HIGH-LEVEL RESOURCE BINDING AND ALLOCATION FOR POWER AND PERFORMANCE OPTIMIZATION

BY

SCOTT A. CROMAR

B.S., University of California, Irvine, 2007

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Adviser:

Professor Deming Chen

ABSTRACT

While technology scaling has presented many new and exciting opportunities, new design challenges have arisen. Smaller feature sizes have led to increased density and large variations in the delay and power characteristics of on-chip devices. Additionally, with the increasing desirability of low-power chips, decreasing power consumption has become a significant priority. Major sources of dynamic power consumption in modern chips include glitches (i.e., spurious signal transitions), the reduction of which are challenges to circuit designers. High-level synthesis has been touted as a solution to these problems, as it can both significantly reduce the number of man hours required for a circuit design, and offer greater opportunities for optimization of design goals, by raising the level of abstraction. In this thesis, we present two resource binding and allocation algorithms that take advantage of the optimization opportunities available at the higher level of abstraction.

The first is a new variation-aware high-level synthesis binding and module selection algorithm, named FastYield, which takes into consideration multiplexers, functional units, registers, and interconnects. FastYield connects with the lower levels of the design hierarchy through its inclusion of a timing-driven floorplanner guided by a statistical static timing analysis engine which is used to modify and enhance the synthesis solution. FastYield is able to incorporate spatial correlations of process variations in its optimization, which are shown to affect performance yield. FastYield is shown to achieve a significant reduction in clock period, and significant gain in performance yield, when compared to a variation-unaware and layout-unaware algorithm.

The second is a glitch-aware, high-level binding algorithm for power, area, and multiplexer reduction targeting field programmable gate arrays (FPGAs), called HLPower. HLPower employs a glitch-aware dynamic power estimation technique derived from an FPGA technology mapper. High-level binding results are converted to VHSIC hardware description language (VHDL), and synthesized with Altera's Quartus II software, targeting the Cyclone II FPGA architecture. Power characteristics are evaluated with the Altera PowerPlay Power Analyzer. The binding results of HLPower are compared to LOPASS, a state-of-the-art low-power high-level synthesis algorithm for FPGAs. Experimental results show that HLPower significantly reduces toggle rate and area, resulting in a large decrease in dynamic power consumption.

To Catherine, for her love and support

ACKNOWLEDGMENTS

I would like to thank my research adviser, Professor Deming Chen, for his guidance of this research. I would also like to thank all of my colleagues in the CAD group, especially Gregory Lucas, who was a significant collaborator for part of this work and contributed sections to Chapter 2.

I also want to thank my family for the constant support over the past couple of years.

This work was supported in part by grants from the NSF.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

1.1 High-Level Synthesis

As designs have increased in complexity, the number of man-hours required to implement a design has skyrocketed. To combat this difficulty, a significant and renewed effort has gone into high-level (or behavioral) synthesis algorithms, which raise the level of abstraction of the design and allow for a more efficient exploration of the design space.

A key step in the design of virtually all modern electronics, high-level synthesis transforms a behavioral description of a digital system to a register-transfer level (RTL) hardware implementation consisting of a datapath and a control unit. The datapath consists of three types of components or resources including functional units (such as arithmetic and logic units (ALUs), and multipliers), storage units (such as registers), and multiplexers. High-level synthesis usually consists of three subtasks: scheduling, resource allocation, and binding [1]. Scheduling involves the selection of *when* an operation will take place; allocation involves determining *how many* and *what types* of each resource are going to be needed (also called module selection); and binding involves selecting *where* an allocated resource will be bound to a specific operation.

In high-level synthesis, the design of the digital circuit is written as a behavioral description, commonly in a language such as C or SystemC. The design specification is first compiled into an internal representation such as a control data flow graph (CDFG), which is then mapped to the three types of resources, selected from the resource library, to optimize design goals (such as delay, power, and area). Previous research has shown that, compared to synthesis at RTL, code density can be reduced by 10 times, and simulation time can be reduced by 100 times with high-level synthesis [2].

High-level synthesis is a well-studied topic [1], [3], [4], [5]. Much work has been done in the areas of scheduling, resource allocation, and binding, and because binding and resource allocation are so interrelated, these two tasks are frequently undertaken simultaneously. In this thesis, we focus on the problem of binding and resource allocation to optimize for the specific design goals of performance optimization and power minimization. The binding algorithms presented target both application specific integrated circuits (ASICs), and field programmable gate arrays (FPGAs).

1.1.1 Challenges in circuit synthesis

High-level synthesis, and even the reduced complexity subtask of resource binding, is a difficult problem to solve due to a very large solution space. The work in [6] proved that the problem of resource binding for multiplexer reduction, itself, is NP-complete. Here, we briefly introduce two challenges to high-level synthesis that are relevant to the binding work to be presented. The challenges—of mitigating the effects of process variations on performance yield and reducing power consumption—are not unique to high-level synthesis. But previous work has shown that there are significantly greater opportunities for the optimization at a high level, than at lower levels of abstraction. For example, [7] showed that system and high-level power optimization techniques can achieve 40% more power reduction than is possible at a lower level of abstraction.

1.1.1.1 Process variations

Aggressive technology scaling to the deep submicron realm has resulted in significant variations in fabricated device parameters, complicating the efficient synthesis of digital circuits. The causes of these variations can be categorized roughly into two areas: process and environmental [8]. Process variations are, in general, static, meaning they do not change over time, and result from intrinsic nonuniformities in the manufacturing process and materials. Environmental variations, on the other hand, are dynamic, meaning they change with the passage of time and changing workload, and are caused by such factors as fluctuations in voltage and temperature on chip. We focus on process variations in this thesis.

Process variations can also be categorized into systematic and random. Systematic variations refer to those variations that are spatially correlated, while random variations have no correlation to one another. Systematic and random variations in structural device parameters (e.g., gate length and width, gate oxide thickness, metal thickness, dopant atom distribution, etc.), create variations in critical device parameters (e.g., threshold voltage) which can affect the performance (delay) and power (leakage) of the chip. The impact on the chip, such as an increase in worst-case critical-path delay and the resulting decrease in maximum operating frequency, can be significant. It has been reported that process variations can cause up to 20 times variation in chip leakage and 50% variation in chip frequency, depending on the process technology [9], [10]. Additionally, the effects of these variations on transistor parameters become more pronounced as technology scales to smaller feature sizes.

In order to overcome this obstacle, designers set a processor's frequency to allow for the worst-case delay, plus a margin of safety. Unfortunately, this option is becoming less and less viable with increasing variation, due to the increasing size of the required safety margin (also known as a guardband). For example, the delay variation for an adder can be up to 27% of its mean value, necessitating an excessive margin of error. The result is that designing for worst-case process margins is no longer a viable option. To ensure the meeting of design constraints, such synthesis methodology may result in a design that is slow, or a design with excess resource usage and/or unexpected discrepancies in power and performance. This has led to the development of statistical design techniques that attempt to consider probabilistic delays and power during circuit synthesis and simulation. In this thesis we focus on the mitigation of static correlated and random process variations through a novel, variation-aware, high-level binding algorithm.

1.1.1.2 Power consumption

In fabricated circuits there are two sources of power consumption: *static power* and *dynamic power*. Static power is power consumed when the circuit is either active or idle. Unless power gating and other transistor-level techniques are built into the chip, static power cannot be easily reduced. Dynamic power, on the other hand, is power consumed when a signal transition occurs at gate outputs, and, being a characteristic of the design implemented on the chip, is more easily mitigated. Signal transitions make up the switching activity (SA) of a circuit, and can be classified into two types: *functional transitions*, and *glitches*. Functional transitions are the signal transitions necessary to perform the required logic function, while glitches are spurious transitions: unnecessary transitions that occur due to unbalanced path delays at the inputs of a gate.

Dynamic power consumption can be estimated as $P_d = 0.5 \times SA \times C \times V_{dd}^2 \times f$, where SA is the switching activity of the circuit, C is the effective capacitance,

V_{dd} is the supply voltage, and f is the operating frequency. Reducing any of these factors will reduce the dynamic power of a circuit.

Dynamic power has been shown to be a significant source of total power consumption in ASICs, but this is true to an even greater extent in FPGAs [11]. The low power efficiency of FPGAs, as compared to ASICs, makes FPGA power minimization especially critical for potential low power applications. In Altera's Stratix II FPGAs in the 90 nm process technology, dynamic power is the dominant type of power consumed [12]. Further, it has been shown that glitches can account for 60% of the dynamic power consumed, and in some data-flow intensive designs glitches can account for 4–5 times more transitions than functional transitions [13], [14].

In this thesis, we target power minimization with our FPGA-targeted low-power binding algorithm, focusing on reducing switching activity (through a glitch-aware switching activity estimator) and effective capacitance (by reducing multiplexer area).

1.1.2 Related work

In this section we will briefly introduce some of the previous work in the areas of binding that are relevant to the work presented in this thesis: binding for performance yield optimization, and FPGA-targeted binding for low power.

1.1.2.1 Binding for performance yield optimization[1]

The significant variations in device parameters in newer fabrication processes have caused many traditional circuit design and analysis techniques to become inadequate. To overcome this obstacle, a shift in the design paradigm from the worst-case deterministic design to a statistical or probabilistic design is critical. A new era of statistical design techniques has begun to emerge in which circuit parameters such as delay and power are no longer modeled as deterministic values, but are represented as probability density functions (PDFs). These statistical design techniques are leading to the reclamation of performance and yield that has been lost when using deterministic design techniques.

The shift to probabilistic design methodologies has produced a number of gate-level variation-aware optimization techniques [15], [16]. While progress at the gate level is encouraging, the large productivity gains available in high-level synthesis make it attractive and necessary to address the issue of process variations at a higher level of abstraction.

A number of works, such as [17], have addressed the topic of simultaneous binding and floorplanning, but with no consideration of spatial correlation or variability. Huang et al. [18] presented a binding algorithm based on bipartite weighted matching. However, their algorithm does not address the critical issues of module selection and delay variability. Likewise, most of the work in high-

[1]This section includes previously published work. © 2009 IEEE. Reprinted, with permission, from G. Lucas, S. Cromar, and D. Chen, "FastYield: Variation-aware, layout-driven simultaneous binding and module selection for performance yield optimization," in Proceedings of the 2009 IEEE/ACM Asia South Pacific Design Automation Conference, 2009.

This material is reprinted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Illinois' products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to `pubs-permissions@ieee.org`.

By choosing to view this material, you agree to all provisions of the copyright laws protecting it.

level synthesis has ignored the issue of process variation as it has not been an important issue, but that has begun to change in the past few years as the need for variation-aware synthesis tools at the high level has been realized.

Hung et al. [19] offer a simultaneous scheduling, binding, and allocation algorithm based on simulated annealing. The simulated annealing algorithm seeks to reduce the overall latency while meeting a performance yield requirement. However, the algorithm does not consider multiplexer use or interconnect delay, both of which can significantly contribute to the clock period of the unit.

Jung et al. [20] propose a timing variation-aware high-level synthesis algorithm which improves resource sharing. While the algorithm is effective, it ignores multiplexers and interconnects, and also relies on the assumption that functional units are independent of each other in its yield calculation given by

$$yield = \prod_{k=1}^{n} P(FU_k < T_{clk}) \tag{1.1}$$

where n is the number of functional units, the function $P(\cdot)$ is the probability, FU_k is the delay distribution of functional unit k, and T_{clk} is the chosen clock period. As has been shown in [21] and [22], and as our results show, correlation among process parameters of the functional units has an effect on the performance yield. Thus, we need an enhanced yield calculation method.

Lastly, Wang et al. [23] propose a simulated annealing based method to consider both power yield and timing yield during high-level synthesis. They use a number of different simulated annealing moves combined with a cost function that penalizes the design if it exceeds a power or timing yield constraint. Spatial correlation and interconnect delay are not considered.

