Variation-Aware Placement Tool for Field Programmable Gate Array Devices

by

Christopher E. Perez

S.B., E.E. M.I.T., 2006

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 2007

[June 2007]

Author_____
Department of Electrical Engineering and Computer Science

Certified by_____
Arif Rahman, Xilinx, Inc.
Thesis Co-Supervisor

Certified by_____
Charles G. Sodini, Professor
Thesis Advisor

Accepted by_____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Acknowledgements

I would like to acknowledge and give special thanks to Dr. Arif Rahman of Xilinx, Inc. for his continual support and guidance throughout the past year and a half. With his help, I've gained a great deal of knowledge about process variation in semiconductor devices. The progress I made in this area of research is due in large part to the guidance I received during my summer stay at Xilinx, Inc. in San Jose, CA. I thank Arif Rahman, Satyaki Das, Tim Tuan, and Steve Trimberger at Xilinx Research Labs for the stimulating support and creative ideas I received during my summer internship. I would also like to thank all the Xilinx Research Labs interns for making my summer an enjoyable experience I will never forget.

In addition to the Xilinx research staff, I am pleased to thank Professor Charles Sodini for all of the encouragement he has given me, from lectures in the Independent Projects Lab class to personal one-to-one meetings in his office. I greatly appreciate the help he has given me in completing this thesis.

Lastly, I would like to thank my family and friends for their patience and words of support throughout the past year of my research. Through their inspirational words, I am motivated to continue learning something new each and every day.

Contents

Abstract 2

List of Figures 6

List of Tables 7

1 Introduction 8

1.1 Motivation for Variation-aware Tools 9

1.2 Relevant Work 10

1.3 Project Goals 11

2 Background Information 13

2.1 Framework for Variation Analysis 13

2.1.1 Process Variation: Design vs. Manufacturing Perspective 13

2.1.2 Environmental vs. Physical Variation 14

2.1.3 Physical Variation Decomposition 15

2.1.4 Building a Spatial Correlation Model 17

2.1.5 The WID Spatial Correlation Function 20

2.2 FPGA Architecture 24

2.3 FPGA Design Tool Flow 28

2.4 VPR Basics 29

3 Proposed Work: VA-VPR 34

3.1 Generating Systematic Within-Die Performance Profile 35

3.2 Variation-aware Cost Function 36

3.2.1 Block Binning 37
3.2.2 Spatial Correlation Model Integration 42
3.3 Incorporation into VPR 44
3.4 Testing VA-VPR 47
3.5 Summary of VA-VPR Development 51

4 VA-VPR Results and Comparisons 53
4.1 Spatial Correlation VA Results 56
4.2 Block Binning VA Results 64

5 Conclusions 72
5.1 Summary 72
5.2 Future Work and Extensions 74
5.3 Applications 77

A Abbreviations and Symbols 79

B VA-VPR C Code 80

C MATLAB Code 95

Bibliography 97

List of Figures

1	Examples of decreasing exponential correlation functions	21
2	Examples of decreasing double exponential correlation functions	21
3	Examples of piece-wise linear correlation functions	22
4	Examples of piece-wise linear correlation functions with non-zero baseline	23
5	Island-style FPGA architecture	25
6	Basic FPGA logic block structure	25
7	Basic structure of N clustered BLEs	26
8	Structure of an FPGA switch box	27
9	Structure of an FPGA connection box	27
10	Circuit design and implementation flow for FPGAs	28
11	Flow diagram for VPR Placer operation	31
12	Example of bounding box	32
13	Example binning scheme using M bins	38
14	Example cost table for block binning	39
15	Example of smooth cost function trend (a) and jagged cost function trend (b)	40
16	Illustration of bin contour running through region of allowed movement	41
17	Flow diagram of cost function computation for VA-VPR	45
18	Binning scheme and cost table used for Trial 1	48
19	Binning scheme and cost table used for Trial 2	48
20	Correlation functions for Trial 3 (dotted), Trial 4 (solid), and Trial 5 (dashed)	49
21	Close-up view of 4-LUT+FF logic block architecture	53
22	3D Performance Profile Surface used during testing	55
23	Flat Performance Profile used during testing	55
24	Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 3 for small arrays	57
25	Trend in Variation Cost Function for Trial 3, circuit *e64*, small array	58
26	Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 3 for large arrays	61
27	Graph of Normalized Δ_{crit} values observed for Trials 3-5	63
28	Graph of Normalized $3\sigma_{crit}$ values observed for Trials 3-5	63
29	Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 1 for small arrays	65
30	Trend in Variation Cost Function for Trial 1, circuit *e64*, small array	66
31	Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 1 for large arrays	68
32	Final placement image, Trial 0 (VPR without variation-aware cost function)	69
33	Final placement images, Trial 1 (a) and Trial 3 (b)	70

List of Tables

1	Settings used for each test trial of VA-VPR	50
2	Listing of MCNC benchmark circuits and their properties	54
3	Placement results for Trial 3 and Trial 0 for small arrays	56
4	Placement results for Trial 3 and Trial 0 for large arrays	60
5	Placement results from testing 3 spatial correlation parameter settings	62
6	Placement results for Trial 1 for small arrays	64
7	Placement results for Trial 1 for large arrays	67
8	Placement results for Trial 2 for small arrays	70

Chapter 1

Introduction

As semiconductor technologies become more advanced, process variations within microelectronic devices, including variations in channel length or in oxide thickness, play a much more important role with regards to circuit delay and leakage power dissipation [1]. These process variations cause variability in circuit performance characteristics that can be minimized by improving process control techniques, resulting in more predictable behavior. However, if we are concerned with programmable logic devices like FPGAs affected by variation, the circuit designer in the field has no control over or knowledge about the manner in which process variations affect the device and the final circuit implementation. Our research here is aimed at developing and incorporating process variation models into circuit design and implementation tools for FPGAs. With variation-aware tools, programmable logic device users will have the opportunity to produce circuit implementations that are fully optimized for performance given the presence of process variations.

1.1 Motivation for Variation-Aware Tools

In this thesis, we address the problem of optimizing the implementation of a circuit targeted for an FPGA device given the presence of process variation. For a given circuit, there are several parametric specifications such as maximum power dissipation or minimum speed that the final circuit implementation must satisfy. Current generation design tools allow us to determine whether or not those specifications will be satisfied given a deterministic set of FPGA performance characteristics. However, the presence of process variation forces us to treat the FPGA performance characteristics as samples from a stochastic process. This being so, we are now presented with the possibility of not only satisfying performance specifications, but also of improving upon those specifications with a certain probability or confidence level. The only way such improvements can be made is if we have a set of 'variation-aware' design and implementation tools capable of interpreting statistics and models that characterize the process variation. If we don't incorporate variation-aware tools in our design flow, the presence of variation in FPGAs may negatively affect the performance of the final circuit implementation. It is important that these tools, most importantly place-and-route tools, be variation-aware so that the final circuit implementation within the FPGA is fully optimized for maximum speed or minimum power dissipation.

1.2 Relevant Work

Much of the research in this area has been focused on the extraction of data related to the process variation in FPGA devices. The process of collecting and interpreting data from chips related to variation, called variation characterization, is now integrated into many yield management techniques. Researchers from Xilinx, Inc. have taken advantage of chip programmability by using FPGAs as a tool for variation characterization and process monitoring [2].

Once variation characterization is performed, analysis of the results is performed. Extensive research in the area of process variation analysis and decomposition has been contributed by S. Nassif, IBM Austin, and D. Boning, M.I.T., in [3] and [4]. These works have led to extensive modeling of process variation. The extent and impact of within-die variation and spatial correlation on circuit performance has been studied and documented by several researchers at Berkeley in [5]. Alongside this research, models describing within-die spatial correlation have been researched by L. He of UCLA in [6]. The spatial correlation model developed in [5] and [6] has proved useful for us in developing a placement tool that fully optimizes circuit implementation assuming some variation model for a given FPGA device.

Recently developed variation aware placement schemes have been researched and documented in [7] and [8]. Research in [7] proposed a variation-aware placement scheme referred to as 'block discarding policy'. The block discarding policy prohibited placement of logic blocks in regions of the FPGA that did not satisfy performance constraints due to process variation. The placement scheme discussed in section 3.2.1, referred to as 'block

binning', builds on this idea of partitioning the FPGA into variation-affected regions with varying performance characteristics. However, instead of prohibiting placement of blocks in poor performance regions, our block binning method performs placement by weighing each region by a cost factor that is inversely proportional to the region's performance.

The work in this thesis also builds upon research done in [8]. In [8], critical path delay is reduced statistically by reading in a variation map and performing chip-wise FPGA placement. Their work computed a distribution for the longest path delay in the placed circuit. Our work similarly reads in a variation map when performing placement by block binning. Furthermore, our placement scheme uses a distribution computation similar to that done in [5] to compute the distribution parameters for the longest path delay. We chose to model our testing strategy on the strategy used in [8] by performing multiple placements across several benchmark circuits and comparing the results.

1.3 Project Goals

This paper focuses on modifying the well-known academic place-and-route tool called Versatile Place-and-Route (VPR) developed by Betz and Rose at the University of Toronto [9]. The original version of VPR carried out the placement and routing of a circuit targeted to a specified 'island style' FPGA. Although several modifications have been made to VPR since its development to increase support for more complex FPGA architectures, few known modifications have been made that incorporate process variation models. Our overall goal is to obtain circuit placements from VPR for variation-affected FPGAs which result in behavior that is more predictable and reliable. This

translates to producing circuit placements with minimal nominal delays and reduced variability in delay distributions. Chapter 2 provides the background information needed to understand the VPR modifications proposed in Chapter 3. Once the modifications were completed, we tested the placement tool on several benchmark circuits of various sizes. The results of these tests are included in Chapter 4. In Chapter 5 we summarize the overall development of the variation-aware placement tool and its performance. Also included in Chapter 5 are possible project extensions and steps that can be taken in the future to further develop a variation-aware placement tool for FPGA devices.

Chapter 2

Background Information

This chapter presents a framework for the analysis of process variation that will be incorporated into our variation-aware placement tool. We base this work on research done in [3], [4], [5], and [6]. Once this foundation is established, we present an overview of the FPGA device and of the circuit design and implementation flow for FPGAs. A discussion of the VPR placement tool and the proposed variation-aware modifications follows.

2.1 Framework for Variation Analysis

2.1.1 Process Variation: Design vs. Manufacturing Perspective

Research concerning IC process variation and circuit performance variability reduction tends to be viewed from one of two perspectives: 1) the manufacturing

perspective, or 2) the design perspective. On the manufacturing side, process engineers have attempted to reduce variation through means such as improved chemical-mechanical polishing (CMP) and etching techniques. On the design side, IC designers have been creating IC architectures that are resistant to or adaptive to process variation through means such as adaptive body-biasing.

A special category of process variation research involves FPGAs affected by variation. This research focuses on improving the FPGA design tool flow so that resulting FPGA circuit implementations reliably satisfy performance and power constraints. The work in this thesis focuses on this class of research, as we work on updating a current-generation FPGA placement tool to account for device and interconnect variability within FPGA ICs.

2.1.2 Environmental vs. Physical Variation

Researchers have documented the impact that variation has on IC performance, with the most problematic variation falling within two distinct classes: environmental variation and physical variation [3]. Environmental variation consists of variability in performance factors (speed and power) due to fluctuation in supply voltage, on-chip noise, and ambient temperature. This type of variability is both spatially and temporally dependent, as it depends not only on the design of the power supply grid, but also on the changes in operating temperature over time. Physical variation consists of performance variability due to fluctuation in the physical and electrical properties of device and interconnect structures. This type of variation is largely dependent on both the processing

technology and circuit layout and is therefore more spatially than temporally dependent. Device characteristics affected by process variation include effective channel length (L_{eff}), oxide thickness (t_{ox}), and dopant concentrations. Interconnect characteristics affected by process variation include the sheet resistance and capacitance of the circuit wiring. The fluctuation in all these quantities determines the overall performance variability of the implemented circuit.

2.1.3 Physical Variation Decomposition

Physical variation can be divided into multiple stages, with each stage specified by a level of the processing technology. The lowest level of physical variation is within-die (WID) variation. This type of variation affects properties of spatially separated structures differently within the chip. For example, poly-silicon gate dimensions in regions more densely packed with transistors have been shown to vary significantly from gate dimensions in regions of the chip that are sparse [10]. This class of variation is largely determined by the circuit's layout pattern. Furthermore, because this type of variation is spatially dependent, we can model the variation with a spatial correlation function that assigns a correlation coefficient to the performance variables for each pair of sites within the die [6]. This correlation function is the basis of the modified variation-aware placement tool and is discussed in more detail in section 2.1.4.

The next level above within-die variation is die-to-die (D2D) variation. This type of variation consists of a slowly-varying, smooth fluctuation across the wafer that affects performance characteristics within the same die equally. This variation is largely

dependent on such factors as the thermal gradient across the wafer during fabrication and other processing non-uniformities across the wafer lot [3].

Both WID and D2D variation can be further decomposed into one of the following two classes depending on the cause of the variation: systematic and random variation. The systematic component of variation is caused by an identifiable, deterministic feature of the design or the process technology. One of the dominant contributors to systematic variation is stepper-induced illumination and imaging non-uniformity due to global lens aberrations [10]. This variability is a growing concern to process control engineers because the optical resolution limit of the stepper systems is slowly being approached as device sizes scale downwards. Another major contributor to systematic variation is the local layout pattern-dependent non-uniformity due to optical proximity effects. IC designers have recently been incorporating optical proximity correction (OPC) design rules into their design flows to account for the effects of this type of variation [6]. The random component of variation however cannot be easily designed for since it behaves more like a 2-D random process that is independent of the process technology and the design layout. One of the causes of this variation is the random fluctuation in dopant concentrations. In the next section, the concepts of WID, D2D, systematic, and random variation are tied into the development of the spatial correlation model to be incorporated into the variation-aware placement tool.

2.1.4 Building a Spatial Correlation Model

This section lays out the details of the spatial correlation model. Let the random variable X represent some quantity performance characteristic affected by physical variation. For example, X may represent device propagation delay. We assume X is affected by WID systematic, WID random, D2D systematic, and D2D random variation components, represented by $X_{WID,S}$, $X_{WID,R}$, $X_{D2D,S}$, and $X_{D2D,R}$, respectively. As was done in [6], we will assume a linear relationship between each variation component and X,

$$X = X_o + X_{WID,S} + X_{WID,R} + X_{D2D,S} + X_{D2D,R}. \quad (2.1)$$

Through variation characterization and analysis, we can extract estimates for the systematic variation components. Because these estimates are known before placement, we can treat the systematic components as deterministic quantities as opposed to random variables. If we lump the systematic components for the WID and D2D variation components together with the nominal performance value, X_o, then we have

$$X = X_{o,s} + X_{WID,R} + X_{D2D,R}, \quad (2.2)$$

where $X_{o,s}$ is the nominal value plus the offset due to systematic variation. For simplicity, the subscripts WID,R and $D2D,R$ are changed to W and D, respectively,

$$X = X_{o,s} + X_W + X_D. \quad (2.3)$$

By independence of the WID and D2D random variation components, the variance of X can easily be computed as

$$\sigma_X^2 = \sigma_W^2 + \sigma_D^2 \quad (2.4)$$

The statistical estimation of the quantities σ_W^2, σ_D^2 from measurement data can be performed as in [6]. For our purposes, we assume these quantities are known ahead of

time. Aside from the variance of X, we can also compute the overall covariance of X taken from two different sites within the IC. We will call X_i and X_j the two random variables representing the quantities of interest affected by variation and spatially located at sites i and j within the die. It follows that

$$Cov(X_i,X_j) = Cov(X_{i,W} + X_{i,D}, X_{j,W} + X_{j,D}) = \\ Cov(X_{i,W}, X_{j,W}) + Cov(X_{i,D}, X_{j,D}). \tag{2.5}$$

To determine the two covariance quantities to the right of the above equation, we make assumptions regarding the WID and D2D components at sites i and j. First, we recall that the D2D component affects all sites within the die equally; therefore, the random variables $X_{i,D}$ and $X_{j,D}$ can both be represented by the random variable X_D. This simplifies the above expression to

$$Cov(X_i,X_j) = Cov(X_{i,W}, X_{j,W}) + Cov(X_D, X_D) = Cov(X_{i,W}, X_{j,W}) + \sigma_D^2. \tag{2.6}$$

Secondly, as is done in [6], we assume that the WID random component is a homogeneous and isotropic random field $F(x,y)$ defined over all sites within the die. This is equivalent to saying that $F(x,y)$ is a second-order stationary process whose autocorrelation function $\rho(x_i,y_i,x_j,y_j)$ is simply a function of the distance between sites i and j, or

$$\rho(x_i,y_i,x_j,y_j) = \rho(d_{ij}). \tag{2.7}$$

Thus, $X_{i,W}$ and $X_{j,W}$, the WID random components of the performance variables for any two sites i and j, will have an associated correlation coefficient $\rho_W(d)$ where d is the distance between sites i and j. The function $\rho_W(d)$ is referred to as the WID spatial correlation function. Details about the forms that $\rho_W(d)$ can take and what conditions $\rho_W(d)$ must satisfy are given in section 2.1.5.

For a given WID spatial correlation function $\rho_W(d)$, we can compute the quantity $Cov(X_{i,W},X_{j,W})$ given in equation (2.6) as

$$Cov(X_{i,W}, X_{j,W}) = \rho_W(d) \times \sigma_W^2. \quad (2.8)$$

The quantity σ_W^2 is the within-die random component variance we assumed is known a priori. It follows from equation (2.6) that the overall covariance between X_i and X_j is

$$Cov(X_i,X_j) = \rho_W(d) \times \sigma_W^2 + \sigma_D^2. \quad (2.9)$$

By definition of covariance, another way of writing this is as

$$Cov(X_i,X_j) = \rho(d) \times \sigma_T^2, \quad (2.10)$$

where σ_T^2 is the overall variance in performance at each site given by

$$\sigma_T^2 = \sigma_{Xi}^2 = \sigma_{Xj}^2 = \sigma_D^2 + \sigma_W^2. \quad (2.11)$$

We refer to $\rho(d)$ as the overall correlation function. It specifies the overall correlation coefficient between the two random variables X_i and X_j indicating performance at sites i and j, separated by distance d. Because there is a one-to-one correspondence between the WID spatial correlation function $\rho_W(d)$ in equation (2.9) and the overall correlation function $\rho(d)$ in equation (2.10), we assume that $\rho(d)$ has the same form as $\rho_W(d)$. The only difference between the function $\rho_W(d)$ and the overall correlation function $\rho(d)$ is that while $\rho_W(d)$ approaches 0 for large distances above the correlation length, $\rho(d)$ approaches a value referred to as the correlation baseline, ρ_B. The correlation baseline indicates the minimum correlation achieved by any two sites on the same die. As the ratio of D2D variation to the total variation increases, the correlation baseline should also increase. The overall correlation function model $\rho(d)$ is used in our work primarily for determining the covariance between the performance variability for any two sites within the die.

2.1.5. The WID Spatial Correlation Function

In this section, we discuss in greater depth the various forms that $\rho_W(d)$ can take, the conditions that this function must satisfy, and the considerations that go into choosing the most appropriate model for the spatial correlation function.

To be a valid correlation function for the WID random component, $\rho_W(d)$ must satisfy several conditions. The first condition is that $\rho_W(d)$ lie between 0 and 1 (more specifically, between -1 and 1, but we will assume all sites within the die are positively correlated to some degree). Another necessary condition $\rho_W(d)$ must satisfy is $\rho_W(0) = 1$ which indicates perfect correlation between $X_{i,W}$ and itself for all i. Furthermore, $\rho_W(d)$ must be a decreasing function of distance; this implies that the farther apart sites are within the die, the smaller their correlation is. This is a phenomenon that has been observed in [5] and [6] and is widely accepted to be true.

