Guarded Evaluation: An Algorithm for Dynamic Power Reduction in FPGAs

by

Chirag Ravishankar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Guarded evaluation is a power reduction technique that involves identifying sub-circuits (within a larger circuit) whose inputs can be held constant (guarded) at specific times during circuit operation, thereby reducing switching activity and lowering dynamic power. The concept is rooted in the property that under certain conditions, some signals within digital designs are not "observable" at design outputs, making the circuitry that generates such signals a candidate for guarding.

Guarded evaluation has been demonstrated successfully for custom ASICs; in this work, we apply the technique to FPGAs. In ASICs, guarded evaluation entails adding additional hardware to the design, increasing silicon area and cost. Here, we apply the technique in a way that imposes minimal area overhead by leveraging existing unused circuitry within the FPGA. The LUT functionality is modified to incorporate the guards and reduce toggle rates.

The primary challenge in guarded evaluation is in determining the specific conditions under which a sub-circuit's inputs can be held constant without impacting the larger circuit's functional correctness. We propose a simple solution to this problem based on discovering gating inputs using"non-inverting paths" and trimming inputs using "partial non-inverting paths" in the circuit's AND-inverter graph representation.

Experimental results show that guarded evaluation can reduce switching activity by as much as 32% for FPGAs with 6-LUT architectures and 25% for 4-LUT architectures, on average, and can reduce power consumption in the FPGA interconnect by 29% for 6-LUTs and 27% for 4-LUTs. A clustered architecture with four LUTs to a cluster and ten LUTs to a cluster produced the best power reduction results.

We implement guarded evaluation at various stages of the FPGA CAD flow and analyze

the reductions. We implement the algorithm as post technology mapping, post packing and post placement optimizations. Guarded Evaluation as a post technology mapping algorithm inserted the most number of guards and hence achieved the highest activity and interconnect reduction. However, guarding signals come with a cost of increased fanout and stress on routing resources. Packing and placement provides the algorithm with additional information of the circuit which is leveraged to insert high quality guards with minimal impact on routing. Experimental results show that post-packing and post-placement methods have comparable reductions to post-mapping with considerably lesser impact on the critical path delay and routability of the circuit.

Acknowledgements

I am eternally grateful to Dr. Andrew Kennings, who has been more than a supervisor, mentor and friend. This work would not have existed without his guidance and encouragement. I would like to thank Dr. Jason Anderson for his technical contributions to the work and introducing me to the world of FPGA CAD research. My heartfelt thanks also extend to Dr. Siddharth Garg and Dr. Hiren Patel for their support and encouragement throughout my time at the University of Waterloo. I feel incredibly fortunate to have the privilege of interacting with such great people. I simply would not be the person that I am today without their unflinching support.

I would like to thank Dr. Alakananda Nath for giving me the gift of music and nurturing my creative and spiritual side, which kept me sane and motivated through the difficult periods of my degree.

I would like to acknowledge the people who have had a significant impact on my life: I am forever indebted to my family for their unconditional love and support; Dr. Parham Aarabi, for introducing me to Electrical and Computer Engineering at the University of Toronto; Dr. Tarek Abdelrahman for his inspirational lectures and guidance when I needed them the most; Dr. Jonathan Rose for his support and inspiration; and Mirel Giugaru for the long talks that helped me gain a better sense of self.

Lastly, I want to specially acknowledge my aunt, Dr. Veena Shekar who always supported and encouraged me. I would like to thank her for always being there, nurturing my creative side, and making me aware of the world. She will always be missed.

Dedication

Dedicated to my supporting and loving family.
My parents, Ravi and Chandrika; sister, Nidhi;
grandparents, Leela and Chandrashekar; aunt, Veena and uncle, Indushekar

Mooshika Vahana, Modhaka Hastha, Chaamara Karna, Vilambhita Sutra
Vaamana Rupa, Maheshwara Putra, Vigna Vinashaka Paada Namaste,
Paada Namaste, Paada Namaste

Om Gam Ganapathaye Namaha

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Field Programmable Gate Arrays (FPGAs) are reconfigurable devices that can implement a digital circuit. It can be programmed by the end-user using high-level hardware description languages such as VHDL or Verilog. They have often been compared to Application Specific Integrated Circuits (ASICs), where the user designs and implements the circuit on a chip that is then fabricated. FPGAs offer quicker time-to-market as an FPGA can be configured in the order of minutes to hours, while ASIC designs take the order of months to reach the market. Furthermore, FPGAs are more cost effective for low volume applications where the cost to produce custom masks for ASICs is not economically viable. Modern FPGAs are widely used in diverse applications, ranging from communications infrastructure, automotive, to industrial electronics.

However, their use in the mainstream market is often elusive due to their high power consumption. Programmability in FPGAs is achieved through higher transistor counts and

larger capacitances, leading to considerably more leakage and dynamic power dissipation compared to ASICs for implementing a given function [24].

Recent years have seen intensive research activity on reducing FPGA power through innovations in CAD, architecture, and circuits. Circuits at the Register-transfer level (RTL), synthesized from HDL, is transformed into the bitstream used to program the FPGA through a number of steps in the CAD flow, which is illustrated in figure 1.1:

1. **Logic Synthesis**: The HDL is elaborated and synthesized into an optimized netlist.
2. **Technology Mapping**: The netlist is mapped to the target FPGA architecture. FPGA architectures consist of look-up tables [17] which are able to implement any K-input logic function.
3. **Packing**: Mapped LUTs are packed into clusters based on the target FPGA architecture.
4. **Placement**: The packed logic blocks (LBs) are physically placed on the device.
5. **Routing**: Necessary connections are made between LBs using physical wires.

We propose an algorithm to reduce FPGA dynamic power consumption. We mainly target the technology mapping stage of the CAD flow using an approach known as *Guarded Evaluation*, which has been used successfully in the custom ASIC domain [48]. We also implement this algorithm in the packing and placement stages to make comparisons and fully explore the benefits of the algorithm. Recall that dynamic power in a CMOS circuit is defined by:

$$P_{avg} = \frac{1}{2} \sum_{i \in nets} C_i \cdot f_i \cdot V^2 \tag{1.1}$$

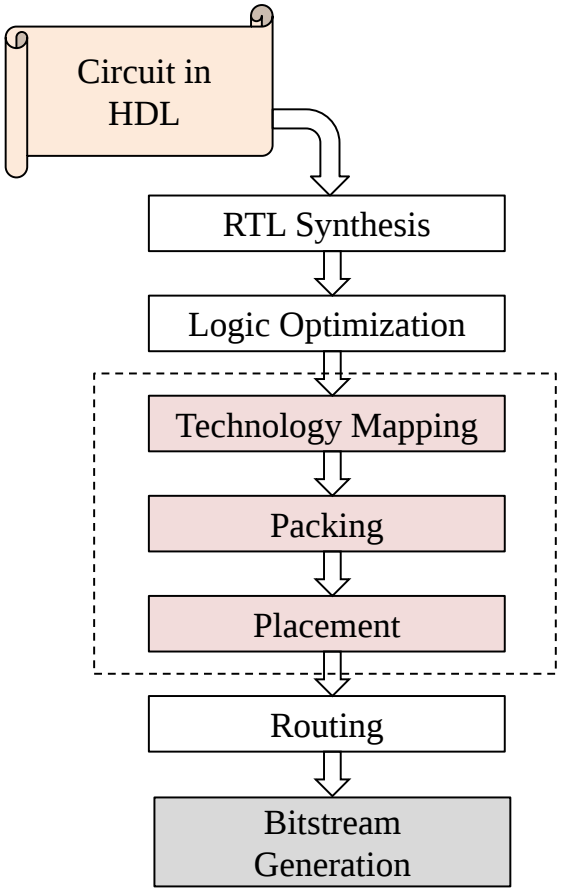


Figure 1.1: Traditional FPGA CAD Flow

where C_i is the capacitance of a net i; f_i is the toggle rate of net i, also known as net i's *switching activity*; V is the voltage supply. In this work, we minimize the *switching activity* of each net to reduce dynamic power consumption.

1.2 Summary of Contributions

A summary of contributions of this thesis are as follows:

1. A power-aware technology mapping cost function that is able to achieve dynamic power reductions over the state-of-the-art [21].
2. An observation in the structure of the netlist (converted to an And-Inverter graph) that allows for fast computation of *observability don't cares*.
3. A post mapping optimization algorithm, called Guarded Evaluation, proposed and implemented on a well known open source synthesis tool [1].
4. Guarded Evaluation extended to be implemented as post-packing and post-placement optimizations.
5. Evaluation and Analysis of the algorithm on a power-aware FPGA CAD flow based on a power model proposed by [42]

1.3 Algorithm Overview

Guarded evaluation seeks to reduce net switching activities by modifying the circuit network. In particular, the approach taken is to eliminate toggles on certain internal signals of a circuit when such toggles are guaranteed to not propagate to overall circuit outputs. This reduces switching activity on logic signals within the interconnection fabric. Prior work has shown that interconnect comprises 60% of an FPGA's dynamic power [45], due primarily to long metal wire segments and the parasitic capacitance of used and unused programmable routing switches.

Guarded evaluation comprises first identifying an internal signal whose value does not propagate to circuit outputs under certain conditions. A straightforward example is an `AND` gate with two input signals, A and B. Values on signal A do not propagate to circuit outputs when B is logic-`0` (the condition). Thus, toggles on A are an unnecessary waste of power when B is logic-`0`. Having found a signal and condition, guarded evaluation

then modifies the circuit to eliminate the toggles on the signal when the condition is true. Returning to the example, the inputs to the circuitry that produce A can be held at a constant value (guarded) when the condition is true, reducing dynamic power. The computationally difficult aspect of the process is in finding signals (such as A) and computing the conditions under which they are not observable, as these steps depend on an analysis of the circuit's logic functionality.

In this work, we propose several techniques which make guarded evaluation appropriate for FPGAs. We modify the technology mapping stage of the FPGA CAD flow to produce mappings with opportunities for guarded evaluation. After mapping, we modify the LUT configurations (logic functions) and alter network connectivity to incorporate guards, reducing switching activity of nets. We also implement guarding after the packing step as well as after placement to reduce interconnect power. We further analyze the algorithm by implementing guarded evaluation on a combination of steps in the CAD flow, namely post-mapping and post-placement. Unlike guarded evaluation in ASICs, which involves adding additional circuitry (increasing area and cost), our approach uses unused circuitry that is already available in the FPGA fabric, making it less expensive from the area perspective. Specifically, input pins on LUTs are frequently not fully utilized in modern designs (only about 39% for 6-LUT architectures [21]), and we use the available free inputs on LUTs for guarded evaluation. This implies that we do not add any additional LUTs to implement guarding, but rather only add a minimal amount of extra connections into the network. In our approach, identifying the conditions under which a given signal can be guarded is accomplished by analyzing properties of the logic synthesis network, which is an And-Inverter Graph (AIG). In particular, we show that the presence of "non-inverting" and "partial non-inverting" paths in the AIG can be used to drive the discovery of guarding opportunities. This structural approach to determining guarding opportunities proves to be very efficient.

Finally, we consider the introduction of different types of guarding logic (as opposed to transparent latches which are used for ASICs) where we force the guarded signals to logic-1 and logic-0 state based on static probability to reduce unnecessary transient switching.

Finally, we consider a variety of different FPGA logic block architectures. In particular, we examine architectures with 4- and 6-input LUTs, as well as architectures with different numbers of LUTs per logic block. Results show that the benefit of guarding on power reduction depends strongly on the underlying architecture of the target FPGA's logic. Also, guarded evaluation implemented after later stages of the CAD flow (i.e. packing and placement) provides more feedback to the algorithm which can be used to insert high quality guards with minimal impact to routing resources.

1.4 Thesis Organization

Chapter 2 presents background and related work on technology mapping for FPGAs, power optimization, and describes guarded evaluation in the ASIC context. The proposed algorithm is described in Chapter 3. Experimental results with various FPGA LUT and cluster architectures appears in Chapter 4. Conclusions and suggestions for future work are offered in Chapter 5.

Chapter 2

Background

This section provides some necessary background on FPGA architecture and technology mapping. We summarize some of the recent works in FPGA dynamic power reduction through CAD. This section also describes some related work on guarded evaluation in both the ASIC and FPGA context. Finally, structural observations in And-Inverter subgraphs, necessary for this work, is described.

2.1 FPGA Architecture

Field-programmable Gate Arrays consist of programmable logic and storage elements that are connected by routing fabric that is also re-configurable. This massively programmable architecture is what allows the chip to be programmed by the end user to implement virtually *any* digital circuit. Figure 2.1 gives an abstract overview of the *island style* FPGA architecture, where logic blocks resemble isolated islands in a sea of interconnection wires. This is the most common type of architecture in commercial FPGAs.