The algorithm we present in Chapter 2 of this thesis is a novel variation-aware simultaneous binding and module selection algorithm, which connects to the layout closely, so layout information can be accurately back-annotated to the synthesis and introduce useful synthesis transformations.

1.1.2.2 FPGA-targeted binding for low power

FPGAs hold significant promise as fast time-to-market replacements for ASICs in many applications. As the price of single-purpose chip development skyrockets in each successive technology iteration, the relative price of the FPGA architecture becomes more and more attractive. This, coupled with the many other advantages of FPGAs, such as rapid prototyping and field reprogrammability, makes them a more and more viable alternative in many current ASIC applications. Unfortunately, the advantages of FPGAs are offset in many cases by high power consumption and large area. In fact, it has been shown that FPGAs can be up to 40 times larger and consume up to 12 times more dynamic power than the equivalent ASIC implementation [11].

Research has shown that, in FPGA architecture, the area and power consumed by multiplexers and interconnects are significantly higher than those of logic cells and registers. In particular, it has been shown that the power consumed by a 32-to-1 multiplexer is nearly equivalent to that of a 18-bit multiplier in a 0.1 μm technology FPGA [24]. It has also been shown that interconnects can contribute up to 80% of the total area, and up to 85% of the total power in FPGAs [13], [25], [26]. Any approach to reducing the power consumption of a design implemented on an FPGA must take into account the significant contributions of multiplexers, interconnects, and glitches to power usage.

Much effort has gone into evaluating and reducing dynamic power in FPGAs. Recent work has included improved SA estimation tools [27], an investigation into implementing logic on embedded memory arrays [28], and architectural changes to reduce glitches [29].

Work in the area of low-power binding has included bipartite graph formulation for multiplexer reduction [18], low-power register binding through a network-flow formulation [30], simultaneous register and resource binding and scheduling algorithms [31], generalized low-power binding formulated as an integer linear programming (ILP) problem with heuristic speedups [32], early evaluation of data flow graphs for low-power binding [33], among many others. [32] provides a good overview of the previous work in low-power binding.

Although the work on high-level synthesis for ASICs is extensive, low-power high-level synthesis and binding for FPGAs is a relatively new area of research. In [34], the switching activity characteristics of the functional units were pre-characterized and used during low-power synthesis targeting FPGAs. However, ignorance of multiplexers created an overly simplistic model. In [35] a low-power, simultaneous resource allocation and binding algorithm for FPGAs was presented, implemented in UCLA's xPilot [36], and included a high-level power estimator. The work in [37] targets FPGA resource reduction, and indirectly multiplexer and power reduction, by identifying patterns in the CDFG that can be synthesized to the same resources.

In [38] and [39], the authors presented a simulated annealing-based algorithm which carried out high-level synthesis subtasks simultaneously, targeting FPGAs for low-power called LOPASS. Their binding algorithm was initially using minimum weight bipartite matching, and then was enhanced using a network flow approach presented in [24] that binds all the resources simultaneously. We compare our own algorithm, presented in Chapter 3, to LOPASS and show that an iterative approach enables greater power savings.

Low-power high-level synthesis is complicated by the difficulty of estimating power from a high level of abstraction. Glitch-power estimation is particularly difficult because calculation of glitches requires a gate-level view of the design.

In Chapter 3, we present an FPGA-targeted, low-power binding algorithm that considers glitches in its power estimation, in addition to targeting area and multiplexer reduction. We overcome the difficulty of estimating glitches during high-level synthesis by employing a low-power FPGA technology mapper, [40], which makes use of a switching activity estimation model considering glitches that has been shown to be effective at capturing glitch power. We focus on reducing switching activity (through the glitch-aware switching activity estimator) and effective capacitance (by reducing multiplexer area) for power minimization.

1.2 Primary Contributions

In this chapter we have provided an overview of high-level synthesis, some challenges to high-level synthesis, and previous work in the area of high-level synthesis relevant to the topic of this thesis. We have shown that there are significant challenges to circuit synthesis in general. Long simulation times and significant man power are necessary to design circuits at the RT level, making high-level synthesis an attractive opportunity. High-level synthesis has the potential for large gains in terms of time and optimization. Still, the large design space even at higher-levels of abstraction requires smart heuristic algorithms that can optimize for design goals.

We have also presented some of the background on work that has been done in the area of high-level synthesis, focusing on binding, for power minimization and performance yield optimization. Work has been done in both of these areas, but it has been lacking in significant ways. Algorithms for performance yield optimization have not always accurately incorporated variation information, or have used overly simplistic models. Algorithms that focus on power minimization have not considered glitches, and have not taken a low-level view for accurate power estimation.

In this thesis we present two novel high-level resource binding and allocation algorithms, FastYield and HLPower, that address many of the shortcomings of previous algorithms in this area. The main goals and contributions of this thesis are:

- A simultaneous binding and module selection algorithm that considers registers, multiplexers, functional units, interconnects, and spatially correlated process variations.
- A timing-driven, simulated annealing-based, statistical floorplanner that considers interconnect delay and spatial correlation between all units in the design.
- An iterative functional unit rebinding based on timing analysis information and register criticality.
- An FPGA-targeted iterative binding algorithm, driven by an accurate dynamic power estimation, that considers registers, multiplexers, and functional units.
- The incorporation of a glitch-aware dynamic power estimator based on low-level FPGA technology mapping, which makes use of an effective switching activity model, into a low-power binding algorithm.

1.3 Thesis Organization

This thesis is organized into four chapters that illustrate the goals and contributions outlined above. In Chapter 1, we have presented some background on the topic of high-level synthesis, outlined the major contributions of this thesis, and in the next section (1.4) we provide some definitions that will be helpful in the following chapters. Chapter 2 outlines the first of the two algorithms presented in this thesis: FastYield, a novel, variation-aware, high-level binding algorithm for performance yield optimization. In Chapter 3, we present the second high-level binding algorithm: HLPower, an FPGA-targeted binding algorithm for low power, considering glitches. Finally, Chapter 4 concludes the thesis and presents directions for future work.

1.4 Definitions

Definition 1. A **control data flow graph (CDFG)** is a directed acyclic graph, $G(V, E)$, where a vertex in V can be either an operation node or a control node, and an edge in E represents a data dependency between two nodes. A directed edge, $e(v_i, v_j)$, represents a transfer of value or control from one node to another. Each of the nodes in a CDFG can be classified as one of the following [41]:

- Operational nodes: responsible for arithmetic, logical or relational operations.
- Call nodes: denoting calls to subprogram modules.
- Control nodes: responsible for operations like conditionals and loop constructs.
- Storage nodes: representing assignment operations associated with variables and signals.

We use a two-level CDFG representation as an input to our binding algorithms. In the first level is the control flow, each node being a control or call

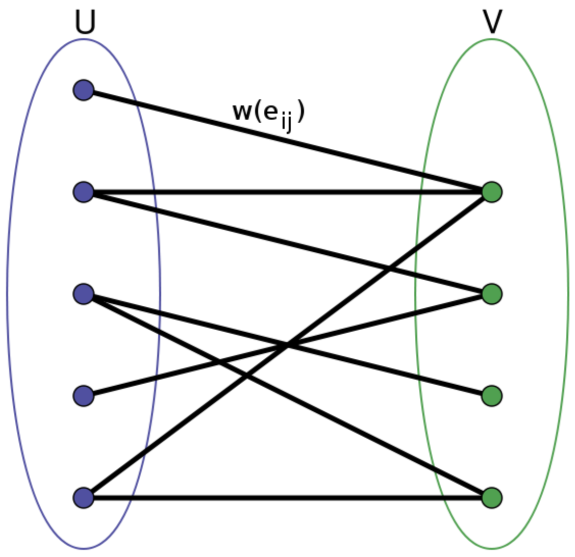


Figure 1.1: An example of a weighted bipartite graph.

node and representing a basic block. In the second level is the data flow of each basic block, each node being an operation or storage node.

Definition 2. A **weighted bipartite graph** is a graph, $G = (U, V, E)$, whose vertices can be divided into two disjoint sets, U and V, and whose edges in E can only connect a vertex in U with a vertex in V. A bipartite graph differs from a general graph in that there are no self-loops (edges which start and end at the same vertex), and no more than one edge between any two vertices. A weight $w(e_{i,j})$ is associated with every edge in the graph. See Figure 1.1.

Definition 3. The **maximum weighted matching** of a graph G is defined as a matching M—a set of edges without common vertices—where the sum of the weights of the edges in the matching has a maximal value.

Bipartite weighted matching has a runtime complexity of $O(|V| * (|E| + |V|log|V|))$, where $|V|$ is the total number of nodes in the graph. In general, weighted bipartite graphs can be solved more efficiently than other types of weighted graphs. We formulate the register binding problem as a bipartite weighted matching problem in our algorithms. We also make use of bipartite weighted matching in the functional unit binding of our algorithms.

CHAPTER 2

FASTYIELD: VARIATION-AWARE, LAYOUT-DRIVEN SIMULTANEOUS BINDING AND MODULE SELECTION FOR PERFORMANCE YIELD OPTIMIZATION

2.1 Overview

In this chapter, we present a variation-aware high-level synthesis binding/module selection algorithm, named FastYield, which takes into consideration multiplexers, functional units, registers, and interconnects. Additionally, FastYield connects with the lower levels of the design hierarchy through its inclusion of a timing-driven floorplanner, guided by a statistical static timing analysis (SSTA) engine, which is used to modify and enhance the synthesis solution. FastYield is able to incorporate spatial correlations of process variations in its optimization, which are shown to affect performance yield. On average, FastYield achieves a clock period that is 14.5% smaller, and a performance yield gain of 78.9%, when compared to a variation-unaware algorithm. By making use of accurate timing

This chapter includes previously published work. © 2009 IEEE. Reprinted, with permission, from G. Lucas, S. Cromar, and D. Chen, "FastYield: Variation-aware, layout-driven simultaneous binding and module selection for performance yield optimization," in Proceedings of the 2009 IEEE/ACM Asia South Pacific Design Automation Conference, 2009.

Gregory Lucas made significant contributions to this chapter, including Section 2.3 and Section 2.4.2.

information, FastYield's rebinding improves performance yield by an average of 9.8% over the initial binding, for the same clock period.

We connect our synthesis engine closely to the layout, so layout information can be accurately back-annotated to the synthesis and introduce useful synthesis transformations. Synthesis and layout are iterated until the performance gain is maximized. The major contributions of our algorithm are summarized below:

1. A simultaneous binding and module selection algorithm that considers registers, multiplexers, functional units, interconnects, and spatially correlated process variations.

2. A timing-driven, simulated annealing-based, statistical floorplanner that considers interconnect delay and spatial correlation between all units in the design.

3. An iterative functional unit rebinding based on timing analysis information and register criticality.

The rest of this chapter is organized as follows: Section 2.2 presents the problem formulation; Section 2.3 presents statistical functional unit modeling; Section 2.4 presents the details of the FastYield algorithm; Section 2.5 presents experimental results.

2.2 Problem Formulation

The input to our binding algorithm is a scheduled CDFG, an area constraint, and a resource library. The input CDFG format holds the scheduling information for all of the operational nodes. The problem to be solved involves the allocation and assignment of registers to variables, and functional units to operations. Module selection and binding are optimized to achieve a high performance yield. This is accomplished with the help of an accurate timing analysis on a layout of the binding solution after each iterative improvement. The binding problem can be formulated as follows:

Given: A scheduled CDFG, an area constraint, and a resource library.

Tasks: Allocate and bind registers to variables; select, allocate, and bind functional units to operations.

Objectives: Produce a valid binding solution while meeting the area constraint and optimizing the solution for performance yield.

