doi:10.1109/TVLSI.2004.824300. → pages 13, 84, 86, 101

[7] Altera Corp. APEX 20K embedded programmable logic family data sheet, . URL http://www.altera.com/literature/ds/apex.pdf. [Online; accessed 15-May-2012]. → pages 13

[8] Altera Corp. Programmable I/O features of the Cyclone FPGA series, . URL http://www.altera.com/support/devices/io/features/io-features.html. [Online; accessed 15-May-2012]. → pages 15

[9] Altera Corp. Quartus II university interface program (QUIP) toolkit, . → pages 47, 58, 82, 88

[10] Altera Corp. *Press Release: Altera Unveils 28-nm Stratix V FPGA Family*. April 19, 2010. → pages 1

[11] J. H. Anderson and F. N. Najm. Power estimation techniques for FPGAs. *Very Large Scale Integrated Systems, IEEE Trans. on*, 12(10):1015–1027, 2004. ISSN 1063-8210. doi:http://dx.doi.org/10.1109/TVLS1.2004.831478. → pages 1, 21

[12] J. H. Anderson and F. N. Najm. Active leakage power optimization for FPGAs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 25(3):423–437, Mar. 2006. ISSN 0278-0070. doi:10.1109/TCAD.2005.853692. → pages 21

[13] J. Antony. *Design of experiments for engineers and scientists*. Butterworth-Heinemann, Oxford :, 2003. ISBN 0750647094. → pages 22

[14] P. J. Ashenden. *The Student's Guide to VHDL*. Morgan Kaufmann, San Fracisco, CA, USA, 1998. → pages 17

[15] S. Balachandran and D. Bhatia. A priori wirelength estimation and interconnect estimation based on circuit characteristics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 24(7):1054–1065, Jul. 2005. → pages 18, 27, 48, 123

[16] M. O. Ball. Computational complexity of network reliability analysis: An overview. *Reliability, IEEE Trans. on*, 35(3):230 –239, 1986. ISSN 0018-9529. doi:10.1109/TR.1986.4335422. → pages 8, 67

[17] J. F. Beetem. Rebel: A clustering algorithm for look-up table FPGA's. *CAD of Integrated Circuits and Systems, IEEE Trans. on*, 17(5):444–451, 1998. → pages 18

[18] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. december 2005 release. URL http://www-cad.eecs.berkeley.edu/~alanmi/abc. [Online; accessed 15-May-2012]. → pages 18

[19] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *FPL '97: Proc. of the Int'l workshop on Field-Programmable Logic and Applications*, pages 213–222, London, UK, 1997. Springer-Verlag. ISBN 3-540-63465-7. → pages 20

[20] V. Betz and J. Rose. Fpga routing architecture: segmentation and buffering to optimize speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, FPGA '99, pages 59–68, 1999. ISBN 1-58113-088-0. → pages 15

[21] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-submicron FPGAs*. Kluwer Academic Publishers, 1999. → pages 1, 11, 18, 19, 20, 21, 30, 42, 44, 47, 55, 63, 65, 82, 101, 102, 123

[22] R. P. Bharadwaj, R. Konar, P. T. Balsara, and D. Bhatia. Exploiting temporal idleness to reduce leakage power in programmable architectures. In *ASP-DAC '05: Proc. of the Asia and South Pacific Design Automation conference*, pages 651–656, New York, NY, USA, 2005. ACM. ISBN 0-7803-8737-6. doi:http://doi.acm.org/10.1145/1120725.1120985. → pages 21

[23] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proc. of the IEEE*, 78(2):264–300, Feb 1990. ISSN 0018-9219. doi:10.1109/5.52213. → pages 18

[24] S. Brown, J. Rose, and Z. G. Vranesic. A detailed router for field-programmable gate arrays. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 11(5):620–628, May 1992. ISSN 0278-0070. doi:10.1109/43.127623. → pages 19, 123

[25] S. D. Brown, J. Rose, and Z. Vranesic. A stochastic model to predict the routability of field-programmable gate arrays. *Computer-Aided Design of Circuits and Systems, IEEE Trans. on*, 12(12):1827–1838, Dec. 1993. → pages 7, 25, 65, 66, 67, 70, 71, 72, 78, 82, 84

[26] B. H. Calhoun, F. A. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in MTCMOS. In *ISLPED '03: Proc. of the Int'l symposium on Low power electronics and design*, pages 104–109, New York, NY, USA, 2003. ACM. ISBN 1-58113-682-X. doi:http://doi.acm.org/10.1145/871506.871535. → pages 21

[27] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. On routability prediction for field-programmable gate arrays. In *DAC '93: Proc. of the Design Automation conference*, pages 326–330, Jun. 1993. → pages 27, 28, 123

[28] V. C. Chan and D. M. Lewis. Area-speed tradeoffs for hierarchical field-programmable gate arrays. In *Proceedings of the 1996 ACM fourth international symposium on Field-programmable gate arrays*, FPGA '96, pages 51–57, 1996. ISBN 0-89791-773-1. → pages 17

[29] C.-C. Chang, J. Cong, and M. Xie. Optimality and scalability study of existing placement algorithms. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ASP-DAC '03, pages 621–627, 2003. ISBN 0-7803-7660-9. → pages 63

[30] S. C. Chang and M. Marek-Sadowska. Technology mapping via transformations of function graphs. In *ICCD '91: Proc. of the IEEE Int'l conference on Computer Design on VLSI in Computer & Processors*, pages 159–162, 1992. → pages 18

[31] D. Chen and J. Cong. DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs. In *ICCAD '04: Proc. of the IEEE/ACM Int'l conference on Computer-aided design*, pages 752–759, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8702-3. doi:http://dx.doi.org/10.1109/ICCAD.2004.1382677. → pages 18

[32] D. Chen, J. Cong, and P. Pan. FPGA design automation: A survey. *Foundations and Trends in Electronic Design Automation*, 1:139–169, January 2006. ISSN 1551-3076. → pages 17, 65