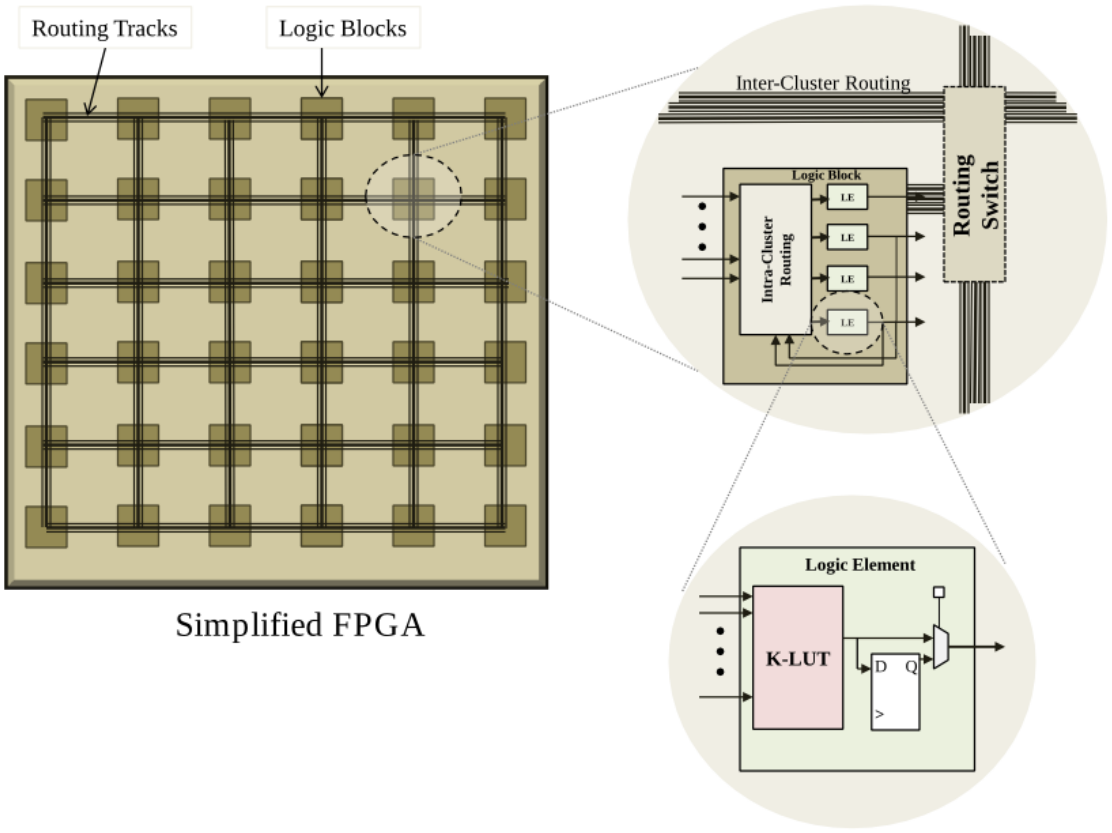


Figure 2.1: Overview of FPGA Architecture

Some top-of-the-line modern FPGAs have upto 2 million logic blocks, about 1000 I/O pins, memory blocks, and other blocks with special features such as DSP, Analog-to-digital converter, PCIe, etc [52]. The programmability is achieved through configuration memory based on static RAM cells. The FPGA is essentially programmed with a serial bitstream that controls every configurable routing track and logic cell. Other technologies exist to implement configuration memory such as antifuses [15] and floating-gate transistors [10]. The focus of this work, however is on SRAM-based FPGAs since they are more prevalent

in the FPGA market.

As shown in figure 2.1, each logic block contains several logic elements that are packed together and routed using local interconnect, which in this thesis is referred to as *intra-cluster* routing. The output of each logic element can also span to logic elements in different logic blocks. In these cases, routing outside the logic block is used which has higher capacitance and impact on performance. In this thesis, this interconnect is referred to as *inter-cluster* routing. The main objective of this work is to minimize the switching frequency in these wires (both intra-cluster and inter-cluster) to reduce dynamic power dissipation.

The basic logic element, shown in figure 2.1 is usually a Look-up Table and a Flip Flop along with its own local interconnect. There are more complex logic element structures such as fracturable LUTs and adaptive logic modules [19]. In this work, we assume a basic logic element that contains a K-input look-up table. Guarded Evaluation primarily is a post-technology mapping algorithm (post-packing and post-placement are also considered) on a traditional K-LUT.

A K-input look-up table has the ability to implement *any* K-input logic function. This flexibility is achieved using 2^K configuration memory based on SRAM cells. The circuit of the LUT is essentially a tree of multiplexers where the select lines are tied to the inputs of the LUT, and each selection corresponds to a programmable SRAM cell. A 3-input LUT is illustrated in figure 2.2. There are 2^3 SRAM cells, where any 3-input function can be implemented by configuring the value in the cells.

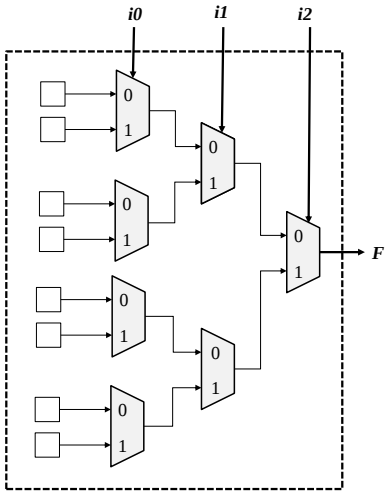


Figure 2.2: 3-input Look-up Table

2.2 FPGA Technology Mapping

FPGA Technology mapping is an important step in the CAD flow where the netlist is mapped to the target architecture, which in this work is a K-LUT. The approach used by modern FPGA technology mappers are based on finding cuts in Boolean networks [44, 13]. The first step is to represent the combinational portion of a circuit as a directed acyclic graph, $G(V, E)$. Each node in G represents a logic function, and edges between nodes represent dependencies among logic functions. Before mapping commences, the number of inputs to each node must be less than the number of inputs of the target look-up-table (K). A primary input node is a node with an in-degree of 0; a primary output node has an out-degree of 0. For a node $z \in V$ in the graph, let $Inputs(z)$ represent the set of nodes that are fanins of z. A node x is said to be a predecessor of node z if there exists a directed path in the graph from x to z.

A *K-feasible cut* at z, N_z, is defined to be a subgraph consisting of z and some of its

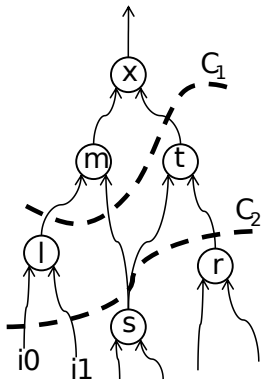


Figure 2.3: Cuts in circuit graph.

predecessors such that $|Inputs(N_z)| \leq K$. Consequently, the technology mapping problem for K-LUTs can be thought of as covering an input DAG with K-feasible cones. Usually, there are many K-feasible cones for each node in the network, each having different area, delay, or power characteristics.

Figure 2.3 illustrates cuts for a node x in a circuit graph. A cut for x is a partition, $(V, \overline{V})$, of the nodes in the subgraph rooted at x, such that $x \in \overline{V}$. For x's cut C_1 in figure 2.3, $\overline{V}$ consists of two nodes, x and m. For x's cut C_2 in the figure, $\overline{V}$ consists of x, m, t, and l. A cut is called K-feasible if the number of nodes in V that drive nodes in $\overline{V}$ is less than or equal to K. In the case of cut C_1, there are 3 nodes that drive nodes in $\overline{V}$ and, the cut is 3-feasible. For a cut $C = (V, \overline{V})$, Inputs(C) represents the nodes in V that drive a node in $\overline{V}$. For the cut C_1 in figure 2.3, Inputs(C_1) $= \{l, s, t\}$. $Nodes(C)$ represents the set of nodes, $\overline{V}$. In figure 2.3, $Nodes(C_1) = \{x, m\}$.

For a K-feasible cut, C, the logic function of the subgraph of nodes, $\overline{V}$, can be implemented by a single K-LUT. The reason for this is that the cut is K-feasible and a K-LUT can implement *any* function of up to K inputs. Hence, the problem of finding all of the possible K-LUTs that generate a node's logic function can be cast as the problem of finding

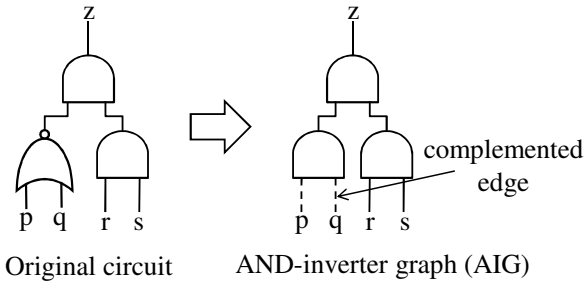


Figure 2.4: And-Inverter Graph (AIG) example.

all K-feasible cuts for the node. There are generally many K-feasible cuts for each node in the network, corresponding to multiple potential LUT implementations.

Enumerating all cuts for each node in the circuit graph is a well-studied problem with an established solution: The cuts for each node in the network can be generated in a topological network traversal, from inputs to outputs. As each node is visited in the traversal, its complete set of K-feasible cuts is generated by merging cuts from its fanin nodes, using the method described in [13, 44].

Having computed the set of K-feasible cuts for each node in the circuit graph, the graph is traversed in topological order again. During this traversal, a "best cut" is chosen for each node. The best cut reflects design optimization criteria, typically, area, power, delay or routability. The best cuts define the LUTs in the technology mapped circuit.

As mentioned, the first step in technology mapping is to represent the network (combinational logic) as a directed acyclic graph such that the number of inputs to each node is less than or equal to K, the number of inputs of the target LUT. A common data structure for this representation is an And-Inverter Graph (AIG). In an AIG, the circuit functionality is represented solely as a network of 2-input `AND` gates and inverters. An example of an `AND`-inverter graph is shown in figure 2.4. Observe that inverters are not represented explicitly as nodes in the graph, but rather as properties on graph edges.

Research has demonstrated the utility of AIGs for many logic synthesis transformations (e.g., [38, 30]). AIGs have also shown value for FPGA technology mapping as exemplified by the ABC tool [39, 37]. We therefore choose to investigate guarded evaluation within the ABC framework and to exploit the properties of AIGs to aid in performing guarded evaluation.

2.3 Power Consumption in FPGAs

FPGAs have two main types of power consumption: **Static** power and **Dynamic** power. Static power occurs due to leakage current in the SRAM-based FPGAs. As transistor technology scales (State-of-the-art FPGAs are currently being implemented at 22nm [54]), the sub threshold leakage power grows significantly. Scaling causes the power supply voltage, V_{dd}, to be scaled down which impacts speed. To maintain the speed, the threshold voltage, V_{th}, is reduced which increases the reverse biased PN-junction current and sub-threshold channel conduction. Static power has been negligible in the past. The work in [45] estimates static power contributes about 5-20% of total power dissipation, depending on temperature, device, frequency, and the implemented circuit. However, as scaling continues, static power dissipation becomes more prevalent and needs to be considered. There are several techniques proposed to minimize leakage power of a gate such as using dual-threshold transistors [20, 27], or minimizing leakage power of a transistor in the OFF state by supplying particular input vectors [40]. The focus of this work, however, is to reduce dynamic power dissipation in FPGAs

As shown in equation. 1.1, dynamic power is directly proportional to the capacitance, swing voltage and operating frequency of each net. Efforts have been made to reduce each of these parameters. To reduce the overall swing voltage, dynamic voltage scaling

techniques [36] are popular where based on the workload and performance requirements of sub-circuits, V_{dd} and V_{th} are modified using body biasing to reduce the power budget for comparable performance. Furthermore, V_{th} can be reduced by intelligently sizing the transistors such that the circuit meets the timing requirements, while reducing V_{dd} [23, 11, 47]. These techniques are also applied to reduce sub-threshold leakage power. These methods, however, require transistor level modifications which is an arduous task for FPGAs as it warrants architecture level changes. Hence, there has been a significant effort to reduce dynamic power consumption through the CAD flow. A brief overview of some of these techniques is provided in the next section.

2.4 Power-Aware CAD Algorithms

2.4.1 Power-aware mapping

Power-aware cut-based technology mapping has been studied recently (e.g., [26, 22]). The core approach taken is to keep signals with high switching activity out of the FPGA's interconnection network. Since the routing tracks used to implement the interconnection network in FPGAs consist high capacitive load, charging and discharging consumes a significant amount of power. This is achieved by costing cuts to encourage such high activity signals to be captured within LUTs, leaving only low activity inter-LUT connections. A second aspect of power-aware mapping pertains to logic replication. Logic replication is needed to achieve mappings with low depth (high speed). However, replication can increase power [26], as replication increases signal fanout and capacitance. Replications can therefore be detected and cost accordingly, trading off their power "cost" with their depth "benefit" [8]

2.4.2 Power-aware packing

Modern FPGAs [51] [4] "pack" LUTs and flip-flops into logic clusters that contain multiple LUTs and registers. This has advantages from the delay and area perspectives [34] as well as reduced compile time due to fewer number of blocks. The packing algorithm has two objectives:

1. Minimize the area by packing the clusters to capacity.
2. Minimize the inputs to each cluster to minimize inter-cluster connections.

The algorithm proceeds by picking an unclustered LUT as the "seed". Corresponding LUTs are packed along with the seed LUT based on an *attraction* function, which is determined by the number of inputs and outputs that the LUTs have in common. There has been considerable work on modifying the attraction function to produce packed netlists with minimal area, delay and power. [33, 14, 46]

Power aware packing has been proposed for clustered FPGA architectures. The idea is to encapsulate high activity signals inside of clusters [26]. In [43] energy is targetted by modifying the attraction function that is used to select the LUTs packed within the same cluster. The function aims to reduce inter-cluster signals as well as the pin-utilization ratio on each cluster.

2.4.3 Power-aware placement

Placement is a well-studied NP-complete problem [49, 28] with several heuristic based approaches. Power can be reduced during placement by modifying the objective function to account for the activity and capacitance in of signals, effectively trying to reduce the length of high activity signals [50]. [25] uses post routing timing analysis to align arrival times of signals to reduce extraneous toggling due to glitches.

It is clear that there is opportunity for dynamic power reduction at each stage of the FPGA CAD flow. In this work, we implement the proposed power reduction algorithm as a post-processing step at the end of each stage.