Other conditions that must be satisfied concern whether $\rho_W(d)$ gives rise to a positive semi-definite correlation matrix for all sites within the die. The work done in [6] proved that correlation functions of the form

$$\rho_W(d) = \exp(-k \times d)$$
$$\rho_W(d) = \exp(-k^2 \times d^2) \qquad (2.12)$$

yield positive semi-definite correlation matrices for any set of sites within the die. Examples of these forms are shown in Figure 1 and Figure 2.

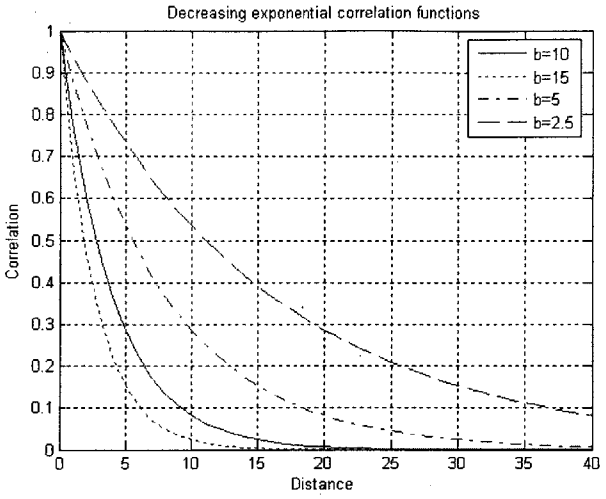


Figure 1: Examples of decreasing exponential correlation functions

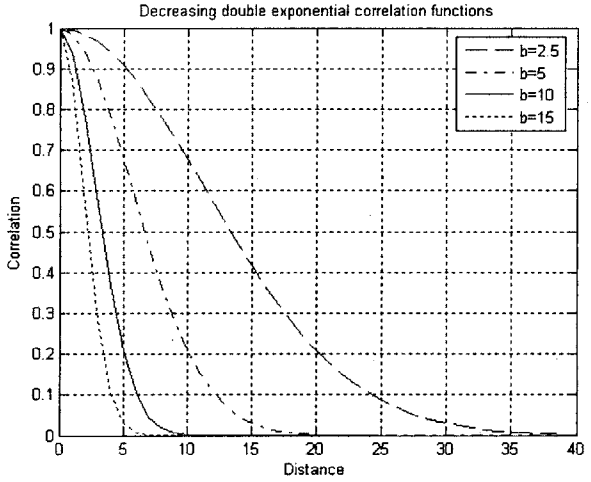


Figure 2. Examples of decreasing double exponential correlation functions

Another correlation function model considered in [5] was a decreasing piece-wise linear function of the form

$$\rho_W(d) = 1 - d/d_L,\ d \le d_L$$
$$\rho_W(d) = 0,\ d > d_L. \tag{2.13}$$

Correlation functions of this form are shown in Figure 3. In (2.1.3), d_L is defined to be the correlation length for the model. The correlation length for a variation-affected

device is assumed to be the within-chip distance beyond which the correlation between performance at sites separated by that distance is sufficiently small [6], [8]. For the exponential model, the correlation length is determined by the parameter k, defined to be the correlation decay rate. The correlation length is largely determined by the shape and gradient of the systematic WID variation component affecting the chip being targeted for placement [5]. For instance, sharp changes in the systematic WID component imply shorter values for the correlation length d_L and larger values for the decay rate k.

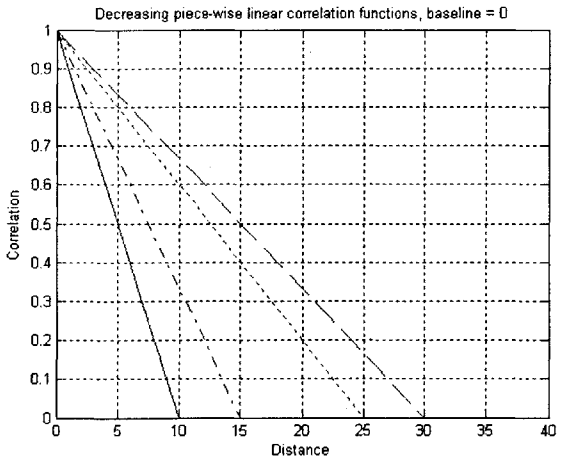


Figure 3: Examples of piece-wise linear correlation functions

The overall correlation function $\rho(d)$ resulting from $\rho_W(d)$ modeled as a piece-wise linear function takes on the form

$$\rho(d) = 1 - d/d_L \times (1 - \rho_B), \quad d \leq d_L$$
$$\rho(d) = \rho_B, \quad d > d_L \tag{2.14}$$

where ρ_B is the correlation baseline defined in the previous section. Figure 4 shows examples of overall correlation function of this form.

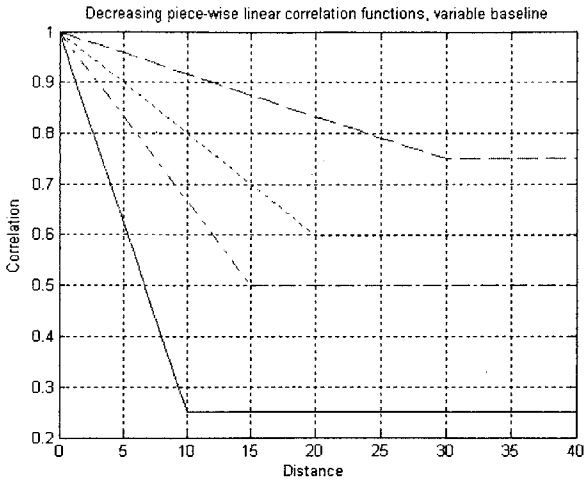


Figure 4: Examples of piece-wise linear correlation functions with non-zero baseline

Although research in [5] showed that the piece-wise linear correlation functions did model variation characterization data reasonably well, [6] proves that piece-wise linear correlation functions in some cases do not give rise to positive semi-definite correlation matrices. Despite this fact, the ease by which the piece-wise linear functions can be implemented over the exponential function implementation makes the correlation functions of the form in (2.13) attractive for our application.

Another important factor to consider in choosing an appropriate spatial correlation model for our application is whether the correlation function is specifiable in terms of a small set of parameters. This is important because when the user chooses to perform variation-aware placement using our tool, the user can completely specify the WID spatial correlation function by setting parameters at the command line. Both the exponential form shown in (2.12) and the piecewise linear form shown in (2.13) can be specified in terms of the parameters k or d_L and the correlation baseline ρ_B.

Ultimately, the ease by which the correlation function form in (2.13) can be implemented in our tool is what motivated our decision to incorporate a piece-wise linear correlation function model. Although this function may not give rise to a positive semi-definite correlation matrix for all cases, this function should serve as a reasonable model and as a good approximation to the true WID spatial correlation function.

In section 3.2.2, we show how the spatial correlation model allows us to determine the delay variability for any path in the circuit, including the circuit's critical path.

2.2 FPGA Architecture

Now that the framework for our variation analysis and correlation model is established, we focus on a special class of ICs known as field programmable gate arrays, or FPGAs.

FPGAs are programmable ICs capable of reproducing any logic function we care to implement. FPGAs come in many flavors, typified by the manner in which the chip is programmed. For example, SRAM FPGAs are programmed by sending the FPGA a bitstream that is loaded into the many SRAM memory cells located throughout the FPGA. These SRAM cells individually determine the functionality of the many components within the FPGA. The Xilinx Virtex FPGA is an example of this type.

Typical FPGAs consist of three main programmable components: logic blocks, input-output blocks (IOBs), and interconnect. Most FPGAs have programmable logic blocks arranged in a two-dimensional array with the IO blocks running along the

periphery of the array and the interconnect running through the vertical and horizontal channels in between the logic blocks. An example of this type of architecture is shown in Figure 5. This type of architecture is typically referred to as an 'Island-Style' architecture.

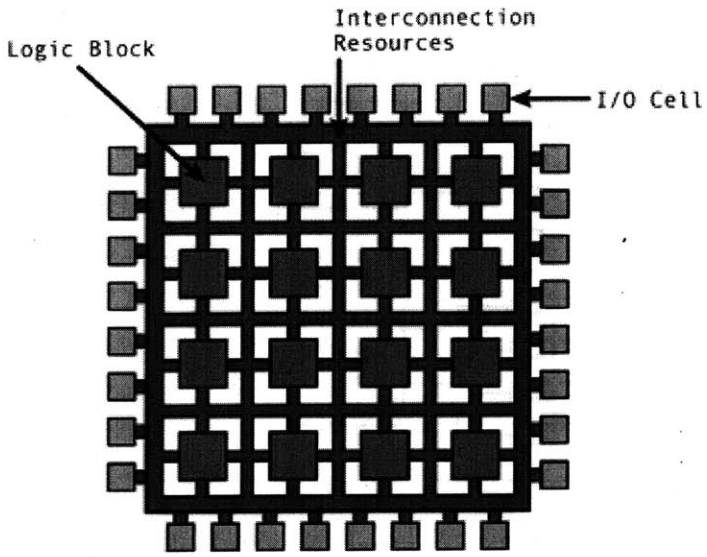


Figure 5: Island-style FPGA architecture

The most basic logic block structure consists of a programmable *K*-input lookup table (LUT) and a flip-flop (FF) device, as is shown in Figure 6.

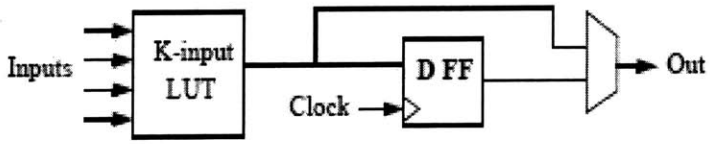


Figure 6: Basic FPGA logic block structure

This structure is typically referred to as a basic logic element (BLE). The BLE can implement any *K*-input logic function. The block can implement both combinational

logic and sequential logic via the D flip-flop. If we assume this structure for our logic blocks, then each block will have $K + 1$ inputs and 1 output.

BLE blocks can also be clustered together into groups of N BLEs to form higher-level blocks referred to as configurable logic blocks (CLBs), as shown in Figure 7.

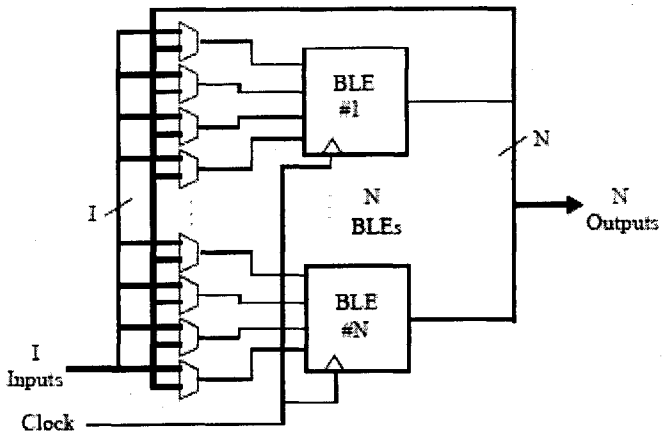


Figure 7: Basic structure of N clustered BLEs

The interconnect of the FPGA consists of an array of vertical and horizontal channels, with each channel made up of some number of routing tracks. FPGAs today normally consist of interconnect with a uniform number of routing tracks, although some research has focused on non-uniform, heterogeneous routing architectures with a variable number of tracks per channel. Programmable switch boxes at the channel intersections give signals the flexibility to propagate in any direction within the 2-D logic array. As shown in Figure 8, each switch box is programmed by specifying the state of each of the 6 switches for each track.

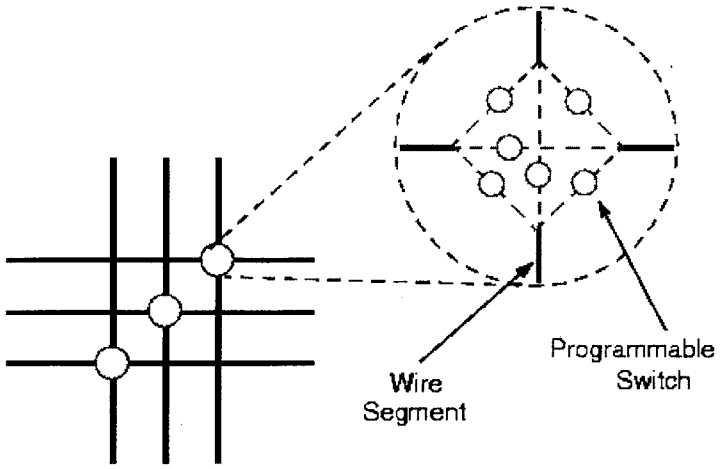


Figure 8: Structure of an FPGA switch box

Every signal that propagates to a logic block must first go through a connection block as it makes its way off the routing track and into the logic block. This connection box is a switch that is usually made up of a tri-state buffer or a pass transistor. The programmability of the switch and connection boxes gives the FPGA user the control over how signals propagate through the circuit logic.

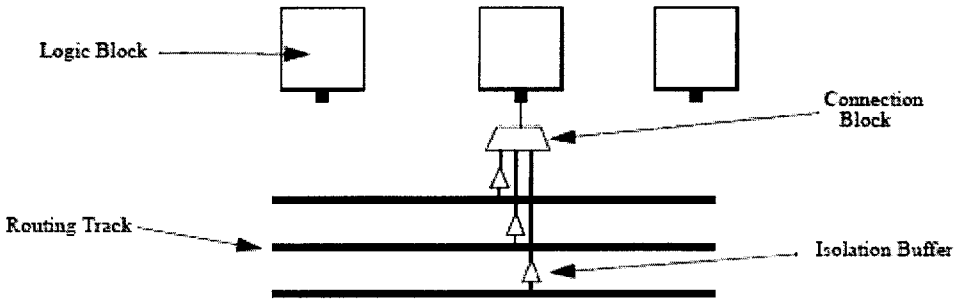


Figure 9: Structure of an FPGA connection box

2.3 FPGA Design Tool Flow

To implement a circuit on an FPGA, the circuit's functionality must be fully described by an HDL. Once this is done, the circuit description goes through a logic-synthesis stage where the circuit is transcribed into a netlist of discrete digital logic components, such as muxes, flip-flops, discrete logic gates, etc. This netlist must then be technology-mapped for a given FPGA architecture. This mapping describes the netlist in terms of the available resources for a given FPGA, such as LUTs and flip-flops.

The technology-mapped netlist is then placed and routed within the FPGA. The resulting circuit implementation within the FPGA is checked and validated by timing-analysis, simulation, and other verification procedures. If the circuit passes all the tests, then a bitstream, or binary file, is created that fully specifies the state of each of the programmable resources within the FPGA necessary to implement the circuit. This entire design-implementation flow is described in the figure below.

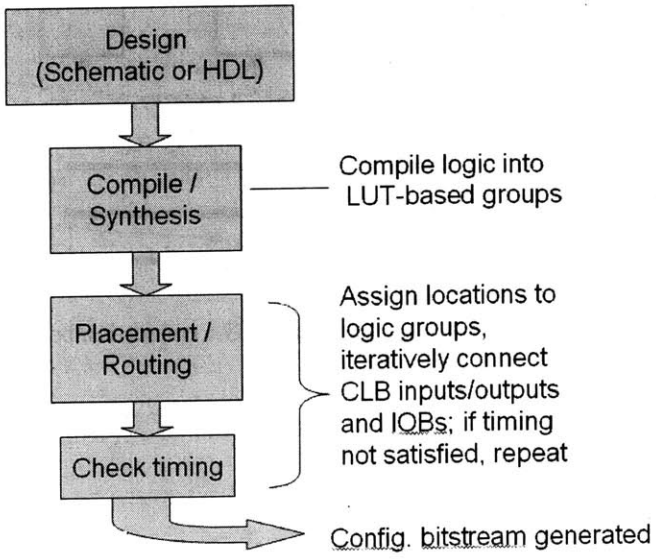


Figure 10: Circuit design and implementation flow for FPGAs

To determine at which stage of the design flow we should incorporate variation models, we must consider which stages require information pertaining to the FPGA device being targeted. The netlist of components produced by the synthesis stage doesn't include information on how variation affects the devices or interconnect of the circuit, and so there is no sense in focusing developmental efforts on variation-aware synthesis or technology-mapping tools. The next stage in the flow is to place the circuit elements within the FPGA. This involves choosing which FPGA resources to use by assigning a spatial location within the 2-D logic array to each circuit logic block. It is natural to focus our efforts on placement because variation affects the circuit devices and interconnect in a spatial manner via the WID variation component. For example, a LUT in the NW corner of the chip may behave differently than a LUT in the center of the chip. Choosing to take advantage of this fact is an important step in developing a variation-aware design flow.

2.4 VPR Basics

Modifying the functionality of the placement tool requires an understanding of inner-workings of the tool. We chose to work on VPR not only because of its prevalence in the FPGA research community, but also because of its ability to handle a multitude of FPGA architectures.

VPR reads in a circuit netlist file that has been synthesized and mapped, as well as a file describing the FPGA architecture we wish to target. The circuit netlist consists of

all the LUTs and FFs within the circuit as well as all the connections between these components. The architecture file lists physical dimensions of the FPGA, number of routing tracks, wire segment descriptions, device and interconnect timing parameters, etc. Once this information is read in, VPR calls the place subroutine.

The goal of an FPGA placement algorithm is to assign a physical location to each circuit CLB or IOB within the FPGA while optimizing the overall quality of the placement. Quality optimization is traditionally achieved in one of two ways: by minimizing the overall wire-length in the circuit, or by maximizing the overall circuit speed. Wire-length-driven placement uses the former as its optimization goal, while timing-driven placement uses the latter. These are the two optimization goals used by the current version of VPR.

VPR performs placement by means of a simulated-annealing process [11]. This process transforms an initial random placement of blocks, CLBs or IOBs, by making random swaps between blocks. After each swap, the change in the cost function resulting from the swap is computed. If the cost goes down, the swap is accepted. The advantage of using the simulated-annealing procedure is that if the cost goes up, the swap still may be accepted, allowing the placer to escape local minima points in the cost function in hopes of finding a global minimum point. The probability of accepting a swap that has increased the cost depends on a parameter called the temperature. At higher temperatures, a swap with increased cost is more likely to be accepted. Initially, the temperature is set extremely high, meaning almost all swaps are accepted. As the placement algorithm proceeds, the temperature decreases according to a pre-determined schedule. As the temperature decreases, fewer swaps resulting in increased cost are accepted. This process

allows for hill-climbing in the optimization procedure and makes convergence to a globally optimal placement configuration more likely.