2.3 Resource and Correlation Modeling

Modeling resources at a higher level of abstraction is critical to attaining an accurate high-level synthesis solution. We employ a Monte Carlo based method to *precharacterize* the functional units. Two types of variation are considered, random variation and correlated variation (or systematic variation). The characterization flow for each unit begins with logic synthesis followed by placement and routing using Synopsys Design Compiler and Cadence SOC Encounter. The characterization was performed on a recently released 45 nm standard cell library provided in the design kit from [42]. From the place and route information, the delay of the unit and placement of the individual gates in the unit are extracted.

Using Monte Carlo analysis, we then characterize the units by specifying a correlated, θ_{cor}, and independent, θ_{ind}, percentage of delay variation for each gate in the resource with respect to its nominal delay value. For each Monte Carlo run, the critical path of the circuit is then found by running a deterministic timing analysis (we used Synopsys PrimeTime). By plotting the critical path for each Monte Carlo run, the mean, μ_{FU}, and standard deviation, σ_{FU}, of the delay distribution are built.

To consider spatial correlation during the binding algorithm, we define two types of delay variation, interunit delay variation and intraunit delay variation. Interunit delay variation is defined to be correlated across units, while intraunit delay variation is defined to be independent across units. The components of inter- and intraunit delay variation are calculated as percentages of the standard deviation that was found from the Monte Carlo analysis of the resource. Equations (2.1) and (2.2) show the calculation of the intra- and interunit delay standard deviations.

$$\sigma_{intra}^2 = \sigma_{FU}^2 \times \theta_{ind}/(\theta_{ind} + \theta_{cor}) \tag{2.1}$$

$$\sigma_{inter}^2 = \sigma_{FU}^2 \times \theta_{cor}/(\theta_{ind} + \theta_{cor}) \tag{2.2}$$

We support different structural implementations of the same arithmetic operation. These implementations provide different delay and area tradeoff characteristics and offer opportunities for better design space exploration targeting higher performance yield given a specific resource or area constraint.

2.4 FastYield Binding Algorithm Description

In this section we will present the FastYield binding/module selection algorithm. FastYield seeks to improve performance yield through a multiplexer- and interconnect-aware delay reduction strategy. Performance yield evaluated at a clock period t, $PY(t)$, is defined as

$$PY(t) = P(r_1 \leq t, r_2 \leq t, \ldots, r_n \leq t) \tag{2.3}$$

where $PY(t)$ is the probability that $r_1, r_2, \ldots, r_n$ meets the clock period requirement, and r_n represents the probability distribution of register n. We assume all delays are jointly Gaussian with an associated covariance matrix, i.e., they are correlated.

The algorithm has three major components: (1) an initial resource allocation and binding; (2) a timing driven floorplanner, which performs both a timing driven placement as well as SSTA; and (3) a functional unit rebinding which incorporates timing analysis information from component 2. FastYield seeks to improve the synthesis solution through iteratively feeding back accurate, floorplan- and interconnect-aware, statistical timing information to the rebinding step.

One of the strengths of FastYield lies in its use of a process correlation model during timing analysis. Enabled by the floorplan, interconnect delay and multiplexer delays are considered during each SSTA step. Performance yield is calculated at the end of each timing analysis to evaluate the success of the algorithm, and the algorithm exits when no further improvement is seen in the binding/module selection solution. Each of the main components of FastYield is described next.

2.4.1 Initial binding

The inputs to the algorithm include: (1) a scheduled CDFG, (2) a resource library, and (3) an area constraint. The resource library contains all the resources including functional units, multiplexers, and registers as well as the precharacterization data for each. FastYield performs an initial allocation and binding in three steps: First, a minimal set of registers is allocated and bound to a set of variables (variables are outputs of operations). Second, a combined functional

Algorithm 2.1 Register Binding Algorithm.

```
1: traverse CDFG, find control step of max density
2: allocate a set of registers = max density
3: initialize clusters of mutually unsharable variables
4: for all clusters of mutually unsharable variables do
5:    initialize bipartite graph $G_r = (U_r, V_r, E_r)$; $V_r$ = variables, $U_r$ = registers
6:    calculate edge weights by equation in [18]
7:    solve $G_r$ for minimal weighted matching
8: end for
```

unit allocation and binding takes place. Third, a minimized set of multiplexers is allocated. We name this section Initial Binding to differentiate from the Rebinding procedure to be covered later.

2.4.1.1 Register allocation and binding

Register binding is accomplished in a manner similar to that described in [18], where variables are bound by solving a weighted bipartite graph. Algorithm 2.1 provides a summary of the register binding algorithm. An allocated set of registers is determined by counting the number of variables present in the control step of maximum density. This set of registers is allocated, and a cluster of mutually unsharable variables (meaning the lifetimes of these variables are overlapping) is bound at a time, by way of a weighted bipartite graph, sorted in ascending order according to their birth times.

The weighting of the edges of the bipartite graph is also very similar to [18]. The weighting is based on the idea that it is advantageous, for multiplexer reduction, to maximize register sharing among variables that have a common operation type, and minimize sharing among variables that cannot share the same operation type.

2.4.1.2 Initial functional unit allocation and binding

Once the registers are allocated and variables are bound to them, functional units are allocated and operations assigned to them one control step at a time. First, control steps are ranked according to the equation

$$Rank_{cstep} = diversity \times numOPs \tag{2.4}$$

where $diversity$ is the number of different types of operations in the control step, and $numOPs$ is the number of operations assigned to the control step. The control steps are then processed from the highest ranked to the lowest ranked. This strategy is similar to the “first fit decreasing” heuristic used in bin packing problems. The items are put in descending order according to their volumes (in this case rank), and then packed one at a time in an effort to make the packing as close to optimal as possible.

The cluster of control step operations to be bound is placed into a set, O_{cstep}, and the available functional units are put into a set FU_{av}. On the first control step to be bound, the set of available functional units consists of, for each operation in the control step, one instance of each functional unit in the resource library that is compatible with that operation (see Figure 2.1). This initial allocation ensures that each operation can bind to any of the compatible functional units in the resource library. In subsequent control steps, FU_{av} is trimmed of any functional units that, if allocated, would exceed the area constraint, with the qualification that a sufficient number of functional units of each type has been allocated to accommodate a successful binding solution. In this way FastYield produces a binding solution that meets the area constraint, while also enabling module selection.

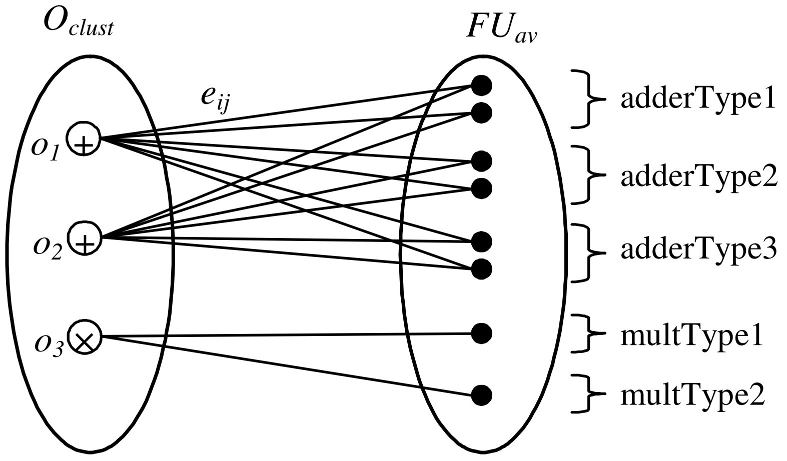


Figure 2.1: Illustration of the bipartite graph created for the functional unit binding of the first control step.

A weighted bipartite graph is constructed where each vertex represents either an operation ($o_i \in O_{cstep}$) or a functional unit ($fu_j \in FU_{av}$), and there is an edge, e_{ij}, between each operation, o_i, and functional unit, fu_j, which can perform the operation, with a corresponding weight. Edge weights are based on multiplexer creation due to the already bound registers. If two operations share the same input register, it is advantageous to bind the two operations to the same functional unit, because no multiplexer is needed (which will in effect potentially reduce the path delay). Likewise, if two operations that share the same output register are bound to the same functional unit, no multiplexer is needed at the register's input port (again having a positive effect on the delay reduction). The initial binding weight, $w_{ij\_initial}$, corresponding to each edge, e_{ij}, is calculated as

$$w_{ij\_initial} = \frac{1}{estDelay(i,j)} \tag{2.5}$$

$$estDelay(i,j) = \mu_{fu_j} + \mu_{mux_{in}} + \mu_{mux_{out}} + 3 \times \sqrt{\sigma^2_{fu_j} + \sigma^2_{mux_{in}} + \sigma^2_{mux_{out}}} \tag{2.6}$$

where μ is the mean, σ is the standard deviation, mux_{in} is the multiplexer that would be created at the input of the functional unit if the operation were bound to it, and mux_{out} is the multiplexer that would be created at the input of the output register if the operation were bound to the functional unit. This weight calculation effectively incorporates the statistical behaviors of all the involved components in the circuit paths, putting a higher weight on the shorter delay paths. The maximum weight solution is then found to minimize the delay, and the operations are bound to functional units for the control step. After all the control steps are processed, functional units and registers are connected with allocated multiplexers.

2.4.2 Statistical timing driven floorplanner

The timing-driven floorplanner is run after each binding iteration is completed to evaluate the performance yield of the solution. As has been shown in previous work, [22], [43], and as we show in the experimental results section, spatial correlation of variation in parameters such as gate length can have an impact on the variance of the timing of a circuit. To achieve accurate timing results, it is important that spatial correlation among units is considered during statistical timing analysis.

2.4.2.1 Unit correlation model

To complement the high-level synthesis resource modeling, we propose a novel unit-based correlation model. In this model, each functional unit, register, or multiplexer is assigned a unit number and the correlation between each unit is found based on the distance between the center points of the units using a correlation function that meets the requirements of [22] so that the correlation matrix for the circuit is positive-semidefinite, a requirement for the SSTA approach that we use.

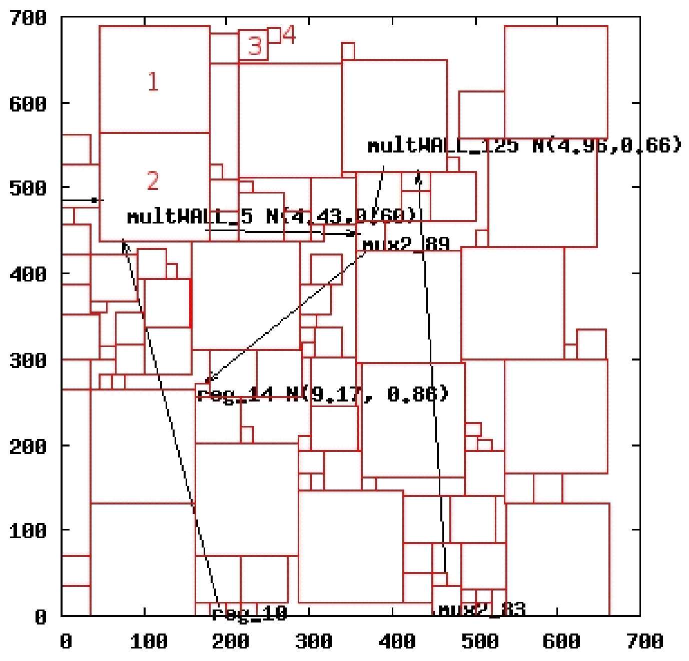


Figure 2.2: Sample floorplan showing data connections.

This model is beneficial to high-level synthesis as it complements the proposed resource modeling (Section 2.3), and also takes into account the different sizes of functional units. On the other hand, a grid based model, as used in [43], does not complement the unit characterization since it is possible for functional units to be split across different grid regions, which complicates both the unit characterization process and the correlation calculation.