[33] G. Chen and E. G. Friedman. An RLC interconnect model based on Fourier analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 24(2):170–183, Feb. 2005. ISSN 0278-0070. doi:10.1109/TCAD.2004.841065. → pages 24

[34] K.-C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar. DAG-Map: graph-based FPGA technology mapping for delay optimization. *IEEE Design & Test of Computers*, 9(3):7–20, 1992. ISSN 0740-7475. doi:http://dx.doi.org/10.1109/54.156154. → pages 18

[35] C. E. Cheng. Risa: Accurate and efficient placement routability modeling. In *ICCAD '94: IEEE/ACM Int'l conference on Computer-Aided Design*, pages 690–695, Nov. 1994. → pages 7, 27, 123

[36] L. Cheng, P. Wong, F. Li, Y. Lin, and L. He. Device and architecture co-optimization for FPGA power reduction. In *Proceedings of the 42nd annual Design Automation Conference*, DAC '05, pages 915–920, 2005. ISBN 1-59593-058-2. → pages 128

[37] L. Cheng, F. Li, Y. Lin, P. Wong, and L. He. Device and architecture cooptimization for FPGA power reduction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(7):1211 –1221, july 2007. ISSN 0278-0070. doi:10.1109/TCAD.2006.888289. → pages 128

[38] S. Chin and S. Wilton. Memory footprint reduction for FPGA routing algorithms. In *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, pages 1 –8, dec. 2007. doi:10.1109/FPT.2007.4439225. → pages 20

[39] S. Y. Chin and S. J. Wilton. Towards scalable FPGA cad through architecture. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 143–152, 2011. ISBN 978-1-4503-0554-9. → pages 19

[40] S. Y. L. Chin. *Improving the run-time and memory scalability of FPGA CAD algorithms*. PhD thesis, Vancouver, BC, Canada, 08 2011. → pages 11, 17, 18, 63

[41] S. Y. L. Chin and S. J. E. Wilton. An analytical model relating FPGA architecture and place and route runtime. In *FPL '09: Int'l conference on Field Programmable Logic and Applications*, pages 146–153, Aug. 2009. → pages 3, 26

[42] S. Y. L. Chin and S. J. E. Wilton. Static and dynamic memory footprint reduction for FPGA routing algorithms. *ACM Trans. Reconfigurable Technol. Syst.*, 1(4):1–20, 2009. ISSN 1936-7406. doi:http://doi.acm.org/10.1145/1462586.1462587. → pages 20

[43] Y. J. Chong and S. Parameswaran. Flexible multi-mode embedded floating-point unit for field programmable gate arrays. In *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '09, pages 171–180, 2009. ISBN 978-1-60558-410-2. → pages 21

[44] P. Christie. Rent exponent prediction methods. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 8(6):679–688, Dec. 2000. → pages 24, 51

[45] P. Christie and J. Pineda de Gyvez. Pre-layout prediction of interconnect manufacturability. In *SLIP '01: Proc. of the Int'l workshop on System-level interconnect prediction*, pages 167–173, New York, NY, USA, 2001. ACM. ISBN 1-58113-315-4. doi:http://doi.acm.org/10.1145/368640.368826. → pages 24

[46] P. Christie and D. Stroobandt. The interpretation and application of Rent's rule. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 8(6):639–648, Dec. 2000. ISSN 1063-8210. doi:http://dx.doi.org/10.1109/92.902258. → pages 24, 78

[47] L. Ciccarelli, D. Loparco, M. Innocenti, A. Lodi, C. Mucci, and P. Rolandi. A low-power routing architecture optimized for deep sub-micron FPGAs. In *CICC '06: IEEE Custom Integrated Circuits conference, 2006*, pages 309–312, Sep. 2006. doi:10.1109/CICC.2006.320889. → pages 21

[48] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, Inc., New York, NY, USA, 1987. ISBN 0195049209. → pages 75

[49] C. J. Colbourn. Combinatorial aspects of network reliability. *Annals of Operations Research*, 33 (1):1 – 15, January 1991. ISSN 0254-5330 (Print) 1572-9338 (Online). doi:http://dx.doi.org/10.1109/TVLS1.2004.831478. → pages 67

[50] J. Cong and Y. Ding. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 13(1), Jan. 1994. → pages 18, 54

[51] J. Cong and K. Minkovich. Optimality study of logic synthesis for LUT-based FPGAs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA '06, pages 33–40, 2006. ISBN 1-59593-292-5. → pages 18

[52] J. Cong and K. Minkovich. Optimality study of logic synthesis for LUT-based FPGAs. In *FPGA '06: Proc. of the ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 33–40, New York, NY, USA, 2006. ACM. ISBN 1-59593-292-5. doi:http://doi.acm.org/10.1145/1117201.1117207. → pages 18

[53] J. Cong, J. Peck, and Y. Ding. RASP: A general logic synthesis system for SRAM-based FPGAs. In *FPGA '96: Proc. of the ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 137–143, 1996. → pages 18

[54] J. Dambre, P. Verplaetse, D. Stroobandt, and J. Van Campenhout. On Rent's rule for rectangular regions. In *Proceedings of the 2001 international workshop on System-level interconnect prediction*, SLIP '01, pages 49–56, 2001. ISBN 1-58113-315-4. → pages 51

[55] J. Dambre, P. Verplaetse, D. Stroobandt, and J. Van Campenhout. Getting more out of Donath's hierarchical model for interconnect prediction. In *Proceedings of the 2002 international workshop on System-level interconnect prediction*, SLIP '02, pages 9–16, 2002. ISBN 1-58113-481-9. → pages 51

[56] J. Dambre, P. Verplaetse, D. Stroobandt, and J. Van Campenhout. A comparison of various terminal-gate relationships for interconnect prediction in VLSI circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(1):24 –34, feb. 2003. ISSN 1063-8210. doi:10.1109/TVLSI.2002.808454. → pages 24, 28