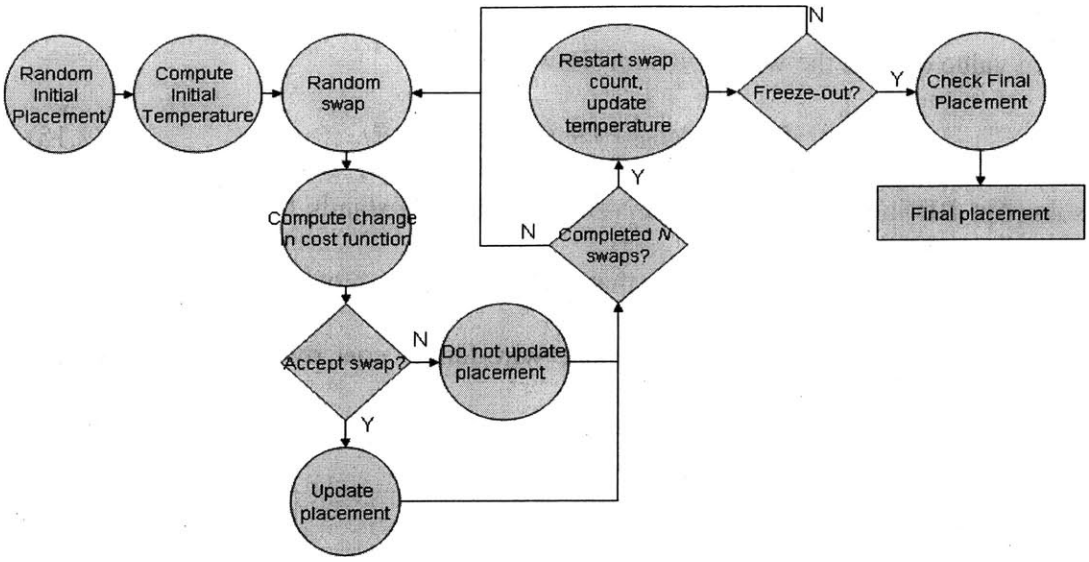


Figure 11: Flow diagram for VPR Placer operation

The distance that a block can move during a swap is also dependent on the temperature. At higher temperatures, the blocks move around within a larger range. As the temperature decreases, the blocks move around within a smaller area. This mimics the physical process of annealing, as the kinetic energy of particles decreases due to the decreased temperature, making the particles less likely to move large distances.

The simulated-annealing algorithm converges on a final placement when the temperature meets an exit criterion. In the diagram shown in Figure 11, we call this condition the freeze-out condition. Once the algorithm satisfies this condition, the resulting placement is output from VPR as the final placement configuration.

The cost function computed at each temperature change or swap is a function of the amount of overall wire-length used and the overall circuit speed. The cost determined

by the total wire-length used is represented as $f_{BB}(P)$ while the cost determined by the overall circuit speed is represented as $f_{TD}(P)$. The user can choose which portion to weigh more heavily by setting the parameter α, normalized with minimum value of 0 and maximum value of 1, in the weighted sum shown below,

$$f(P) = \alpha \times f_{BB}(P) + (1 - \alpha) \times f_{TD}(P). \quad (2.15)$$

The subscript BB in the wire-length-driven cost function stands for bounding box. The bounding box for a given net is a box that contains all the blocks connected to that net and has minimum area. VPR determines the wire-length cost by computing the length and width of the bounding box for each net in the circuit. Figure 12 below shows an example bounding box.

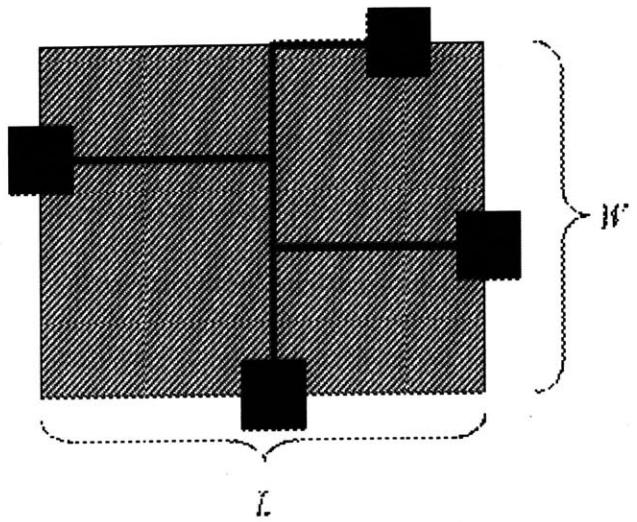


Figure 12: Example of bounding box

The wire-length-driven cost function is then computed by summing the bounding box width and length for all nets in the circuit,

$$f_{BB}(P) = \sum_i (L + W)_i. \quad (2.16)$$

To determine the timing-driven portion of the cost function, $f_{TD}(P)$, VPR computes estimates for the delays for each source-sink connection in the circuit. In addition to computing these delay estimates, VPR also computes criticalities for each source-sink connection in the circuit. The timing-driven cost function is then computed as

$$f_{TD}(P) = \sum_i \sum_j \delta_{ij} \times crit(i,j) \tag{2.17}$$

where i ranges over all nets in the circuit, j ranges over all the sink pins for the i-th net, δ_{ij} is the associated delay, and $crit(i,j)$ is the associated criticality.

Along with the final placement, VPR also outputs metrics that allow us to determine the overall quality of the final configuration of logic blocks. These metrics include the wire-length cost and timing-driven cost described above. VPR also outputs an estimate for the critical path delay. Another metric not produced by the current version of VPR but proposed here is an estimate for the critical path delay variability. This metric indicates the extent that within-die variation has widened the critical path delay distribution. Each of these metrics will prove important for our work as they allow for comparison between the quality of the placements produced by traditional VPR and our modified, variation-aware version of VPR.

Chapter 3

Proposed Work: VA-VPR

The variation-aware version of VPR, VA-VPR, consists of two modifications made to the placement sub-routine in VPR. These modifications incorporate information about the systematic WID variation component and the spatial correlation model introduced in Chapter 2. The first modification employs a method referred to as 'block binning'. This method designates regions of the FPGA logic fabric as being either 'slow', 'typical', or 'fast' on average, depending on how the systematic WID variation component has affected the local performance in that region. The second modification uses the WID spatial correlation model to reduce the overall variability in path delay for each path in the circuit. Both these variation-aware methods require knowledge of the variation present in the FPGA device we wish to target for implementation. Information about the systematic WID variation component can be represented by a performance profile of the FPGA indicating how systematic WID variation has affected performance throughout the chip.

3.1 Systematic Within-Die Performance Profile

All FPGAs are affected by WID variation that affects performance in a spatial manner. As stated above in Chapter 2, this type of variation can further be broken down into a systematic and random component. The systematic component, determined mainly by the FPGA layout, can be modeled as a deterministic quantity. In other words, if we let $F(x,y)$ be the stochastic process modeling the overall WID component of variation, then it follows that

$$F(x,y) = s(x,y) + R(x,y), \tag{3.1}$$

where $s(x,y)$ models the systematic component and $R(x,y)$ models the zero-mean random component.

The random component, as stated in Chapter 2, is assumed to be a homogeneous and isotropic field, meaning that for every site (x,y), $R(x,y)$ is identically distributed with some mean and variance. In this case, $R(x,y)$, for all (x,y), is zero-mean with variance equaling the D2D variance σ_D^2 discussed in Chapter 2.

If the performance parameter represented by the profile is the LUT delay at site (x,y), then a reasonable upper-bound for LUT delay at position (x,y) in the FPGA logic array is the 3σ point in the LUT delay distribution, computed as

$$\delta_{UB}(x,y) = \delta_o + s(x,y) + 3 \times \sigma_D, \tag{3.2}$$

where δ_o is the nominal value for LUT delay and $s(x,y)$ is the offset due to systematic WID variation. The 3σ point in the LUT delay distribution at (x,y) is chosen as the upper-bound because more than 99% of the LUT delay distribution falls below this point, assuming a Gaussian distribution for the random variation at site (x,y).

Because we assumed $s(x,y)$ and σ_D are known from variation characterization, we can assume that we know prior to placement what the upper-bound values for LUT delay are at position (x,y), given by $\delta_{UB}(x,y)$. We refer to $\delta_{UB}(x,y)$ as the performance profile for the target FPGA.

Current-generation place-and-route tools assume uniform performance characteristics for the entire die. These parameters are modeled as worst-case parameters and are usually set well above the 3σ point of the overall performance distribution to guard-band designs during place and route. Let this worst-case performance value be $\delta_{+3\sigma}$. If we take advantage of the spatial dependence of WID process variation, then we can provide better upper-bounds for LUT delays by specifying $\delta_{UB}(x,y)$ for each site in the die, since the 3σ point in the overall performance distribution is at least as large as the 3σ point in the local performance distribution for a particular site (x,y). This implies

$$\delta_{+3\sigma} \geq \delta_{UB}(x,y) \text{ for all } (x,y). \tag{3.3}$$

Details about how this performance profile for the FPGA is incorporated into the modified version of VPR are included in section 3.2.1.

3.2 Variation-aware Cost Function

Our variation-aware placer uses the same simulated annealing placement algorithm present in the original version of VPR. The changes we propose making to the placer concern the placement cost function. The current version of VPR supports a wire-length minimization cost function as well as a delay minimization cost function. We

propose adding a third cost function, referred to as $f_{VAR}(P)$, that is variation-aware. The breakdown of the overall cost function is then written as

$$f(P) = \alpha \times f_{BB}(P) + \beta \times f_{TD}(P) + \gamma \times f_{VAR}(P). \quad (3.4)$$

In this equation, the weights α, β, and γ sum to 1 for normalization purposes.

In the next section, we discuss how the variation-aware cost function $f_{VAR}(P)$ is produced by the method of block binning. In section 3.2.2, we discuss how $f_{VAR}(P)$ is produced by incorporating the spatial correlation model developed in section 2.1.4.

3.2.1 Block Binning

Block binning requires the performance profile $\delta_{UB}(x,y)$, defined in section 3.1, for the target FPGA device to account for the effects of systematic WID variation. The motivation for performing placement by block binning is to maximize the number of logic blocks placed in fast performance regions of the FPGA. This placement does not differentiate between whether or not a given logic block is part of the circuit's frequency-limiting path. Instead, the placement scheme treats every source-sink logic block connection in the circuit equally. For each source-sink connection in the circuit, the source block and sink block are classified into one of M classes. The classification is based on how that block is affected by systematic WID variation. Once the blocks are classified, a $M \times M$ cost table is used to determine what cost value should be added to the variation cost function $f_{VAR}(P)$. In other words,

$$f_{VAR}(P) = \sum_i \sum_j t(c(i),c(j)), \quad (3.5)$$

where i ranges over all nets, j ranges over all sinks connected to the i-th net, and $t(.,.)$ is the cost table addressed by the i-th and j-th classes, $c(i)$ and $c(j)$ respectively. Minimization of $f_{VAR}(P)$ is achieved by minimizing the cost value taken from the cost table for each pair of source-sink blocks. These cost values are minimized by placing the i-th source and j-th sink blocks in faster performance bins. This translates to placing the source and sink blocks in regions of the FPGA logic array that typically exhibit better performance characteristics such as lower LUT delay. A useful metric to look at when block binning is the total number of logic blocks placed in regions with poor performance. A smaller value for this metric would indicate a good placement that has compensated for the effect of the systematic WID variation component.

The classification scheme used for block binning sub-divides the range of values given by the performance profile $\delta_{UB}(x,y)$ into M bins. The spacing between the boundary values of the sub-divisions depends on the shape of the profile. A block i is placed in a bin by taking its spatial position (i_x,i_y) and determining which sub-division $\delta_{UB}(i_x,i_y)$ falls in.

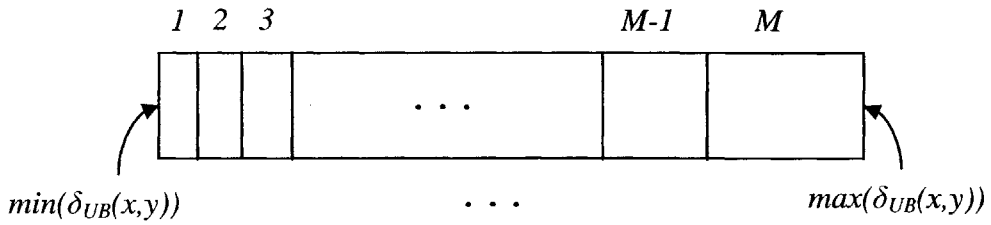


Figure 13: Example binning scheme using M bins

Once a source-sink pair of blocks is binned, the cost factor to be added for that connection is looked up in the cost table. The entries in the cost table indicate the quality of a given source-sink connection. Good quality connections have both source and sink

blocks placed in lower cost, higher performance bins while poor quality connections have both blocks placed in higher cost, lower performance bins. An example cost table with cost entries ranging from the best-case, minimum value of 0 to the worst-case, maximum value of 1 is given below for $M = 4$.

Bins	*i=1*	*i=2*	*i=3*	*i=4*
j=1	0.0	0.166	0.25	0.7
j=2	0.166	0.333	0.5	0.75
j=3	0.25	0.5	0.7	0.85
j=4	0.7	0.75	0.85	1.0

Figure 14: Example cost table for block binning with minimum cost = 0 and maximum cost = 1

One problem with block binning is that when computing the change in $f_{VAR}(P)$ for a given swap, the cost change can only take on a discrete set of values as opposed to a continuous range of values. This poses a potential problem for our placement scheme because of the nature of the simulated-annealing process. Annealing is a controlled, gradual cooling process with scheduled reheating used to prevent defects in materials. If a material undergoes cooling that is too rapid during this cooling process, then the material will not settle into an optimum defect-free state. Simulated-annealing works the same way. The placement algorithm operates on the premise that as temperature decreases, logic blocks undergo random movements within smaller and smaller ranges, resulting in smaller changes to the cost function until a globally optimal solution is found. This results in a cost function that evolves smoothly over time, as shown in part (a) of Figure

15. However, with block binning, there is the potential that as temperature decreases and random movements are restricted to smaller ranges, a movement of a block may bring about a large, sharp change in $f_{VAR}(P)$; this behavior may result in a cost function evolution similar to that shown in part (b) of the figure below.

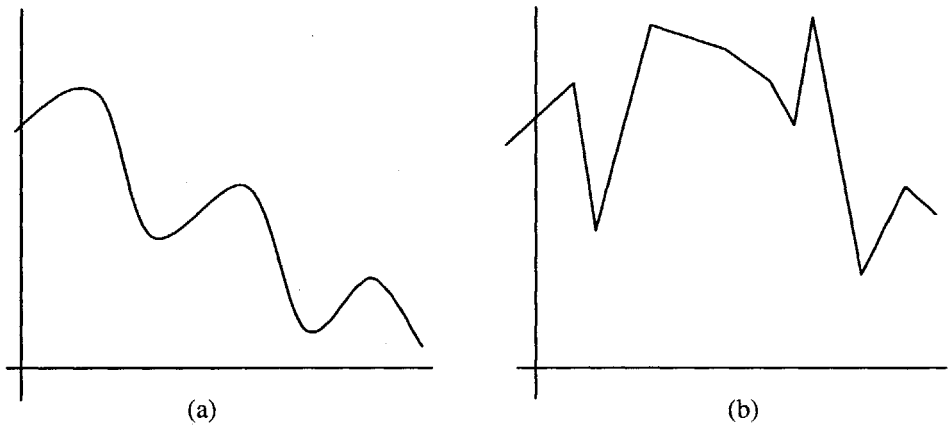


Figure 15: Example of smooth cost function trend (a) and jagged cost function trend (b)

To explain further, assume sink block j is involved in a random swap and that it is moved by 1 unit, where 1 unit corresponds to the length of 1 CLB. Assume this move is such that the performance at the blocks initial position is significantly better than the performance at the blocks new position as indicated by $\delta_{UB}(x,y)$. When block j is binned, we find that its new bin, bin 4, is 3 bins away from block j's initial bin, bin 1. If we use the cost table given in Figure 14, then this move would result in a cost change of 0.7, assuming that the source block connected to j is binned into bin 1. Such a significant increase in $f_{VAR}(P)$ may push the overall cost function, $f(P)$, above the rejection threshold for the current temperature. This would result in that move being rejected, even though the random move placed block j just 1 unit away.

It is important that the placer avoid such disturbances in the cost function evolution in pursuit of the globally optimal placement configuration. If the placer encounters sharp disturbances to the cost function during random moves within small ranges, then convergence to a placement configuration with minimum $f(P)$ cannot be guaranteed. The impact this problem poses depends mostly on the shape of the performance profile $\delta_{UB}(x,y)$ and the binning scheme employed. If $\delta_{UB}(x,y)$ is a slowly varying surface, then block movements within small ranges are not likely to bring about abrupt changes to $f(P)$. However, there is still the possibility that one of the surface contours of $\delta_{UB}(x,y)$ corresponding to a bin boundary goes through the allowed region of movement for a given logic block and temperature, as shown in Figure 16.

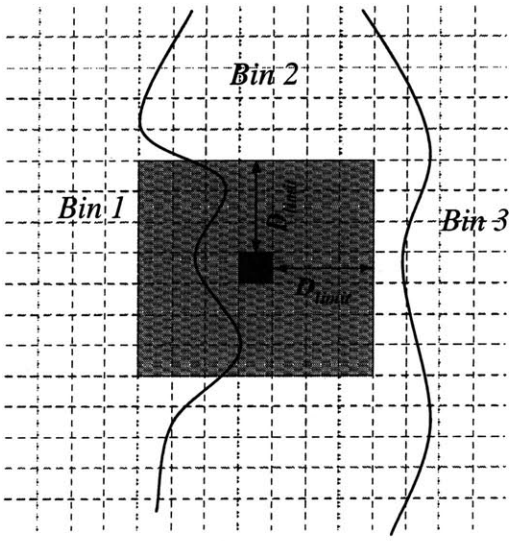


Figure 16: Illustration of bin contour running through region of allowed movement

In this example, a logic block is allowed to move anywhere in a region of 7 units ×7 units centered at that particular block. The performance profile is such that the surface contour specifying the boundary value between bin 1 and bin 2 runs through the allowed

movement region for the block shown. There is then a chance that the block changes bins during its random movement. This would bring about an abrupt change to $f_{VAR}(P)$. In this example, however, the block can only stay in the same bin or move 1 bin away, resulting in smaller cost changes than would occur if multiple bin boundary surface contours ran through the region of allowed movement.

In practice, we suspect this problem will have a greater negative impact on the placement algorithm the performance profile $\delta_{UB}(x,y)$ is not smooth or slowly-varying and is instead a surface with abrupt peaks and valleys. Fortunately, research on FPGA variation characterization has shown that $\delta_{UB}(x,y)$ can be modeled as a slowly-varying polynomial surface. Furthermore, characterization has also shown that most chips tend to show a large region of fast performance around the center of the die, while exhibiting a gradual change to slower performance along the edges and corners of the die [2]. Such a smooth profile would not likely pose any convergence issues.

3.2.2 Spatial Correlation Model Integration

The second proposed method utilizes the spatial correlation model from equation (2.14),

$$\rho(d) = 1 - d/d_L \times (1 - \rho_B), \quad d \leq d_L$$

$$\rho(d) = \rho_B, \; d > d_L.$$

We define the variation-aware cost function such that its minimization results in placements with minimal variability in path delays. If we assume that each delay stage

along a path affected by variation has variance σ_T^2, as defined in Chapter 2, then we can similarly compute the variance for the path delay distribution as

$$\Sigma = n \times \sigma_T^2 + 2 \times \sum_{i \to n} \sum_{j=i+1 \to n} Cov(i,j)$$
$$\Sigma = n \times \sigma_T^2 + 2 \times \sum_{i \to n} \sum_{j=i+1 \to n} \rho(d_{ij}) \times \sigma_T^2, \quad (3.6)$$

where i and j range over the delay stages along the path of interest [5]. The double-sum term in the above equation is taking the stages two at a time out of the set of n stages, giving us ${}_nC_2$ covariance terms.

In equation (3.6), we are assuming that the delay for a given stage is only dependent on the associated LUT delay at that particular stage. This is not quite an accurate assumption since the stage delay for any particular stage will also be dependent on characteristics from the previous and next stages along the path [5]. However, this assumption is good enough to provide us with a first-cut approximation of the variance of the circuit path delay distribution.

Equation (3.6) implies that the path delay variability is increased if stages along the circuit path are highly correlated in performance. From equation (3.6), we see that the only ways to minimize the variance for the path delay distribution are by minimizing the overall variation in LUT delay, σ_T^2, or by minimizing the spatial correlation between the LUT delays at any two stages along the circuit path.