Figure 2.2 shows an example of the unit correlation model. Two multipliers (1 and 2), an adder (3), and a register (4) are labeled in the picture. It can be seen that when an adder and a register (small area) are placed next to each other, the correlation is higher than when two multipliers (large area) are placed next to each other. This scenario can be accurately modeled using our unit-based model. Our model can also be viewed as an extension of the grid based model where each logic gate/functional unit is its own grid.

Algorithm 2.2 Timing Driven Floorplanner Pseudo Code.

```
1: Parquet:
2: while time > time_cool do
3:    Perform_moves(num_moves);
4:    Calc_wire_delay();
5:    Calc_Correlation();
6:    Perform_PCA(); //principle component analysis
7:    Perform_timing_analysis();
8:    Calculate_Cost();
9: end while
```

The proposed correlation model, in conjunction with the interunit and intraunit variation found during the resource characterization, allows correlated variation to be represented at a higher level of abstraction with accuracy and runtime efficiency.

2.4.2.2 SSTA algorithm

To obtain layout information during the timing analysis, we use a modified version of the Parquet floorplanner [44]. The modified flooplanner employs a simulated annealing approach where, after a number of unit moves, a statistical timing analysis is performed to evaluate the solution. Algorithm 2.2 shows the pseudo code for the timing-driven floorplanner.

The method for statistical timing analysis considering spatial correlation is based on the work of Chang et al. [43]. This work relies on principal component analysis (PCA) to transform a set of correlated random variables into a new set of independent random variables.

To perform PCA, a correlation matrix for the binding solution is found using the unit correlation model described above. The interconnect delay between the units is modeled based on distance. Since no detailed routing information is available, we model the connection between two functional units as a two-pin net with the length being the Manhattan distance between the two connecting terminals of the functional units. The mean Elmore delay with optimal buffer placement is then found using Equation (2.7) which follows from the results of [45]:

$$\mu_{wire} = 2.5 \times \sqrt{R_{buff} C_{buff} R_{length} C_{length} l^2} \tag{2.7}$$

$$\sigma_{wire} = \alpha \times \mu_{wire} \tag{2.8}$$

where μ_{wire} is the mean wire delay, R_{buff} is the output resistance of the buffer, C_{buff} is the input capacitance of the buffer, R_{length} is the resistance per unit length, C_{length} is the capacitance per unit length, and l is the net length. The standard deviation of the wire length is calculated using Equation (2.8), where α is a percentage of wire variation. The value of α is found in accordance with the results from [46] as follows[1]:

$$\alpha = 0.3836 \times exp(-0.1537h) \tag{2.9}$$

where h is the optimal buffer size as calculated by [45]. We consider the wire variation to be independent across wires.

2.4.2.3 Floorplanner cost function

The cost function for simulated annealing moves in the floorplanner is given by Equation (2.12):

$$Z \sim N(\mu_z, \sigma_z) = max(reg_1(\mu_1, \sigma_1), reg_2(\mu_2, \sigma_2), \ldots, reg_n(\mu_n, \sigma_n)) \tag{2.10}$$

$$T_R = \frac{\mu_z + \sigma_z}{\mu_{best} + \sigma_{best}} \tag{2.11}$$

$$Cost = \alpha \times area + \beta \times T_R \tag{2.12}$$

[1]We plotted the equation based on the buffer size vs. wire variation data reported in [46].

where $max(reg_1(\mu_1, \sigma_1), reg_2(\mu_2, \sigma_2), \ldots, reg_n(\mu_n, \sigma_n))$ represents the statistical max operation [43] on the timing distributions at the inputs to all output registers (pseudo primary outputs), μ_{best} and σ_{best} represent the mean and standard deviation of the best solution found so far, and α and β are weighting parameters. The T_R cost is then found by adding the mean and standard deviation of the max distribution, normalized by the mean and standard deviation of the best solution. In the calculation of T_R, we chose to use the sum of the mean and standard deviation since the result corresponds to the required clock frequency for an ~84% yield, for which we target in this study. After a specified number of moves, a timing analysis is performed on all paths in the design as described in Algorithm 2.2.

Upon completion of the timing analysis, the delay PDF for each register is known. The distributions, as well as the required clock frequency for an 85% performance yield, are then passed back to FastYield for the criticality analysis of the rebinding step. Figure 2.2 shows the example floorplan obtained from the timing driven floorplanner, with the arrows representing the flow of data through the critical path.

2.4.3 Rebinding

Functional unit rebinding is performed after the initial solution has been analyzed by the timing driven floorplanner, and then continues in an iterative fashion until the floorplanner reports that no improvement has been made. The rebinding algorithm works by determining which functional units along the critical paths are causing the majority of the delay. It then attempts to reduce the delay in two ways: one, by swapping slower functional units on critical paths for faster functional units; and two, by rebinding individual operations on the

Algorithm 2.3 Rebinding Algorithm Pseudo Code.

```
1: Calc_reg_and_FU_ranks();
2: if Swap_critial_FUs() then
3:    Break;
4: end if
5: Order_rebind_operations($O_{rebind}$);
6: for all op in $O_{rebind}$ do
7:    Calc_op_to_FU_weights();
8:    Bind_largest_weight_pair();
9:    Estimate_timing();
10:   Recalc_reg_and_FU_ranks();
11: end for
```

critical paths. Algorithm 2.3 shows the pseudo code for the rebinding algorithm, which will be explained next.

2.4.3.1 Register and functional unit ranking

The algorithm begins by ranking the output registers in order of their criticality. The slowest register is identified by finding the worst case delay based on the mean and standard deviation from the floorplanning information. The rank of register r is then calculated:

$$RegRank_r = \frac{\mu_r + 3\sigma_r}{\mu_{slowest} + 3\sigma_{slowest}} \tag{2.13}$$

where μ_r and σ_r are the mean and standard deviation for register r, and $\mu_{slowest}$ and $\sigma_{slowest}$ are the mean and standard deviation of the slowest register. The registers are then ordered according to their criticality, or rank, starting from the most critical.

With the registers ranked, the algorithm then proceeds to rank each functional unit that is connected to each register. The rank of functional unit k

connected to register r is found by

$$FURank_k = RegRank_r \times (0.5 \times \frac{\mu_k}{\mu_r} + 0.5 \times \frac{\sigma_k}{\sigma_r}) \tag{2.14}$$

where μ_r and σ_r are the mean and standard deviation for output register r, μ_k and σ_k are the mean and standard deviation of the functional unit, and $RegRank_r$ is the rank for register r. The $RegRank_r$ weight provides an estimate of the global impact the register has on the overall clock period, while the ratio of the means and standard deviations considers how much the overall mean or variance of the functional unit impacts the final timing at the register. The end of Section 2.4.3.3 will present an example of how this ranking is accomplished.

2.4.3.2 Swapping critical functional units

The rebinding algorithm examines the set of allocated functional units, and based on their rank, finds any higher ranked functional units that are slower than lower ranked, faster functional units of the same type, and swaps them. The net effect is to place the fastest functional units on the most critical paths. If no functional units meet the criteria for swapping, the rebinding proceeds to the next step. If functional units are swapped, then the timing analysis is rerun before rebinding proceeds.

2.4.3.3 Selection of operations to be rebound

The rankings of the registers and functional units are used in the selection of particular operations that will be rebound. Operations are chosen that both contribute to a critical path delay, and have the potential to reduce that delay. Briefly, this is done as follows: First, a set of output registers are selected for their delay criticality based on their rank. For each chosen register, the functional unit connected to it with the highest rank (denoting its greater contribution to the criticality of the register) is selected, and an operation, or multiple operations, that are bound to that functional unit are selected to be rebound. The criterion for selection of the particular operations associated with each functional unit to be rebound is the operation's potential, if rebound, to reduce multiplexer size on that critical path. The example in the next paragraph serves to clarify the process.

An example of the register and functional unit ranking, and operation selection, is illustrated in Figure 2.3. The method is presented step-by-step: The slowest register has a mean $\mu = 3.1$, and a standard deviation $\sigma = 0.3$. (1) Based on the slowest register information, by Equation (2.13) register 5 is found to have a rank of 0.9. (Register 5 is determined to be critical based on its rank.) (2) The functional units connected to register 5 are ranked according to Equation (2.14). fu_2 is found to have a higher rank than fu_1, so it is from fu_2 that an operation, or operations, will be selected for rebinding. (3) The inputs to fu_2 are examined, and port 1 is found to have a larger multiplexer than port 0. (4) The registers connected to the inputs of the 3-input multiplexer are evaluated. One of the three registers has two variables bound to it, and the other two have one variable bound to them. Since fewer variables bound to a register is preferred (more likely to reduce the multiplexer size if moved), register 3 is randomly chosen from the two registers with only one variable bound to them. The operation corresponding to that register/variable, operation 1 in this case, is assigned the rank of the target functional unit, and is chosen for rebinding. This same process is carried out for each critical register, and the selected operations (along with their ranking) are placed in the set O_{rebind} to be rebound.

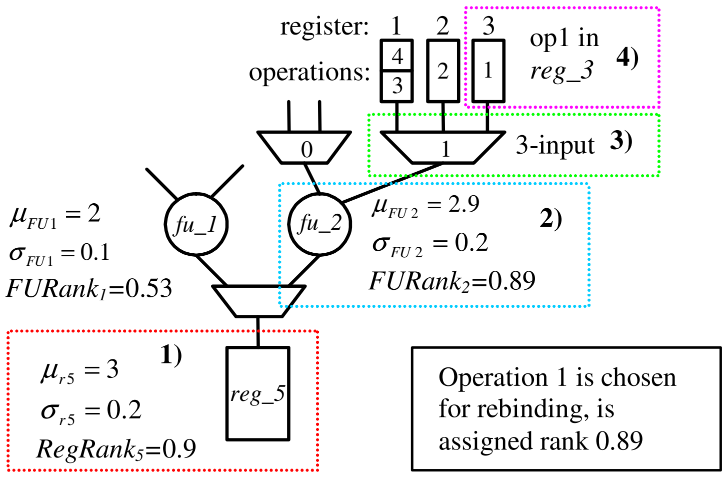


Figure 2.3: Example of functional unit ranking and the selection of operations for rebinding.

2.4.3.4 Operation rebinding

The rebinding is performed for each operation $o_i \in O_{rebind}$, one operation at a time, starting with the operation with the highest rank. Previous bindings that have not been selected for rebinding are left untouched. For a given operation, o_i, a rebind weight is calculated for each functional unit, fu_j, in the previously allocated functional unit set. The weight, w_{ij_rebind}, for each operation functional unit pair is calculated as follows:

$$w_{ij\_rebind} = \frac{w_{ij\_rebindPrevious}}{max(w_{i\_rebind})} \times (1 - FURank_j) \tag{2.15}$$

where $w_{ij\_rebindPrevious}$ is the weight of the operation-to-functional unit pair in the previous iteration of rebinding (or the initial binding if this is the first iteration), $max(w_{i\_rebind})$ is the maximum weight from all of the operation-to-functional unit pairs, and $FURank_j$ is the functional unit rank as described earlier. The first part of the weighting considers the likelihood operation o_i had of being assigned to fu_j in the previous binding. If o_i was close to being assigned to fu_j during the previous binding, then rebinding o_i to fu_j will be a good choice, if the rank of the functional unit is low (meaning it currently is not a part of the critical path). The second part of the equation adds this rank consideration to the weight.

The operation-to-functional unit pair with the largest weight is then chosen, and that operation is bound to the functional unit. The process repeats for each operation that belongs to the set O_{rebind}. However, after each operation is rebound it is possible that the multiplexer size has changed, which in turn reduces the critical path of the circuit and changes the ranks of the registers. Therefore, after each operation is rebound, a fast estimated timing analysis is performed on the paths that are affected by the rebinding of the operation and the register and functional unit ranks are recalculated. After every operation in O_{rebind} has been processed, one iteration of rebinding is complete and the solution is sent to the floorplanner for analysis.