[57] J. Das and S. Wilton. An analytical model relating FPGA architecture parameters to routability. submitted for publication. → pages 7

[58] J. Das and S. J. Wilton. An analytical model relating FPGA architecture parameters to routability. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 181–184, 2011. ISBN 978-1-4503-0554-9. → pages 3, 7, 63, 95, 124

[59] J. Das and S. J. Wilton. Accelerated FPGA architecture design: Capabilities and limitations of analytical model. In *FPT '11: IEEE International conference on Field Programmable Technology*, FPT '11, 2011. → pages 3, 7, 95, 124

[60] J. Das, S. Wilton, P. Leong, and W. Luk. Modeling post-techmapping and post-clustering FPGA circuit depth. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 205 –211, 31 2009-sept. 2 2009. doi:10.1109/FPL.2009.5272315. → pages 7, 30, 45, 46, 83, 123

[61] J. Das, A. Lam, S. Wilton, P. Leong, and W. Luk. An analytical model relating FPGA architecture to logic density and depth. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(12):2229 –2242, dec. 2011. ISSN 1063-8210. doi:10.1109/TVLSI.2010.2079339. → pages 3, 6, 7, 30, 83, 95, 123

[62] J. Davis, V. De, and J. Meindl. A stochastic wire-length distribution for gigascale integration (GSI). part I. Derivation and validation. *Electron Devices, IEEE Trans. on*, 45(3):580–589, Mar. 1998. → pages 24, 37, 38, 51, 78, 125

[63] J. Davis, V. De, and J. Meindl. A stochastic wire-length distribution for gigascale integration (GSI). part II. Applications to clock frequency, power dissipation, and chip size estimation. *Electron Devices, IEEE Trans. on*, 45(3):590–597, Mar. 1998. → pages 24, 37, 38, 125

[64] J. A. Davis and J. D. Meindl. Compact distributed RLC interconnect models-Part II: Coupled line transient expressions and peak crosstalk in multilevel networks. *Electron Devices, IEEE Trans. on*, 47(11):2078–2087, Nov. 2000. ISSN 0018-9383. doi:10.1109/16.877169. → pages 24

[65] J. A. Davis and J. D. Meindl. Compact distributed RLC interconnect models. I. Single line transient, time delay, and overshoot expressions. *Electron Devices, IEEE Trans. on*, 47(11): 2068–2077, Nov. 2000. ISSN 0018-9383. doi:10.1109/16.877168. → pages 24

[66] A. DeHon. Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization). In *FPGA '99: Proc. of the ACM/SIGDA Int'l symposium on Field Programmable Gate Arrays*, pages 69–78, New York, NY, USA, 1999. ACM. ISBN 1-58113-088-0. doi:http://doi.acm.org/10.1145/296399.296431. → pages 14, 18

[67] W. E. Donath. Placement and average interconnect lengths of computer logic. *Circuits and Systems, IEEE Trans. on*, 26(4):272–277, Apr. 1979. → pages 24, 51

[68] A. A. El Gamal. Two-dimensional stochastic models for interconnections in master-slice integrated circuits. *Circuits and Systems, IEEE Trans. on*, 26(4):127–138, Feb. 1981. → pages 7, 24

[69] E. Elmallah and H. AboElFotoh. Circular layout cutsets: An approach for improving consecutive cutset bounds for network reliability. *Reliability, IEEE Trans. on*, 55(4):602 –612, Dec. 2006. ISSN 0018-9529. doi:10.1109/TR.2006.884595. → pages 8, 67

[70] W. M. Fang and J. Rose. Modeling routing demand for early-stage FPGA architecture development. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA '08, pages 139–148, 2008. ISBN 978-1-59593-934-0. → pages 3, 4, 14, 25, 30, 31, 32, 34, 36, 37, 39, 100, 112, 123, 128

[71] W. Feng and J. W. Greene. Post-placement interconnect entropy. *IEEE Trans. VLSI Syst.*, 15(8): 945–948, 2007. → pages 26

[72] W. Feng and S. Kaptanoglu. Designing efficient input interconnect blocks for LUT clusters using counting and entropy. In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, FPGA '07, pages 23–32, 2007. ISBN 978-1-59593-600-4. → pages 26

[73] M. Feuer. Connectivity of random logic. *Computers, IEEE Trans. on*, CAS-26(4):29–33, Jan. 1982. → pages 24, 27, 37

[74] G. Fiorenza, R. Rand, and M. Y. Lanzerotti. Accounting for circuitry type in assessments of wire-length distribution models for ULSI chips. *IBM Research Report RC23506*, 2005. → pages 51

[75] R. A. Fisher. The arrangement of field experiments. *The J. of the Ministry of Agriculture*, 33: 503–513, 1926. → pages 22

[76] R. J. Francis, J. Rose, and Z. Vranesic. Technology mapping of lookup table-based FPGAs for performance. In *ICCAD '91: IEEE Int'l conference on Computer-Aided Design. Digest of Technical Papers*, pages 568–571, Nov. 1991. doi:10.1109/ICCAD.1991.185334. → pages 18

[77] H. Gao, Y. Yang, X. Ma, and G. Dong. Analysis of the effect of LUT size on FPGA area and delay using theoretical derivations. In *QED '05: Int'l symposium on Quality Electronic Design*, pages 370–374, Mar. 2005. → pages 25

[78] V. George, H. Zhang, and J. Rabaey. The design of a low energy FPGA. In *ISLPED '99: Proc. of the Int'l symposium on Low power electronics and design*, pages 188–193, New York, NY, USA, 1999. ACM. ISBN 1-58113-133-X. doi:http://doi.acm.org/10.1145/313817.313920. → pages 21

[79] T. Givargis and F. Vahid. Platune: a tuning framework for system-on-a-chip platforms. *IEEE Trans. on CAD of IC and Sys.*, 21(11):1317–1327, 2002. → pages 106