The total LUT delay variation σ_T^2 for any stage along the path, or for any logic block along the path, is set by the manufacturing process and the chip layout. We therefore cannot reduce the delay variability by reducing σ_T^2. We can, however, reduce delay variability by minimizing the correlation between any two stages of the circuit path of interest. The spatial correlation between the LUT delays of any two logic blocks along

a particular circuit path is decreased by increasing the distance between these two logic blocks up to the correlation length, according to the correlation models discussed in section 2.1.5.

This brings us to the goal of spatial correlation model integration. We define the variation-aware cost function $f_{VAR}(P)$ to be proportional to the total delay variability between all source and sink blocks for the circuit to be placed. The definition we used for $f_{VAR}(P)$ is

$$f_{VAR}(P) = \sum_i \sum_j \rho(d_{ij}). \tag{3.7}$$

In equation (3.7), i ranges over all nets while j ranges over all sink blocks in the circuit. The minimization of this variation-aware cost function will directly result in placements with lower spatial correlation. According to equation (3.6), this will result in circuit placements with minimum variance in path delay distributions.

A useful metric to look at when checking the quality of the final placement is the variance for the critical path delay distribution. This metric is computed using equation (3.6) from above.

3.3 Updating VPR

To test the effectiveness of the two variation-aware cost function implementations discussed above, we updated the source code for the VPR program and added sub-routines that efficiently compute and update the proposed variation-aware cost function.

The diagram shown below explains in more detail how the modified placement algorithm proceeds to calculate the various cost functions.

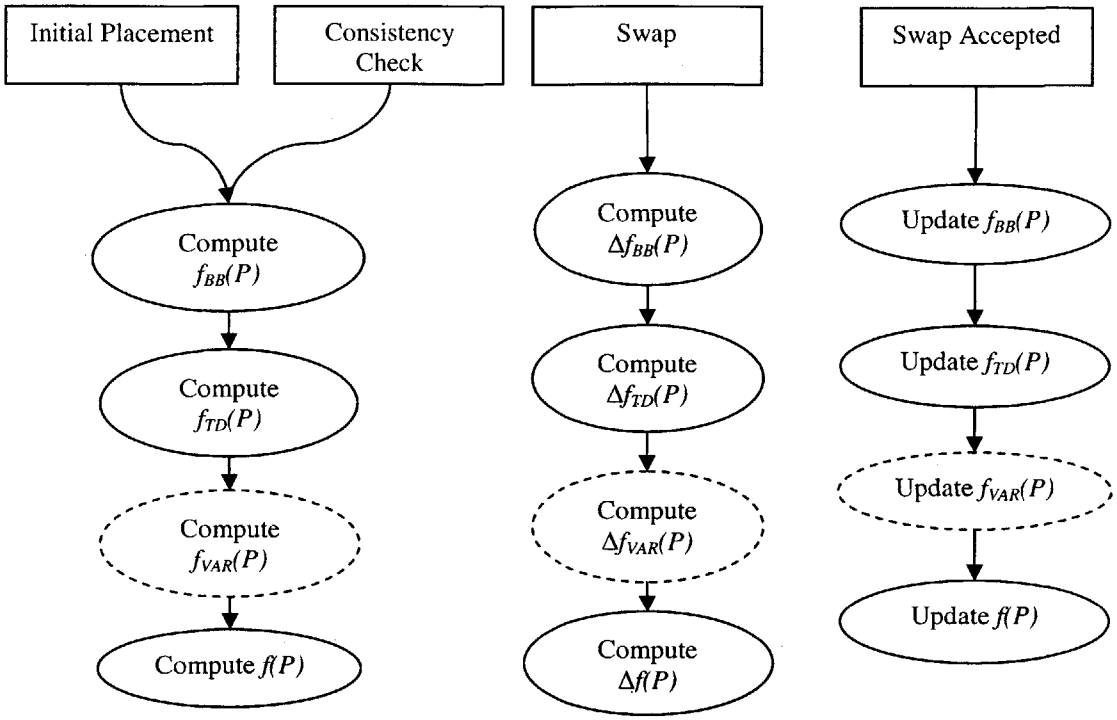


Figure 17: Flow diagram of cost function computation for VA-VPR

The equations below summarize the cost function equations that are computed during placement.

$$f_{BB}(P) = \sum_i (L + W)_i$$

$$f_{TD}(P) = \sum_i \sum_j \delta_{ij} \times crit(i,j)$$

$$f_{VAR}(P) = \sum_i \sum_j t(c(i),c(j))$$

$$f_{VAR}(P) = \sum_i \sum_j \rho(d_{ij})$$

$$f(P) = \alpha \times f_{BB}(P) + \beta \times f_{TD}(P) + \gamma \times f_{VAR}(P)$$

The user of VA-VPR can control whether the placer skips computation of the variation-aware cost function and operates just as the original version of VPR does. This choice is made at the command line with the flag *–correct_var*. If this flag is not set, then VA-VPR behaves exactly as VPR does, computing and updating a bounding box cost

function, a timing-driven cost function, and an estimate for the critical path delay. If *–correct_var* is set, the variation-aware cost function is also computed and updated.

The user can also specify the weights α, β, and γ for the bounding box cost function, timing-driven cost function, and variation-aware cost function, respectively, when computing the overall cost function, as shown in equation (3.4). In addition to specifying the cost function weights, when operating with *–correct_var* set, the user can specify which implementation of the variation-aware cost function is used. This is done by setting the flag *–va_type* to 0 for block binning, or to 1 for spatial correlation adjustment. At this point, the tool cannot handle both variation-aware cost functions, and so only one implementation method can be specified, with the default choice being the correlation based variation-aware function.

When the placer is variation-aware, the user has the option of setting the correlation baseline and correlation length for the spatial correlation model. Another parameter the user can set is the total delay variation for any given logic block, σ_T^2. This is for informational purposes only as it does not affect the variation-aware cost function. It is used when computing the estimated variance for the path delay distributions.

VPR requires 4 user arguments given at the command line: a circuit netlist file, an architecture file, a placement file, and a routing file. VA-VPR requires another two arguments: a performance profile file, and a results file. The profile file is a listing of the performance profile for each location in the logic array. For our current version of VA-VPR, the performance profile must be the same size as the FPGA size being targeted. For instance, if the user chooses to target a 20 × 20 FPGA, then the profile must contain 20 ×

20 = 400 points. An error is output if the performance profile size and the FPGA size do not match.

Although the results file is required in the current VA-VPR, it is only needed for informational purposes. Once the final costs and key metrics are computed, they are output to this file to facilitate an analysis and comparison of the results.

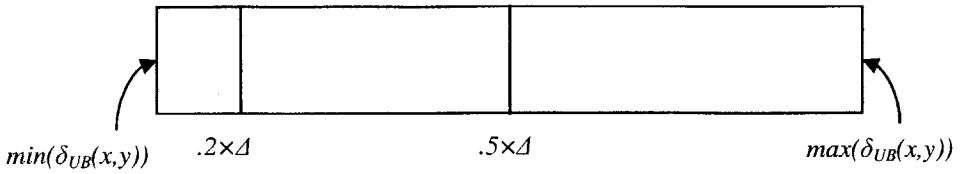
3.4 Testing VA-VPR

Our implementation of VA-VPR was tested on several circuits from the MCNC benchmark circuit set targeting a simple *4*-LUT+FF logic block architecture. For each test circuit, we targeted 2 different FPGA sizes, with the sizes set so that device utilization rates for the 2 sizes were 90% and 40%. We chose to target two different sizes to see if there is any dependence of the effectiveness of VA-VPR on device utilization.

For every FPGA size we choose to target, we must be able to produce a sample performance profile file for testing purposes. For this purpose, MATLAB scripts were used to generate various slowly-varying, smooth polynomial surfaces that can be used to model the performance profile. These surfaces are written to files that can be used as inputs to VA-VPR.

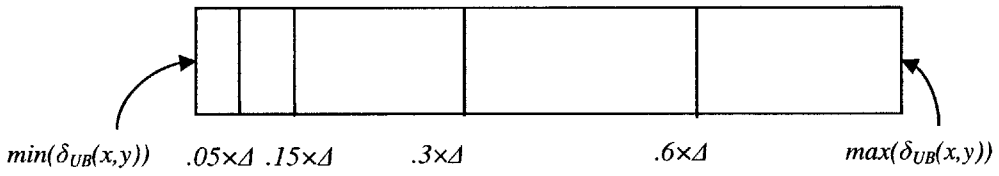
For a given test circuit and target FPGA size, we performed 6 different trial runs of VA-VPR. Trial 0 tested the original version of VPR. For Trial 1 and 2 we tested the block binning-based VA-VPR using 2 different binning schemes, the first of which is shown in Figure 18 while the second scheme is shown in Figure 19. The cost entries in

both cost tables shown were normalized to a range between 0 and 1 and were determined experimentally.



Bins	*i=1*	*i=2*	*i=3*
j=1	0.0	0.2	0.5
j=2	0.2	0.45	0.7
j=3	0.5	0.7	1.0

Figure 18: Binning scheme and cost table used for Trial 1



Bins	*i=1*	*i=2*	*i=3*	*i=4*	*i=5*
j=1	0.0	0.1	0.25	0.35	0.5
j=2	0.1	0.2	0.3	0.45	0.7
j=3	0.25	0.3	0.4	0.65	0.8
j=4	0.35	0.45	0.65	0.75	0.9
j=5	0.5	0.6	0.8	0.9	1.0

Figure 19: Binning scheme and cost table used for Trial 2

We chose to test both these block binning schemes in hopes of showing how the quality of the placements is affected by how we choose to bin the circuit logic blocks.

For Trials 3, 4, and 5, we tested the spatial correlation-based VA-VPR using the piece-wise linear spatial correlation model shown in section 2.1.5 with different settings for the correlation model parameters ρ_b and d_L. This is done to see how changes to the model affect the final placement quality. Assuming a target FPGA size of 40×40, these levels correspond to the spatial correlation functions shown in the figure below.

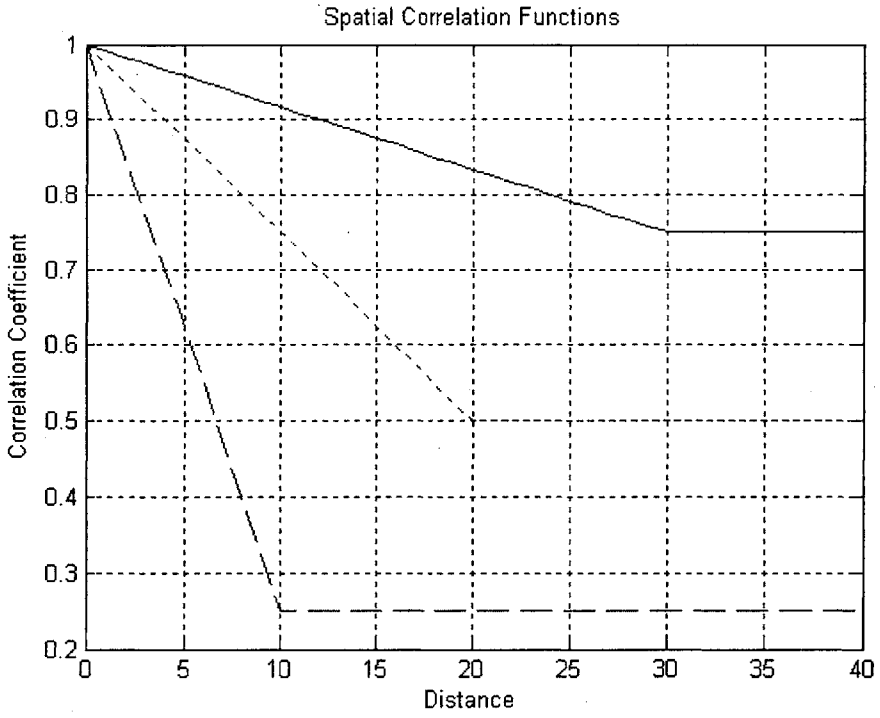


Figure 20: Spatial correlation functions used for Trial 3 (dotted), Trial 4 (solid), and Trial 5 (dashed)

The table below summarizes VPR settings for each test trial.

Trial	*Settings*
0	Not variation-aware, $\alpha = ½$, $\beta = ½$
1	VA, va_type = block binning, 3 bins, $\alpha = ⅓$, $\beta = ⅓$, $\gamma = ⅓$
2	VA, va_type = block binning, 5 bins, $\alpha = ⅓$, $\beta = ⅓$, $\gamma = ⅓$
3	VA, va_type = correlation-based, $\rho_b = 0.5$, d_L= chip-length/2, $\alpha = ⅓$, $\beta = ⅓$, $\gamma = ⅓$
4	VA, va_type = correlation-based, $\rho_b = 0.25$, d_L = chip-length/4, $\alpha = ⅓$, $\beta = ⅓$, $\gamma = ⅓$
5	VA, va_type = correlation-based, $\rho_b = 0.75$, d_L = 3×chip-length/4, $\alpha = ⅓$, $\beta = ⅓$, $\gamma = ⅓$

Table 1: Settings used for each test trial of VA-VPR

In the table above, *va_type* is used to indicate which variation-aware cost function implementation should be used. For all variation-aware trials, the weight parameters α, β, and γ were set equal to each other. We wanted to test VA-VPR without weighing the variation-aware portion of the cost function more heavily than the bounding box and timing-driven portions.

In comparing results, we chose to look at the following key metrics: $f_{BB}(P)$, $f_{TD}(P)$, $f_{VAR}(P)$, the estimated critical path delay Δ_{crit}, the estimated 3σ point for the critical path delay distribution, and the number of blocks placed in bin i, represented as N_i. By comparing the cost function values for each test trial, we can determine which settings result in decreased wire-length, decreased circuit path delay, and decreased variance for circuit path delay distributions. We also chose to look at the number of blocks placed in each bin to see which test settings minimized the number of blocks placed in poor performance regions.

3.5 Summary of VA-VPR Development

This section summarizes the modifications made to VPR to produce a variation aware placement tool capable of reducing nominal path delays and path delay variabilities. Also included in this section is a summary of the testing procedure used to produce the results shown in Chapter 4.

The simulated-annealing placement algorithm in VPR proceeds by making random moves of logic blocks while computing the changes in the total cost function resulting from those moves. The total cost function computed at each step consists of two components, a wirelength-driven component and a timing-driven component. Our work modified VPR so that a third component of the cost function is computed and updated after each move. This component, referred to as the variation-aware cost function, is computed by incorporating information about the variation present in the target device.

We proposed two different variation-aware cost functions. The first uses a known performance profile of the device to perform a method referred to as block binning. This cost function is used to minimize the number of logic blocks placed in regions of the FPGA that have slower performance on average due to systematic WID variation. The second variation-aware cost function assumes a WID spatial correlation model. This function is computed by summing over all the spatial correlation coefficients between each source and sink logic block in the circuit. The goal of this cost function is to minimize the delay variability for each circuit path, most importantly the critical path.

After the variation-aware cost functions were implemented, we tested the variation-aware placer on 9 different circuits of various sizes taken from the MCNC

benchmark circuit collection. For any given test circuit, we performed placements for each variation aware cost function while targeting FPGAs with two different utilization rates. For the spatial correlation-based cost function, we performed placements using 3 settings for the parameters defining the spatial correlation model. For the block binning based cost function, we performed placements using two different binning schemes. For each placement, we recorded the estimated critical path delay, critical path delay variability captured by the associated 3σ value, and the number of blocks placed in each performance bin. We also computed the percent changes of the critical path metrics with respect to those produced by VPR without the variation-aware cost function in hopes of comparing the performance of both placers and determining any drawbacks or advantages of the variation-aware placement tool. The results from these tests are shown in Chapter 4 and are summarized in Chapter 5.

Chapter 4

VA-VPR Results and Comparisons

In this chapter, we include the results produced by running test trials of VA-VPR. Tables containing values for the key metrics are included here, as are graphs indicating trends and example images of final placements.

Figure 21 shows a close-up view of a CLB for the simple *4*-LUT+FF logic block architecture we are targeting.

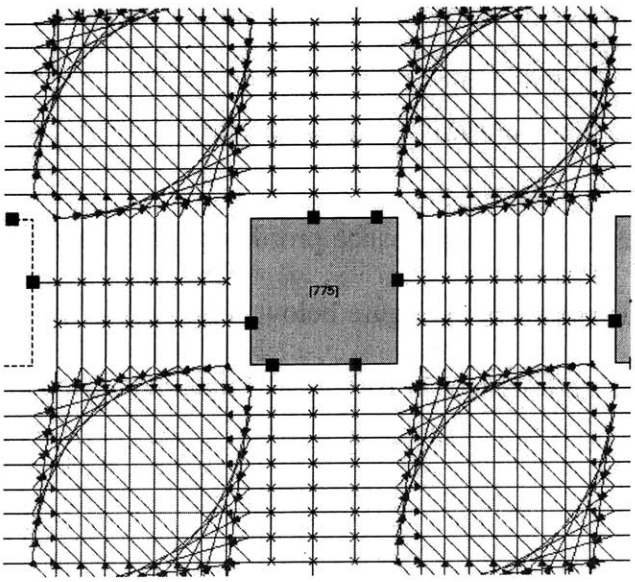


Figure 21: Close-up view of 4-LUT+FF logic block architecture

The set of circuits we chose to test our placement tool on were all taken from the MCNC benchmark circuit set. The table below lists and describes key properties for each of the circuits tested. The specific sizes targeted for each circuit are also listed in Table 2.

Circuit	Properties	FPGA sizes
e64	274 CLBs (274 LUTs, 0 latches), 130 IOBs	18x18, 30x30
alu4	1522 CLBs (1522 LUTs, 0 latches), 22 IOBs, 1536 nets	42x42, 71x71
apex2	1878 CLBs (1878 LUTs, 0 latches), 42 IOBs, 1916 nets	47x47, 78x78
apex4	1262 CLBs (1262 LUTs, 0 latches), 28 IOBs, 1271 nets	39x39, 65x65
ex5p	1064 CLBs (1064 LUTs, 0 latches), 71 IOBs, 1072 nets	36x36, 60x60
misex3	1397 CLBs (1397 LUTs, 0 latches), 28 IOBs, 1411 nets	41x41, 68x68
Seq	1750 CLBs (1750 LUTs, 0 latches), 76 IOBs, 1791 nets	46x46, 76x76
Diffeq	1497 CLBs (1494 LUTs, 377 latches), 103 IOBs, 1561 nets	42x42, 70x70
Tseng	1047 CLBs (1046 LUTs, 385 latches), 174 IOBs, 1099 nets	35x35, 59x59

Table 2: Listing of MCNC benchmark circuits and their properties

The general shape of the performance profile used for all trials, assuming a target FPGA size of 40×40, is shown in the figure below. This profile was generated assuming that the worst-case LUT performance is 10% greater than best-case LUT performance.