2.5 Guarded Evaluation

Tiwari et al. [48] first described important techniques for guarded evaluation in ASICs. The key idea is shown in figure 2.5. Figure 2.5(a) shows a multiplexer receiving its inputs from a shifter and a subtraction unit, depending on the value of select signal *Sel*. Figure 2.5(b) shows the circuit after guarded evaluation. *Guard logic*, comprised of transparent latches, is inserted before the functional units. The latches are transparent only when the output of the corresponding functional unit is selected by the multiplexer, i.e., depending on signal *Sel*. When the output of a functional unit is not needed, the latches hold its input constant, eliminating toggles within the unit. Here, one can view *Sel* as the "guarding signal". Tiwari applied this concept to gate-level networks, where the difficulty was in determining which signals could be used as guarding signals for particular sub-circuits. Tiwari used binary decision diagrams to discover logical implications that permit certain sub-circuits to be disabled at certain times.

Abdollahi et al. proposed using guarded evaluation in ASICs to attack both leakage and dynamic power [2]. The guarding signals were used to drive the gate terminals of NMOS sleep transistors incorporated into CMOS gate pull-down networks, putting sub-circuits into low-leakage states when their outputs were not needed.

Howland and Tessier studied guarded evaluation at the RTL level for FPGAs [16]. Their approach produced encouraging power reduction results by exploiting select signals on steering elements (multiplexers) to serve as guarding signals and is therefore limited to

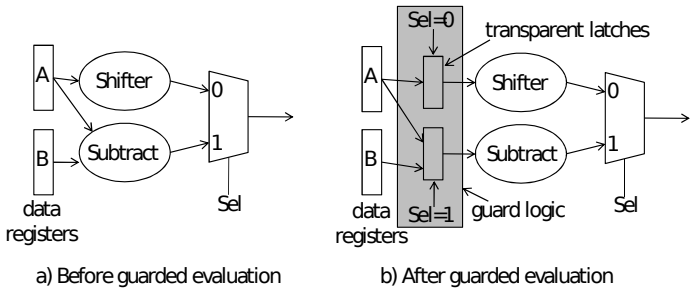


Figure 2.5: Guarded evaluation (adapted from [48]).

specific types of circuits; e.g., datapath circuits in which multiplexers are used for resource sharing. Our approach is not directly comparable since we work on a synthesized LUT network, avoid adding additional logic into the network, and are not limited to using only the select lines on multiplexers to act as guarding signals.

In contrast to prior works, which discover only a limited number of candidate guarding opportunities, our approach exposes many guarding opportunities through easy-to-compute properties of the logic synthesis network. Furthermore, while prior approaches required additional hardware to be added to the design (e.g., transparent latches in figure 2.5), our approach incurs no overhead (in terms of LUT count) by using existing yet unused FPGA circuitry, although additional wires are required to perform guarding.

2.6 Gating Inputs and Non-Inverting AIG Paths

Technology mapping covers the circuit AIG with LUTs – each LUT in the mapped network implements a portion of the underlying AIG logic functionality. A recent work suggested a new FPGA architecture using properties of the AIG to discover *gating inputs* to LUTs [6]. A gating input to a LUT has the property that when the input is in a particular logic state

(either logic-0 or logic-1), then the LUT output is logic-0, irrespective of the logic states of the other inputs to the LUT. We borrow the idea of gating inputs for guarded evaluation and therefore briefly review the concept here.

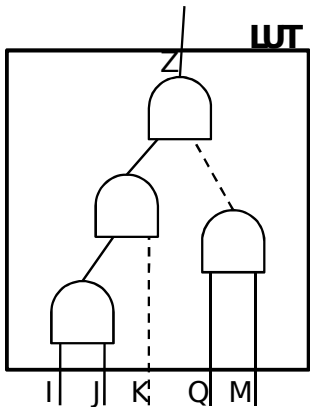


Figure 2.6: Identifying gating inputs on LUTs using non-inverting paths.

Figure 2.6 gives an example of a LUT and the corresponding portion of a covered AIG. The logic function implemented by the LUT is: $Z = I \cdot J \cdot \overline{K} \cdot \overline{Q \cdot M}$. Examine the AIG path from the input I to the root gate of the AIG, Z. The path comprises a sequence of `AND` gates with none of the path edges being complemented. Recall that the output of an `AND` gate is logic-0 when either of its inputs is logic-0. For the path from I to Z, when I is logic-0, the output of each `AND` gate along the path will be logic-0, ultimately producing logic-0 on the LUT output. We therefore conclude that I is a gating input to the LUT. The LUT in figure 2.6, in fact, has three gating inputs, I, J, and K. Input J is the same form as input I in that there exists a path of `AND` gates from J to root gate Z and none of the edges along the path are inverted.

Observe, however, that the situation is slightly different for input K. For input K, the "frontier" edge crossing into the LUT is inverted, however, aside from this frontier edge, the remaining edges along the path from K to the root node Z are "true" edges.

This means that when K is logic-1, the output of the AND gate it drives will be logic-0, eventually making the LUT's output signal Z logic-0. K is indeed a gating input, though it is K's logic-1 state (rather than its logic-0 state) that causes the LUT output to be logic-0. In contrast with inputs I, J and K, LUT inputs Q and M are not gating inputs to the LUT as neither logic state of these inputs causes the LUT output to be logic-0. The question of which inputs are gating inputs is also apparent by inspection of the LUT's Boolean equation.

In [6], the gating input idea was generalized and it was observed that the defining feature of such inputs is the presence of a *non-inverting path* from the input through the AIG to the root node of the AIG. Since by definition, an AIG contains only AND gates with inversions on some edges, one does not need to be concerned with other gates appearing in the AIG (e.g. EXOR). Non-inverting paths are therefore chains of AND gates without edge inversions. Gating inputs to LUTs can be easily discovered through a traversal of the underlying AIG. In [6], the notions of gating inputs and non-inverting paths were applied to map circuits into a new logic block architecture that delivers improved area-efficiency. Here, we apply the ideas for power reduction through guarded evaluation.

2.7 Trimming Inputs and Partial Non-Inverting AIG Paths

As previously described, gating inputs are determined by searching for non-inverting paths from the input to output of a LUT in the LUT's underlying AIG representation. However, more opportunities for guarding can be found by considering *trimming inputs* in addition to gating inputs. Consider the logic function $Z = \overline{(A \cdot B \cdot \overline{C \cdot D})} \cdot \overline{(\overline{D \cdot E} \cdot F)}$ illustrated in

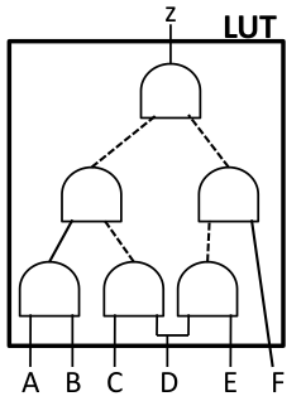


Figure 2.7: Identifying trimming inputs on LUTs using partial non-inverting paths

figure 2.7.

There is no non-inverting path from any LUT input to the LUT output, Z. However, we can observe that a logic-`0` on input A will still force the output on some `AND` gates to be logic-`0` as its value propagates towards Z. We can identify the `AND` gate that drives the first inverted edge on the path from A to Z and, subsequently, find the fanout-free cone rooted at the identified `AND` gate; the set of LUT inputs to this fanout-free cone (excluding A) can be *trimmed* by A when A is a logic-`0`. In this example, this means inputs B and C can be *trimmed* when input A is a logic-`0`. Note that input D cannot be trimmed since it is not in the fanout free fanin cone of the affected `AND` gates. Input F can be used to trim input E (but not input D) when F is a logic-`0` following a similar analysis. We refer to, and discover, trimming inputs by considering *partial non-inverting paths* which are simply defined as non-inverting paths which are internal to the LUT's underlying AIG representation and begin at LUT inputs.

The idea of trimming and gating inputs are related to the Shannon decomposition of a LUT's logic function as described in [7]. Recall that any n-variable logic function f can be co-factored with respect to variable x_k as follows:

$$f = x_k \cdot f(x_0, \cdots, x_{k-1}, \mathbf{1}, x_{k+1}, \cdots, x_n) + \overline{x_k} \cdot f(x_0, \cdots, x_{k-1}, \mathbf{0}, x_{k+1}, \cdots, x_n) \qquad (2.1)$$

Here, $f(x_0, \cdots, x_{k-1}, 1, x_{k+1}, \cdots, x_n)$ is the 1-co-factor of f with respect to x_k and $f(x_0, \cdots, x_{k-1}, 0, x_{k+1}, \cdots, x_n)$ is the 0-co-factor of f with respect to variable x_k. Each co-factor is itself a logic function with at most $n-1$ variables. In [7], a trimming input was defined as an input to a n-variable function in which the Shannon decomposition produced a co-factor having strictly less that $n-1$ inputs. In the case of a gating input, the Shannon decomposition produces a decomposition in which one of the co-factors is logic-0. Hence, with respect to [7], the use of non-inverting paths and partial non-inverting paths are structural techniques to identify gating and trimming inputs, respectively.

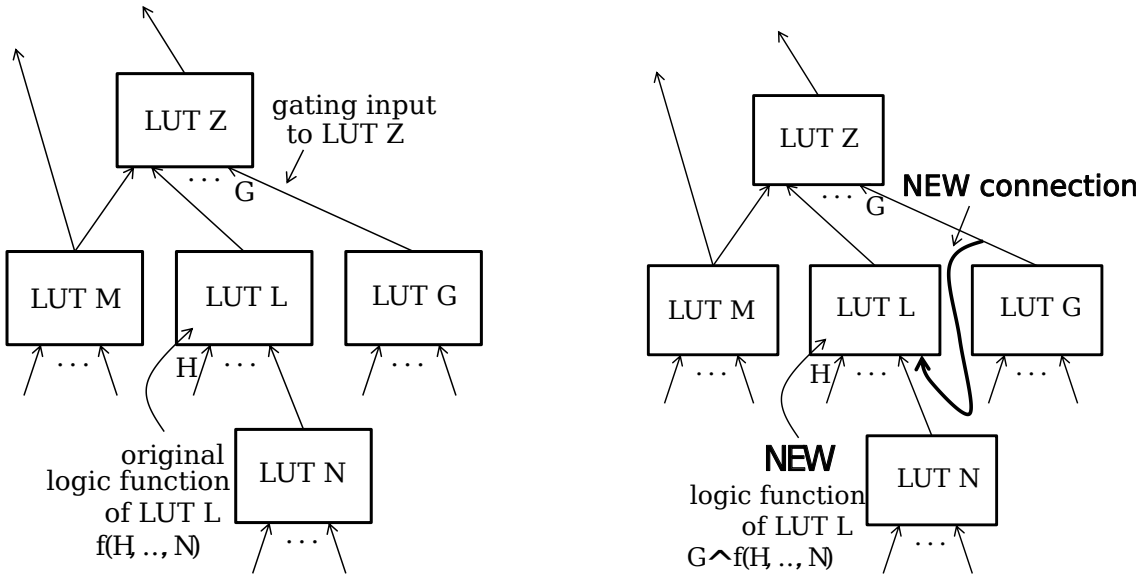
Chapter 3

Guarded Evaluation for FPGAs

We now describe our approach to guarded evaluation, beginning with a top-level overview, and then describing how guarding opportunities can be created during technology mapping, and finally discussing the post-mapping guarding transformation. We also describe the approach taken to implement guarded evaluation on post-packed and post-placed netlists.

3.1 Overview

Figure 3.1(a) illustrates how gating and trimming inputs to LUTs can be applied for guarded evaluation. Without loss of generality, assume that logic-0 is the state of the gating input, G, that causes LUT Z's output to be logic-0. When G is logic-0, Z is also logic-0, and any toggles on the other inputs of Z are guaranteed not to propagate through Z to circuit outputs. Similarly, if G is a trimming input of, say, input L (i.e., a logic-0 on G blocks toggles on signal L from propagating to signal Z), then L can also be guarded by signal G.



(a) Original LUT network

(b) Network after Guarded Evaluation

Figure 3.1: Network before and after guarding

Since L's single fanout is to Z, L's output value will not affect overall circuit outputs when G is logic-0. Toggles that occur in computing L's output when G is logic-0 are an unnecessary waste of dynamic power.

In figure 3.1(a), L is a candidate for guarded evaluation by signal G. If LUT L has a free input, we modify the mapped network by attaching G to L, and then modifying L's logic functionality as shown in figure 3.1(b). The question is how to modify L's logic functionality. In [5], logic functions were modified to force the LUT output to a logic-0 when guarded. Here we also consider different types of guards based on signal probabilities and guarding values. For a signal L, define its static probability, $P(L)$, as the probability that the signal is logic-1. Static probability is a property of logic signals widely used in the power estimation domain [41]. Assume a guarding value of logic-0 for signal G; the new logic function for L is determined based on L's static probability, $P(L)$, of signal L. If the

signal spends most of its time at logic-0 (i.e., $P(L) \leq 0.5$), it is set equal to a logical AND of its previous logic function and signal G. Hence, we force the signal to logic-0 when it is guarded. If $P(L) > 0.5$, the logic function is set equal to the logical OR of the previous logic function and the inverted version of signal G, hence forcing the signal to logic-1 when it is guarded. This distinction is made to avoid inducing additional toggles on the guarded signal. Consider the case where the output of LUT L in figure 3.1(b) was logic-1 the instant prior to guarding. If it was guarded using a logical AND of its previous function and signal G, then the gate would induce one additional toggle from logic-1 to logic-0. Hence, the static probability of the guarded signal is examined prior to inserting the guarding logic to avoid such additional (and unnecessary toggles).