[80] M. Gort and J. Anderson. Deterministic multi-core parallel routing for FPGAs. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 78 –86, dec. 2010. doi:10.1109/FPT.2010.5681758. → pages 19

[81] M. Gort and J. H. Anderson. Reducing FPGA router run-time through algorithm and architecture. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 336 –342, sept. 2011. doi:10.1109/FPL.2011.67. → pages 19, 20

[82] D. Grant, C. Wang, and G. G. Lemieux. A CAD framework for malibu: an FPGA with time-multiplexed coarse-grained elements. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 123–132, 2011. ISBN 978-1-4503-0554-9. → pages 16

[83] J. W. Greene, S. Kaptanoglu, W. Feng, V. Hecht, J. Landry, F. Li, A. Krouglyanskiy, M. Morosan, and V. Pevzner. A 65nm flash-based FPGA fabric optimized for low cost and power. In *FPGA*, pages 87–96, 2011. → pages 16

[84] S. Gupta, J. Anderson, L. Farragher, and Q. Wang. CAD techniques for power optimization in Virtex-5 FPGAs. In *CICC '07: IEEE Custom Integrated Circuits conference, 2007*, pages 85–88, Sep. 2007. doi:10.1109/CICC.2007.4405687. → pages 21

[85] M. Hammerquist and R. Lysecky. Design space exploration for application specific FPGAS in system-on-a-chip designs. In *SOC Conference, 2008 IEEE International*, pages 279 –282, sept. 2008. doi:10.1109/SOCC.2008.4641527. → pages 16, 21

[86] Y. V. Heyden, M. Khots, and D. Massart. Three-level screening designs for the optimisation or the ruggedness testing of analytical procedures. *Analytica Chimica Acta*, 276(1):189 – 195, 1993. ISSN 0003-2670. doi:DOI:10.1016/0003-2670(93)85055-O. → pages 108

[87] C. H. Ho, R. H. W. Leong, W. Luk, S. J. E. Wilton, and S. Lopez-Buedo. Virtual embedded blocks: A methodology for evaluating embedded elements in FPGAs. In *FCCM '06: IEEE symposium on Field-Programmable Custom Computing Machines*, pages 35–44, Apr. 2006. doi:10.1109/FCCM.2006.71. → pages 21, 127

[88] M. Holland and S. Hauck. Improving performance and robustness of domain-specific CPLDs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA '06, pages 50–59, 2006. ISBN 1-59593-292-5. → pages 16

[89] M. Holland and S. Hauck. Automatic creation of domain-specific reconfigurable CPLDs for SoC. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):291 –295, feb. 2007. ISSN 0278-0070. doi:10.1109/TCAD.2006.887926. → pages 16

[90] E. Hung, S. J. E. Wilton, H. Yu, T. C. P. Chau, and P. H. W. Leong. An analytical FPGA delay path model. In *FPT '09: IEEE International conference on Field Programmable Technology*, Dec. 2009. → pages 3, 26, 30, 31, 33, 34, 37, 45, 100, 103, 112, 113, 123, 128

[91] M. Hutton. Interconnect prediction for programmable logic devices. In *SLIP ’01: Proc. of the Int’l workshop on System-level interconnect prediction*, pages 125–131, New York, NY, USA, 2001. ACM. ISBN 1-58113-315-4. doi:http://doi.acm.org/10.1145/368640.368816. → pages 27

[92] M. Hutton, D. Lewis, B. Pedersen, J. Schleicher, R. Yuan, G. Baeckler, A. Lee, R. Saini, and H. Kim. Fracturable FPGA logic elements, Altera CP-01006-1.0. URL http://www.altera.com/literature/lit-index.html. [Online; accessed 15-May-2012]. → pages 13

[93] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics*, 30(1):105–114, 1976. → pages 39, 48

[94] J. Inoue, H. Ito, S. Gomi, T. Kyogoku, T. Uezono, K. Okada, and K. Masu. Evaluation of on-chip transmission line interconnect using wire length distribution. In *ASP-DAC ’05: Proc. of the Asia and South Pacific Design Automation conference*, volume 1, pages 133–138 Vol. 1, Jan. 2005. doi:10.1109/ASPDAC.2005.1466145. → pages 24

[95] S. Jang, B. Chan, K. Chung, and A. Mishchenko. Wiremap: FPGA technology mapping for improved routability. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA ’08, pages 47–55, 2008. ISBN 978-1-59593-934-0. → pages 18

[96] W. Jia, K. A. Shaw, and M. Martonosi. Stargazer: Automated regression-based gpu design space exploration. In *ISPASS*, pages 2–13, 2012. → pages 22, 23, 126

[97] X. Jia and R. Vemuri. The gapla: a globally asynchronous locally synchronous fpga architecture. In *Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on*, pages 291 – 292, april 2005. doi:10.1109/FCCM.2005.64. → pages 16

[98] A. B. Kahng and D. Stroobandt. Wiring layer assignments with consistent stage delays. In *SLIP ’00: Proc. of the Int’l workshop on System-level Interconnect Prediction*, pages 115–122, New York, NY, USA, 2000. ACM. ISBN 1-58113-249-2. doi:http://doi.acm.org/10.1145/333032.333041. → pages 24

[99] P. Kannan, S. Balachandran, and D. Bhatia. fGREP - fast generic routing demand estimation for placed FPGA circuits. In *FPL ’01: Int’l conference on Field Programmable Logic and Applications*, pages 37–47, 2001. → pages 7, 27, 123

[100] T. Karnik and S.-M. Kang. An empirical model for accurate estimation of routing delay in FPGAs. In *ICCAD ’95: Int’l conference on Computer-Aided Design*, volume 0, page 0328, Los Alamitos, CA, USA, 1995. IEEE Computer Society. ISBN 0-8186-7213-7. doi:http://doi.ieeecomputersociety.org/10.1109/ICCAD.1995.480136. → pages 28