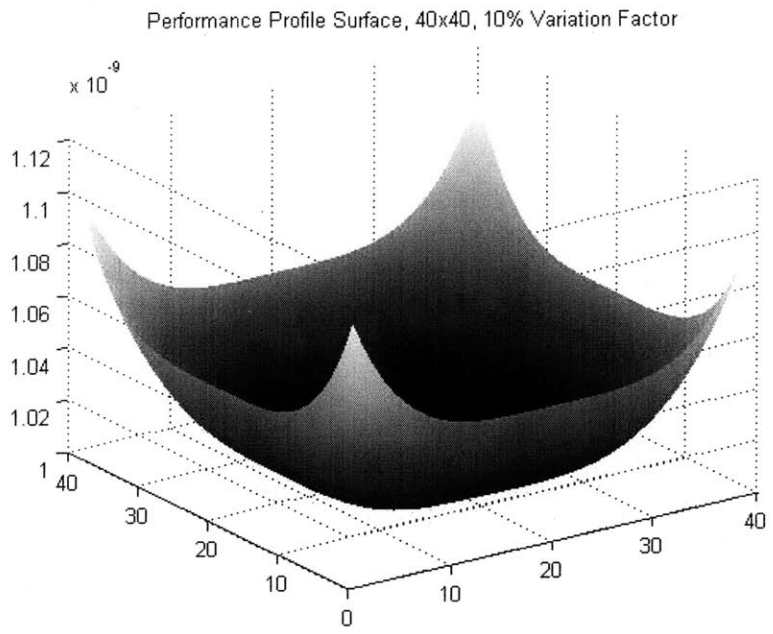


Figure 22: 3D Performance Profile Surface used during testing

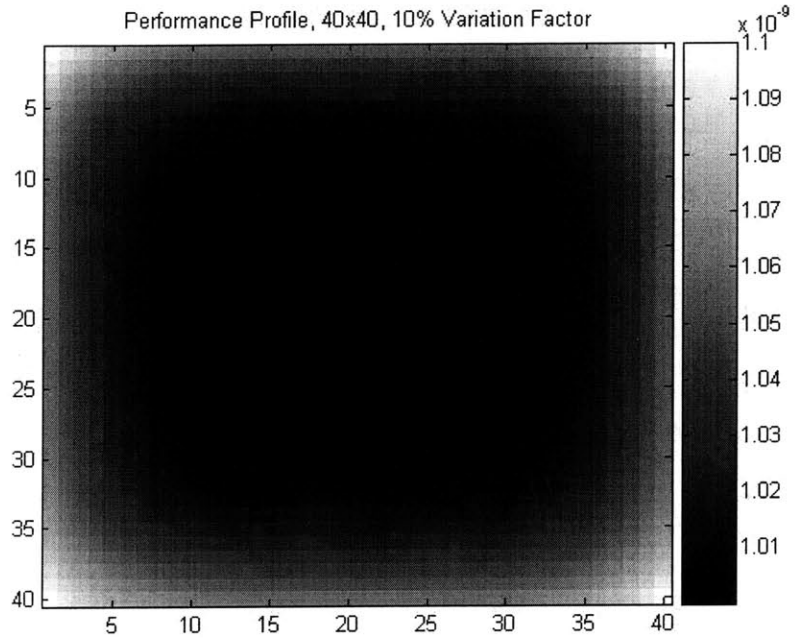


Figure 23: Flat Performance Profile used during testing

4.1 Spatial Correlation VA Results

Table 3 shows the estimated critical path delay and estimated 3σ point for the critical path delay distribution produced by Trial 3 for each of the 9 benchmark circuits of interest targeted to arrays with 90% utilization. Also included for comparison are the same metrics computed by the original version of VPR (Trial 0). The percentages shown in the 4th and 5th columns of the table represent the percent differences of Δ_{crit} and $3\sigma_{crit}$, respectively, with respect to the results from Trial 0.

e64, 20x20	Δ_{crit}	$3\sigma_{crit}$			N_1	N_2	N_3
Trial 0	3.44E-08	5.58E-10			207	66	1
Trial 3	3.32E-08	5.57E-10	-3.50%	-0.22%			
alu4, 44x44							
Trial 0	7.91E-08	9.63E-10			1080	418	24
Trial 3	8.01E-08	7.73E-10	1.32%	-19.67%			
ex5p, 38x38							
Trial 0	7.61E-08	7.66E-10			815	242	7
Trial 3	7.32E-08	8.68E-10	-3.73%	13.41%			
apex2, 49x49							
Trial 0	9.65E-08	1.03E-09			1357	508	13
Trial 3	9.98E-08	1.04E-09	3.43%	1.46%	1357	519	2
apex4, 41x41							
Trial 0	7.71E-08	7.59E-10			913	348	1
Trial 3	7.89E-08	7.82E-10	2.37%	3.01%	913	348	1
misex3, 43x43							
Trial 0	7.12E-08	8.38E-10			1023	355	19
Trial 3	7.41E-08	8.18E-10	3.98%	-2.34%	1019	360	18
seq, 48x48							
Trial 0	8.30E-08	7.85E-10			1312	436	2
Trial 3	8.68E-08	9.14E-10	4.66%	16.50%	1310	438	2
Diffeq, 44x44							
Trial 0	7.60E-08	1.63E-09			1051	428	18
Trial 3	7.39E-08	1.02E-09	-2.81%	-37.51%	1067	422	8
Tseng, 37x37							
Trial 0	6.47E-08	1.65E-09			716	328	3
Trial 3	6.36E-08	1.65E-09	-1.79%	0.00%	735	311	1

Table 3: Placement results for Trial 3 and Trial 0 for arrays with 90% utilization

The graph shown in Figure 24 indicates the Δ_{crit} and $3\sigma_{crit}$ values for each circuit normalized (divided) by the same results taken from Trial 0. In this graph, values less than 1 indicate better critical path performance by VA-VPR compared to VPR, while values greater than 1 indicate worse performance for the critical path. From this graph, we see that there is a significant dependence of the quality of VA-VPR placements on the type of circuit placed. The best critical path variability reductions were seen for the circuits *alu4* and *diffeq*, where we saw a 19.67% and 37.51% reduction in critical path delay variability, respectively.

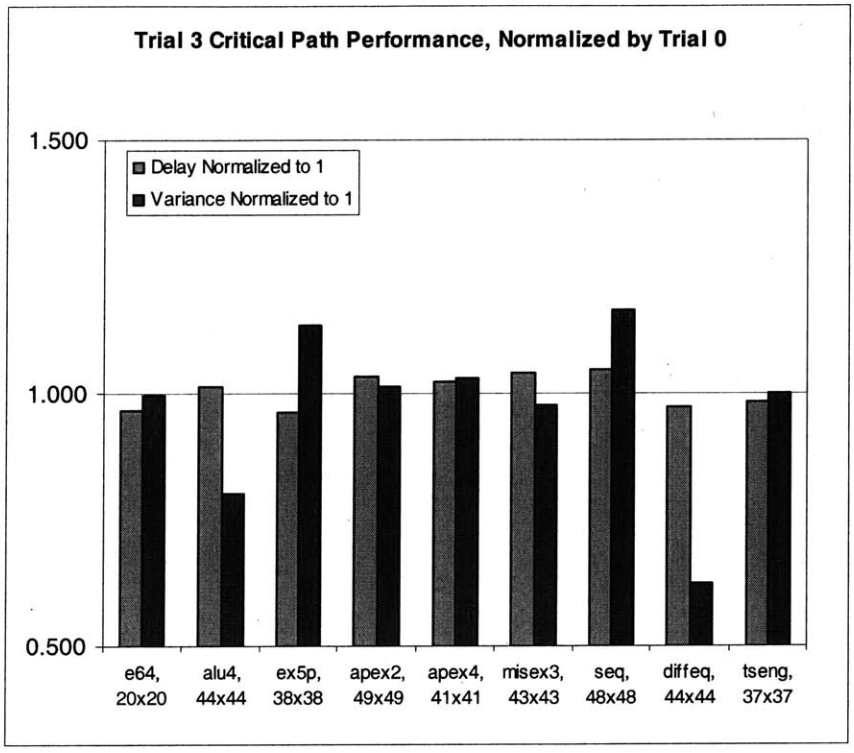


Figure 24: Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 3 for arrays with 90% utilization

Although VA-VPR performed reasonably well for most circuits tested, poor placements were observed for circuits *ex5p* and *seq*. For both these, there was more than a 10% increase in estimated critical path delay variability. Despite this result, the

estimated critical path delays predicted by VA-VPR for all circuits fell within ±5% of the delays predicted by VPR.

One possible explanation for the increase in critical path delay variability for a small subset of the test circuits may be due to the relation between the bounding box cost function $f_{BB}(P)$ and the variation-aware cost function $f_{VAR}(P)$. The graph in Figure 25 shows the trends in the correlation-based variation-aware cost function, indicating the manner by which the simulated-annealing placement algorithm converges to a final placement solution. The cost function trend consists of cost function values computed by the placer each time the temperature decreases. We expect the cost function to decrease as the placement algorithm proceeds, with larger changes in cost at higher temperatures and smaller changes at lower temperatures. The solid plot shows the trend of $f_{VAR}(P)$ with cost function weights set to α=1/3, β=1/3, and γ=1/3. The dashed plot corresponds to the weights α=0.25, β=0.25, and γ=0.5, while the dotted plot corresponds to the weights α=0.15, β=0.15, and γ=0.7.

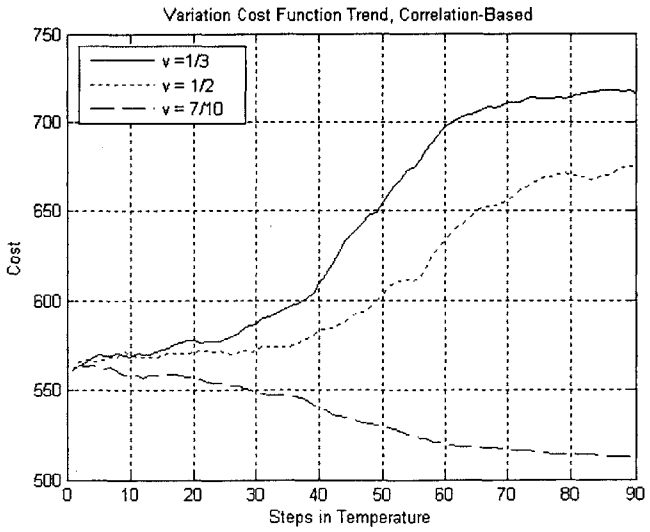


Figure 25: Trend in Variation Cost Function for Trial 3, circuit *e64*, small array

These plots show unexpected behavior in the cost function trends for smaller values of the variation-aware cost function weight γ. For the first two weight settings, $f_{VAR}(P)$ does not decrease as the placer proceeds, but instead increases. Although we expect the final cost value to decrease as we increase the weight γ, as is verified by the plots, there seems to be some inconsistency in the trend of $f_{VAR}(P)$. One explanation for this may be that for the correlation-based placer, $f_{VAR}(P)$ decreases as the sum of the correlations between all source-sink pairs decreases. Based on our spatial correlation model, this implies a decrease in $f_{VAR}(P)$ as the distance between source and sink blocks increases up to the correlation length. However, as the distance between source and sink blocks increase, the bounding box length and width for that source-sink pair increases, and as a consequence, so does the bounding box cost function $f_{BB}(P)$. This produces a tradeoff when attempting to minimize the two cost functions $f_{BB}(P)$ and $f_{VAR}(P)$. We suspect that this competition between the minimization of the two cost functions is the main cause for the correlation-based placer's poor performance for a small subset of the test circuits.

One way to get around this, as indicated by Figure 25, is to increase the weight γ for the variation-aware cost function. Doing so will weigh the benefit of placing logic blocks farther apart to reduce correlation more heavily than the benefit of placing blocks closer together to minimize the total wire-length used. This would result in path delay distributions with lower variances at the cost of an increased nominal delay value. Care should be taken however when setting γ greater than the wire-length and timing-driven weights, α and β, because a small improvement due to reduced path delay variabilities

may be overshadowed by a large degradation due to increased nominal path delays. More information regarding the weight settings and their implications is given in section 5.2.

Now that we have summarized the results produced by targeting logic arrays with 90% utilization for placement, we include results of the same tests performed targeted arrays with 40% utilization. This is done to highlight any device utilization rate dependencies that exist in the correlation-based variation-aware placer.

e64, 32x32	Δ_{crit}	$3\sigma_{crit}$			N_1	N_2	N_3
Trial 0	4.39E-08	5.89E-10			148	117	9
Trial 3	4.67E-08	5.85E-10	6.32%	-0.61%	197	74	3
alu4, 73x73							
Trial 0	9.15E-08	9.65E-10			1167	350	5
Trial 3	9.23E-08	8.46E-10	0.93%	-12.31%	1217	304	1
ex5p, 62x62							
Trial 0	8.81E-08	9.84E-10			850	214	0
Trial 3	8.96E-08	8.35E-10	1.67%	-15.08%	900	164	0
apex2, 80x80							
Trial 0	1.06E-07	1.09E-09			1253	569	56
Trial 3	1.05E-07	1.06E-09	-0.47%	-3.40%	1452	425	1
apex4, 67x67							
Trial 0	8.75E-08	8.12E-10			1063	198	1
Trial 3	9.81E-08	8.14E-10	12.17%	0.32%	1111	150	1
Misex3, 70x70							
Trial 0	8.19E-08	8.41E-10			1030	352	15
Trial 3	8.16E-08	8.37E-10	-0.31%	-0.50%	1093	303	1
seq, 78x78							
Trial 0	9.17E-08	8.38E-10			1317	427	6
Trial 3	9.97E-08	8.59E-10	8.73%	2.49%	1389	361	0
diffeq, 72x72							
Trial 0	7.34E-08	1.89E-09			1497	0	0
Trial 3	7.96E-08	1.68E-09	8.41%	-10.96%	1432	65	0
tseng, 61x61							
Trial 0	6.65E-08	1.71E-09			1047	0	0
Trial 3	6.89E-08	1.71E-09	3.58%	0.08%	1047	0	0

Table 4: Placement results for Trial 3 and Trial 0 for arrays with 40% utilization

To summarize the results of testing on larger-sized target arrays, we include the graph in Figure 26 indicating the estimated critical path delays and 3σ point for the critical path delay distributions normalized by the results taken from Trial 0. The best

improvements were seen for the circuits *alu4*, *ex5p*, and *diffeq*, where we saw a reduction in delay variability greater than 10%.

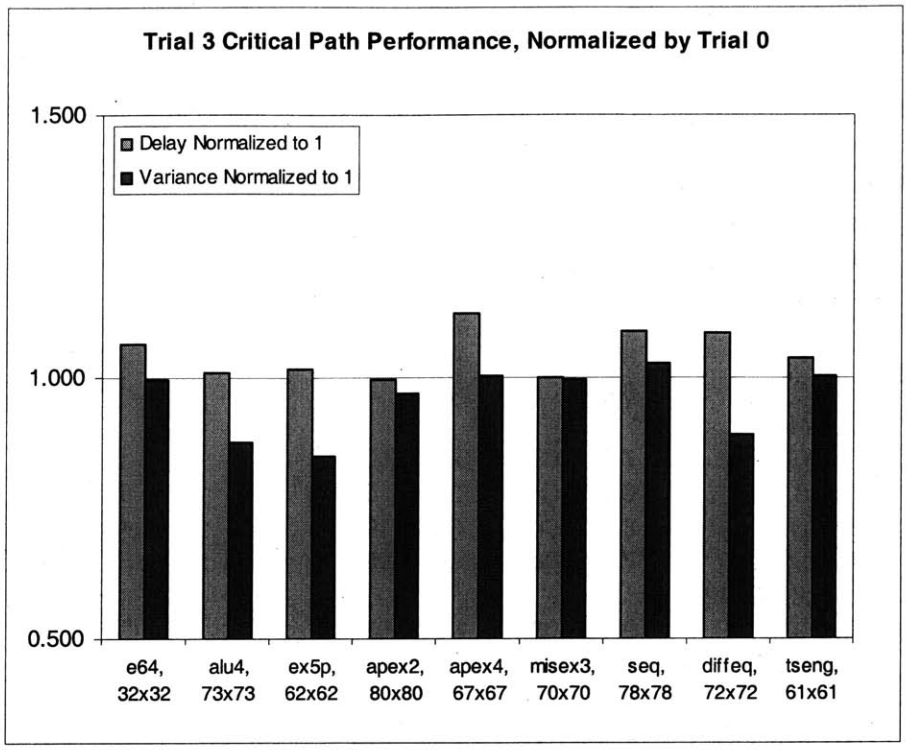


Figure 26: Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 3 for arrays with 40% utilization

The correlation-based placer performed better when targeting arrays with lower utilization rates during placement. Although the increase in critical path delay varied from <1% to 12% for most circuits, we did not observe any unexpectedly large increases in critical path delay variability, as we had observed for devices with 90% utilization. This improvement in placer performance due to the increased array sizes may be because with more available space for placement, minimization of $f_{BB}(P)$ does not necessarily result in an increase in $f_{VAR}(P)$. As the placer minimizes bounding box lengths and widths, the placer also has more freedom in choosing source-sink locations with minimal correlation coefficients.

To show how the correlation-based placer's performance depends on the spatial correlation model parameters ρ_B and d_L defined above, we have included placement results for Trials 3, 4, and 5, each corresponding to one of the three spatial correlation functions shown in Figure 20. Also shown below is a graph indicating critical path delay and delay variability values normalized by results from Trial 0.

e64, 20x20	Δ_{crit}	$3\sigma_{crit}$		
Trial 0	3.44E-08	5.58E-10		
Trial 3	3.32E-08	5.57E-10	-3.50%	-0.22%
Trial 4	3.61E-08	4.33E-10	4.86%	-5.59%
Trial 5	3.44E-08	6.24E-10	-0.02%	-0.28%
alu4, 44x44				
Trial 0	7.91E-08	9.63E-10		
Trial 3	8.01E-08	7.73E-10	1.32%	-19.67%
Trial 4	8.31E-08	7.23E-10	5.03%	-7.86%
Trial 5	7.51E-08	1.04E-09	-4.97%	0.09%
Apex2, 49x49				
Trial 0	9.65E-08	1.03E-09		
Trial 3	9.98E-08	1.04E-09	3.43%	1.46%
Trial 4	9.25E-08	7.84E-10	-4.15%	-3.67%
Trial 5	9.59E-08	1.04E-09	-0.64%	-9.47%
Apex4, 41x41				
Trial 0	7.71E-08	7.59E-10		
Trial 3	7.89E-08	7.82E-10	2.37%	3.01%
Trial 4	7.93E-08	5.95E-10	2.96%	0.25%
Trial 5	8.01E-08	8.67E-10	3.92%	0.16%

Table 5: Placement results from testing 3 spatial correlation parameter settings

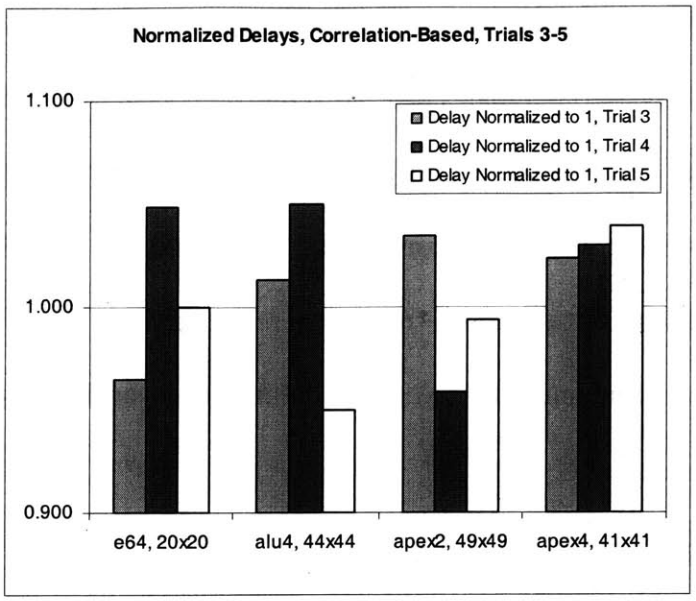


Figure 27: Graph of Normalized Δ_{crit} values observed for Trials 3-5

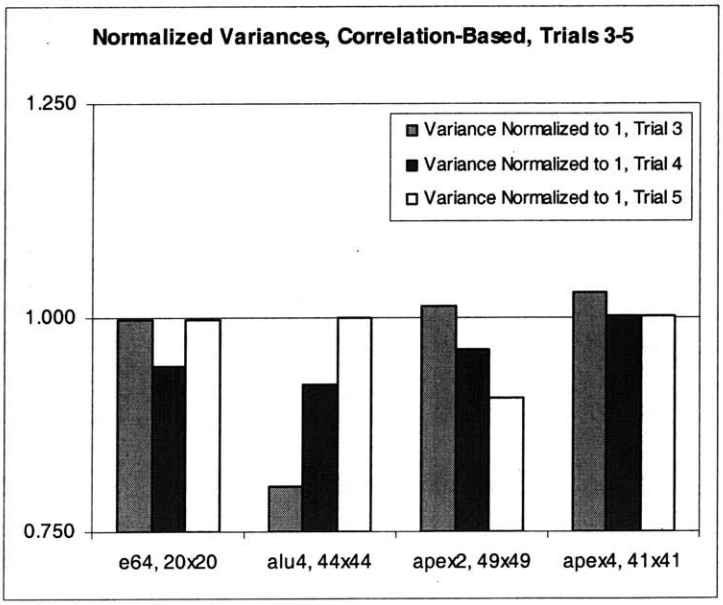


Figure 28: Graph of Normalized $3\sigma_{crit}$ values observed for Trials 3-5

From these results, it is difficult to spot any trends or dependencies on the specific spatial correlation function used. Fortunately, no increases in variability greater than 5% were observed for the 4 test circuits under consideration here. For Trial 5 ($\rho_B = 0.75$, $d_L =$

3×chiplength / 4) little changes were observed in the delay variabilities, as expected. An increase in correlation baseline implies that the WID component of variation contribution to the total variation has decreased. However, VA-VPR can only optimize designs for variation by considering the WID component. Therefore, as the WID component decreases, improvements in variability reduction resulting from VA-VPR decrease as well, resulting in behavior similar to the original VPR; this is verified by the graph above.