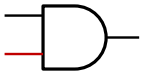
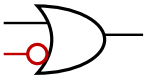
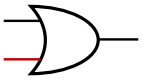
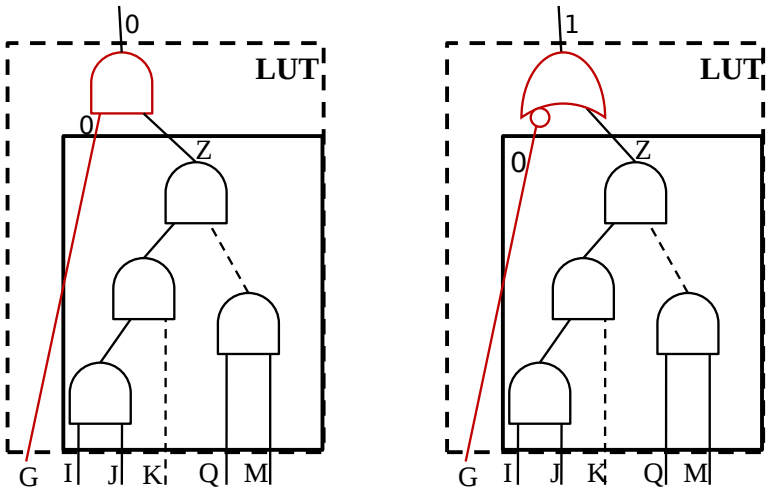
Guard	**Static Probability**	
	P <= 0.5	**P > 0.5**
Logic-0		
Logic-1		

Figure 3.2: Inserting guards based on static probability

Figure. 3.2 provides an illustration of the type of guarding used based on the static probability and the guarding value[1]. No additional LUTs are required to perform guarding since we are modifying the function of the guarded LUT, which is logic internal to the LUT, as illustrated in figure 3.3. After guarding, switching activity on L's output signal may be reduced, lowering the power consumed by the signal. Note, however, that guarding must

[1]We note that, since we are using AIG representations, we do not insert explicit OR gates, but rather AND gates with appropriate inversions on inputs and outputs.

be done judiciously, as guarding increases the fanout (and likely the capacitance) of signal G. The benefit of guarding from the perspective of activity reduction on L's output signal must be weighed against such cost.



(a) Guarding with AND (b) Guarding with OR

Figure 3.3: Modified function internal to LUTs

The guarded evaluation procedure can be applied recursively by walking the mapped network in reverse topological order. For example, after considering guarding LUT L with signal G, we examine L's fanin LUTs and consider them for guarding by G. Since LUT N in figure 3.1(a) only drives LUT L, N is also a candidate for guarding by signal G. We traverse the network to build up a list of guarding options.

There may exist multiple guarding candidates for a given LUT. For example, if signal H in figure 3.1(a) were a gating or trimming input to LUT L, then H is also a candidate for guarding LUT N (in addition to the option of using G to guard N). Furthermore, if a LUT has multiple free inputs, we can guard it multiple times. We discuss the ranking and selection of guarding options in section 3.3. The ease with which we can use AIGs to

identify gating and trimming inputs (via finding non-inverting and partial non-inverting paths) circumvents one of the key difficulties encountered by Tiwari et al. [48], specifically, the problem of determining which signals can be used to guard which gates.

While we can guard L with G in figure 3.1, we cannot necessarily guard LUT M with G. The reason is that M is multi-fanout, and it fans out to LUTs aside from Z. In Section 3.3.1, we discuss using circuit "don't cares" to enable guarding in *some* cases such as M. Note, however, that there do exist multi-fanout LUTs in circuits where guarding is "obviously" possible, such as LUT Q in figure 3.4. LUT Q fans out to two LUTs, however, both fanout paths from Q pass through LUT Z. LUT Q is said to have *reconvergent* fanout. If all fanout paths from a LUT pass through the "root" LUT that receives the gating input, then guarding the multi-fanout LUT can be done without damaging circuit functionality. A fast network traversal can be used to determine if all transitive fanout paths from a LUT pass through a second LUT. Such a traversal is applied to qualify multi-fanout LUTs as guarding candidates. In general, for a guarding signal G driving a LUT Z, we can safely use G to guard any LUT within Z's fanout-free fanin cone.

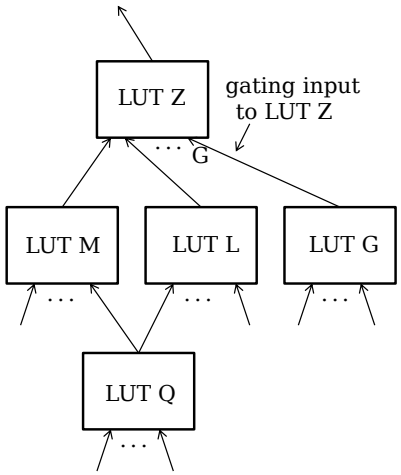


Figure 3.4: Guarding with re-convergent fanout

It is worthwhile to highlight an important difference between our approach and the prior ASIC approach, shown in figure 2.5. In figure 2.5, transparent latches are used to hold inputs to blocks constant while the blocks are guarded. Our approach, on the other hand, takes the logical `AND` or logical `OR` of an existing LUT function with the guarding signal, making the LUT output logic-`0` or logic-`1` while guarded. Our method requires the guarded LUT to have additional inputs that are free to insert the guarding signal, which constrains some guarding opportunities. Nonetheless, our results show that a significant number of guards were inserted effectively reducing dynamic power. Moreover, our method has the advantage of adding no LUTs to the circuit. The only overhead is the wires added to connect guarding signals to the fanin of the guarded LUTs.

It is also worth mentioning that LUTs in today's commercial FPGAs have 6 inputs [4, 51], which provide better speed performance than the 4-LUTs used traditionally. Many logic functions in circuits require less than 6 variables and consequently, LUTs in mapped circuits commonly have unused inputs. A recent work from Xilinx demonstrated that in commercial 6-LUT circuits, only 39% of the LUTs in the mappings use all 6 inputs [21]. A similar observation was made earlier in [29] when describing the Altera Stratix II architecture where it was observed that only 36% of the LUTs in a mapped set of designs required full 6-LUTs. The considerable number of LUTs with unused inputs bodes well for our guarding scheme.

3.2 Creating Guarding Opportunities During Mapping

Having introduced how guarded evaluation can be applied to a mapped network, we now consider the influence of the mapping step itself on guarding. We aim to encourage the creation of LUT mapping solutions containing "good" guarding opportunities, while maintaining the quality of other circuit criteria, such as area and depth. We propose a cost function for cuts to reflect cut value from the guarding perspective.

For a set of inputs to a cut C, Inputs(C), define $Gating$[Inputs(C)] to be the subset of inputs that are gating inputs, as defined in Section 2.6. We define a *GuardCost* for a cut, such that minimization of *GuardCost* will encourage the creation of mapping solutions containing high-quality guarding opportunities, while at the same time minimizing the dynamic power of the mapped network:

$$GuardCost(C) = \frac{1 + \sum_{i \in Inputs(C)} \alpha(i)}{1 + |Gating[Inputs(C)]|} \tag{3.1}$$

where $\alpha(i)$ represents the switching activity on LUT input i. The numerator of equation 3.1 tallies the switching activities on cut inputs, minimizing activity on inter-LUT connections in the mapped network. Higher input activities yield higher values of *GuardCost*. A similar approach to activity minimization has been used in other works on power-aware FPGA technology mapping [26, 22]. The denominator of equation 3.1 reflects the desire to have LUTs with gating inputs (i.e., inputs that drive non-inverting paths in the AIG). The signals on such inputs can naturally be used to guard other LUTs, as described in Section 3.1. Cuts with higher numbers of such non-inverting path inputs will have lower values of equation 3.1.

3.3 Post-Mapping Guarded Evaluation

Following mapping, the circuit is represented as a network of LUTs. Consider a guarding option, O, comprising L as the candidate LUT to guard, and G being the candidate guarding signal (produced by some other LUT in the design). The guarding option O is scored as follows:

$$Score(O) = \frac{|Outputs(L)| \cdot \alpha(L) \cdot P(L,G)}{1 + \alpha(G)} \tag{3.2}$$

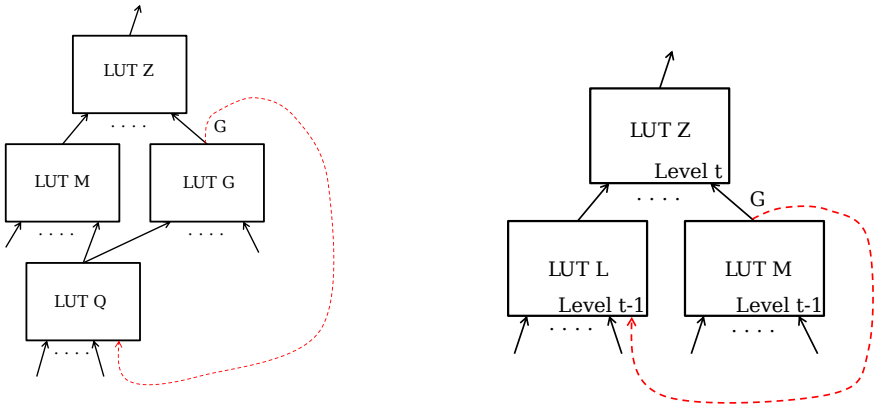
where $|Outputs(L)|$ represents the fanout of LUT L; $\alpha(L)$ and $\alpha(G)$ are the switching activities on L and G's outputs, respectively; and $P(L,G)$ is the fraction of time that G spends at the value that gates L. The numerator of equation 3.2 represents the benefit of guarding, which increases in proportion to L's fanout, its activity and the fraction of time G serves to gate L. The more time that G spends at its gating value, the higher the likely activity reduction on L. The denominator of equation 3.2 represents the cost of guarding, which is an increase of G's fanout (and likely capacitance). The cost is proportional to the activity of signal G, as it is less desirable to increase the fanout of high activity signals. Higher values of equation 3.2 are associated with what we expect will be better guarding candidates.

For a mapped network, we capture all possible guarding options in an array and sort the array in descending order of each option's score, as computed through equation 3.2. The guarding then proceeds as follows: We iteratively walk through the list of guarding options and for each one, we consider introducing the guard into the mapping. To guard some LUT L with some signal G, the following rules must be obeyed:

1. LUT L must have a free input (to attach G).

2. Attaching G to an input of L must not form a combinational loop in the circuit.
3. Signal G must not already be attached to an input of LUT L.
4. The guard should not increase the depth of the mapped network beyond a user-specified limit.
5. The guard must not affect the circuit's functional correctness (discussed in Section 3.3.1 below).

A few of the conditions warrant further discussion. Rule #2 is illustrated in the LUT network of figure 3.5(a). The candidate guarding option is illustrated by the dashed line. If we were to introduce the guard, a combinational loop would be created, as the LUT producing the guarding signal G lies in the transitive fanout of the LUT being guarded, L. We detect and disqualify such guarding options.



(a) Guard creating combinational loop

(b) Guard increasing depth

Figure 3.5: Illustration of impracticable guards

In the case of rule #3, where G is already connected to an input of L, we can alter L's logic function to make G a gating input of L, if it is not already so. We can attain the

benefit of guarding without routing G to an additional load LUT (i.e., without increasing G's fanout).

Regarding rule #4, guarding can have a deleterious impact on network depth, as illustrated by the example in figure 3.5(b). In this case, a root LUT Z at level t receives inputs from two LUTs at level $t-1$: L and M. The candidate guarding option is again shown using a dashed line. If the signal G produced by M is used to guard LUT L, the network depth is increased to $t+1$. Generally, if the level of the LUT producing the guarding signal G is less than the level of the guarded LUT L, the maximum network depth is guaranteed not to increase. Conversely, if the level of the LUT producing G is greater than or equal to the level of L, the network depth *may* increase, depending on whether the LUT L has any slack in the mapping (i.e., depending on whether L lies on the critical path of the mapped network). Naturally, if more flexibility is permitted with respect to increasing network depth, more guarding options can be applied. The allowable increase to network depth is a user-supplied parameter to our guarding procedure.

Introducing a guard on a LUT may reduce the switching activity on the LUT's output and may also reduce activities throughout the LUT's transitive fanout cone. Consequently, activity and probability values become "stale" after guards are introduced.

To deal with this, the activity and probability values are periodically updated during guarding. This is akin to invoking regular timing analysis passes during routing (e.g., as done in [35]). In particular, after introducing T guards into the mapped circuit, we recompute the switching activities and probabilities for all circuit signals. We score the remaining guarding options with the revised activities and probabilities using equation 3.2, and then re-sort the list of guarding options. We resume iterating through the newly sorted list and introducing guards. T is a parameter that permits a user to trade-off run-time with guarding quality. Lower T values will result in better activity reduction, at the expense of

additional computation.

The overall post-mapping guarding process terminates when either there are no profitable guards remaining, or there are no remaining guarding candidates with a free LUT input. We will refer to this guarding flow as the *PostMap* flow in future sections.

3.3.1 Leveraging Non-Obvious "Don't Cares"

"Don't cares" are an inherent property of logic circuits that can be exploited in circuit optimization. Combinational don't cares are tied to the idea of observability. Under certain input conditions, the output of a particular LUT does not affect overall circuit outputs; that is, the LUT output is not observable under certain conditions. Sequential and combinational don't care-based circuit optimization has been an active research area recently. Don't cares were applied for power optimization in [22], wherein high activity connections in a mapped network were removed from the network, or interchanged with other low activity connections in the network. Don't cares can also be used to achieve a considerable reduction in the area of LUT mapped networks [37].

As noted in Section 3.2, guarding inputs on LUTs can be identified through non-inverting and partial non-inverting paths in AIGs and the signals attached to such inputs can be applied to guard certain single and multi-fanout LUTs in the mapped network. This takes advantage of don't cares that are easily discoverable through non-inverting paths. We refer to these as *obvious* don't cares. For cases like that of figure 3.1(b), where LUT L is guarded with signal G, we can be confident that the transformation does not impact the circuit's overall logic functionality. The reason is that G is a gating input to Z in the figure, and L is in the fanout-free fanin cone of Z.