[101] L. T. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Inf.*, 15: 141–145, 1981. → pages 39

[102] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA ’06, pages 21–30, 2006. ISBN 1-59593-292-5. → pages 16

[103] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 26(2):203–215, Feb. 2007. ISSN 0278-0070. doi:10.1109/TCAD.2006.884574. → pages 13

[104] A. Lam, S. J. E. Wilton, P. Leong, and W. Luk. An analytical model describing the relationships between logic architecture and FPGA density. In *FPL '08: Int'l conference on Field Programmable Logic and Applications, 2008*, pages 221–226, Sep. 8-10 2008. → pages 3, 6, 25, 26, 30, 31, 32, 34, 35, 36, 39, 43, 50, 57, 60, 82, 96, 112, 123

[105] J. Lamoureux and S. J. E. Wilton. On the interaction between power-aware FPGA CAD algorithms. In *ICCAD '03: Proc. of the IEEE/ACM Int'l conference on Computer-aided Design*, page 701, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 1-58113-762-1. doi:http://dx.doi.org/10.1109/ICCAD.2003.106. → pages 21

[106] J. Lamoureux, G. G. Lemieux, and S. J. E. Wilton. Glitchless: an active glitch minimization technique for FPGAs. In *FPGA '07: Proc. of the ACM/SIGDA Int'l symposium on Field Programmable Gate Arrays*, pages 156–165, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-600-4. doi:http://doi.acm.org/10.1145/1216919.1216946. → pages 21

[107] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. *Computers, IEEE Trans. on*, C-20(12):1469–1479, Dec. 1971. → pages 4, 23

[108] M. Y. Lanzerotti, G. Fiorenza, and R. A. Rand. Interpretation of Rent's rule for ultralarge-scale integrated circuit designs, with an application to wirelength distribution models. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 12(12):1330–1347, Dec. 2004. ISSN 1063-8210. doi:10.1109/TVLSI.2004.837990. → pages 24, 51, 52

[109] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGARCH Comput. Archit. News*, 34(5):185–194, Oct. 2006. ISSN 0163-5964. → pages 22, 23, 126

[110] Y.-S. Lee and A.-H. Wu. A performance and routability-driven router for FPGAs considering path delays. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(2): 179 –185, feb 1997. ISSN 0278-0070. doi:10.1109/43.573832. → pages 65

[111] G. Lemieux and D. Lewis. *Design of Interconnection Networks for Programmable Logic*. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN 1402077009. → pages 15, 101, 102

[112] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and single-driver wires in FPGA interconnect. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 41 – 48, dec. 2004. doi:10.1109/FPT.2004.1393249. → pages 14

[113] G. G. Lemieux and S. D. Brown. A detailed routing algorithm for allocating wire segments in field-programmable gate arrays. In *in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA*, pages 215–226, 1993. → pages 19

[114] G. G. F. Lemieux, S. D. Brown, and D. Vranesic. On two-step routing for FPGAs. In *Proceedings of the 1997 international symposium on Physical design*, ISPD '97, pages 60–66, 1997. ISBN 0-89791-927-0. → pages 65

[115] 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 *FPGA '05: Proc. of the ACM/SIGDA Int'l symposium on Field-programmable gate arrays*, pages 14–20, New York, NY, USA, 2005. ACM. ISBN 1-59593-029-9. doi:http://doi.acm.org/10.1145/1046192.1046195. → pages 13, 15

[116] F. Li, D. Chen, L. He, and J. Cong. Architecture evaluation for power-efficient FPGAs. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, FPGA ’03, pages 175–184, 2003. ISBN 1-58113-651-X. → pages 1, 13, 21

[117] F. Li, Y. Lin, and L. He. FPGA power reduction using configurable dual-Vdd. In *DAC ’04: Proc. of the annual Design Automation conference*, pages 735–740, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-8. doi:http://doi.acm.org/10.1145/996566.996767. → pages 21

[118] D. K. J. Lin. A new class of supersaturated designs. *Technometrics*, 35:28–31, February 1993. ISSN 0040-1706. → pages 108

[119] Y. Lin, F. Li, and L. He. Power modeling and architecture evaluation for FPGA with novel circuits for Vdd programmability. In *FPGA ’05*, pages 199–207, 2005. doi:10.1145/1046192.1046218. → pages 1

[120] J. Lou, S. Krishnamoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. In *ISPD ’01*, pages 112–117, New York, NY, USA, 2001. ACM. ISBN 1-58113-347-2. doi:http://doi.acm.org/10.1145/369691.369749. → pages 7, 27, 123

[121] A. Ludwin, V. Betz, and K. Padalia. High-quality, deterministic parallel placement for FPGAs on commodity hardware. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA ’08, pages 14–23, 2008. ISBN 978-1-59593-934-0. → pages 19

[122] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *FPGA ’09: Proc. of the ACM/SIGDA Int’l symposium on Field Programmable Gate Arrays*, pages 133–142, Feb. 2009. → pages 13, 20, 30, 47, 82, 102

[123] J. Luu, J. H. Anderson, and J. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *FPGA*, pages 227–236, 2011. → pages 20

[124] P. Maidee, C. Ababei, and K. Bazargan. Timing-driven partitioning-based placement for island style FPGAs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 24(3): 395–406, Mar. 2005. ISSN 0278-0070. doi:10.1109/TCAD.2004.842812. → pages 18

[125] V. Manohararajah, G. R. Chiu, D. P. Singh, and S. D. Brown. Difficulty of predicting interconnect delay in a timing driven FPGA CAD flow. In *SLIP ’06: Proc. of the 2006 Int’l workshop on System-level interconnect prediction*, pages 3–8, New York, NY, USA, 2006. ACM. ISBN 1-59593-255-0. doi:http://doi.acm.org/10.1145/1117278.1117280. → pages 28