4.2 Block Binning VA Results

Table 6 below shows the results from Trial 1 produced by VA-VPR using the block binning methodology for all 9 benchmark circuits.

e64, 20x20	Δ_{crit}	$3\sigma_{crit}$			N_1	N_2	N_3
Trial 1	3.73E-08	5.59E-10	8.34%	0.22%	216	58	0
alu4, 44x44							
Trial 1	9.11E-08	9.48E-10	15.21%	-1.60%	1080	442	0
ex5p, 38x38							
Trial 1	8.00E-08	8.48E-10	5.13%	10.81%	816	248	0
apex2, 49x49							
Trial 1	1.06E-07	8.92E-10	9.36%	-13.07%	1357	521	0
apex4, 41x41							
Trial 1	9.37E-08	7.38E-10	21.55%	-2.81%	913	349	0
Misex3, 43x43							
Trial 1	7.90E-08	7.61E-10	10.96%	-9.16%	1025	372	0
seq, 48x48							
Trial 1	9.02E-08	7.92E-10	8.66%	0.89%	1312	438	0
diffeq, 44x44							
Trial 1	7.27E-08	1.86E-09	-4.34%	14.14%	1080	417	0
tseng, 37x37							
Trial 1	6.47E-08	1.64E-09	0.04%	-0.42%	769	278	0

Table 6: Placement results for Trial 1 for arrays with 90% utilization

The critical path delay and delay variability for Trial 1 normalized by Trial 0 results are shown in Figure 29 below.

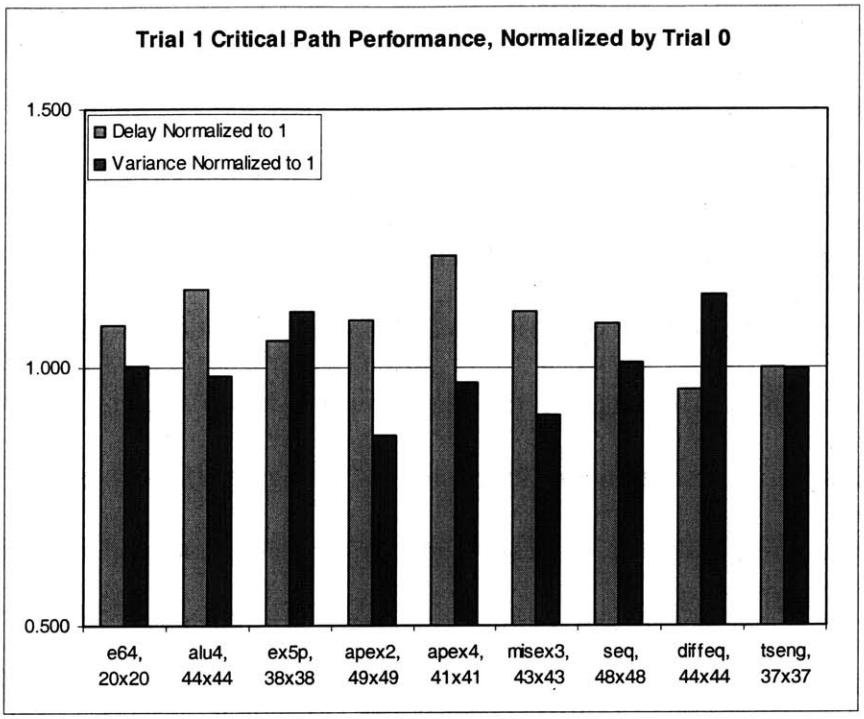


Figure 29: Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 1 for arrays with 90% utilization

The predicted delays for block-binning VA-VPR increase significantly from the delays predicted by VPR. Although there may be reductions in critical path delay variability, as is seen for circuits *apex2* and *misex3*, the overall goal of block binning-based VA-VPR is not to reduce critical path delay variability, but instead to place logic blocks in regions that are on the average faster than other regions of the FPGA. This translates to increasing the number of blocks placed in the faster bins (bin N_1 for our binning scheme) and reducing the number of blocks that are placed in slower bins (bin N_3). Our block-binning based VA-VPR does exactly this, as the percentages in Table 6

indicate that for all 9 circuits tested, there was a 100% reduction in the number of blocks placed in the bin N_3.

One advantage of the block binning-based placer is that the trend in the variation cost function does not increase as occurs for the correlation-based placer. The cost function trends, shown below in Figure 30, indicate that the variation cost function decreases as temperature decreases. This behavior implies no competition between the minimization of the bounding box cost function and the variation-aware cost function. However, the main disadvantage of the block binning-based placer, as discussed in section 3.2.1, is that the variation cost function trend in Figure 30 shows sharp transitions at initial temperatures. Fortunately, these sharp transitions did not raise any convergence issues due to the smoothness of the performance profile used during testing.

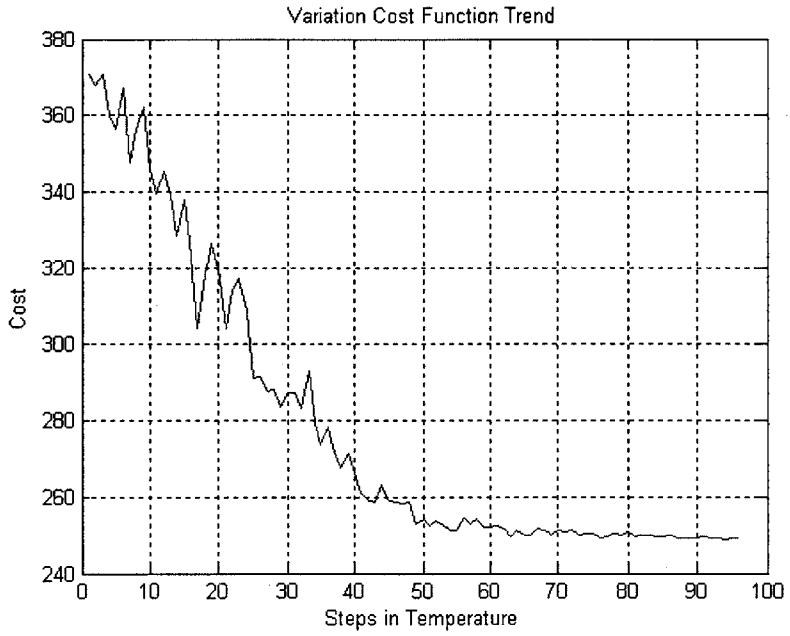


Figure 30: Trend in Variation Cost Function for Trial 1, circuit *e64*, array with 90% utilization

The results of the block-binning based VA-VPR placer, most notably the number of blocks placed in slower performance bins, are even more pronounced when targeting

arrays with lower device utilization rates. Table 7 includes the results from testing the block binning-based VA-VPR placer on each benchmark circuit targeting devices with 40% utilization.

e64, 32x32	Δ_{crit}	$3\sigma_{crit}$			N_1	N_2	N_3
Trial 1	4.56E-08	5.84E-10	3.95%	-0.80%	274	0	0
alu4, 73x73							
Trial 1	1.02E-07	9.89E-10	11.99%	2.47%	1522	0	0
ex5p, 62x62							
Trial 1	1.07E-07	9.69E-10	22.03%	-1.52%	1064	0	0
apex2, 80x80							
Trial 1	1.19E-07	1.06E-09	12.69%	-3.50%	1878	0	0
apex4, 67x67							
Trial 1	1.08E-07	7.94E-10	23.59%	-2.18%	1262	0	0
misex3, 70x70							
Trial 1	1.05E-07	9.89E-10	28.85%	17.64%	1397	0	0
seq, 78x78							
Trial 1	1.05E-07	7.98E-10	14.34%	-4.82%	1750	0	0
diffeq, 72x72							
Trial 1	8.07E-08	1.66E-09	10.00%	-11.93%	1497	0	0
tseng, 61x61							
Trial 1	6.53E-08	1.72E-09	-1.73%	0.48%	1047	0	0

Table 7: Placement results for Trial 1 for arrays with 40% utilization

As the array size increases, the estimated critical path delay reported by the block binning-based VA-VPR placer becomes much worse than that predicted by VPR. This problem is due in large part to the fact that the bulk of the circuit logic blocks are placed in the center of the chip away from the periphery IOBs, as shown in Figure 33. As the bulk of the blocks moves away from the periphery IOBs, the propagation delay from any input IOB to the logic blocks increases, as does the delay from the logic blocks to any output IOB. This increase in delay results in an increased estimate of the critical path delay predicted by the block binning-based placement. This problem of over-estimating the critical path delay can be fixed by accounting for the smaller LUT delays at the center

of the chip, as indicated by the performance profile in Figure 22. More accurate critical path delays can be obtained by updating the timing graph used by VPR during timing analysis and path delay prediction. Currently, VPR's timing graph specifies the same LUT delay for each LUT in the chip. If we can update the graph so that each LUT delay is dependent on position within the chip, then the estimated critical path delay predicted during block binning placement would decrease relative to the delay predicted by non-variation aware VPR. More information on improving the timing graph is discussed in section 5.2.

From the results shown in Table 7, we see no trend in the estimated variance of the critical path delay distribution. The estimated variability predicted by block binning VA-VPR falls within ±5% of the variability predicted by VPR, with the exception circuits being *diffeq* and *misex3*. These results are summarized graphically in Figure 31.

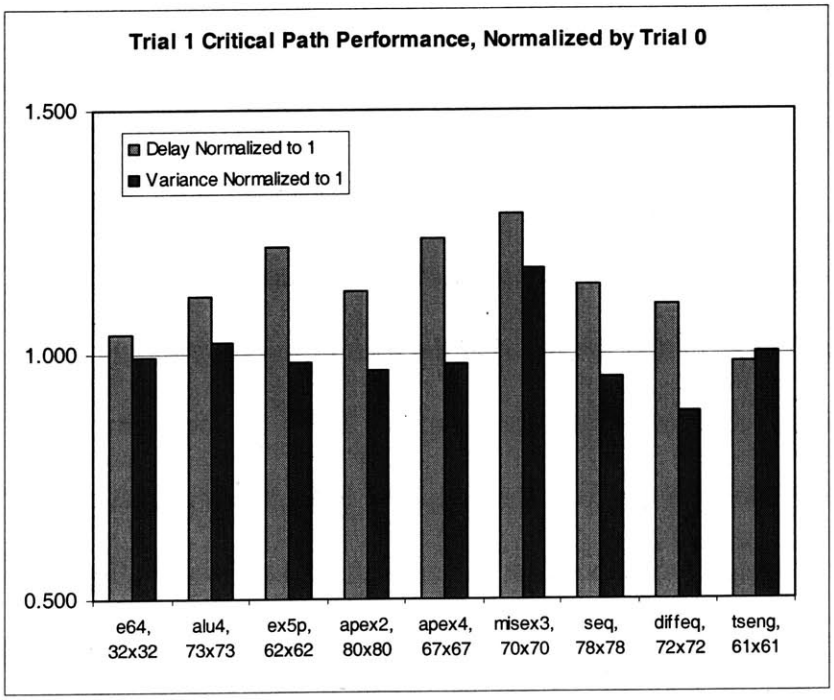


Figure 31: Graph of Normalized Δ_{crit} and $3\sigma_{crit}$ observed for Trial 1 for arrays with 40% utilization

From Table 7, we observe that block-binning VA-VPR performs much better when targeting devices with lower utilization rates. When compared to Trial 0 results, VA-VPR increased the number of blocks in N_1 by more than 20% for most circuits, while decreasing the number of blocks placed in N_2 and N_3 by 100%. These improvements can be explained by the fact that as the logic array size increases, there are more locations available for placement within the array that exhibit better performance characteristics on average.

Images of final placements produced for Trials 0, 1, and 3 are shown in Figures 32-33. These images show that the block binning placer attempts to place logic blocks in regions of the FPGA that have faster performance characteristics, as is indicated by the performance profile shown in Figure 22. By comparing the block binning placement image with the correlation-based placement image, we see that the correlation-based placer is not concerned with placing blocks in faster performance regions of the FPGA. The goal of that placer is instead to reduce circuit path delay variability for all paths.

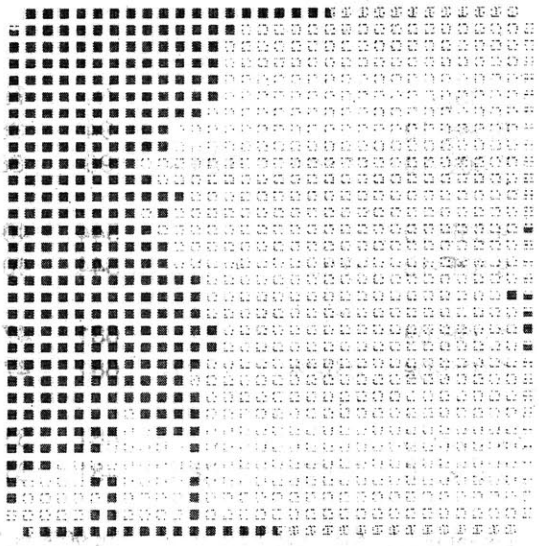
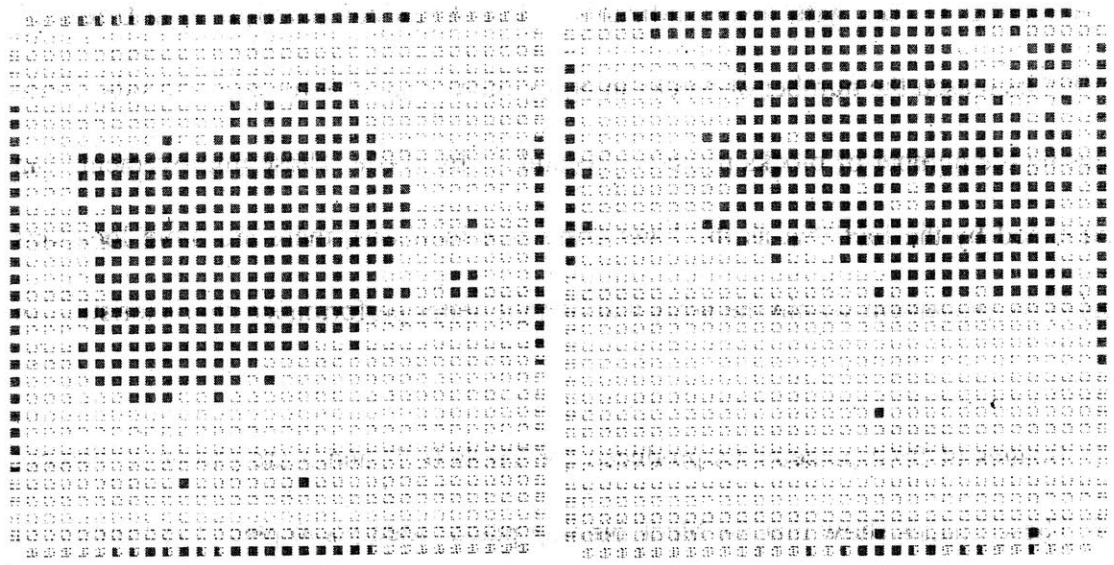


Figure 32: Final placement image, Trial 0 (VPR without variation-aware cost function)



(a) (b)

Figure 33: Final placement images, Trial 1 (a) and Trial 3 (b)

To show how the block binning method is affected by changes to the binning scheme, we have included the results from placements produced during Trial 2, corresponding to the 5-bin scheme shown in Figure 19. In the table below, the Trial 0 results correspond with placements produced by not taking into account variation.

e64, 20x20	Δ_{crit}	$3\sigma_{crit}$			N_1	N_2	N_3	N_4	N_5
Trial 0	3.44E-08	5.58E-10			94	85	94	1	0
Trial 2	3.38E-08	5.60E-10	-1.74%	0.33%	96	88	90	0	0
alu4, 44x44									
Trial 0	7.91E-08	9.63E-10			544	404	531	41	2
Trial 2	7.86E-08	9.52E-10	-0.65%	-1.11%	544	404	569	5	0
apex2, 49x49									
Trial 0	9.65E-08	1.03E-09			681	476	689	32	0
Trial 2	9.71E-08	1.01E-09	0.62%	-1.22%	681	476	721	0	0
apex4, 41x41									
Trial 0	7.71E-08	7.59E-10			481	312	465	3	1
Trial 2	7.88E-08	7.67E-10	2.20%	1.02%	481	312	469	0	0

Table 8. Placement results for Trial 2 for arrays with 90% utilization

During Trial 2 placement, most of the work went into moving blocks from bins N_3 and N_4 to higher performance bins N_2 and N_1. Although we see improvements in terms of number of blocks placed in higher performance bins, block binning with 5 bins still provides no improvement in critical path performance parameters. The variation-aware cost function trend is similar to that corresponding to the 3-bin scheme. These results indicate that switching to a scheme that uses a larger number of bins provides little gain in terms of reducing path delay variability. Although we can rigorously check this by testing many more block-binning schemes, the complexity in implementation increases significantly with an increase in the number of bins used.

Chapter 5

Conclusion

This section summarizes the results from testing our variation-aware placement tool. Many more steps need to be taken in the future if we are to establish a complete design tool flow that optimizes circuit implementation at each step by minimizing the impact of process variation on circuit performance. Before we go over these future steps, it is important that we summarize the work done thus far in developing and testing an FPGA placement tool that is variation-aware.

5.1 Summary

Our work focused on modifying the current functionality of the FPGA placement tool VPR to incorporate a model for process variation. We incorporated two variation-aware cost function implementations, a correlation-based function and a function based on block binning, and left the user with the choice over which cost function was to be used.