2.5 Experimental Results

In this section we present a number of results that demonstrate the importance of considering process variation and correlation during high-level synthesis, and the effectiveness of FastYield at accomplishing these tasks. FastYield reads in a benchmark, which has been prescheduled with list scheduling, and a resource library, and runs it through the initial binding, floorplanning and timing analysis, and rebinding. The resource library contains the precharacterized resources, which include functional units, multiplexers, and registers. The resources were precharacterized with 10% random variation and 10% spatially correlated variation with a correlation distance of 1 mm (such assumptions are compatible with the variation predictions laid out in [47]). The characterization was per-

Table 2.1: Benchmark Profiles.

Bench-marks	No. of PIs	No. of POs	No. of Adds	No. of Mults	Total No. of Edges
chem	20	10	171	176	731
dir	8	8	84	64	314
honda	9	2	45	52	214
mcm	8	8	64	30	252
pr	8	8	26	16	134
steam	5	5	105	115	472
wang	8	8	26	22	134

formed on a 45 nm library provided in the design kit from [42], as described in Section 2.3.

A number of data-intensive benchmarks are used in our experiments with FastYield. The benchmark CDFGs include several different discrete cosine transform (DCT) algorithms including *pr*, *wang*, and *dir*, and several digital signal processing (DSP) programs including *chem*, *steam*, *mcm* and *honda* [48]. The benchmarks are profiled in Table 2.1. Each node in the benchmarks is either an addition/subtraction or a multiplication.

2.5.1 Spatial correlation in timing analysis

In order to show the importance of considering spatially correlated process parameters during the timing analysis, we performed a floorplanning and timing analysis on the same binding solution with spatial correlation and θ_{ind} from Equation (2.1) set to 0 (Corr), and without correlation (No Corr) with $\theta_{ind} = 1$. Setting $\theta_{ind} = 0$ makes $\sigma^2_{inter} = \sigma^2_{FU} = 1$. This makes it possible for all the functional unit variation to be correlated between units; however, the actual correlation between the functional units is still found based on the distance between them. The results are shown in Table 2.2. Columns 2 and 3 show the clock period obtained for an 85% yield with Corr and No Corr, respectively.

Table 2.2: Correlation vs. No-Correlation Experimental Results.

	85% Yield Clk (ns)			
Benchmarks	Corr	No Corr	Corr reduction in Clk over No Corr (%)	Corr 85% PY Gain over No Corr (%)
chem	5.91	6.20	4.70	14.97
dir	4.91	5.14	4.49	14.98
honda	5.14	5.30	3.03	14.37
mcm	4.09	4.28	4.35	10.56
pr	4.45	4.66	4.51	14.99
steam	5.54	5.80	4.51	14.98
wang	4.91	5.11	3.98	14.99

Column 4 shows the reduction in clock period of the Corr result over the No Corr result, which averages 4.22%. Column 5 reports the performance yield (PY) gain of Corr over No Corr, which averages about 14.25%. That is, for the No Corr clock period given, one would expect to achieve an 85% PY based on the No Corr timing analysis, but would achieve a 85% + 14.25% = 99.25% PY based on the Corr timing analysis. In other words, timing analysis without consideration of correlated process parameters is conservative compared to correlated timing analysis. This shows the importance of using spatial correlation information to guide the floorplanner, as well as performing accurate SSTA.

2.5.2 FastYield compared to BindBWM and rebinding improvement

We compare the results of FastYield after rebinding (FY Rebind) to an enhanced version of the weighted bipartite graph based binding (here referred to as BindBWM) of Huang et al. [18]. The enhancements to [18] include module selection and the ability to specify an area constraint, making the comparison demonstrative of the performance yield gains that can be achieved when considering process variation during binding. The same schedules, area constraints, and library were used in both algorithms. We also compare FY Rebind perfor-

Table 2.3: FastYield Experimental Results.

	BindBWM		FastYield Initial		FastYield Rebind	
Bench-marks	85% Yield Clk (ns)	PY at FY Rebind 85% Clk (%)	85% Yield Clk (ns)	PY at FY Rebind 85% Clk (%)	85% Yield Clk (ns)	Total FY Run Time (min)
chem	6.9	12.5	6.1	67.7	6.0	75
dir	5.8	1.5	4.9	70.9	4.8	43
honda	5.7	8.1	4.9	82.6	4.9	28
mcm	4.9	11.4	4.3	78.0	4.2	40
pr	5.2	0.1	4.5	70.1	4.3	24
steam	6.2	7.6	5.5	76.3	5.5	64
wang	5.3	1.6	4.7	80.8	4.6	16

mance to the performance attained by FastYield before rebinding (FY Initial) to show the effect of timing information on the rebinding solution. In all of the benchmarks, the same number of adders and multipliers were allocated in the binding solution for BindBWM, FY Initial, and FY Rebind.

Tables 2.3 and 2.4 summarize the experimental results. Columns 2, 4, and 6 of Table 2.3 give the clock periods for each BindBWM, FY Initial and FY Rebind, respectively. Columns 3 and 5 of Table 2.3 give the PY attainable by the respective binding solutions if clocked at the 85% PY clock of FY Rebind. Figure 2.4 demonstrates this graphically by plotting the cumulative density functions (CDFs) for the different binding results of *chem* (PDFs are inset). If clocked at the 85% PY clock period of FY Rebind, FY Initial and BindBWM have PYs of 67.7% and 12.5%, respectively.

Table 2.4: FastYield Experimental Results, Continued.

	Comparison			
Bench-marks	FY Rebind reduction in Clk over BindBWM (%)	FY Rebind 85% PY Gain over BindBWM (%)	FY Rebind reduction in Clk over FY Initial (%)	FY Rebind 85% PY Gain over FY Initial (%)
chem	14.17	72.5	2.35	17.3
dir	16.71	83.5	1.76	14.1
honda	14.39	76.9	0.32	2.4
mcm	14.57	73.6	3.34	7.0
pr	16.47	84.9	3.04	14.9
steam	11.88	77.4	1.14	8.7
wang	13.29	83.4	0.95	4.2
Average	14.50	78.9	1.84	9.8

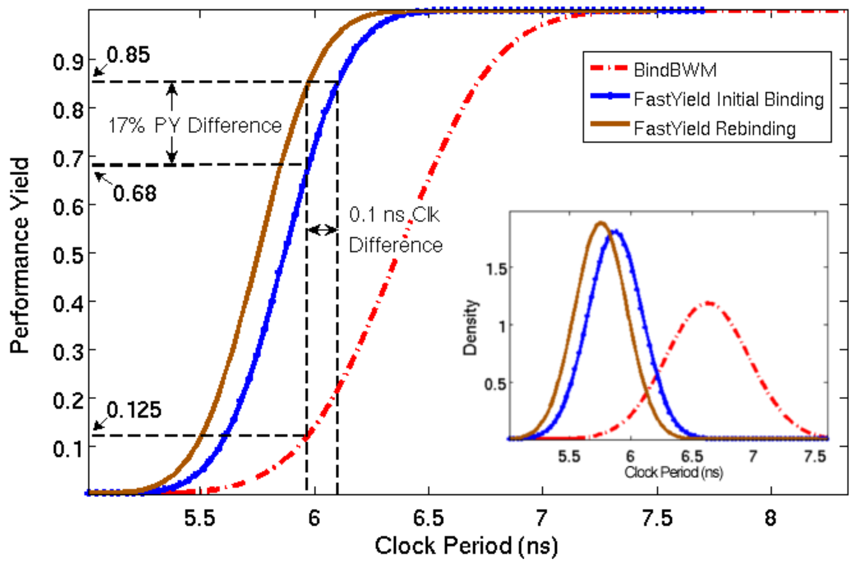


Figure 2.4: *Chem* delay distributions of BindBWM, FY Initial, and FY Rebind.

In Table 2.3 , Column 7 gives the total FY runtime in minutes. Columns 2 and 4 of Table 2.4 give the FY Rebind percentage reduction in clock period when compared to BindBWM and FY Initial, respectively. Columns 3 and 5 of Table 2.4 give the PY gain (in percent) of FY Rebind over BindBWM and FY Initial, respectively. This means that if BindBWM or FY Initial were clocked at the 85% PY clock period of FY Rebind, they would have a PY smaller than 85% by the given amount.

By considering process variation and layout, FY Rebind is able to reduce the clock period of the benchmarks by an average of 14.5% and increase the performance yield an average of 78.9%, when compared to BindBWM. It is also able to improve clock period and PY by an average of 1.84% and 9.8%, respectively, over FY Initial.

In some cases the amount of clock period improvement that rebinding can achieve is limited by the number of the type of unit that is on the critical path. For example, if there are 4 allocated multipliers, all of which are found to be critical, then rebinding cannot offer much improvement. However, if only 3 of 4 allocated multipliers are found to be critical, then rebinding can offer more improvement.

Often, though, even if the reassignment of operations has a small effect on mean clock period, it can have a large impact on the variance of the clock period, thus improving the PY significantly. This can be seen in Figure 2.4, where there is a large improvement in the delay CDF between the BindBWM and FY Rebind. This explains the results in Column 3 of Table 2.3 , where we see that when BindBWM is clocked at the 85% PY clock value of FY Rebind, the PY is very small. The difference between FY Rebind and FY Initial is not as drastic, but there are two key improvements. First, the mean of the PDF has been shifted to a lower clock value. Second, the variance has been reduced. Combining these two improvements results in a significant PY jump for a relatively minor change in the mean clock period (17% PY difference in the example).

CHAPTER 3

HLPOWER: FPGA-TARGETED HIGH-LEVEL BINDING ALGORITHM FOR POWER AND AREA REDUCTION WITH GLITCH-ESTIMATION

3.1 Overview

Glitches (i.e., spurious signal transitions) and multiplexers are major sources of dynamic power consumption in modern FPGAs. In this chapter we present an FPGA-targeted, glitch-aware, high-level binding algorithm for power, area, and multiplexer reduction. Our binding algorithm employs a glitch-aware dynamic power estimation technique derived from the FPGA technology mapper in [40], which makes use of a switching activity estimation model considering glitches that has been shown to be effective at capturing glitch power. High-level binding results are converted to VHSIC hardware description language (VHDL), and synthesized with Altera's Quartus II software, targeting the Cyclone II FPGA architecture [49]. Power characteristics are evaluated with the Altera PowerPlay Power Analyzer [50]. The binding results of our algorithm are compared to LOPASS, a state-of-the-art low-power high-level synthesis algorithm for FPGAs. Experimental results show that our algorithm, on average, reduces toggle rate by 22% and area by 9%, resulting in a decrease in dynamic power consumption of 19%.

The major contributions of our algorithm are summarized below:

1. An FPGA-targeted iterative binding algorithm, driven by an accurate dynamic power estimation, that considers registers, multiplexers, and functional units.

2. The incorporation of a glitch-aware dynamic power estimator based on low-level FPGA technology mapping, which makes use of an effective switching activity model, into a low-power binding algorithm.

The rest of this chapter is organized as follows: In Section 3.2 we present the problem formulation. In Section 3.3 we describe the technique used for switching activity estimation considering glitches. In Section 3.4 we describe the binding algorithm, herein referred to as HLPower, in detail. In Section 3.5 we present experimental results.

3.2 Problem Formulation

The input to our binding algorithm is a scheduled CDFG, a resource constraint, and a resource library. The input CDFG format holds the scheduling information for all of the operational nodes. The problem to be solved involves the allocation and assignment of registers to variables, and functional units to operations. Efficient sharing of functional units by operations, and registers by variables, in order to reduce power and multiplexer usage, are the challenges of binding. The binding algorithm is driven by the glitch-aware dynamic power estimation, in an effort to reduce the total power usage of the design. The binding problem can be formulated as follows:

Given: A scheduled CDFG, a resource constraint, and a resource library.

Tasks: Allocate and bind registers to variables, and allocate and bind functional units to operations.