[126] V. Manohararajah, G. R. Chiu, D. P. Singh, and S. D. Brown. Predicting interconnect delay for physical synthesis in a FPGA CAD flow. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 15(8):895–903, Aug. 2007. ISSN 1063-8210. doi:10.1109/TVLSI.2007.900744. → pages 18, 28

[127] A. Marquardt, V. Betz, and J. Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *FPGA ’99: Proc. of the ACM/SIGDA Int’l symposium on Field Programmable Gate Arrays*, pages 37–46, 1999. → pages 18

[128] I. M. Masud and S. J. E. Wilton. A new switch block for segmented FPGAs. In *FPL ’99: Proc. of the Int’l workshop on Field-Programmable Logic and Applications*, pages 274–281, London, UK. Springer-Verlag. ISBN 3540664572. → pages 15

[129] L. McMurchie and C. Ebeling. PathFinder: a negotiation-based performance-driven router for FPGAs. In *FPGA '95: Proc. of the ACM third Int'l symposium on Field-programmable gate arrays*, pages 111–117, New York, NY, USA, 1995. ACM. ISBN 0-89791-743-X. doi:http://doi.acm.org/10.1145/201310.201328. → pages 21

[130] L. McMurchie and C. Ebeling. Pathfinder: a negotiation-based performance-driven router for FPGAs. In *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, pages 111–117, 1995. → pages 82

[131] R. Meeuws, K. Sigdel, Y. Yankova, and K. Bertels. High level quantitative interconnect estimation for early design space exploration. In *ICECE Technology, 2008. FPT 2008. International Conference on*, pages 317 –320, dec. 2008. doi:10.1109/FPT.2008.4762407. → pages 27, 123

[132] Microsemi SOC Products Group. IGLOO PLUS FPGAs. URL http://www.actel.com/products/IGLOOplus/default.aspx. [Online; accessed 15-May-2012]. → pages 15

[133] Microsemi SOC Products Group. Introduction to Actel FPGA architecture, 2011. URL http://www.actel.com/documents/Actel_Architecture_AN.pdf. [Online; accessed 15-May-2012]. → pages 16

[134] R. Murgai, N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Improved logic synthesis algorithms for table look up architectures. In *ICCAD '91*. → pages 119

[135] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee. Accurate area and delay estimators for FPGAs. In *DATE '02: Proc. of the conference on Design, automation and test in Europe*, page 862, Washington, DC, USA, 2002. IEEE Computer Society. → pages 17, 28

[136] K. Nepal, O. Ulusul, R. I. Bahar, and S. Reda. Fast multi-objective algorithmic-design co-exploration for FPGA-based accelerators. In *FCCM '12: IEEE symposium on Field-Programmable Custom Computing Machines*, (To appear) 2012. → pages 22, 23, 126

[137] I. Page. Closing the gap between hardware and software: hardware-software cosynthesis at Oxford. In *Hardware-Software Cosynthesis for Reconfigurable Systems (Digest No: 1996/036), IEE Colloquium on*, pages 2/1–211, Feb 1996. → pages 17

[138] J. Pistorius and M. Hutton. Placement Rent exponent calculation methods, temporal behaviour and FPGA architecture evaluation. In *SLIP '03: Int'l workshop on System-level Interconnect Prediction*, pages 31–38, New York, NY, USA, 2003. ACM. ISBN 1-58113-627-7. doi:http://doi.acm.org/10.1145/639929.639936. → pages 27

[139] J. Pistorius and M. Hutton. Placement Rent exponent calculation methods, temporal behaviour and FPGA architecture evaluation. In *SLIP '03: Int'l workshop on System-level Interconnect Prediction*, pages 31–38, New York, NY, USA, 2003. ACM. ISBN 1-58113-627-7. doi:http://doi.acm.org/10.1145/639929.639936. → pages 27, 28, 47, 52

[140] R. L. Plackett and J. P. Burman. The design of optimum multifactorial experiments. *Biometrika*, 33(4):305–325, 1946. doi:10.1093/biomet/33.4.305. → pages 107, 108

[141] K. K. W. Poon, S. J. E. Wilton, and A. Yan. A detailed power model for field-programmable gate arrays. *Design Automation of Electronic Systems, ACM Trans. on*, 10(2):279–302, 2005. ISSN 1084-4309. doi:http://doi.acm.org/10.1145/1059876.1059881. → pages 1, 18, 21

[142] J. S. Provan and M. O. Ball. Computing network reliability in time polynomial in the number of cuts. *Operations Research*, 32(3):516–526, 1984. → pages 8, 67

[143] A. Rahman, A. Fan, and R. Reif. Wire-length distribution of three-dimensional integrated circuit. In *SLIP '99*. → pages 82, 83

[144] S. T. Rajavel and A. Akoglu. An analytical energy model to accelerate FPGA logic architecture investigation. In *Proceedings of the 2011 IEEE International Conference on Field-Programmable Technology (FPT'11)*, FPT '11, 2011. → pages 26, 127

[145] J. Rose. LocusRoute: a parallel global router for standard cells. volume 0, pages 189–195, Los Alamitos, CA, USA, 1988. IEEE Computer Society. ISBN 0-8186-0864-1. doi:http://doi.ieeecomputersociety.org/10.1109/DAC.1988.14757. → pages 19

[146] J. Rose. Closing the gap between FPGAs and ASICs, Keynote Address. In *FPT '06: Int'l conference on Field-Programmable Technology*, pages viii–viii, dec 2006. → pages 24

[147] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. H. Anderson. The VTR project: architecture and CAD for FPGAs from verilog to routing. In *Proceedings of the 2012 ACM/SIGDA international symposium on Field programmable gate arrays*, pages 77–86, 2012. → pages 20, 21

[148] A. Royal and P. Y. K. Cheung. Globally asynchronous locally synchronous FPGA architectures. In *Field Programmable Logic and Applications, LNCS 2778*. Springer, 2003. → pages 16