To summarize the results for the correlation-based variation-aware placer, we computed the average percent changes in both critical path delay and delay variability over all test circuits. For the target FPGAs with 90% utilization, we observed an average of a 0.44% increase in critical path delay and a 2.82% reduction in critical path delay variability. For the larger target FPGAs with 40% utilization, we observed a 4.56% increase in critical path delay and a 4.44% reduction in the delay variability. These results indicate that if we are willing to give up a small increase in critical path delay, then we can achieve a greater reduction in critical path delay variability by performing placement with the correlation based variation-aware VPR. We also obtained a reduction in the number of logic blocks placed in the slower performance bins for each test circuit and target size.

The results for the block binning variation-aware placer indicated poorer performance regarding the critical path metrics due to the over-estimation of critical path delay mentioned in section 4.2. We observed an average of 0.11% reduction in critical path delay variability and an 8.32% increase in critical path delay when targeting the smaller sized FPGA devices. For the larger sizes, we observed an average of 0.47% reduction in variability and a 13.97% increase in critical path delay. Despite the poor results predicted for the critical path metrics, the block binning placer did achieve a 100% reduction in the number of blocks placed in poor performance bins for both small and large-sized FPGAs. This indicated that the block binning placer achieved its goal of maximizing the number of logic blocks placed in regions of the FPGA that, on the average, show the best performance characteristics.

5.2 Future Work and Extensions

There are several ways we can update VPR to account for models of process variation and produce placement solutions that have been fully optimized not only for wire-length and delay but also for other metrics indicative of circuit performance reliability. For instance, although thus far we have only discussed two methods for incorporating a variation-aware cost function into VPR, there are many additional, practical ways by which this can be achieved.

If in the future we wish to incorporate a different type of variation-aware cost function, then only one sub-routine need be updated in the source code for VA-VPR, making VA-VPR easily extensible. This routine computes the extra cost contributed to the total variation-aware cost by one source-sink connection in the circuit, or equivalently by one net within the circuit netlist. For instance, for the case of the spatial correlation-based placer, this sub-routine computed the correlation coefficient between the given source and sink block of the connection. This leads to the idea of researching various variation-aware cost functions in pursuit of one that achieves the best performance in terms of minimal nominal path delays and delay variabilities.

We can further extend our VA-VPR to include information on the variability in not only LUT delay but also in terms of other device parameters that can be measured at any specific site within the FPGA. For instance, through various variation characterization procedures, we can collect systematic WID variation profiles for interconnect parameters such as delay per unit length for each wire segment in the FPGA architecture, or even for some other performance parameters such as leakage power

dissipation. For each of these profiles, we can synthesize an overall performance profile that can be used for block binning or for any other variation-aware cost function dependent on some performance profile.

Another modification, discussed in section 4.2, is to update the generation of VPR's timing graph used during timing analysis. The graph is a directed, weighted-edge graph whose nodes are components of the circuit (pin, block, etc.) and whose edges indicate delays between the connected components. The timing graph in VPR is computed once during the initialization of the placer operation. Because WID variation causes the delay between two components to depend on the relative position of those two components within the FPGA, we can provide more accurate predicted path delays by extending the timing graph to be dependent on position. The graph can be continuously updated during placement so that the location assigned to each node in the graph is synchronized with the location of the physical component corresponding to that node. Such a modification would provide more accurate critical path results, especially when performing variation-aware placement by block binning.

Another way we can modify VPR to account for variation is by moving our attention away from the placer and instead focusing on both place and route operations. Routing chooses the best path to take from any source pin to any sink pin given a certain placement. This is currently done by an algorithm that attempts to minimize the delay between the two pins while ensuring that there is no congestion or overuse of routing resources. One idea for future work may be to ensure that the path taken from the source pin runs through a region of high spatial correlation, ensuring that circuit path metrics are computed with some level of reliability or confidence. Ideally, we would like the place-

and-route tool to predict an upper-bound for the path delay for any given path at some specified confidence level. The work done so far on adding a spatial correlation model to the placer is a first step in this direction.

Another area of research to pursue in the future concerns the selection of the weights for the three cost functions, an issue briefly mentioned in section 4.1. Although on average our correlation-based VA-VPR was able to reduce critical path delay variability by 2.82% in FPGAs with 90% utilization, the actual factor by which the variability was reduced varied from circuit to circuit. We hypothesize that the weight settings that results in circuit placements with the greatest reduction in critical path delay variability are dependent on several key properties of the circuit being placed. One suggestion for future research is to propose a framework for determining, either experimentally or theoretically, the best weight settings for each individual circuit to be placed. This framework can be similar to a response surface modeling methodology where the response variables are the predicted critical path nominal delay and delay variability and the controllable factors are the cost function weights. Another suggestion for future work is to research how key properties of each placed circuit determine the responsiveness of the circuit to path delay variability reduction via correlation-based placement. For example, the average logic depth in the circuit to be placed may be a property that determines to what degree the path delay variability can be reduced by correlation-based placement. The goal of this research is to understand, at a fundamental level, how such properties of circuits determine the overall circuit behavior after correlation-based variation aware placement. This knowledge would help in determining

the best cost function weight settings to be used for each individual circuit to obtain the greatest path delay variability reduction.

5.3 Applications

The work done in this project is applicable for any general place-and-route tool targeting FPGA devices. As the device sizes in current and future generation technologies continue to scale downwards, the impact of process variation on circuit design and implementation becomes much more important. Not only is it important to perform thorough variation characterization of any FPGA product released, but it should also be just as important to release CAD toolsets that have models of variation built in and that can fully optimize designs so that circuit performance is reliably predicted. The modifications to VPR discussed above are just the first steps towards incorporating variation awareness into a general CAD flow for FPGA devices. If we can make every step aware of the presence of process variations, then we will be more capable of predicting the final circuit implementation's performance.

Appendix A: Abbreviations and Symbols

VPR – Versatile Place and Route tool
VA-VPR – Variation-aware Versatile Place and Route tool
D2D – Die-toDie variation component
WID – Within-die variation component
ρ_B – Correlation baseline
d_L – Correlation length
Δ – Path delay
Σ – Path variance
$f_{BB}(P)$ – Bounding-box (Wirelength-driven) Cost Function
$f_{TD}(P)$ – Timing-driven Cost Function
$f_{VAR}(P)$ – Variation-aware Cost Function
$f(P)$ – Total Cost Function
α – Wirelength-driven Cost Function Weight
β – Timing-driven Cost Function Weight
γ – Variation-aware Cost Function Weight
LUT – Lookup Table
FF – Flip-Flop
CLB – Configurable Logic Block
IOB – Input/Output Block
FPGA – Field Programmable Gate Array
ASIC – Application-Specific Integrated Circuit
MCNC - Microelectronics Center of North Carolina

Appendix B: VA-VPR C Code

Lines added to `globals.h`:

```
/* Global parameter declarations required to     *
 * run VA-VPR                                    */

extern float corr_length, corr_baseline, local_var;
// run va-vpr flag
extern boolean correct_var;
// indicate variation-aware type
extern int va_type;
// array storing performance profile
extern t_var_dist **var_params;
```

Lines modified/added to `vpr_types.h`:

```
/* Stores the performance offset due to systematic variation    *
 * and the performance profile value (LUT delay at site (x,y)   */

typedef struct {float sys_var_offset; float lut_del; } t_var_dist;

struct s_placer_opts {
       enum e_place_algorithm place_algorithm;
       float timing_tradeoff;       //used if not VA
       float tradeoff_bb;     //wirelength cost function weight
       float tradeoff_td;     //timing-driven cost function weight
       float tradeoff_var;    //variation-aware cost function weight
       int block_dist;
       enum place_c_types place_cost_type; float place_cost_exp;
       int place_chan_width; enum e_pad_loc_type pad_loc_type;
       char *pad_loc_file; enum pfreq place_freq; int num_regions;
       int recompute_crit_iter;
       boolean enable_timing_computations;
       int inner_loop_recompute_divider;
       float td_place_exp_first;
       float td_place_exp_last;};
```

Lines modified/added to `main.c`:

```
#include "read_var.h"

. . .

            /* Physical architecture stuff */

float corr_length, corr_baseline;
float local_var;
boolean correct_var;
int va_type;
/* var_params[0..nx-1][0..ny-1].
   Performance profile used for
   block binning. */
t_var_dist **var_params;
```

```
. . .

static void parse_command (int argc, char *argv[], char *net_file,
char *arch_file, char *var_file, char *place_file, char *route_file,
char *results_file, enum e_operation *operation, float *aspect_ratio,
boolean *full_stats, boolean *user_sized, boolean *verify_binary_search,
int *gr_automode, boolean *show_graphics, struct s_annealing_sched
*annealing_sched, struct s_placer_opts *placer_opts, struct
s_router_opts *router_opts, boolean *timing_analysis_enabled, float
*constant_net_delay) {

. . .

  corr_length = 0;
  corr_baseline = 0;
  local_var = 0.;
  va_type = 0;

  correct_var = FALSE;

. . .

  placer_opts->tradeoff_bb = 0.3333333;
  placer_opts->tradeoff_td = 0.3333333;
  placer_opts->tradeoff_var = 0.3333333;

. . .

while (i < argc) {

. . .

    if (strcmp(argv[i],"-correct_var") == 0) {
      correct_var = TRUE;
      i++;
      continue;
    }

    if (strcmp(argv[i],"-tt_bb") == 0) {
      placer_opts->tradeoff_bb = read_float_option (argc, argv, i);

      if (placer_opts->tradeoff_bb < 0 || placer_opts->tradeoff_bb > 1){
      printf("Error:  -tt_bb value must be between 0 and 1.\n");
      exit(1);
      }
      i += 2;
      continue;
    }

    if (strcmp(argv[i],"-tt_td") == 0) {
      placer_opts->tradeoff_td = read_float_option (argc, argv, i);

      if (placer_opts->tradeoff_td < 0 || placer_opts->tradeoff_td > 1){
      printf("Error:  -tt_td value must be between 0 and 1.\n");
      exit(1);
      }
      i += 2;
      continue;
    }

    if (strcmp(argv[i],"-tt_var") == 0) {
      placer_opts->tradeoff_var = read_float_option (argc, argv, i);

      if (placer_opts->tradeoff_var <0||placer_opts->tradeoff_var >1) {
```

```
      printf("Error:  -tt_var value must be between 0 and 1.\n");
      exit(1);
      }
      i += 2;
      continue;
    }

    if (strcmp(argv[i],"-corr_length") == 0) {
      corr_length = read_int_option (argc, argv, i);

      if (corr_length <= 0) {
      printf("Error:  -corr_length value must be greater than 0.\n");
      exit(1);
      }
      i += 2;
      continue;
    }

    if (strcmp(argv[i],"-corr_baseline") == 0) {
      corr_baseline = read_float_option (argc, argv, i);

      if (corr_baseline <= 0 || corr_baseline >=1) {
      printf("Error:  -corr_baseline value must be between 0 and 1.\n");
      exit(1);
      }
      i += 2;
      continue;
    }

    if (strcmp(argv[i],"-local_var") == 0) {
      local_var = read_float_option (argc, argv, i);

      if (local_var <= 0) {
      printf("Error:  -local_var value must be positive.\n");
      exit(1);
      }
      i += 2;
      continue;
    }

    if (strcmp(argv[i],"-va_type") == 0) {
      va_type = read_int_option (argc, argv, i);

      if (va_type != 0 && va_type != 1) {
      printf("Error:  -va_type value must be 0 or 1.\n");
      exit(1);
      }
      i += 2;
      continue;
    }
. . .

static void get_input (char *net_file, char *arch_file, char *var_file,
      int place_cost_type, int num_regions, float aspect_ratio, boolean
      user_sized, enum e_route_type route_type, struct
      s_det_routing_arch *det_routing_arch, t_segment_inf
      **segment_inf_ptr, t_timing_inf *timing_inf_ptr, t_subblock_data
      *subblock_data_ptr, t_chan_width_dist *chan_width_dist_ptr) {


. . .
read_var (var_file);
. . .
}
```

Lines modified/added to `read_var.c`:

```
#include <string.h>
#include <stdio.h>
#include <math.h>
#include "util.h"
#include "vpr_types.h"
#include "globals.h"
#include "read_var.h"

void read_var (char *var_file) {

 /* Reads in the performance profile */

 int i,j,n;
 FILE *fp_wid;
 float tmp_lut_var, tmp_sys_offset;
 int x, y;


 fp_wid = my_fopen (var_file, "r", 0);

 var_params = (t_var_dist **) alloc_matrix (0, nx-1+io_rat, 0, ny-
           1+io_rat, sizeof (t_var_dist));

 for (i = 0; i < nx+io_rat; i++)
   for (j = 0; j < ny+io_rat; j++) {
       var_params[i][j].sys_var_offset = 0.;
       var_params[i][j].lut_var = 0.;
   }

  while ((n=fscanf (fp_wid, "%d,%d,%f,%f", &x, &y, &tmp_sys_offset,
            &tmp_lut_var)) != EOF) {
    var_params[x][y].lut_var = tmp_lut_var;
    var_params[x][y].sys_var_offset = tmp_sys_offset;
  }

  fclose(fp_wid);

}
```