Objectives: Produce a valid binding solution while meeting the resource constraint and optimizing the solution for power and area on the targeted FPGA.

3.3 Switching Activity Estimation

As dynamic power estimation is a central driver of our binding algorithm, the way this is accomplished is described in this section. Dynamic power is estimated in the form of a switching activity model based on probabilistic techniques developed originally in [51], extended in [52], and further developed to target FPGA mapping and include glitches in [40].

In [51] the ideas of *transition density* (also referred to as *toggle rate* or *switching activity*) and *signal probability* are initially developed. The *transition density* of a logic signal is defined as the average number of transitions per unit time, while the *signal probability* is defined as the fraction of the time that the logic signal is in the 1 state (i.e., the average value of the logic signal over all time). An efficient technique is presented that allows for the calculation of the total circuit switching activity by means of propagation of transition densities and signal probabilities from input nodes to output nodes. For a node y with independent fanin nodes $x_1, x_2, \ldots, x_n$, and given the transition density (switching activity) $s(x_i)$ of each fanin node x_i, the transition density of node y, $s(y)$, can be computed using the Boolean difference $(\partial y/\partial x)$ of y with respect to x_i:

$$s(y) = \sum_{i=1}^{n} P(\frac{\partial y}{\partial x_i})s(x_i) \tag{3.1}$$

where $P(\partial y/\partial x_i)$ is the signal probability of the Boolean difference.

This technique was extended in [52] to take into account simultaneous switching, something that the technique in [51] lacked. Let y be a Boolean expression, $y(t)$ be its value at time t, $P(y)$ be the signal probability of y, and $s(y)$ now be the normalized switching activity of y. $s(y)$ is the probability of y having different values at time t and $t + T$, where T is a unit time period, and is thus given by $s(y) = P(y(t)\overline{y(t+T)}) + P(\overline{y(t)}y(t+T))$. Additionally, note that

$P(y(t)\overline{y(t+T)}) = P(\overline{y(t)}y(t+T))$. Thus, $P(y(t)\overline{y(t+T)}) = P(\overline{y(t)}y(t+T)) = 1/2s(y)$. Since $P(y(t)) = P(y(t)y(t+T)) + P(y(t)\overline{y(t+T)})$, we find

$$s(y) = 2(P(y(t)) - P(y(t)y(t+T))) \tag{3.2}$$

And, as noted in [40], the term $P(y(t)y(t+T))$ can be calculated from the probabilities and switching activities of fanin nodes of y using the procedure in [52].

Finally, the technique for switching activity estimation was applied to FPGA technology mapping in [40]. The algorithm in [40] reads in a netlist, and uses a cut-enumeration technique [53] to select K-input cuts that will be mapped to the FPGA (K-input lookup tables). Primary inputs are assumed to have signal probabilities and switching activities of 0.5. For each node, the signal probability of all of the K-input feasible cuts of that node are computed using the weighted averaging algorithm from [54]. When calculating the switching activities for each cut, the widely accepted unit delay model is assumed for the FPGA lookup tables. This means that signal transitions are assumed to happen only at discrete time units: $1, 2, \ldots, D(C)$, where $D(C)$ is the depth of the cut. The transition that takes place at time $D(C)$ is considered the functional transition, while the transitions that occur at the other time steps are considered glitches. Switching activities are then calculated and propagated through the cut according to Equation (3.2). For a given cut, the effective switching activity is a summation of the switching activities at each time step. An example of how this is accomplished can be found in [40].

The best cuts, those with the lowest switching activities, are then chosen for implementation of the node in the FPGA. Summing up the switching activities, sa_i, for all of the selected cuts, $1, 2, \ldots n$, provides the *total estimated switching* *activity*, SA, for the netlist:

$$SA = \sum_{i=1}^{n} sa_i \tag{3.3}$$

The *total estimated switching activity*, SA, is used in the binding algorithm. This technique for switching activity estimation has the advantages over previous techniques of being mapping-aware and considering glitches.

3.4 HLPower Binding Algorithm Description

The HLPower binding algorithm proceeds in two major parts. First, registers are allocated and bound, and second, functional units are allocated and bound. In this description we focus on the functional unit binding. The functional unit binding proceeds in an iterative fashion until the resource constraint is met, driven by the estimated dynamic power usage, and multiplexer sizes, of various operation-to-functional unit bindings, as will be explained below. Algorithm 3.1 provides a summary of the HLPower binding algorithm.

3.4.1 Register binding

Register binding in HLPower is accomplished in the same way as described in FastYield, Section 2.4.1.1.

3.4.2 Functional unit binding

3.4.2.1 Algorithm overview

Functional unit binding iteratively constructs weighted bipartite graphs, finds a maximum matching, and combines nodes that are matched. Before the first iteration of the functional unit binding, the scheduled CDFG is traversed, and

Algorithm 3.1 HLPower Binding Algorithm.

```
1: **Input:** Scheduled CDFG, library, resource constraint
2: **Output:** Scheduled and bound CDFG
3:
4: precalc $SA$ values for all functional unit & multiplexer combinations
5:
6: bind registers as described in Section 2.4.1.1
7:
8: /* Functional Unit Binding */
9: traverse CDFG, select nodes for set $U$
10: put remaining nodes in set $V$
11: while resource constraint is not met do
12:    initialize bipartite graph $G = (U, V, E)$
13:    for all edges in $E$ do
14:       calculate input multiplexer sizes (if nodes were combined)
15:       look up $SA$ value for particular functional unit & multiplexers
16:       calculate edge weight
17:    end for
18:    solve $G$ for maximum weight
19:    combine matched nodes & allocate functional units
20: end while
```

for each operation type, the control step with the largest number of operations of that type is found. This gives a lower bound on the possible resource constraint. These operations are selected to make up one set of vertices (or nodes), U, in the bipartite graph. The second set of vertices, V, includes all of the other nodes. See Figure 3.1.

During functional unit binding, the nodes of the graph are each considered an allocated functional unit. Initially, as none of the operations have been bound to functional units, every operation is considered to be bound to its own functional unit, and each is represented by an individual node of the graph. On subsequent iterations, each node (functional unit) of the graph may contain more than one operation. Edges (making up the set E) are created between compatible nodes in the graph. Two nodes are compatible if they meet the following two criteria:

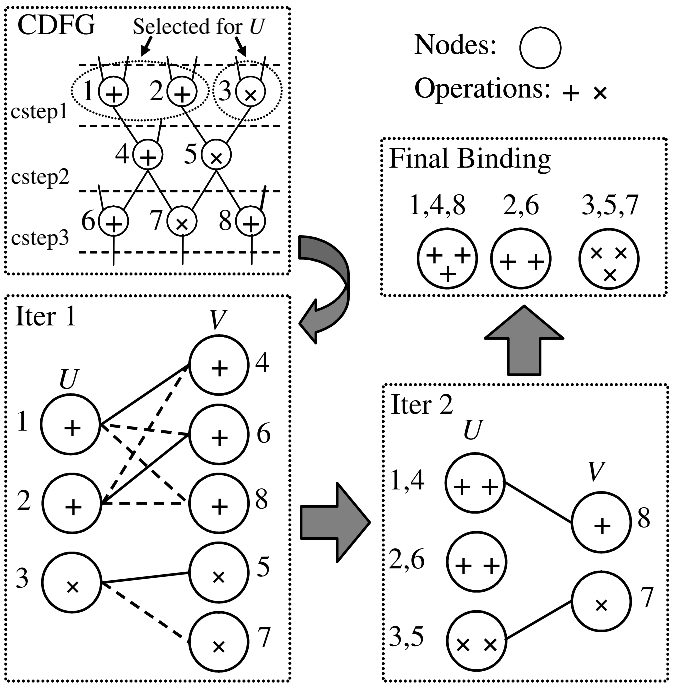


Figure 3.1: An example of the binding algorithm formulated as a bipartite graph. Solid edges indicate matches selected. The final functional unit allocation is 2 adders and 1 multiplier.

1. They perform the same type of operation; e.g., both are multiplications, or one is an addition and the other is a subtraction (and can thus both be performed by the same ALU).

2. They do not contain any operations that have overlapping lifetimes in the schedule.

Each edge of the graph represents a possible binding of two sets of operations to the same functional unit. Edge weights are then assigned as described in the next section. This formulation is similar to binding that works with a compatibility graph, but not all nodes will be bound in a single iteration. As the graph will be solved for the maximum weight, larger edge weights should be assigned to those edges that would produce lower power consumption, if the two sets of operations were bound to the same functional unit.

Figure 3.1 illustrates the bipartite graph formulation. In iteration 1, add operations 1 and 2, and mult operation 3 are selected for set U, because they come from the control steps of maximum density for their respective types in the scheduled CDFG. (Note that, alternatively, any of the mult operations could have been chosen, or add operations 6 and 8 could have been chosen.) Solid edges represent those selected in the maximum weighted matching. Nodes are combined, and in iteration 2 nodes are further combined. In iteration 3, there is no longer any compatibility between the nodes, and the algorithm is completed. The final allocation is 2 adders and 1 multiplier.

Theorem 1. *A weighted bipartite graph $G = (U, V, E)$, representing the operations of a scheduled CDFG (as previously described), if iteratively generated, solved, and matching nodes are combined, guarantees that the minimum possible resource constraints can be met.*

Proof. Suppose on the contrary that the minimum resource constraint cannot be met. This would mean that there exists a node in the set V that is incompatible with all nodes in the set U. Since this incompatibility could not be due to compatibility criterion 1 given above—the nonexistence of a compatible operation type (if, for example, there was only one operation of a particular type in a CDFG, then it would already lie in set U)—it must be due to criterion 2—operations that have overlapping lifetimes. That would imply that there were more operations in the incompatible operation's control step than were in the control step chosen initially for set U. This could not be the case due to the selection criteria for set U. □

Theorem 1 guarantees that, in the bipartite graph formulation, the minimum resource constraint for the given scheduled CDFG can be met. The worst case runtime complexity of the algorithm is $O(|N|^2 * (|E| + |N|log|N|))$, where

$|N|$ is the total number of nodes in the CDFG. This is because the number of bipartite graphs solved is, in the worst case, linear with the number of nodes in the CDFG.

3.4.2.2 Edge weight calculation

For each of the edges in the graph, edge weights are calculated as follows:

1. The sizes of the input multiplexers to the functional unit to which the operations connected by the edge would be bound (if the matching included the given edge) are found. This is possible because the registers have already been assigned, enabling the calculation of the exact multiplexer sizes. This virtual binding creates a partial datapath.

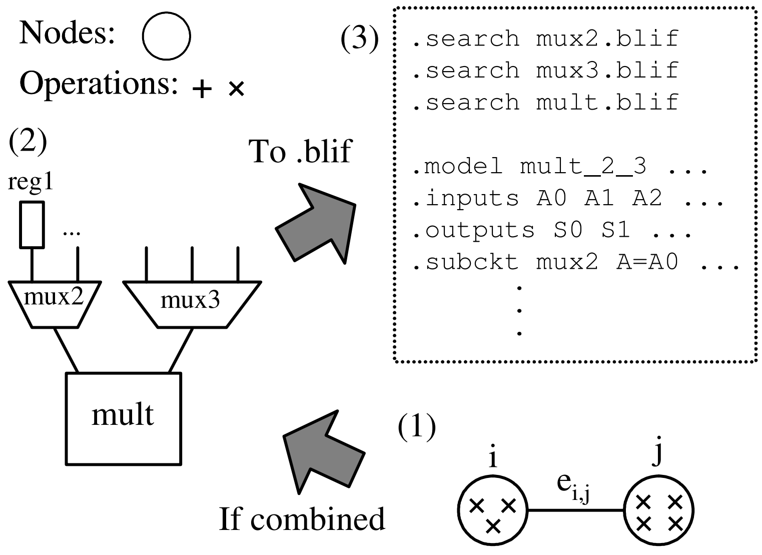
2. A gate-level netlist of the partial datapath (including the functional unit and multiplexers) is generated in .blif format [55]. This is accomplished by creating a new .blif file with proper input and output ports, importing existing instantiations of the multiplexers and functional units, and making the necessary connections. See Figure 3.2.