[149] R. Y. Rubin and A. M. DeHon. Timing-driven pathfinder pathology and remediation: quantifying and reducing delay noise in vpr-pathfinder. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA, pages 173–176, 2011. ISBN 978-1-4503-0554-9. → pages 19, 95

[150] T. Sakurai. Closed-form expressions for interconnection delay, coupling, and crosstalk in VLSIs. *Electron Devices, IEEE Trans. on*, 40(1):118–124, Jan 1993. ISSN 0018-9383. doi:10.1109/16.249433. → pages 24

[151] Y. Sankar and J. Rose. Trading quality for compile time: ultra-fast placement for FPGAs. In *FPGA '99: Proc. of the 1999 ACM/SIGDA seventh Int'l symposium on Field programmable gate arrays*, pages 157–166, New York, NY, USA, 1999. ACM. ISBN 1-58113-088-0. doi:http://doi.acm.org/10.1145/296399.296449. → pages 19

[152] S. Sastry and A. Parker. Stochastic models for wireability analysis of gate arrays. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 5(1):52 – 65, January 1986. ISSN 0278-0070. → pages 7

[153] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *Solid-State Circuits, IEEE Journal of*, 20(2):510 – 522, apr 1985. ISSN 0018-9200. doi:10.1109/JSSC.1985.1052337. → pages 19

[154] L. Shang, A. S. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. In *FPGA '02: Proc. of the ACM/SIGDA Int'l symposium on Field-programmable gate arrays*, pages 157–164, New York, NY, USA, 2002. ACM. ISBN 1-58113-452-5. doi:http://doi.acm.org/10.1145/503048.503072. → pages 21

[155] J. Shanthikumar. Bounding network-reliability using consecutive minimal cutsets. *Reliability, IEEE Trans.*, 37(1):45–49, Apr 1988. ISSN 0018-9529. doi:10.1109/24.3711. → pages 8, 67, 68, 76, 77

[156] J. G. Shanthikumar. Reliability of systems with consecutive minimal cutsets. *Reliability, IEEE Trans.*, R-36(5):546–550, Dec. 1987. ISSN 0018-9529. doi:10.1109/TR.1987.5222467. → pages 8, 67, 68, 75, 76, 77

[157] D. Sheldon and F. Vahid. Making good points: application-specific Pareto-point generation for design space exploration using statistical methods. In *FPGA '09*. → pages 22, 105, 106, 107, 109, 126

[158] H. Shin and C. Kim. A simple yet effective technique for partitioning. *VLSI Systems, IEEE Trans. on*, 1:380–386, 1993. → pages 18

[159] A. W. Shogan. Sequential bounding of the reliability of a stochastic network. *Operations Research*, 24 (6):1027–1044, 1976. doi:10.1287/opre.24.6.1027. → pages 8

[160] M. Shyu, G.-M. Wu, Y.-D. Chang, and Y.-W. Chang. Generic universal switch blocks. *Computers, IEEE Transactions on*, 49(4):348 –359, apr 2000. ISSN 0018-9340. doi:10.1109/12.844347. → pages 78

[161] A. Singh and M. Marek-Sadowska. FPGA interconnect planning. In *Proceedings of the 2002 international workshop on System-level interconnect prediction*, SLIP '02, pages 23–30, 2002. ISBN 1-58113-481-9. → pages 25, 44, 55

[162] A. Singh and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, FPGA '02, pages 59–66, 2002. ISBN 1-58113-452-5. → pages 18

[163] A. Singh, G. Parthasarathy, and M. Marek-Sadowska. Interconnect resource-aware placement for hierarchical FPGAs. In *ICCAD '01: Proc. of the 2001 IEEE/ACM Int'l conference on Computer-aided design*, pages 132–136, 2001. ISBN 0-7803-7249-2. → pages 7

[164] A. Smith, G. Constantinides, and P. Cheung. FPGA architecture optimization using geometric programming. *CAD of IC and Systems, IEEE Trans. on*, 29(8):1163 –1176, aug. 2010. ISSN 0278-0070. doi:10.1109/TCAD.2010.2049046. → pages 26, 128

[165] A. M. Smith. *Heterogeneous Reconfigurable Architecture Design: An Optimisation Approach*. PhD thesis, Imperial College, United Kingdom, 2006. → pages 13, 17

[166] A. M. Smith, G. A. Constantinides, and P. Y. K. Cheung. Area estimation and optimization of FPGA routing fabrics. In *FPL '09: Int'l conference on Field Programmable Logic and Applications*, pages 256–261, Aug. 2009. → pages 26, 40, 125, 128

[167] A. M. Smith, G. A. Constantinides, S. J. E. Wilton, and P. Y. K. Cheung. Concurrently optimizing FPGA architecture parameters and transistor sizing: Implications for FPGA design. In *FPT '09: IEEE International conference on Field Programmable Technology*, Dec. 2009. → pages 26, 128

[168] A. M. Smith, S. J. Wilton, and J. Das. Wirelength modeling for homogeneous and heterogeneous FPGA architectural development. In *Proceeding of the ACM/SIGDA international symposium on*

Field programmable gate arrays, FPGA ’09, pages 181–190, 2009. ISBN 978-1-60558-410-2. → pages 3, 7, 25, 30, 31, 82, 83, 123, 125, 127

[169] D. Stroobandt. *Analytical Methods for a Priori Wire Length Estimation in Computer Systems*. PhD thesis, Gent, 11 1998. → pages 24, 51, 52

[170] D. Stroobandt. *A Priori Wire Length Estimates for Digital Design*. Kluwer Academic Publishers, 2001. → pages 24, 39

[171] S. Sutanthavibul, E. Shragowitz, and J. B. Rosen. An analytical approach to floorplan design and optimization. In *DAC ’90: Proc. of the ACM/IEEE Design Automation conference*, pages 187–192, Jun 1990. doi:10.1109/DAC.1990.114852. → pages 17