Surprisingly, however, we have observed that due to don't cares, it is possible to perform

guarding in additional non-obvious cases, such as guarding LUTs like M with signal G in figure 3.1(a). Here, M is not in the fanout-free fanin cone of Z, so it is not obvious that guarding M with G should be possible. If we can indeed guard M with G, we refer to this as leveraging *non-obvious* don't cares. We experimented with allowing non-obvious guarding cases to be executed. Section 3.1 above describes the process by which guarding opportunities are identified, namely, by identifying a gating or trimming input, G, to a LUT, Z, and then walking the mapped network upstream from Z's other inputs. The same procedure is employed to discover non-obvious guarding options, except that the uphill traversal is more extensive. Specifically, we consider using G to guard LUTs that lie outside of Z's fanout-free fanin cone.

For guarded evaluation with don't cares, we use the same flow as described above, namely, sorting all possible guarding candidates and iteratively implementing/evaluating each one in the sorted order. We use simulation and combinational logic verification (`cec` command in ABC) to check that guarding (in the case of non-obvious don't cares) does not damage functional correctness (we "undo" the guarding if needed). In particular, we use a fast random vector simulation to ascertain if correct functionality was disrupted. SAT-based formal verification is used if the simulation check was successful. Certainly, performing a full circuit-wise verification after guarding is compute-intensive. However, the aim of this work is to demonstrate the *potential* of guarded evaluation for activity and power reduction. Moreover, recent work on scalable window-based verification strategies, such as [37], can be incorporated to mitigate run-times for large industrial circuits. Power optimization is frequently done as a post-pass conducted after other design objectives are met, specifically, performance and area. Power optimization algorithms are likely not executed during the initial iterative design process, making longer run-times acceptable for such algorithms.

3.4 Post-Packing Guarded Evaluation

We investigate guarded evaluation on clustered netlists by implementing it after the packing stage of the FPGA CAD flow. Clustering information can be incorporated into the algorithm and, possibly, improve the resulting guarding due to the additional information available. Clustering information can be used to determine whether or not a new guard will:

1. be inserted entirely within a Logic Block (LB);
2. use an existing connection between the pair of LBs; or
3. require a new connection between a pair of LBs.

When a guard is required entirely within a LB or can exploit an existing connection between LBs, the new connection will require only additional *intra-cluster* routing. Intra-cluster routing are shorter, have smaller capacitances and consume significantly less power compared to *inter-cluster* connections. Further, an intra-cluster connection is expected to have less impact on the overall usage of routing resources inside of the FPGA and reduce any potential for routing issues such as congestion. With consideration to clustering information, when guarded evaluation finds a guard that requires an additional inter-cluster connection, the LBs must be checked to ensure that the input and output usage of the LBs are not violated. In other words, even though a LUT may have a spare input to allow a guard to be inserted, the LB containing the LUT to be guarded may not have a free input. In such cases, the guard must be rejected in order to prevent the clustering solution from becoming infeasible. Hence, when guarding after clustering, the guarding algorithm can be enhanced to:

1. include design rule checks to avoid the creation of infeasible LBs.

2. include a term in the scoring function which prefers intra-cluster connections (either through guarding entirely within a LB or through the use of an existing connection between a pair of LBs).

To this end, we propose a modified scoring function compared to equation 3.2 as follows:

$$PackScore(O) = (IntraCluster) \times GuardScore(O) \tag{3.3}$$

where $IntraCluster$ is set to number greater than 1 to "boost" the score if the intra-cluster routing, otherwise it is set to 1; it is possible that other scoring functions could be considered. Given a modified guarding algorithm which accounts for clustering information, we consider two flows which guard after clustering. First, we consider guarding with the modified scoring function and design rule checking of LBs to ensure no infeasibilities are created — this is referred to as the *PostPack* guarding flow. We also consider one modification of this flow where we prevent any new inter-cluster connections (and therefore only allow additional intra-cluster connections). We refer to this as the *PostPack(NoNewWires)* guarding flow. The idea of the second post clustering flow is to prevent guarding from having any impact on the number of inter-cluster connections in the design which affords a minimum impact on placement and routing of the netlist. In either flow, re-clustering is not required since all the LBs are guaranteed to be feasible.

3.5 Post-Placement Guarded Evaluation

It is also possible to implement guarded evaluation as a post-placement optimization with additional information (when compared to guarding after mapping or after clustering). Here, guarding can not only take into account the assignment of LUTs to LBs, but also

the physical separation between the LUTs involved in a guarding candidate. As in post-clustering guarding, the scoring function used to rank candidate guards should be modified to account for the positional information available for LUTs. In our experiments, we modify the cost function in Equation 3.3 to account for the physical separation between LUTs in a guarding candidate; the modification is given by

$$PlaceScore(O) = PackScore(O) \times \frac{1}{1 + (6 \times distance(Z, G))} \tag{3.4}$$

where $distance(Z, G)$ represents the Manhattan distance between the LUT Z (the destination of the guard candidate) and LUT G (the source, or controlling signal, of the guard candidate). This scoring function is intended to encourage guarding opportunities between nearby LUTs such that any new inter-cluster signals require short connections, thereby minimizing the impact on routing resource usage. Of course, the cost function does not prevent the insertion of guards when the LUTs are far apart, but only discourages such connections. The constant of 6 in the denominator was found to be a good scaling factor in our experiments. We note that more complicated scoring functions could be taken into account; e.g., even though routing is not completed, it would be possible to estimate the routing resources required by each signals to get a better estimation of the impact of a new connection. However, as will see in our numerical results, encouraging guarding through distance appears sufficient to illustrate the benefit of guarding after placement. We refer to guarding after placement as the *PostPlace* flow.

3.5.1 Multiple Step Guarded Evaluation

Due to the added design rule check to prevent LB infeasibilites, several high-quality guards are discarded in the *PostPack* and *PostPlace* flows. Hence, we also implemented guarded evaluation in two stages of the CAD flow simultaneously, namely post tech mapping and

post placement. Only the guarding options that meet a certain score threshold according to equation 3.2 were considered after mapping to ensure that a minimal number of high-quality guards are inserted. We do a post-placement optimization where we again insert guards considering packing and placement information. We refer to this as the *PostMap-Place* flow.

Figure 1.1 is modified in figure 3.6 to summarize the different flows considered to implement guarded evaluation in this work. Technology mapping is modified to Activity-Driven Tech-mapping which corresponds to the equation 3.1 added to encourage LUTs with low activity inputs and more guarding opportunities. *PostMap*, *PostPack*, and *PostPlace* op 3.3timizations are represented by the green circles. *PostMap-Place* flow is omitted in the illustration for clarity. u

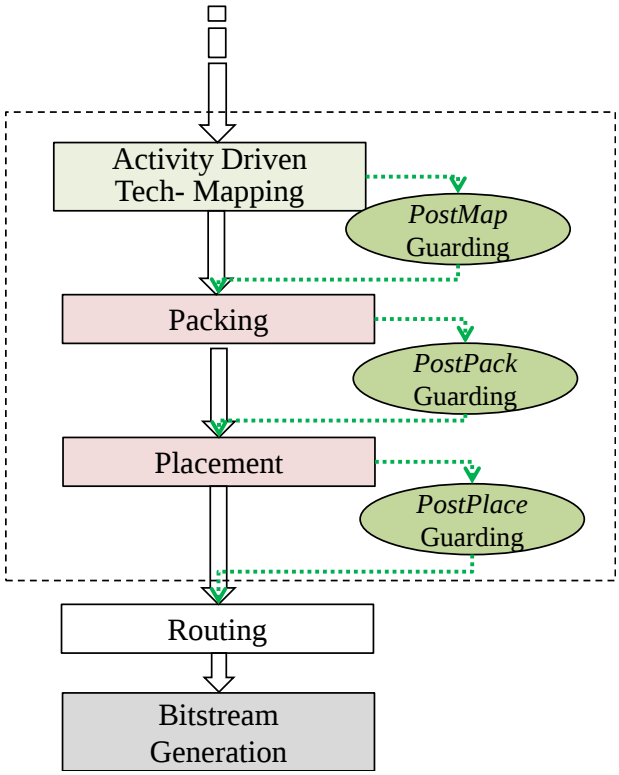


Figure 3.6: Modified FPGA CAD Flow

Chapter 4

Experimental Results

In this chapter, the experimental methodology used to implement the various guarding strategies and flows is described and results are presented. Switching activity numbers refer to the results obtained after the mapping and guarding step. We thoroughly explore the different guarding strategies for the *PostMap* flow described above and make conclusions on the various strategies based on these results. Then we pick the best two guarding strategies in *PostMap* and implement them in the different guarding flows (i.e. *PostPack*, *PostPlace*, etc). The Power reduction numbers are also thoroughly analyzed and discussed for the *PostMap* flow. Power reduction results for the other flows are presented for a comparative analysis. Hence, we analyze the results and try to answer two different questions:

1. Which guarding **strategy** is best?

   The different guarding **strategies** are implemented in the *PostMap* flow and compared with respect to switching activity and interconnect power reduction.

2. Which guarding **flow** is best?

   The best deemed guarding strategies are chosen and the different guarding **flows** are

analyzed and compared with respect to switching activity and interconnect power reduction.

4.1 Methodology

Guarded Evaluation is implemented on an open-source logic synthesis framework called ABC [1] and targeted both 6- and 4-LUT architectures. We compare the results of guarded networks with several different baseline mappings: 1) LUT mapping based on priority cuts [39] (the `if` command in ABC), 2) WireMap [21], and 3) activity-driven WireMap. Briefly stated, WireMap is a technique that reduces the number of inter-LUT connections which tends to be beneficial for power. Activity-driven WireMap has its cut selection cost function altered to break ties using the sum of switching activity on cut inputs, as described in section 3.2. In all cases, prior to mapping, we execute the ABC `choice` command [12] which provides added mapping flexibility and has been shown to provide superior results. Guarded evaluation was applied to a modified WireMap mapper, where ties in cut selection were broken with the values returned by equation (3.1) to improve guarding opportunities. In all cases, guarded networks were verified using the ABC `cec` command. To determine the benefits of guarded evaluation, we evaluate our ideas using two different power metrics: 1) total switching activity after technology mapping, and 2) power dissipated in the FPGA interconnect after placement and routing[1].

The activity across all nets of a circuit are summed to produce total switching activity. To generate switching activity information, we used the simulator built-in to ABC. Each combinational input (primary input or register output) is assigned a random toggle probability between 0.1 and 0.5. Random input vectors were then generated in a manner

[1]Prior work has shown that interconnect comprises ∼2/3 of dynamic power in FPGAs [45].

consistent with the input toggle probabilities. ABC's logic simulator was used to produce activity values for internal signals, considering the logic functionality. The same set of input vectors were used for each circuit across all runs. All generated simulation and activity information is used when performing packing, placement and routing to determine actual power dissipation for consistent results throughout the experiments.

For dissipated power, we use the VPR framework described in [26], which is based on VPR4.3, and integrates the FPGA power model of [42][2]. Since guarding may adversely impact circuit speed, and since circuits that run slower will naturally consume less dynamic power, it is desirable to evaluate the power impact of guarding separately from the impact of guarding on speed performance. With this in mind, in computing the power numbers, we assume a constant clock frequency (40 MHz) for all circuits/implementations. Hence, the power numbers for a benchmark represent the average power consumed by the benchmark to perform its computations in a given (fixed) amount of time. Differences in observed power for a benchmark across its various implementations (e.g. guarding off/on) are consequently due to differences in switching activities on the benchmark's logic signals and not due to the implementations being clocked at different frequencies. Hence, the power improvements reported in this thesis are essentially energy improvements, and energy is the key metric in determining operational cost and battery life.

Since FPGA architectures are quite varied, we target three different sizes of clustered FPGA architectures when performing both 6- and 4-LUT mapping. Specifically, we target FPGA architectures in which each LUT is possibly paired (packed) with a flip-flop (FF). Then, LUT/FF pairs are clustered into logic blocks (LBs) with 1, 4 or 10 LUT/FF pair(s). In all cases, the routing architecture is composed of length-4 segments. Hence, the following

[2]Newer versions of VPR are available [31, 32], but these newer versions do not include a power model which is required for our investigations.

architectures are targeted:

In all architectures the number of inputs, I, on the logic block clusters is set to $I = K/2 \cdot (N+1)$ where K is the number of LUT inputs and N is the number of LUT/flip-flop pairs per cluster which is a typical value [3].

1. Logic blocks containing one 4-LUT/FF pair per block (flat architecture).
2. Logic blocks containing four 4-LUT/FF pairs per block with 10 inputs to each block.
3. Logic blocks containing ten 4-LUT/FF pairs per block with 22 inputs to each block.
4. Logic blocks containing one 6-LUT/FF pair per block (flat architecture).
5. Logic blocks containing four 6-LUT/FF pairs per block with 15 inputs to each block.
6. Logic blocks containing ten 6-LUT/FF pairs per block with 33 inputs to each block.

To be consistent, for each mapping strategy and each architecture, we force the number of routing tracks to be same; we compute the minimum channel width W needed for the priority cuts mapping and then increase this value by 30%. Therefore, the routing fabric is invariant for each circuit/architecture regardless of the mapping algorithm used. This allows for a fair comparison in terms of dissipated power. In this thesis, an $N \times K$ architecture refers to one with N K-LUT/FF pairs per logic block. Since the FPGA mapper in ABC can operate in depth or area mode, we consider the consequences of guarding on both area-oriented and depth-oriented mappings. For the case of depth-oriented mapping, we also consider the trade-offs between power and depth.