3. The switching activity is estimated for the gate-level netlist (.blif), based on the technique described in Section 4. This produces an estimate of the dynamic power, including glitch power, which will be used to estimate part of the cost of this particular binding of operations to functional unit.

4. The weight on the edge is computed according to the formula

$$w(e_{i,j}) = \alpha \times \frac{1}{SA} + (1 - \alpha) \times \frac{1}{(muxDiff + 1) \times \beta} \tag{3.4}$$

   where SA is the *total estimated switching activity* as defined in Equation (3.3), α is a weighting coefficient, β is a value used to adjust the size



```
.search mux2.blif
.search mux3.blif
.search mult.blif

.model mult_2_3 ...
.inputs A0 A1 A2 ...
.outputs S0 S1 ...
.subckt mux2 A=A0 ...
        .
        .
        .
```

Figure 3.2: An example of gate-level partial data-path netlist generation. Based on the register binding, and operations assigned to the edge nodes (1), it is determined that a 2-input multiplexer is needed on the left, and a 3-input multiplexer on the right input of the multiplier (2). The .blif netlist is then generated (3) for mapping and switching activity estimation.

of the $muxDiff$ factor relative to SA, and $muxDiff$ is defined as the absolute difference in the sizes of the two multiplexers that input to the functional unit.

Equation (3.4) uses the weighting coefficient α to balance the contribution to the weight of two important factors: the *total estimated switching activity*, or SA, and the difference in size between the two multiplexers, or $muxDiff$.

SA provides a *low-level* consideration of the circuit. It explicitly estimates dynamic power usage (including glitches) at the gate-level, taking into account the multiplexers in the partial data-path. It also implicitly considers area through the number of lookup tables required to implement the partial data-path, because a larger area correlates with a higher SA. The term $muxDiff$, on the other hand, provides a *high-level* consideration of the circuit. It explicitly considers multiplexer balancing, which would have a direct impact on glitch reduction, even if the SA estimation is not 100% accurate. This combination of *high-level* multiplexer balancing and *low-level* SA estimation works to select the best matches for power and area reduction in each iteration of the binding algorithm. In our experiments we weight these two factors equally.

As dynamic calculation of the switching activities for each edge during the binding iterations can be time-consuming, in our experiments we precalculate the switching activities for all combinations of multiplexers and functional units. This is done by generating the gate-level netlists for the partial data-path of each combination of functional unit and multiplexers, and running the SA estimation on each. The calculated SA values are then stored in a text file. A hash table is then generated when HLPower is initially run by reading in the precalculated values from the text file. This allows fast lookup of the estimated SA value for a particular combination of input multiplexer sizes, and functional unit. Experimental results show that this method provided us with the same results as running the algorithm with dynamic SA estimation, but with a much shorter run time.

The iterative approach to functional unit binding allows the multiplexer size to be better controlled than is possible with single iteration approaches, such as with a network flow algorithm. By iteratively building up the numbers of operations assigned to allocated functional units, the multiplexer sizes, balance among multiplexers, and the contributions of the multiplexers to dynamic power (including glitch power) consumption can be carefully controlled and evaluated.

Table 3.1: Resource Constraints, Scheduling Length, and Number of Registers Used for Both LOPASS and HLPower Binding. Identical Schedules and Register Bindings were Used by Both LOPASS and HLPower.

Benchmarks	Add	Mult	Cycle	Reg	HLPower Runtime (s)
chem	9	7	39	70	812
dir	3	2	41	25	56
honda	4	4	18	13	14
mcm	4	2	27	54	16
pr	2	2	16	32	2
steam	7	6	28	39	189
wang	2	2	18	39	2

3.5 Experimental Results

3.5.1 Experimental setup

Our experiments are carried out on a 2.8 GHz Intel Pentium 4 Linux machine, with 2 GB of memory. A number of data-intensive benchmarks are used; the same benchmarks are used in Section 2.5 and outlined in Table 2.1. Each node in the benchmarks is either an addition/subtraction or a multiplication.

We compare our binding algorithm, HLPower, to a state-of-the-art low-power high-level synthesis algorithm for FPGAs, LOPASS [38], which can perform simultaneous scheduling, allocation, and binding. We use a resource library containing single-cycle resources, including a multiplier, an adder, a register, and multiplexers. The benchmarks are first run through LOPASS, and a binding solution is obtained. Then they are run through HLPower with the same schedule, register allocation, and resource constraints, to obtain the HLPower binding solution. Table 3.1 summarizes the scheduled benchmark characteristics used for both the LOPASS and the HLPower solutions.

The binding solutions, in CDFG format, are then converted to RTL design in VHDL with a CDFG to VHDL tool. To verify our results using a commercial tool, the designs are put into Quartus II for RTL synthesis, placement and routing, timing analysis, simulation, and power analysis. This is done by first building a project on each benchmark's VHDL, setting the device family to Cyclone II, and selecting the same device for each benchmark. We use the Quartus II vector waveform file (.vwf) editor to generate 1000 random input vectors for each benchmark. We also set the simulator settings *glitch filtering* to *never*, *max balancing dsp blocks* to *0*, *wysiwyg remap* to *on*, *optimization technique* to *speed*, and *synthesis effort* to *fast*. These settings help to ensure that the benchmarks for both LOPASS and HLPower are synthesized in the same way without Quartus II optimizations that would invalidate the power results produced by both algorithms. The same .vwf file is used for both LOPASS and HLPower. Then we run the command *quartus_sh –flow compile* (which runs the synthesis, placement and routing, and timing analysis), *quartus_sim* (which makes use of the .vwf file, and generates a switching activity file, .saf), and *quartus_pow* (which makes use of the .saf file). The command *quartus_pow* runs PowerPlay Power Analyzer, and reports the dynamic power consumption.

3.5.2 FPGA area and power reduction results

Tables 3.2 and 3.3 summarize the synthesis and power analysis results for both LOPASS and HLPower, for each of the benchmarks. There was an average reduction in dynamic power of 19.3%, and area (in the form of look-up tables (LUTs)) of 9.1%. These reductions came at the expense of 0.6% of the clock period, on average.

Table 3.2 columns 5 and 6, and Table 3.3 columns 5 and 6 show the multiplexer reduction results. In the tables, *Largest MUX* is the largest multiplexer needed to implement the binding solution, while *MUX length* is a measure of

Table 3.2: Power, Clock Period, Number of LUTs, and Multiplexer Results for LOPASS and HLPower Bindings.

	LOPASS/HLPower				
Benchmarks	Dynamic Power (mW)	Clk Per. (ns)	LUTs	Largest MUX	MUX Length
chem	1602.3/1468.6	26.0/27.5	9,806/9,613	26/23	672/637
dir	709.1/405.8	23.8/24.2	4,527/3,453	18/15	167/157
honda	658.7/534.1	23.5/23.2	3,352/3,057	15/13	165/162
mcm	351.3/208.7	24.1/24.2	3,274/2,548	17/14	159/153
pr	232.7/192.9	20.9/21.7	1,714/1,732	11/8	70/57
steam	729.6/690.6	24.4/23.6	5,121/4,469	19/22	429/321
wang	161.5/158.5	20.5/19.9	1,697/1,775	12/8	69/76

Table 3.3: Power, Clock Period, Number of LUTs, and Multiplexer Results for LOPASS and HLPower Bindings, Continued.

	Change				
Benchmarks	Dynamic Pow.(%)	Clk Per. (%)	LUTs (%)	Lrgst MUX	MUX Len.(%)
chem	-8.35	5.67	-1.97	-6	-5.2
dir	-42.78	2.04	-23.72	-3	-6.0
honda	-18.92	-1.40	-8.80	-2	-1.8
mcm	-40.60	0.38	-22.17	-3	-3.8
pr	-17.09	3.60	1.05	-3	-18.6
steam	-5.35	-3.32	-12.73	3	-25.2
wang	-1.85	-2.88	4.60	-4	10.1
Average	-19.28	0.58	-9.11	-2.6	-7.2

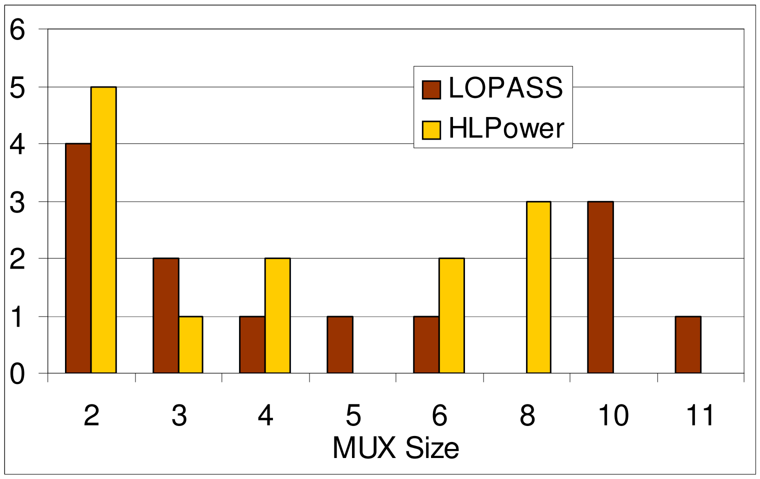


Figure 3.3: Number of multiplexers of each size allocated by LOPASS and HLPower bindings, for benchmark *pr*.

the total number of multiplexers implemented, and is calculated by adding up the total number of multiplexer inputs (sizes).

HLPower reduced the largest multiplexer size by an average of 2.6, and the length by an average of 6.1%, over LOPASS. Figures 3.3 and 3.4 show the number of each size of multiplexer for *pr* and *wang*. In these examples we can see not only that HLPower creates fewer multiplexers, and multiplexers of smaller sizes (reducing area and power), but also that the multiplexers are more balanced, with fewer single multiplexers of a particular size. This balance of multiplexers contributes to a power reduction by balancing paths, reducing interconnect, and eliminating extra glitch transitions.

Evidence for the decrease in glitches is given in Figure 3.5. Average toggle rate is a number reported by Quartus II that provides a measure of the total switching activity of the circuit. HLPower reduces the toggle rate for each benchmark, averaging 21.9% overall. The combination of area savings through multiplexer reduction and reduced glitching (as evidenced by the significant decrease in toggle rate) produces an aggregate reduction in dynamic power, for a negligible change in clock period.

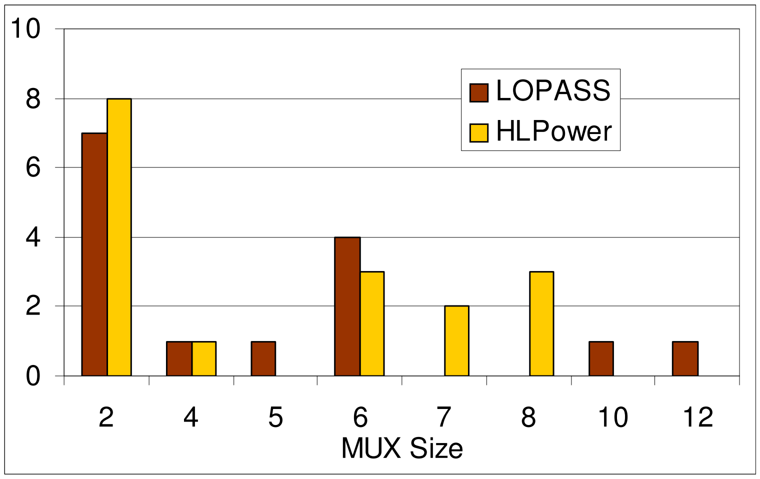


Figure 3.4: Number of multiplexers of each size allocated by LOPASS and HLPower bindings, for benchmark *wang*.