[172] R. Syed, X. Chen, Y. Ha, and B. Veeravalli. sfpga2 - a scalable gals fpga architecture and design methodology. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 314 –319, 31 2009-sept. 2 2009. doi:10.1109/FPL.2009.5272278. → pages 16

[173] Tabula Inc. Spacetime architecture: white paper, 2011. URL http://www.tabula.com/technology/TabulaSpacetime_WhitePaper.pdf. [Online; accessed 15-May-2012]. → pages 16

[174] W. C. Tang, W. H. Lo, Y. L. Wu, and S. C. Chang. FPGA technology mapping optimization by rewiring algorithms. In *ISCAS ’05: IEEE Int’l symposium on Circuits and Systems*, pages 5653–5656 Vol. 6, May 2005. doi:10.1109/ISCAS.2005.1465920. → pages 18

[175] R. Tessier and H. Giza. Balancing logic utilization and area efficiency in FPGAs. In *FPL ’00: Proc. of the Int’l workshop on Field-Programmable Logic and Applications*, pages 535–544, London, UK, 2000. Springer-Verlag. ISBN 3-540-67899-9. → pages 27, 123

[176] M. Tom and G. Lemieux. Logic block clustering of large designs for channel-width constrained FPGAs. In *Design Automation Conference, 2005. Proceedings. 42nd*, pages 726 – 731, june 2005. doi:10.1109/DAC.2005.193907. → pages 14

[177] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon. HSRA: high-speed, hierarchical synchronous reconfigurable array. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, FPGA ’99, pages 125–134, 1999. ISBN 1-58113-088-0. → pages 17

[178] T. Tuan and S. Trimberger. The power of FPGA architectures. *Xcell Journal*, pages 12–15, Second Quarter, 2007. → pages 21

[179] University of Toronto. VPR and T-VPack user’s manual. URL http://www.eecg.utoronto.ca/vpr/VPR_5.pdf. [Online; accessed 15-May-2012]. → pages xiii, 49, 84

[180] University of Wisconsin. Introduction FPGA CAD tools requirement. URL http://www.ece.wisc.edu/~kati/fpgacad/CAD_Flow0.pdf. [Online; accessed 15-May-2012]. → pages 119

[181] D. Wang, N. E. Jerger, and J. G. Steffan. DART: Fast and flexible NoC simulation using FPGAs. In *Intl. Symp. on NoC*, 2011. → pages 19

[182] S. Wilton. *Architecture and Algorithms for Field-Programmable Gate Arrays with Embedded Memory*. PhD thesis, U of Toronto, 1997. → pages 14, 15

[183] S. J. E. Wilton. *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory*. PhD thesis, University of Toronto, Ontario, Canada, 1997. → pages 65

[184] H.-Y. Wong, L. Cheng, Y. Lin, and L. He. FPGA device and architecture evaluation considering process variations. In *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pages 19 – 24, nov. 2005. doi:10.1109/ICCAD.2005.1560034. → pages 128

[185] Y.-W. C. Wong, D. F. Wong, and C. K. Wong. Universal switch modules for FPGA design. *Design Automation of Electronic Systems, ACM Trans. on*, 1:80–101, 1996. → pages 15

[186] B. Y. WU and K.-M. CHAO. *Spanning trees and optimization problems*. Chapman and Hall/CRC, 2004. → pages 39, 48

[187] Y.-L. Wu and M. Marek-Sadowska. An efficient router for 2-D field programmable gate array. In *European Design and Test Conference, 1994. EDAC, The European Conference on Design Automation. ETC European Test Conference. EUROASIC, The European Event in ASIC Design, Proceedings.*, pages 412 –416, feb-3 mar 1994. doi:10.1109/EDTC.1994.326843. → pages 65

[188] Xilinx Inc. *Press Release: Xilinx 7 Series FPGAs Slash Power Consumption by 50% and Reach 2 Million Logic Cells on Industrys First Scalable Architecture*. June 21, 2010. → pages 1

[189] Xilinx Inc. Virtex-6 family overview, June 24, 2009. URL http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. [Online; accessed 15-May-2012]. → pages 13, 15

[190] M. Yamashita. A combined clustering and placement algorithm for FPGAs. Master's thesis, The University of British Columbia, British Columbia, Canada, 2007. → pages 44

[191] A. Yan, R. Cheng, and S. J. E. Wilton. On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques. In *FPGA'02*, pages 147–156, 2002. → pages 9, 95, 98

[192] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. *Technical Report MCNC*, 1991. → pages 47, 82, 100

[193] X. Yang, E. Bozorgzadeh, and M. Sarrafzadeh. Wirelength estimation based on rent exponents of partitioning and placement. In *Proceedings of the 2001 international workshop on System-level interconnect prediction*, SLIP '01, pages 25–31, 2001. ISBN 1-58113-315-4. → pages 51

[194] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion estimation during top-down placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(1):72 –80, jan 2002. ISSN 0278-0070. doi:10.1109/43.974139. → pages 27, 28, 123

[195] P. Zarkesh-Ha, J. A. Davis, W. Loh, and J. D. Meindl. Prediction of interconnect fan-out distribution using Rent's rule. In *Proceedings of the 2000 international workshop on System-level interconnect prediction*, SLIP '00, pages 107–112, 2000. ISBN 1-58113-249-2. → pages 31, 35, 36

[196] P. Zarkesh-Ha, J. A. Davis, and J. D. Meindl. Prediction of net-length distribution for global interconnects in a heterogeneous system-on-a-chip. *Very Large Scale Integrated Systems, IEEE Trans. on*, 8(6):649–659, 2000. → pages 24, 26

[197] Y. Zhu, Y. Hu, M. B. Taylor, and C.-K. Cheng. Energy and switch area optimizations for FPGA global routing architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 14:13:1–13:25, January 2009. ISSN 1084-4309. → pages 78