To implement post-packing and post-placement Guarded Evaluation, the mapped netlist is read back into ABC along with the respective packing and placement information. Guarding is performed by considering the underlying AIGs of each LUT in the mapped netlist. The scoring function is modified as described in sections 3.4 and 3.5 and feasible guards are inserted. Switching activities and functions implemented by each LUT are

modified to reflect the guarded solution and the netlist is finally routed. As stated previously, power aware T-V-Pack and VPR [26] are used to perform the packing, placement and routing.

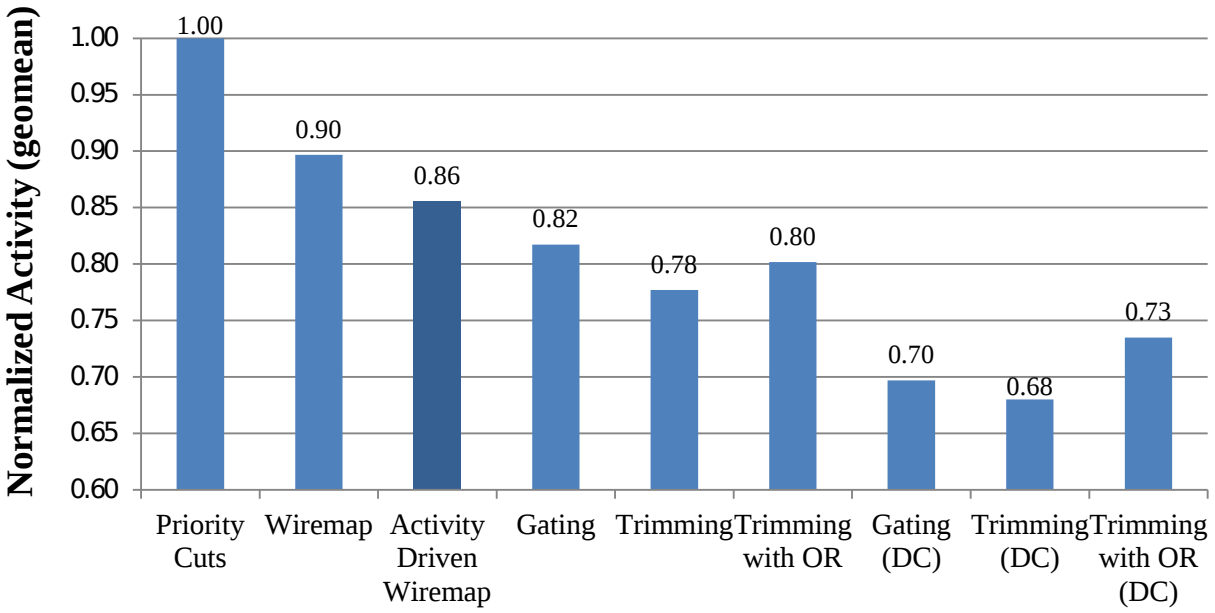
Finally, for benchmarks, we use the larger designs from the MCNC suite [53] which are distributed with the VPR package. When mapped to 4- and 6-LUT architectures, these designs range in size from a few hundred to a few thousand LUTs.

4.2 Switching Activity Results

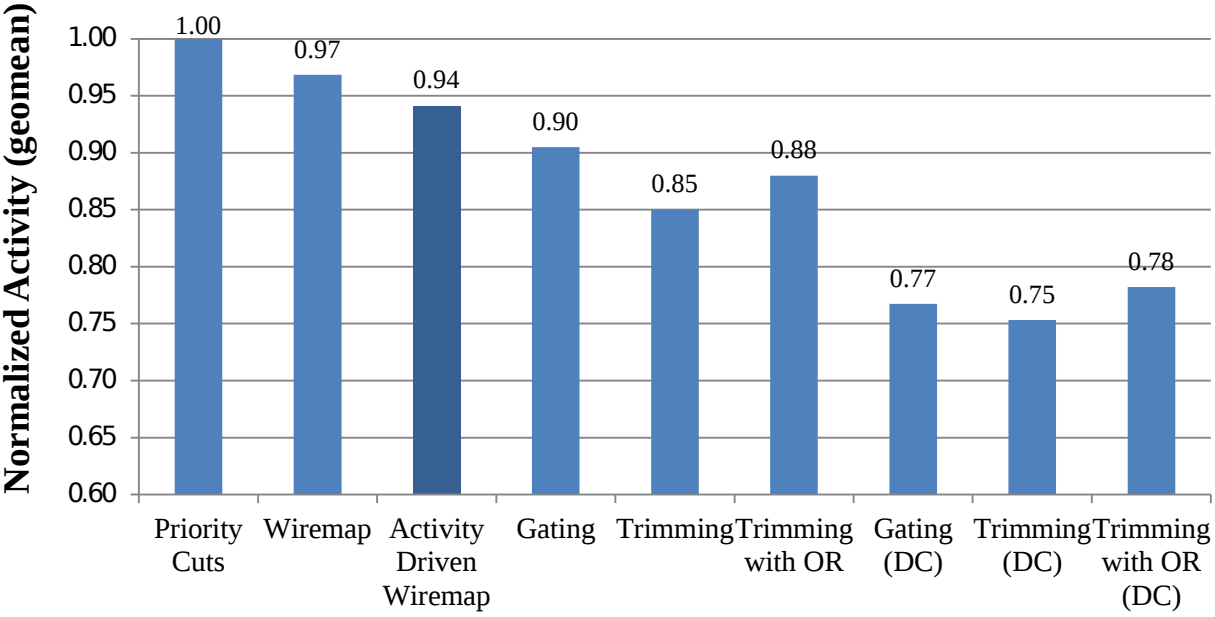
The reduction in total switching activity for area-oriented mapping (using all different mapping techniques) is shown in figure 4.1(a) and figure 4.1(b) for 6-LUT and 4-LUT architectures, respectively. Reported numbers represent the total switching activity averaged across a benchmark suite of 20 circuits normalized to the results obtained using the priority cut-based mapper.

In both figure 4.1(a) and (b), the left-most bar shows total switching activity for priority cut-based mapping [39] and represents the baseline result. The second bar shows activity values for WireMap [21]. On average, WireMap reduces total switching activity by 10% and 3% on average for 6-LUT and 4-LUT architectures, respectively. The third bar shows results for activity-driven WireMap; total switching activity is further reduced by 4% and 3% for 6-LUT and 4-LUT architectures, respectively.

The fourth bar in figure 4.1 shows results for guarding with only gating inputs (c.f. Section 2.6) without any consideration of trimming inputs (c.f. Section 2.7) or non-obvious don't cares (c.f. Section 3.3.1). Further, the fourth bar does not consider the guard insertion based on static probabilities, but only inserts `AND` gates (c.f. Section 3.1) to force



(a) 6-LUT Architectures



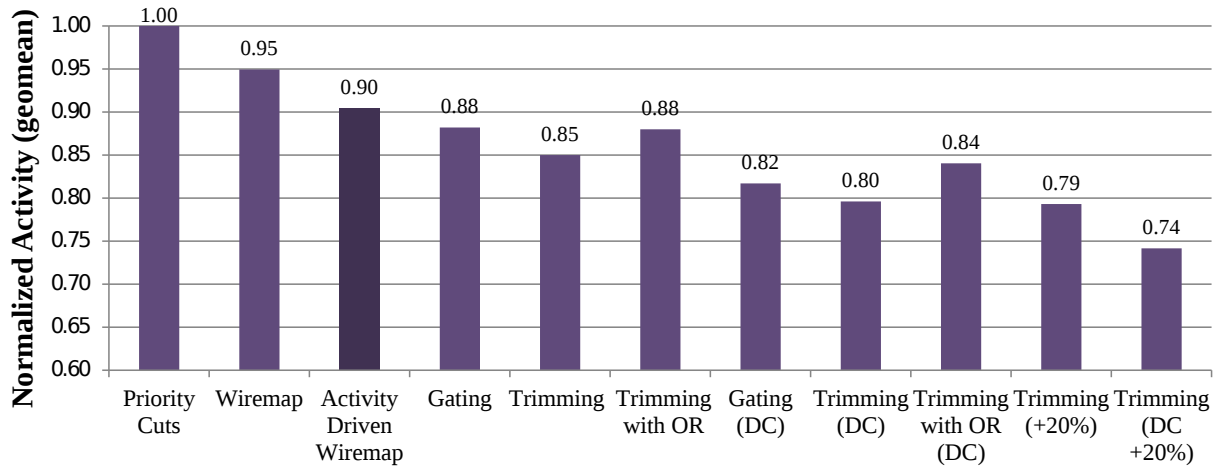
(b) 4-LUT Architectures

Figure 4.1: Normalized Reduction in Switching Activity for Area-Oriented Mapping

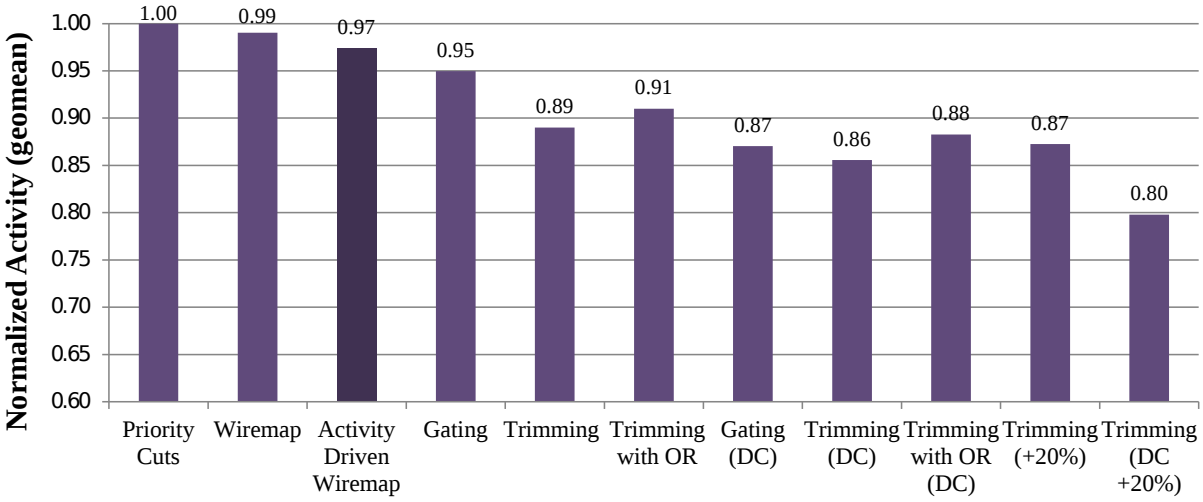
signals to logic-0 when guarded. Guarding with only gating inputs and `AND` gates reduces the total switching activity by an additional 4% for both 6-LUT and 4-LUT architectures when compared to activity-driven WireMap. The use of trimming inputs significantly improves results as shown in the fifth bar in figure 4.1(a) and (b); the total switching activity is further reduced by an additional 4% and 5% for 6-LUT and 4-LUT architectures, respectively, when compared to guarding with only gating inputs. Significantly more guarding opportunities were revealed when trimming inputs (i.e., partial non-inverting paths) are considered.

The sixth bar shows guarding with gating and trimming inputs while considering the guarding value and the static probability of the guarded LUT when determining whether to guard with an `AND` gate or an `OR` gate (c.f. Section 3.1). Recall that, intuitively, by considering different types of guarding logic, it should be true that unnecessary toggling is reduced and, consequently, a further reduction in switching activity can be obtained. However, as demonstrated by the sixth bar in figure 4.1(a) and (b), we see that results are worsened by $2-3\%$ for both 6-LUT and 4-LUT architectures. This result is analyzed and considered further in Section 4.4.

Finally, the last three bars (bars 7 through 9) in figure 4.1(a) and (b) shows results for guarding with consideration for non-obvious "don't cares" under the same conditions as the previous three bars (bars 4 through 6). We see a similar pattern to bars 4 through 6 with the exception that the use of non-obvious don't cares serve to further improve results. If we consider all different mapping strategies, we can see that it is possible to obtain significant reductions in total switching activity compared to the priority cut-based mapper; with minor modifications to WireMap and by proper selection of guarding techniques, average reductions of 32% and 25% in total switching activity can be obtained for 6-LUT and 4-LUT architectures, respectively.



(a) 6-LUT Architectures



(b) 4-LUT Architectures

Figure 4.2: Normalized Reduction in Switching Activity for Depth-Oriented and Depth-Relaxed Mapping

Figure 4.2(a) and (b) show the reductions in total switching activity for 6-LUT and 4-LUT architectures, respectively, once depth optimization is taken into account. During the insertion of guards, an additional constraint is enforced such that the depth of the network cannot increase due to the addition of a guard. Intuitively, this constraint will restrict (reduce) the number of possible guards that can be successfully inserted and, consequently, many guarding options are discarded.

Columns 1 through 9 in figure 4.2(a) and (b) show the depth-oriented results for the different mappers in the same order as presented previously in figure 4.1(a) and (b) for area-oriented mapping. Without further detail, we can see the same trend when comparing the different mapping strategies; the obtained reductions in switching activity, however, are less for depth-oriented mapping due to the enforcement of the additional constraint on logic depth when inserting guards. The best reduction in total switching activity obtained was, on average, 20% and 14% for 6-LUT and 4-LUT architectures, respectively. This result was obtained when guarding was performed with both gating and trimming inputs, and consideration of non-obvious don't cares.

One final experiment was performed with respect to depth-oriented mapping to analyze the impact of the depth constraint enforced during the insertion of guards. Network depth was relaxed and allowed to increase by up to 20% of its optimal depth[3]. Depth relaxation was allowed for the two best mapping strategies; namely (1) guarding with gating and trimming inputs and (2) guarding with gating and trimming inputs and with consideration to non-obvious don't cares. The tenth and eleventh bars in figure 4.2(a) and (b) show the results for 6-LUT and 4-LUT architectures, respectively. We can see that by allowing only a small amount of depth relaxation, a further reduction in the total switching activity is

[3]That is, if the optimal mapped circuit depth was originally L levels, the depth was permitted to grow to $\lceil L \cdot 1.2 \rceil$ levels.