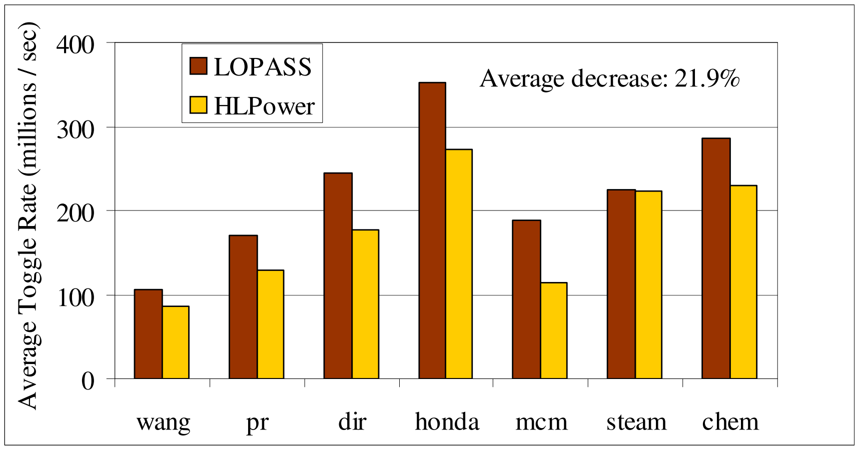


Figure 3.5: Average toggle rate as reported by Quartus II for LOPASS and HLPower bindings.

CHAPTER 4

CONCLUSION

4.1 Summary and Conclusion

In this thesis we have presented two novel high-level binding algorithms. The first, named FastYield, was a variation-aware algorithm for simultaneous binding and module selection. FastYield incorporated many competing factors into its algorithm that are not found in previous variation-aware algorithms. It considered register, multiplexer, and functional unit usage, as well as spatial correlation among the resources during SSTA embedded in a floorplanner. The importance of spatial correlation during SSTA was demonstrated. On average, FastYield achieved an 85% performance yield clock period that is 14.5% smaller, and a performance yield gain of 78.9%, when compared to a variation-unaware and layout-unaware algorithm based on [18]. Also, by making use of accurate timing information, FastYield's rebinding improved performance yield by an average of 9.8% over the initial binding, for the same clock period. These results showed that by performing statistical layout-driven synthesis, substantial gains in performance yield can be made.

The second algorithm, dubbed HLPower, was a high-level binding algorithm for power and area reduction, targeting FPGAs. The binding algorithm was based on iterative weighted bipartite matching, and made use of glitch power-aware, dynamic power estimation. The power estimations came from an FPGA technology mapper that made use of an effective switching activity model.

Experimental results were obtained that compared HLPower to LOPASS, a state-of-the-art low-power high-level synthesis algorithm for FPGAs. High-level binding results were converted to VHDL, and synthesized with Altera's Quartus II software, targeting the Cyclone II FPGA architecture. Power characteristics were evaluated with the Altera PowerPlay Power Analyzer. The results showed that our algorithm, on average, reduces toggle rate by 22% and area by 9%, resulting in a decrease in dynamic power consumption of 19%.

4.2 Future Work

In the algorithms presented, we have focused mainly on binding and module selection for performance yield optimization and low power. Although we have successfully improved binding results over previous algorithms, the next step will be to integrate these ideas into a whole high-level synthesis framework, including the scheduling of resources. Additionally, simultaneous binding of registers and functional units should be considered within that framework. This is an important area of research, as it has the potential to significantly improve the characteristics of circuits designed both as ASICs and as FPGAs, while also reducing the time required to complete those designs. Tools that are variation-aware will also enable the continued shrinking of device sizes in next-generation processes, providing greater resources and potential on each chip.

REFERENCES

[1] G. De Micheli, *Synthesis and Optimization of Digital Circuits.* New York, NY: McGraw Hill, Inc., 1994.

[2] K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: An ASIC and system vendor perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1507–1522, Dec. 2000.

[3] D. Gajski, N. Dutt, and A. Wu, *High-Level Synthesis: Introduction to Chip and System Design.* New York, NY: Springer, 1992.

[4] A. Raghunathan, N. K. Jha, and S. Dey, *High-Level Power Analysis and Optimization.* Norwell, MA: Kluwer Academic Publishers, 1998.

[5] R. Camposano and W. Wolf, *High-Level VLSI Synthesis.* New York, NY: Springer-Verlag, 2001.

[6] B. Pangrle, "On the complexity of connectivity binding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 11, pp. 1460–1465, Nov. 1991.

[7] K. Srinivasan and K. S. Chatha, "Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints," *Integration, The VLSI Journal*, vol. 40, no. 3, pp. 326–354, 2007.

[8] O. S. Unsal, J. W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzlez, and O. Ergin, "Impact of parameter variations on circuits and microarchitecture," *IEEE Micro*, vol. 26, no. 6, pp. 30–39, 2006.

[9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the 2003 40th Annual Conference on Design Automation*, 2003, pp. 338–342.

[10] S. R. Nassif, "Statistical design on the verge of maturity: Revisiting the foundation," presented at the 2009 IEEE/ACM Asia South Pacific Design Automation Conference Tutorials, pp. 264–311, 2009.

[11] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, 2006, pp. 21–30.

[12] Altera Corporation, "Stratix II vs. Virtex-4 power comparison & estimation accuracy white paper," Aug. 2005. [Online]. Available: http://www.altera.com/literature/wp/wp_s2v4_pwr_acc.pdf.

[13] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *Proceedings of the 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays*, 2003, pp. 175–184.

[14] A. Raghunathan, S. Dey, and N. Jha, "Register transfer level power optimization with emphasis on glitch analysis and reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 8, pp. 1114–1131, Aug. 1999.

[15] M. R. Guthaus, N. Venkateswarant, C. Visweswariaht, and V. Zolotov, "Gate sizing using incremental parameterized statistical timing analysis," in *Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 1029–1036.

[16] I.-J. Lin, T.-Y. Ling, and Y.-W. Chang, "Statistical circuit optimization considering device and interconnect process variations," in *Proceedings of the 2007 International Workshop on System Level Interconnect Prediction*, 2007, pp. 47–54.

[17] Y.-M. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning," in *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design*, 1994, pp. 317–321.

[18] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu, "Data path allocation based on bipartite weighted matching," in *Proceedings of the 1990 ACM/IEEE 27th Annual Conference on Design Automation*, 1990, pp. 499–504.

[19] W.-L. Hung, X. Wu, and Y. Xie, "Guaranteeing performance yield in high-level synthesis," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 303–309.

[20] J. Jung and T. Kim, "Timing variation-aware high-level synthesis," in *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 424–428.

[21] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos, "Modeling within-die spatial correlation effects for process-design co-optimization," in *Proceedings of the 2005 6th International Symposium on Quality of Electronic Design*, 2005, pp. 516–521.

[22] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," in *Proceedings of the 2006 International Symposium on Physical Design*, 2006, pp. 2–9.

[23] F. Wang, G. Sun, and Y. Xie, "A variation aware high level synthesis framework," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2008, pp. 1063–1068.

[24] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *Proceedings of the 2004 IEEE/ACM Asia South Pacific Design Automation Conference*, 2004, pp. 68–73.

[25] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 643–663, 2002.

[26] E. Kusse and J. Rabaey, "Low-energy embedded FPGA structures," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, 1998, pp. 155–160.

[27] J. Lamoureux and S. Wilton, "Activity estimation for field-programmable gate arrays," in *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.

[28] S. Chin, C. Lee, and S. Wilton, "Power implications of implementing logic using FPGA embedded memory arrays," in *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.

[29] J. Lamoureux, G. G. Lemieux, and S. J. E. Wilton, "GlitchLess: An active glitch minimization technique for FPGAs," in *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, 2007, pp. 156–165.

[30] J.-M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proceedings of the 1995 32nd Annual Conference on Design Automation*, 1995, pp. 29–35.

[31] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," in *Proceedings of the 1995 International Symposium on Low Power Design*, 1995, pp. 69–74.

[32] A. Davoodi and A. Srivastava, "Effective techniques for the generalized low-power binding problem," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 52–69, 2006.

[33] E. Kursun, A. Srivastava, S. O. Memik, and M. Sarrafzadeh, "Early evaluation techniques for low power binding," in *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, 2002, pp. 160–165.

[34] F. Wolff, M. Knieser, D. Weyer, and C. Papachristou, "High-level low power FPGA design methodology," in *Proceedings of the 2000 IEEE National Aerospace and Electronics Conference*, 2000, pp. 554–559.

[35] D. Chen, J. Cong, Y. Fan, and Z. Zhang, "High-level power estimation and low-power design space exploration for FPGAs," in *Proceedings of the 2007 IEEE/ACM Asia South Pacific Design Automation Conference*, 2007, pp. 529–534.

[36] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "xPilot: A platform-based behavioral synthesis system," presented at the 2005 SRC Techcon Conference, Oct. 2005.

[37] J. Cong and W. Jiang, "Pattern-based behavior synthesis for FPGA resource reduction," in *Proceedings of the 2008 ACM/SIGDA 16th International Symposium on Field Programmable Gate Arrays*, 2008, pp. 107–116.

[38] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architectures," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003, pp. 134–139.

[39] D. Chen, J. Cong, Y. Fan, and L. Wan, "LOPASS: A low-power architectural synthesis system for FPGAs with interconnect estimation and optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, to be published.

[40] L. Cheng, D. Chen, and M. D. F. Wong, "GlitchMap: An FPGA technology mapper for low power considering glitches," in *Proceedings of the 2007 44th Annual Conference on Design Automation*, 2007, pp. 318–323.

[41] S. Amellal and B. Kaminska, "Scheduling of a control data flow graph," in *Proceedings of the 1993 IEEE International Symposium on Circuits and Systems*, 1993, pp. 1666–1669.

[42] "OSU design flows for MOSIS SCMOS_SUBM design flow, FreePDK 45nm variation-aware design flow," 2008. [Online]. Available: http://vcag.ecen.okstate.edu/projects/scells/.

[43] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*, 2003, pp. 621–625.

[44] S. Adya and I. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[45] H. Bakoglu and J. Meindl, "Optimal interconnection circuits for VLSI," *IEEE Transactions on Electron Devices*, vol. 32, no. 5, pp. 903–909, May 1985.

[46] N. NS, T. Bonifield, A. Singh, C. Bittlestone, U. Narasimha, V. Le, and A. Hill, "BEOL variability and impact on RC extraction," in *Proceedings of the 2005 42nd Annual Conference on Design Automation*, 2005, pp. 758–759.

[47] S. Nassif, "Design for variability in DSM technologies [deep submicron technologies]," in *Proceedings of the 2000 IEEE 1st International Symposium on Quality Electronic Design*, 2000, pp. 451–454.

[48] M. Srivastava and M. Potkonjak, "Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 1, pp. 2–19, March 1995.

[49] Altera Corporation, "Cyclone II device handbook," 2008. [Online]. Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf.

[50] Altera Corporation, "PowerPlay power analysis," Nov. 2005. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii53013.pdf.

[51] F. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 310–323, Feb. 1993.

[52] T.-L. Chou and K. Roy, "Estimation of activity for static and domino CMOS circuits considering signal correlations and simultaneous switching," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 10, pp. 1257–1265, Oct. 1996.

[53] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proceedings of the 1999 ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays*, 1999, pp. 29–35.

[54] B. Krishnamurthy and I. Tollis, "Improved techniques for estimating signal probabilities," *IEEE Transactions on Computers*, vol. 38, no. 7, pp. 1041–1045, July 1989.

[55] E. Sentovich et al., “SIS: A system for sequential circuit synthesis,” UCB/ERL Memorandum M89/49, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Tech. Rep., Nov. 1992.