Lines modified/added to `place.c`:

```
/* Variables local to place.c   */

. . .

/* Assume 3 bins used for block-binning */
static int blocks_per_bin[3]; // Stores number of bins in each of 3 bins
static float max_sysvar_val, min_sysvar_val;    // Used for binning
static float bin_cost_table[3][3];           // block binning cost table

. . .

/* [0..num_nets-1][1..num_pins-1]. Stores the value of the variation */
/* cost for each source-sink connection in the circuit */
static float **point_to_point_var_cost = NULL;
static float **temp_point_to_point_var_cost = NULL;
```

```
. . .

/* Static subroutines local to place.c  */

. . .
// Compute total variation-aware cost function
static float comp_var_cost (void);

// Swap blocks
static int try_swap (float t, float *cost, float *bb_cost, float
      *timing_cost, float *var_cost, float rlim, int *pins_on_block,
      int place_cost_type, float **old_region_occ_x,
      float **old_region_occ_y, int num_regions, boolean fixed_pins,
      enum e_place_algorithm place_algorithm, float timing_tradeoff,
      float tradeoff_bb, float tradeoff_td, float tradeoff_var,
      float inverse_prev_bb_cost, float inverse_prev_timing_cost,
      float inverse_prev_var_cost, float *delay_cost);

// Check final placement
static void check_place (float bb_cost, float timing_cost, float
      var_cost, int place_cost_type,
      int num_regions, enum e_place_algorithm place_algorithm,
      float delay_cost);

// Determine number of blocks per bin
static void compute_blocks_per_bin(void);

static void compute_maxmin_sysvar(void);
// Compute correlation-based VA cost for 1 source-sink connection
static float comp_point_to_point_scvar (int source_x, int source_y, int
      sink_x, int sink_y);
// Compute block binning VA cost for 1 source-sink connection
static float comp_point_to_point_bbvar (int source_x, int source_y, int
      sink_x, int sink_y);
// Determine bin number for given performance profile value
static int get_bin(float var_value);
// Set cost table entries
static void set_cost_table(void);
// Compute correlation between two sites
static float comp_corr_coef(int x_1, int y_1, int x_2, int y_2);
// Update VA cost function after swap
static void update_var_cost(int b_from, int b_to, int num_of_pins);
// Compute change in VA cost function
static void comp_delta_var_cost(float *var_delta, int old_x, int old_y,
                        int new_x, int new_y, int b_from, int b_to,
                        int num_of_pins);
// Estimate critical path delay variability
static float estimate_cp_var(void);

. . .



/*****************************************************/

void try_place (struct s_placer_opts placer_opts, struct
            s_annealing_sched annealing_sched, t_chan_width_dist
            chan_width_dist, struct s_router_opts router_opts,
            struct s_det_routing_arch det_routing_arch,
            t_segment_inf *segment_inf,
            t_timing_inf timing_inf,
            t_subblock_data *subblock_data_ptr, char *results_file) {

. . .
```

```
if (placer_opts.place_algorithm == NET_TIMING_DRIVEN_PLACE ||
     placer_opts.place_algorithm == PATH_TIMING_DRIVEN_PLACE) {
. . .
   // Used for block binning VA cost function
   compute_maxmin_sysvar();
   set_cost_table();

if (correct_var) {
     var_cost = comp_var_cost();
     inverse_prev_var_cost = 1/var_cost;
   }
   else {
     var_cost = 0.;
     inverse_prev_var_cost = 0.;
   }
. . .
}
. . .

while (exit_crit(t, cost, annealing_sched) == 0) {

   if (placer_opts.place_algorithm == NET_TIMING_DRIVEN_PLACE ||
       placer_opts.place_algorithm == PATH_TIMING_DRIVEN_PLACE) {
     cost = 1;
   }
    av_cost = 0.;
    av_bb_cost = 0.;
    av_delay_cost = 0.;
    av_timing_cost = 0.;
    av_var_cost = 0.;
    sum_of_squares = 0.;
    success_sum = 0;

. . .

  for (inner_iter=0; inner_iter < move_lim; inner_iter++) {
      if (try_swap(t, &cost, &bb_cost, &timing_cost, &var_cost,
           rlim, pins_on_block, placer_opts.place_cost_type,
           old_region_occ_x, old_region_occ_y, placer_opts.num_regions,
           fixed_pins, placer_opts.place_algorithm,
           placer_opts.timing_tradeoff, placer_opts.tradeoff_bb,
           placer_opts.tradeoff_td,
           placer_opts.tradeoff_var, inverse_prev_bb_cost,
           inverse_prev_timing_cost, inverse_prev_var_cost, &delay_cost)
            == 1) {

      . . .
      }
      . . .
    }
}

 for (inner_iter=0; inner_iter < move_lim; inner_iter++) {
   if (try_swap(t, &cost, &bb_cost, &timing_cost, &var_cost,
        rlim, pins_on_block, placer_opts.place_cost_type,
        old_region_occ_x, old_region_occ_y, placer_opts.num_regions,
        fixed_pins, placer_opts.place_algorithm,
        placer_opts.timing_tradeoff,
        placer_opts.tradeoff_bb, placer_opts.tradeoff_td,
        placer_opts.tradeoff_var,
        inverse_prev_bb_cost, inverse_prev_timing_cost,
        inverse_prev_var_cost, &delay_cost) == 1) {
```

```
        . . .

        }
    }

check_place(bb_cost, timing_cost, var_cost, placer_opts.place_cost_type,
            placer_opts.num_regions,
            placer_opts.place_algorithm, delay_cost);

if (placer_opts.place_algorithm == NET_TIMING_DRIVEN_PLACE ||
     placer_opts.place_algorithm == PATH_TIMING_DRIVEN_PLACE ||
     placer_opts.enable_timing_computations) {

       . . .
       net_delay = point_to_point_delay_cost;
       load_timing_graph_net_delays(net_delay);
       est_crit = load_net_slack(net_slack, 0);
       est_crit_var = estimate_cp_var();
       . . .
}

 /*****************************************************/

static int try_swap (float t, float *cost, float *bb_cost, float
       *timing_cost, float *var_cost, float rlim, int *pins_on_block,
       int place_cost_type, float **old_region_occ_x,
       float **old_region_occ_y, int num_regions, boolean fixed_pins,
       enum e_place_algorithm place_algorithm, float timing_tradeoff,
       float tradeoff_bb, float tradeoff_td, float tradeoff_var,
       float inverse_prev_bb_cost, float inverse_prev_timing_cost,
       float inverse_prev_var_cost, float *delay_cost) {


. . .
 delta_c = 0;                          /* Change in costs due to this swap. */
 bb_delta_c = 0;
 timing_delta_c = 0;
 var_delta_c = 0;
. . .
if (place_algorithm == NET_TIMING_DRIVEN_PLACE ||
     place_algorithm == PATH_TIMING_DRIVEN_PLACE) {

       /* Compute change in timing-driven cost after swap */
       comp_delta_td_cost(b_from, b_to, num_of_pins, &timing_delta_c,
                  &delay_delta_c);

       /* Compute change in variation-aware cost after swap */
   if (correct_var) {
     comp_delta_var_cost(&var_delta_c, x_from, y_from, x_to, y_to,
            b_from, b_to, num_of_pins);

     delta_c = (bb_delta_c * inverse_prev_bb_cost)*tradeoff_bb +
       (timing_delta_c * inverse_prev_timing_cost)*tradeoff_td +
       (var_delta_c * inverse_prev_var_cost)*tradeoff_var;
   }
   else delta_c = (1-timing_tradeoff)*bb_delta_c*inverse_prev_bb_cost +
            timing_tradeoff*timing_delta_c*inverse_prev_timing_cost;
 }
 else delta_c = bb_delta_c;

keep_switch = assess_swap (delta_c, t);

if (keep_switch) {
    *cost = *cost + delta_c;
    *bb_cost = *bb_cost + bb_delta_c;
```

```
    if (place_algorithm == NET_TIMING_DRIVEN_PLACE ||
      place_algorithm == PATH_TIMING_DRIVEN_PLACE) {
      /*update the point_to_point_timing_cost and
      point_to_point_var_cost values from the temporary values*/
      *timing_cost = *timing_cost + timing_delta_c;
      *delay_cost = *delay_cost + delay_delta_c;
      update_td_cost(b_from, b_to, num_of_pins);

      if (correct_var){
      *var_cost = *var_cost + var_delta_c;
      update_var_cost(b_from, b_to, num_of_pins);
      }
    }
}
else {    /* Move was rejected.  */  }

return(keep_switch);

}

/*************************************************/

static float estimate_cp_var() {

   t_linked_int *critical_path_head, *critical_path_node;
   int *x_positions, *y_positions;
   int num_drivers, inode, iblk, inet;
   t_tnode_type type;
   int i,j;
   float corr_coef, cp_var;

   critical_path_head = allocate_and_load_critical_path ();
   critical_path_node = critical_path_head;

   num_drivers = 0;

   while (critical_path_node != NULL) {
     inode = critical_path_node->data;
     type = tnode_descript[inode].type;
     if (type == INPAD_OPIN || type == CLB_OPIN)
       num_drivers++;
     critical_path_node = critical_path_node->next;
   }

  critical_path_node = critical_path_head;


   x_positions = (int *) my_malloc (num_drivers * sizeof (int));
   y_positions = (int *) my_malloc (num_drivers * sizeof (int));


   i=0;
   while (critical_path_node != NULL) {
     inode = critical_path_node->data;
     type = tnode_descript[inode].type;

     if (type == INPAD_OPIN || type == CLB_OPIN) {
       get_tnode_block_and_output_net (inode, &iblk, &inet);
       x_positions[i] = block[iblk].x;
       y_positions[i] = block[iblk].y;
       i++;
       }
```

```
    critical_path_node = critical_path_node->next;
  }


  free_int_list (&critical_path_head);

  cp_var = 0.;

  for (i=1; i<num_drivers; i++)
    for (j=i+1; j<=num_drivers; j++) {
      corr_coef = comp_corr_coef(x_positions[i], y_positions[i],
                  x_positions[j], y_positions[j]);
      cp_var += 2 * corr_coef * local_var;
    }

  cp_var += num_drivers * local_var;

  cp_var = sqrt(cp_var)*3;
  return(cp_var);

 }
/*************************************************************/

static float comp_point_to_point_scvar (int source_x, int source_y, int
            sink_x, int sink_y) {

  /* returns the variation-aware cost function evaluated for
     one point to point connection using
     the spatial correlation model */

  float var_cost_pt2pt;


  var_cost_pt2pt = comp_corr_coef(source_x, source_y, sink_x, sink_y);

  if (var_cost_pt2pt < 0) {
    printf("Error in comp_point_to_point_scvar in place.c, bad corr_coef
            value\n");
    exit(1);
  }

  return(var_cost_pt2pt);

}
/*************************************************************/
static float comp_point_to_point_bbvar (int source_x, int source_y, int
sink_x, int sink_y) {

  /* returns the variation-aware cost function evaluated for
     one point to point connection using
     the block binning */

  float var_cost_pt2pt;
  float source_val, sink_val;
  int source_bin, sink_bin;

  source_val = var_params[source_x][source_y].sys_var_offset;
  sink_val = var_params[sink_x][sink_y].sys_var_offset;

  /* check source and sink bins */
  source_bin = get_bin(source_val);
  sink_bin = get_bin(sink_val);
```

```
  var_cost_pt2pt = bin_cost_table[source_bin-1][sink_bin-1];

  return(var_cost_pt2pt);

}
/*************************************************************/

static float comp_corr_coef(int x_1, int y_1, int x_2, int y_2) {

  /* compute the correlation coefficient between two sites
     located at given coordinates */

  float d_x, d_y, dist;
  float rho;

  d_x = abs(x_2 - x_1);
  d_y = abs(y_2 - y_1);
  dist = sqrt(d_x * d_x + d_y * d_y);

  if (dist <= corr_length)
    rho = 1 - dist / corr_length * (1 - corr_baseline);
  else rho = corr_baseline;


  return(rho);

}

/*************************************************************/
static void update_var_cost(int b_from, int b_to, int num_of_pins) {

  /* update the point_to_point_var_cost values from the temporary *
   * values for all connections that have changed */

  int blkpin, net_pin, inet, ipin;

    for (blkpin=0; blkpin<num_of_pins; blkpin++) {

      inet = block[b_from].nets[blkpin];

      if (inet == OPEN)
      continue;

      if (is_global[inet])
      continue;

      net_pin = net_pin_index[b_from][blkpin];

      if (net_pin != 0) {

      /*the following "if" prevents the value from being updated twice*/
      if (net[inet].blocks[0] != b_to && net[inet].blocks[0] != b_from)
      {

        point_to_point_var_cost[inet][net_pin] =
          temp_point_to_point_var_cost[inet][net_pin];
        temp_point_to_point_var_cost[inet][net_pin] = -1;

      }
      }
      else { /*this net is being driven by a moved block, recompute */
      /*all point to point connections on this net.*/
      for (ipin=1; ipin<net[inet].num_pins; ipin++) {
```

```
          point_to_point_var_cost[inet][ipin] =
            temp_point_to_point_var_cost[inet][ipin];
          temp_point_to_point_var_cost[inet][ipin] = -1;

        }
        }
      }

      if (b_to != EMPTY) {
        for (blkpin=0; blkpin<num_of_pins; blkpin++) {

        inet = block[b_to].nets[blkpin];

        if (inet == OPEN)
          continue;

        if (is_global[inet])
          continue;

        net_pin = net_pin_index[b_to][blkpin];

        if (net_pin != 0) {

        /*the following "if" prevents the value from being updated 2x*/
        if (net[inet].blocks[0] != b_to && net[inet].blocks[0] != b_from)
{

            point_to_point_var_cost[inet][net_pin] =
              temp_point_to_point_var_cost[inet][net_pin];
            temp_point_to_point_var_cost[inet][net_pin] = -1;

          }
        }
        else { /*this net is being driven by a moved block, recompute */
          /*all point to point connections on this net.*/
          for (ipin=1; ipin<net[inet].num_pins; ipin++) {

              point_to_point_var_cost[inet][ipin] =
              temp_point_to_point_var_cost[inet][ipin];
              temp_point_to_point_var_cost[inet][ipin] = -1;

          }
        }
        }
      }
}
/*************************************************************/
static void comp_delta_var_cost(float *var_delta, int old_x, int old_y,
            int new_x, int new_y, int b_from, int b_to, int num_of_pins)
{

  /* computes the change in var_cost given the swap between blocks
     b_from and b_to */

  int j;
  int inet, ipin, iblk, net_pin;

  float delta_var_cost, temp_var;

  delta_var_cost = 0.;
```

```
for (j=0; j<num_of_pins; j++) { /* for each net connected to b_from */
  inet = block[b_from].nets[j];

  if (inet == OPEN) /* if not connected or if global, skip net */
    continue;

  if (is_global[inet])
    continue;

  net_pin = net_pin_index[b_from][j];  /* get net pin number for
                                          b_from pin */

  if (net_pin != 0) {    /* if not driver pin, only compute change in
                            var cost of one net branch */
    iblk = net[inet].blocks[0];
    if (iblk != b_to && iblk != b_from) {  /* if driver block of net
                                              left untouched in swap */
    if (va_type == 0)
      temp_var = comp_point_to_point_bbvar(block[iblk].x,
                        block[iblk].y, new_x, new_y);
    else temp_var = comp_point_to_point_scvar(block[iblk].x,
                        block[iblk].y, new_x, new_y);

    temp_point_to_point_var_cost[inet][net_pin] = temp_var;

    delta_var_cost += temp_point_to_point_var_cost[inet][net_pin] -
      point_to_point_var_cost[inet][net_pin];
    }
  }
  else {  /* if pin is driver, compute change in var cost of entire
             net */

     for (ipin=1; ipin<net[inet].num_pins; ipin++) {
     iblk=net[inet].blocks[ipin];
     if (va_type == 0)
       temp_var = comp_point_to_point_bbvar(new_x, new_y,
                        block[iblk].x, block[iblk].y);
     else temp_var = comp_point_to_point_scvar(new_x, new_y,
                        block[iblk].x, block[iblk].y);

     temp_point_to_point_var_cost[inet][ipin] = temp_var;

     delta_var_cost += temp_point_to_point_var_cost[inet][ipin] -
       point_to_point_var_cost[inet][ipin];
     }
  }

}

if (b_to != EMPTY) {    /* b_to contained a block, must account for
                           change in cost of b_to nets */

  for (j=0; j<num_of_pins; j++) {
    inet = block[b_to].nets[j];

    if (inet == OPEN) /* if not connected or if global, skip net */
    continue;

    if (is_global[inet])
    continue;

    net_pin = net_pin_index[b_to][j];    /* get net pin number for b_to
                                            pin */
```

```
      if (net_pin != 0) {     /* if not driver pin, only compute change
                                 in var cost of one net branch */
      iblk = net[inet].blocks[0];
      if (iblk != b_to && iblk != b_from) {  /* if driver block of net
                                                left untouched in swap */
        if (va_type == 0)
          temp_var = comp_point_to_point_bbvar(block[iblk].x,
                        block[iblk].y, old_x, old_y);
        else  temp_var = comp_point_to_point_scvar(block[iblk].x,
                        block[iblk].y, old_x, old_y);

        temp_point_to_point_var_cost[inet][net_pin] = temp_var;

        delta_var_cost += temp_point_to_point_var_cost[inet][net_pin] -
          point_to_point_var_cost[inet][net_pin];
      }
     }
     else {    /* if pin is driver, compute change in var cost of entire
                  net */

      for (ipin=1; ipin<net[inet].num_pins; ipin++) {
        iblk = net[inet].blocks[ipin];
        if (va_type == 0)
          temp_var = comp_point_to_point_bbvar(old_x, old_y,
                  block[iblk].x, block[iblk].y);
        else temp_var = comp_point_to_point_scvar(old_x, old_y,
                  block[iblk].x, block[iblk].y);

        temp_point_to_point_var_cost[inet][ipin] = temp_var;

        delta_var_cost += temp_point_to_point_var_cost[inet][ipin] -
          point_to_point_var_cost[inet][ipin];
      }
      }
    }
  }

  *var_delta = delta_var_cost;

}
/*****************************************************/
static float comp_var_cost () {

  /* computes the cost (from scratch) due to the variation along each
     point to point connection */

  int inet, ipin, source_blk, sink_blk;
  float loc_var_cost, temp_var;
  int source_x, source_y, sink_x, sink_y;

  loc_var_cost = 0.;

  for (inet =0; inet<num_nets; inet++) {    /* for each net ... */
    if (is_global[inet] == FALSE) {    /* Do only if not global. */
      source_blk = net[inet].blocks[0];
      source_x = block[source_blk].x;
      source_y = block[source_blk].y;

      for (ipin=1; ipin<net[inet].num_pins; ipin++) {

            sink_blk = net[inet].blocks[ipin];
            sink_x = block[sink_blk].x;
            sink_y = block[sink_blk].y;
            if (va_type == 0)
```

```
                    temp_var = comp_point_to_point_bbvar(source_x,
                                source_y, sink_x, sink_y);
                else temp_var = comp_point_to_point_scvar(source_x,
                                source_y, sink_x, sink_y);


                loc_var_cost += temp_var;
                point_to_point_var_cost[inet][ipin] = temp_var;


                temp_point_to_point_var_cost[inet][ipin] = -1;

        }
      }
    }

    return (loc_var_cost);
}
/*************************************************************/
static void compute_blocks_per_bin() {

    /* Compute number of blocks placed in each bin */
    int i;
    int block_x, block_y;
    float block_val;
    int count_1, count_2, count_3;
    int block_bin;

    count_1 =0; count_2=0; count_3=0;

    for (i=0; i<num_blocks; i++){
      if (block[i].type == CLB){
        block_x = block[i].x;
        block_y = block[i].y;
        block_val = var_params[block_x][block_y].sys_var_offset;

        /* check block bin */
        block_bin = get_bin(block_val);
        if (block_bin == 1)
              count_1++;
        else if (block_bin == 2)
              count_2++;
        else count_3++;
      }
    }

    blocks_per_bin[0] = count_1;
    blocks_per_bin[1] = count_2;
    blocks_per_bin[2] = count_3;
}
/*************************************************************/
static void compute_maxmin_sysvar() {

    /* Compute max and min values for the
        offset due to systematic variation */
    int i, j;
    float temp_max, temp_min;

    temp_max = 0;
    temp_min = 1;

    for (i=0; i<nx+io_rat; i++)
      for (j=0; j<ny+io_rat; j++){
        if (var_params[i][j].sys_var_offset > temp_max)
```

```
      temp_max = var_params[i][j].sys_var_offset;
      if (var_params[i][j].sys_var_offset < temp_min)
      temp_min = var_params[i][j].sys_var_offset;
    }

  max_sysvar_val = temp_max;
  min_sysvar_val = temp_min;

}
/*******************************************************/
static int get_bin(float var_value){


  float diff_sysvar_val, bin1_lb, bin2_lb;// bin3_lb, bin4_lb;

  diff_sysvar_val = max_sysvar_val - min_sysvar_val;
  bin1_lb = max_sysvar_val -  diff_sysvar_val * 0.1;
  bin2_lb = max_sysvar_val -  diff_sysvar_val * 0.5;

  /* check block bin */
  if (min_sysvar_val <= var_value && var_value < bin2_lb)
    return 3;
  else if (bin2_lb <= var_value && var_value < bin1_lb)
    return 2;
  else return 1;


}
/*******************************************************/

static void set_cost_table() {

  bin_cost_table[0][0] = 0;
  bin_cost_table[0][1] =  0.2;
  bin_cost_table[0][2] =  0.5;
  bin_cost_table[1][0] = 0.2;
  bin_cost_table[1][1] =  0.45;
  bin_cost_table[1][2] =  0.7;
  bin_cost_table[2][0] = 0.5;
  bin_cost_table[2][1] =  0.7;
  bin_cost_table[2][2] =  1.0;

}
```

Appendix C: MATLAB Code

make_lut_del_prof.m Code:

```
function [z] = make_lut_del_prof(nx, ny, alphax, px, py, var_fact,
cent_mean)
%% This function creates a sample performance profile to be
%% written into a file by write_var_file.m which is read by VA-VPR
%% The surface model used has the form
%% z = alpha*x^px + (1-alpha)*y^py, 0<alpha<1, px>0, py>0
%% nx: size of chip in x-direction
%% ny: size of chip in y-direction
%% var_fact: percent difference between worst-case and best-case
%% cent_mean: best-case value

x=-1:2/(nx-1):1;
y=-1:2/(ny-1):1;

z=zeros(length(x),length(y));
for i=1:length(x),
    for j=1:length(y),
        z(i,j) = alphax*x(i)^px + (1-alphax)*y(j)^py;
        if z(i,j)<0,
            z(i,j) = -z(i,j);
        end
    end
end
z = z ./ max(max(z));
z = z .* var_fact;
z = z .* cent_mean;
```

/***/

make_sys_offset_prof.m Code:

```
function [sys_offset_prof] = make_sys_offset_prof(nx, ny, lut_del_prof)
%% This function creates the systematic offset profile that is to be
%% written by write_var_file.m and read by VA-VPR using read_var.c
worst_case = 0;
for i=0:(nx-1),
    for j=0:(ny-1),
        if lut_del_prof(i+1, j+1) > worst_case,
            worst_case = lut_del_prof(i+1, j+1);
        end
    end
end
sys_offset_prof = worst_case - lut_del_prof;
figure; surf(sys_offset_prof); shading interp; rotate3d;
```

/***/

write_var_file.m Code:

```
function write_var_file(nx, ny, sys_offset_prof, lut_del_prof, filename)
%% This function is used to write the performance profile into a
%% comma-separated-value list file to be read by VA-VPR using read_var.c

tmp_prof = zeros(nx*ny, 4);
ind=1;
for i=0:(nx-1),
    for j=0:(ny-1),
```

```
    tmp_prof(ind,1) = i;
    tmp_prof(ind,2) = j;
    tmp_prof(ind,3) = sys_offset_prof(i+1, j+1);
    tmp_prof(ind,4) = lut_del_prof(i+1, j+1);
    ind = ind + 1;
    end
end

csvwrite(filename, tmp_prof);

/*********************************************************************/
```

make_var_file.m Code:

```
function make_var_file(nx, ny, alpha, px, py, var_fact, center_mean,
local_var_fact, filename)

lut_del_prof = make_lut_del_prof(nx, ny, alpha, px, py, var_fact,
center_mean);
sys_offset_prof = make_sys_offset_prof(nx, ny, lut_del_prof);
write_var_file(nx, ny, sys_offset_prof, lut_del_prof, filename);
```

Bibliography

[1] R. Rao *et al.*, "Parametric yield analysis and contrained-based supply voltage optimization." ACM/IEEE International Symposium on Quality Electronic Design, March 2005.

[2] Xiao-Yu Li *et al.*, "FPGA as process monitor – an effective method to characterize poly gate CD variation and its impact on product performance and yield", *IEEE Transactions on Semiconductor Manufacturing,* vol. 17, no.3, pp.267-272, 2004.

[3] S. R. Nassif, "Modeling and Analysis of Manufacturing Variations", IEEE Conference on Custom Integrated Circuits, 2001.

[4] D. Boning, et al., "Analysis and Decomposition of Spatial Variation in Integrated Circuit Processes and Devices", *IEEE Transactions on Semiconductor Manufacturing*, vol. 10, no. 1, pp. 24-41, February 1997.

[5] P. Friedberg, et al., "Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization", *Proceedings of the Sixth International Symposium on Quality Electronic Design, 2005. IEEE Computer Society.*

[6] L. He, J. Xiong, et al., "Robust Extraction of Spatial Correlation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 4, pp. 619-631, April 2007.

[7] S. Srinivasan, et al., "Variation Aware Placement in FPGAs", *IEEE Computer Society Annual Symposium on VLSI*, March 2006.

[8] L. He, J. Xiong, et al., "FPGA Performance Optimization Via Chipwise Placement Considering Process Variations", *International Conference on Field Programmable Logic and Applications*, 2006.

[9] V. Betz, J. Rose, A. Marquardt, "Architecture and CAD for Deep Sub-Micron FPGAs", Kluwer Academic Publishers, 1999.

[10] M. Orshansky, et al., "Impact of systematic spatial intra-chip gate length variability on performance of high-speed digital circuits", *IEEE/ACM International Conference on Computer Aided Design*, pp. 62-67, November 2000.

[11] S. Kirkpatrick, et al., "Optimization by Simulated Annealing", *Science*, no. 4598, May 1983.