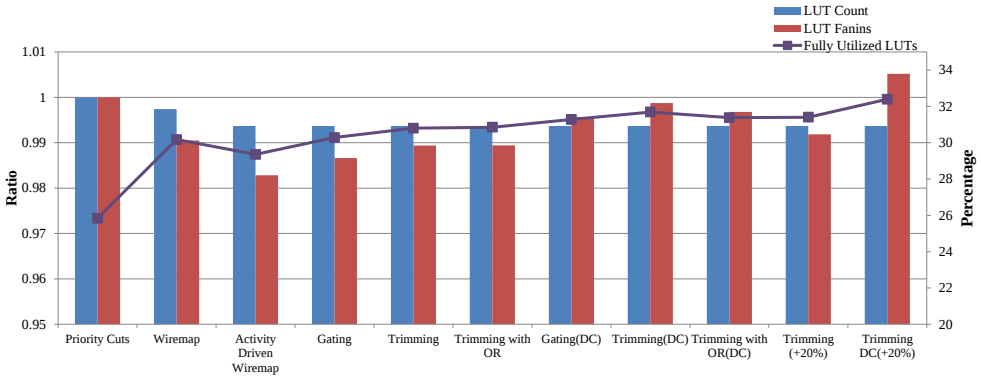
possible.

In summary, the best results in terms of total switching activity for all mappings (area and depth) for both 6-LUT and 4-LUT architectures was produced when guarded with gating and trimming inputs while always using `AND` gates for guarding. The use of non-obvious don't cares served to further improve results. Results for 6-LUT architectures are generally better than those for 4-LUT architectures due to the availability of more free inputs on LUTs which allow for the insertion of more guards.

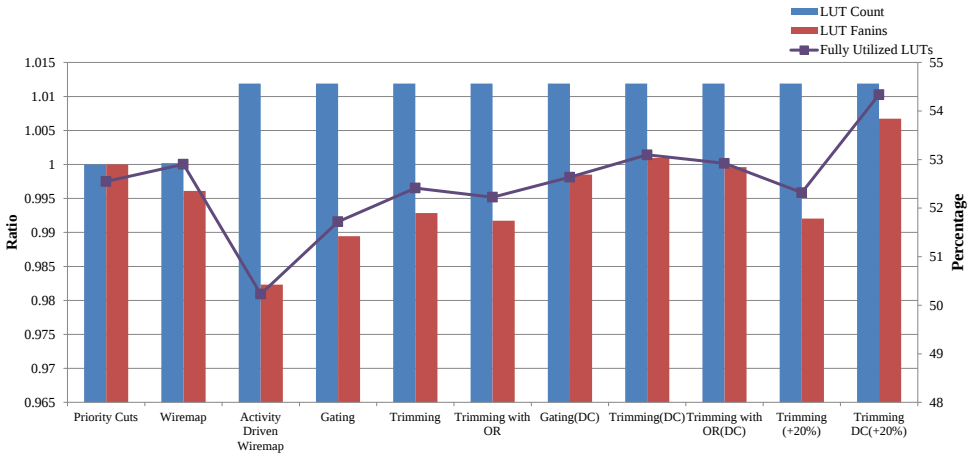
For additional insight into the obtained reductions in total switching activity, Tables A.1 and A.2 present the circuit-by-circuit results for the 6-LUT architectures for area-oriented and depth-oriented mapping, respectively[4]. The last two rows in both Tables A.1 and A.2 show the ratio (of geometric means) of the total switching activity for each mapping technique with respect to the baseline mappers (priority cuts and activity-driven WireMap). Generally, on a per-design basis, the application of guarding aids in reducing the total switching activity. In some cases (e.g., *bigkey*), guarding provided little benefit due to the lack of free inputs on LUTs.

Figure 4.3 shows some additional network statistics to help evaluate the impact of guarding for depth-oriented mapping (area-oriented results are omitted for brevity). The bars in the figure represent LUT count and average number of fanins to each LUT, normalized to the priority cuts mapping. The line in the figure represents the percentage of fully-utilized LUTs (i.e. all inputs used). The bars should be interpreted using the left vertical axis; the line goes with the right vertical axis. Observe that guarding does not increase the LUT count with respect to activity-driven WireMap (see blue bars). Observe also that the average LUT fanin is increased (as expected) due to guarding (see red bars) and that naturally, guarding tends to increase the number of fully utilized LUTs (line).

[4]Circuit-by-circuit results are omitted for 4-LUT architectures for sake of brevity.



(a) 6-LUT Architectures

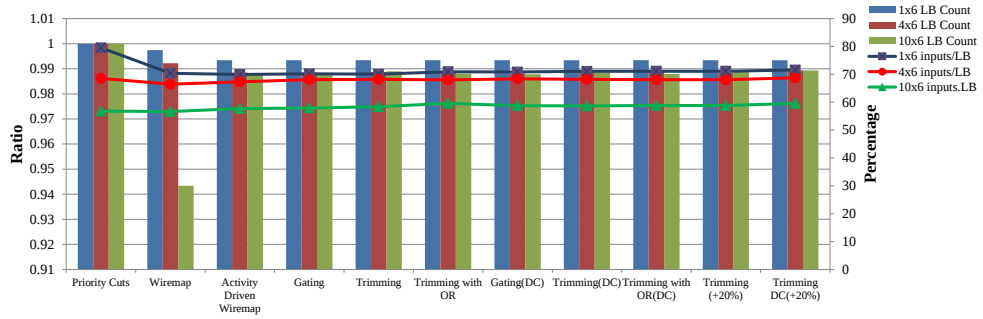


(b) 4-LUT Architectures

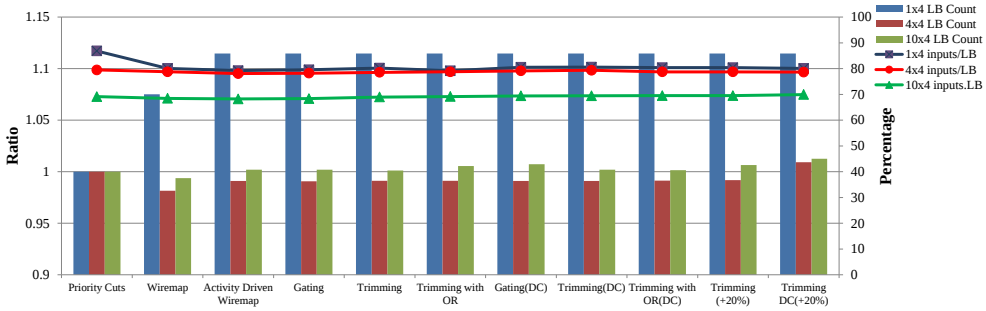
Figure 4.3: Impact of guarding for depth-oriented mapping

The LUT count (blue bars) remain the same as guarding only adds additional connections.

Figure 4.4 shows how guarding affects characteristics of the post-packing netlist, i.e. the netlist *after* LUTs have been packed into LBs. Part (a) gives results for depth-oriented 6-LUT mappings; part (b) gives results for depth-oriented 4-LUT mappings. The bars in the figure illustrate geometric mean LB count for the various flows, normalized to priority



(a) 6-LUT Architectures



(b) 4-LUT Architectures

Figure 4.4: Post-packing statistics on LB count and fanin for depth-oriented mapping

cuts mapping. The line shows average LB fanin (% of used LB inputs) for the architecture having 1 LUT/LB, 4 LUTs/LB and 10 LUTs/LB respectively. Observe that for both 6-LUTs and 4-LUTs, guarding does not have an appreciable impact on LB count – the swings lie within the range of 1-2%, at most. The line in both parts of the figure shows a slight increase towards the more permissive guarding scenarios on the right, where greater numbers of guarding connections are introduced. However, as with LB count, the impact of guarding on LB fanin is evidently quite small. The statistics in the figure are encouraging, as guarding adds connections to the mapped netlist, yet the additional connections appear to have a modest impact post-packing.

We consider runtimes for guarding as follows. Without exploiting don't cares, the worst runtime encountered was 46 seconds[5]. The breakdown of runtime was 33.5% for combinational loop checks, 62.2% for simulation, and 2.1% for guard identification. The small amount of remaining runtime was overhead. Both the simulation runtimes and combinational loop checking can be improved. For example, combinational loops could be checked via node levels rather than via depth-first search in many cases. Similarly, less simulation or incremental simulation could be used. Hence, guarding without don't cares is expected to scale to larger designs. When don't cares are used, however, the runtime situation changes. The worst runtime encountered was ∼8000 seconds. Here, only ∼3% of the runtime was taken for simulation, combinational loop checking and guard identification. Almost all the runtime was used to perform combinational equivalence checking (CEC) via SAT solving. However, it is important to recognize that, as our goal was to evaluate the power benefits of guarding, we made no effort to reduce runtime. The runtime situation is straightforward to improve in a number of ways: In the present implementation, the CEC is always performed on the entire network, but it in fact only needs to be performed on certain points in the fanout of the guarded LUTs. More judicious application of don't cares can be considered. Finally, it is likely that the guarding with don't cares could be better integrated with scalable don't care analysis.

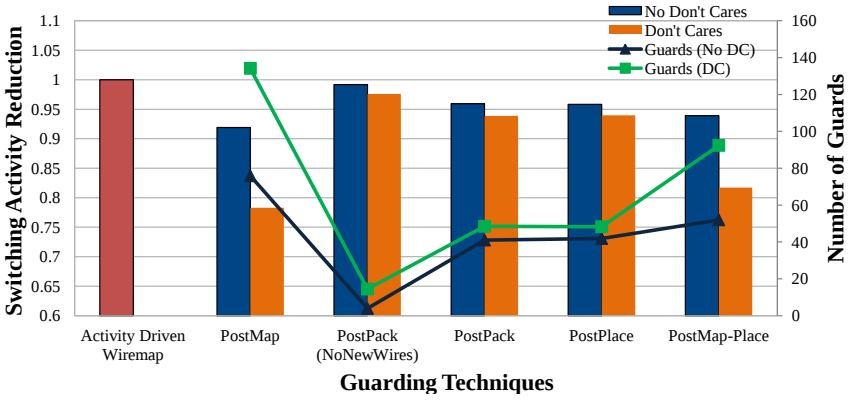
4.2.1 Guarding Post Packing and Placement

This section briefly analyzes the impact of guarded evaluation implemented as a post-packing and post-placement optimizations.

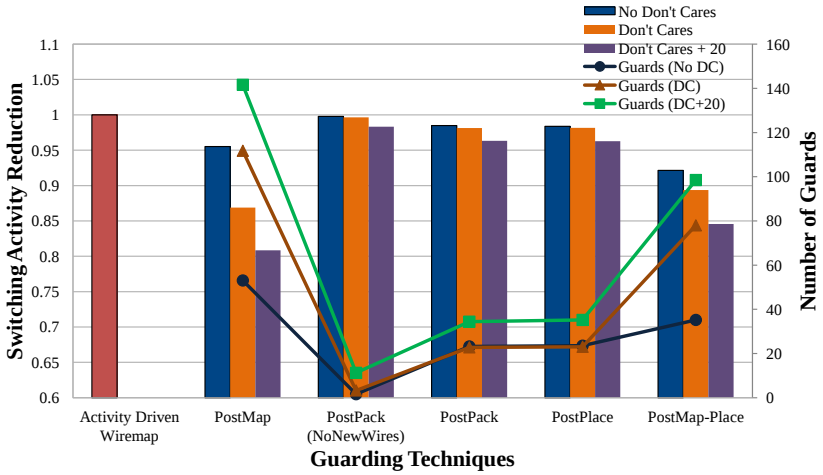
[5]The platform was a 3.2 GHz Intel i7 PC running Ubuntu Linux v11.10. The particular design was *clma* which, when mapped to 6-LUTs, required ∼3000 LUTs.

Figure 4.5 shows a comparison of the switching activity reduction obtained by various flows for 6-LUT architectures. The blue bar represents guarding with `AND` considering Trimming and Gating inputs since this was deemed to be the best guarding strategy from the analysis in the previous section. The Orange bar represents guarding with `AND` considering Trimming and Gating with non-observable don't cares leveraged. For depth-oriented mappings, we also considered guarding with 20% depth-relaxation, represented by the purple bar. The line graphs show the number of guards inserted for each flow. All the numbers are normalized to activity-driven Wiremap (vs. Priority cuts, as in the previous section).

For area-oriented mapping, *PostMap* achieves the most reduction for both guarding strategies. This is the consequence of the number of guards inserted, which were also the highest for this flow. *PostMap-Place*, which inserts high-quality guards post-mapping and performs a post-placement optimization, is able to achieve comparable results with about 22% fewer guards inserted. *PostPack(NoNewWires)* inserts minimal guards due to the added constraint of using only intra-cluster wires. *PostPack* and *PostPlace* are both able to achieve some activity reduction, however due to the feasibility constraint, several guards are discarded. Since at this point we are only considering switching activities post-guarding, the benefits of *PostPack* and *PostPlace* are not visible. The next section, which presents the interconnect power reductions, provides more understanding of the "real" benefits of *PostPack* and *PostPlace*. The reductions are similar for depth-oriented mapping where all guarding options that increase the critical path are discarded. Some additional reductions are gained due to depth-relaxation. It is necessary to consider the number of new netlist connections inserted due to guarding because these signals have impact during routing. As explained previously, the cost of guarding is proportional to the increased fanout of the guarding signal. Furthermore, increased netlist connections give rise to routability



(a) Area-oriented Mapping



(b) Depth Oriented Mapping

Figure 4.5: Average reduction in switching activity for 6-LUT architectures

issues such as congestion and maintaining a minimum routing channel width. Hence, the problem lies in identifying high quality guards that have good activity reductions with minimal impact to routing resources.

4.3 Power Results

While the results above demonstrate a benefit to switching activity, dynamic power scales with the product of activity and capacitance. Guarded evaluation increases the fanout of signals in the network, likely increasing their capacitance and power. Consequently, it is not adequate to focus solely on activity reduction to evaluate the benefit of the technique—actual power measurements after placement and routing are useful.

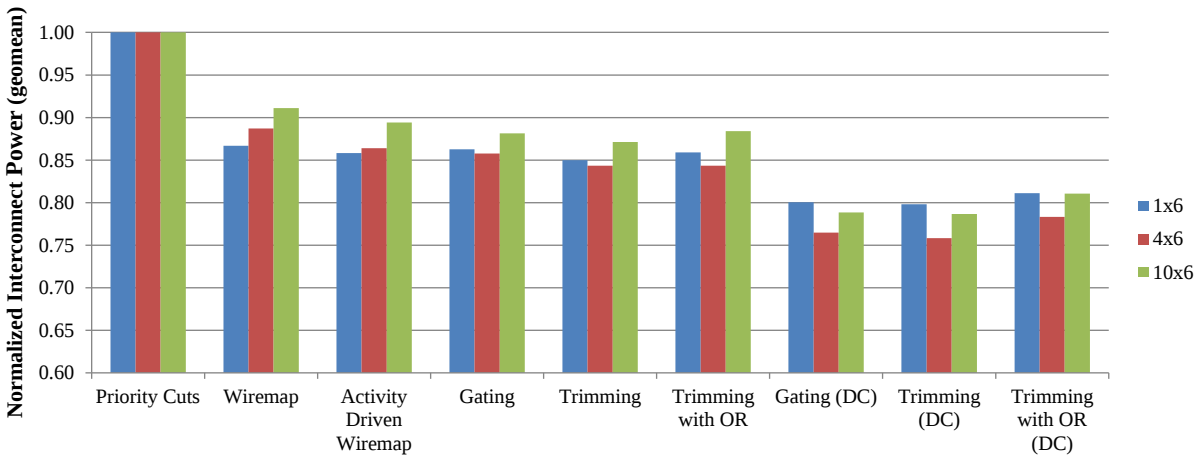
Furthermore, modern FPGA architectures cluster LUT/FF pairs into LBs. Since guarded evaluation reduces the switching activity on wires with the cost of increased fanout of some signals, it is relevant to analyze the impact of this approach on architectures with different cluster sizes. The expectation is that more heavily clustered architectures would benefit from guarding the most since LUTs with identical guards (in effect, shared input signals) would tend to be placed into the same LB. The consequence is that the additional wires added by guarding will not impact inter-clustering routing significantly; i.e., the fanout when measured in terms of the number of logic blocks will not increase as much when guarding is targeted towards heavily clustered architectures.

Figure 4.6(a) and (b) gives the average power consumed in the FPGA interconnect for area-oriented mapping for 6-LUT and 4-LUT architectures of different cluster sizes, respectively. The results consider post-routing interconnect capacitance on architectures with cluster sizes of 1 (flat), 4 and 10. The pattern is similar to that shown when considering total switching activity. The best results are obtained when both gating and trimming inputs were used, with consideration to non-obvious don't-cares, and guarding was done using only `AND` gates. For 6-LUT architectures, figure 4.6(a) shows an average improvement of 14%, 28% and 27% for cluster sizes of 1, 4 and 10, respectively, relative to priority cuts-based mapping. For 4-LUT architectures, figure 4.6(b) shows an average improvement of

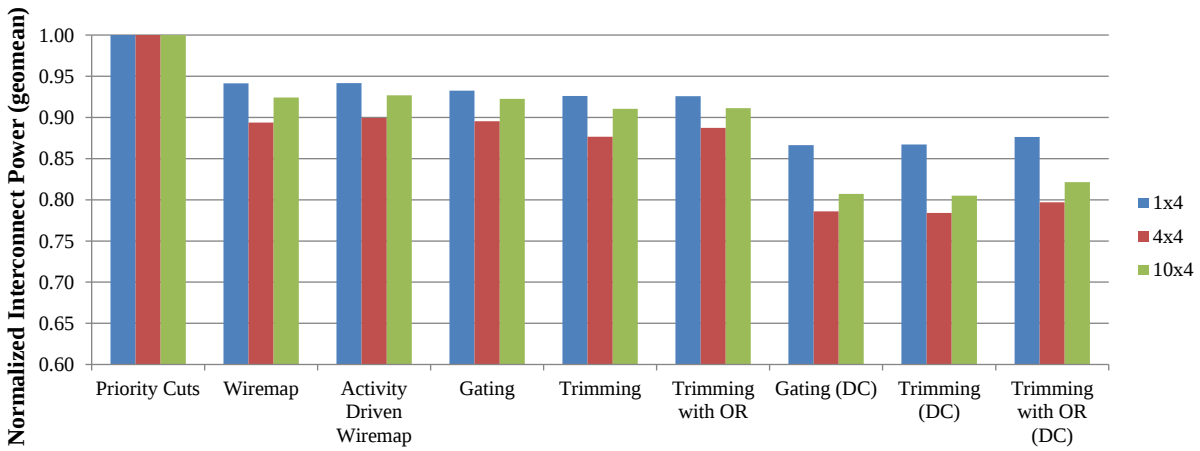
14%, 22% and 21% for cluster sizes of 1, 4 and 10, respectively. From these experimental results, it appears that more heavily clustered architectures benefit the most from guarding. This observation is considered further in Section 4.4.

Figure 4.7(a) and (b) show the results for depth-oriented and depth-relaxed mapping. Once again, the best results are produced when guarding with gating and trimming inputs while considering non-obvious don't cares, which is consistent with the observations made during the investigation of total switching activity. Similar to area-oriented mapping, the most benefit is seen for the more heavily clustered architectures. Specifically, For 6-LUT architectures, figure 4.7(a) shows reductions of 15%, 16% and 15% for clusters sizes of 1, 4 and 10, respectively, relative to priority cuts-based mapping. With depth-relaxation, these results improved to 16%, 20% and 21% for cluster sizes of 1, 4 and 10, respectively For 4-LUT architectures, interconnect power was reduced by 10%, 14% and 14% for cluster sizes of 1, 4 and 10, respectively. Similar to the 6-LUT result, further improvements were obtained using depth relaxation; reductions of 11%, 17%, and 15% were obtained for clusters sizes of 1, 4 and 10, respectively.

For reference, Table 4.1 provides the raw interconnect power results for area-oriented and depth-oriented mappings across the different architectures. Each entry is produced by taking the geometric mean of interconnect power across the 20 benchmark circuits. Appendix A.4 provides the circuit-by-circuit interconnect results for the 4x6 architectures for both area and depth-oriented mappings. Circuit-by-circuit results for 1x6, 10x6, 1x4, 4x4 and 10x4 are omitted for the sake of brevity.

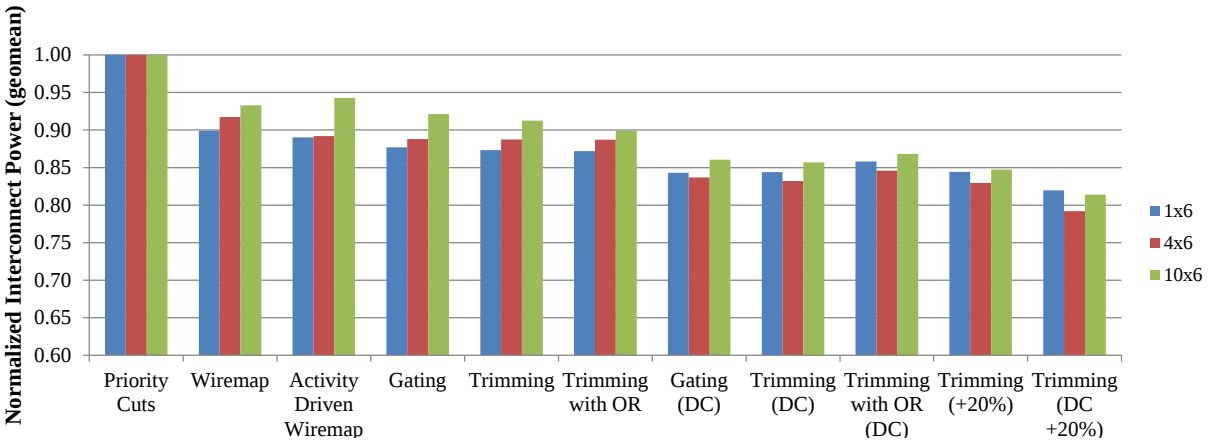


(a) 6-LUT Architectures

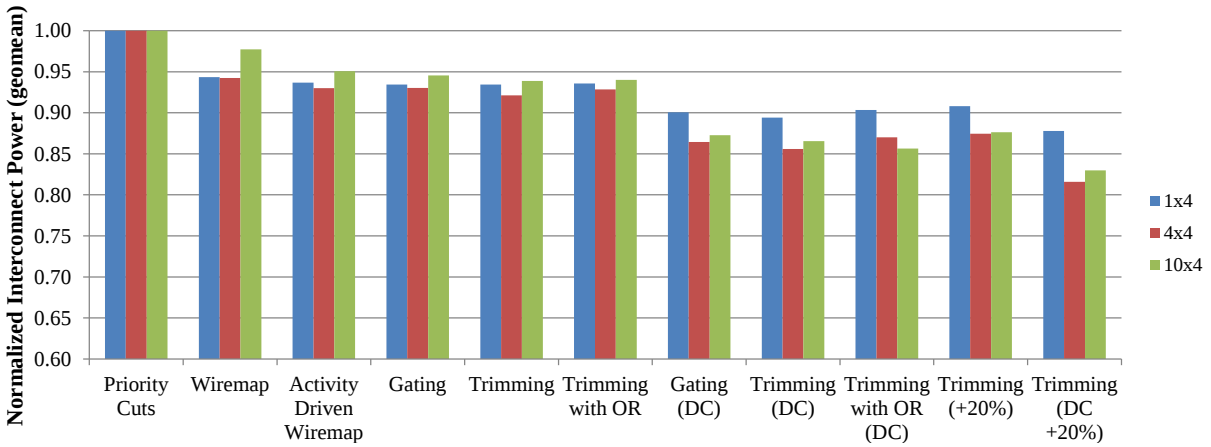


(b) 4-LUT Architectures

Figure 4.6: Normalized Reduction in Interconnect Power for Area-Oriented Mappings



(a) 6-LUT Architectures



(b) 4-LUT Architectures

Figure 4.7: Normalized Reduction in Interconnect Power for Depth-Oriented Mappings

Table 4.1: Power reduction results for different architectures (power given in Watts reported by [26, 42]).

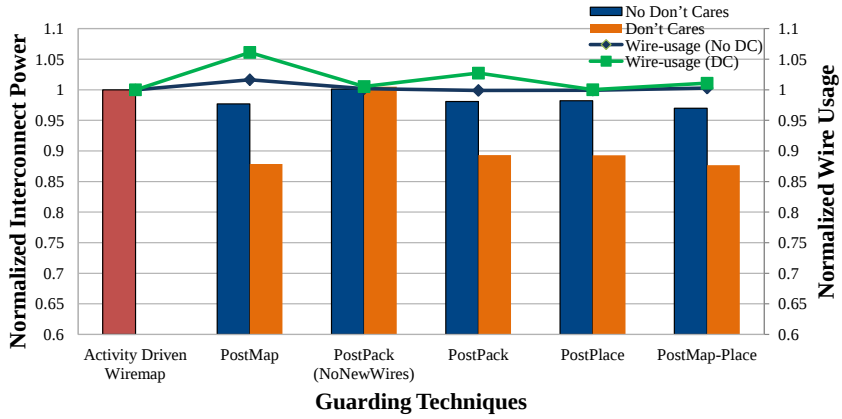
Guarding Strategy	Architecture and Mapping Objective											
	1x4		4x4		10x4		1x6		4x6		10x6	
	Area	Depth	Area	Depth	Area	Depth	Area	Depth	Area	Depth	Area	Depth
Priority Cuts	0.340	0.360	0.116	0.121	0.071	0.073	0.338	0.376	0.116	0.114	0.071	0.071
WireMap	0.316	0.339	0.104	0.114	0.065	0.071	0.313	0.336	0.097	0.106	0.061	0.068
Activity Driven WireMap	0.317	0.335	0.104	0.113	0.065	0.070	0.313	0.333	0.096	0.103	0.060	0.066
Gating	0.316	0.334	0.104	0.112	0.065	0.069	0.311	0.332	0.094	0.102	0.059	0.065
Trimming	0.309	0.329	0.098	0.109	0.063	0.067	0.299	0.325	0.087	0.099	0.055	0.063
Trimming with OR	0.308	0.330	0.099	0.109	0.061	0.068	0.304	0.325	0.089	0.099	0.057	0.062
Gating (DC)	0.295	0.320	0.097	0.105	0.056	0.065	0.291	0.320	0.084	0.096	0.053	0.060
Trimming (DC)	0.294	0.321	0.091	0.104	0.056	0.063	0.291	0.318	0.084	0.096	0.052	0.060
Trimming with OR (DC)	0.297	0.323	0.093	0.107	0.057	0.065	0.294	0.321	0.087	0.100	0.054	0.061
Trimming (+20%)	-	0.329	-	0.106	-	0.066	-	0.320	-	0.095	-	0.061
Trimming (DC +20%)	-	0.319	-	0.101	-	0.062	-	0.316	-	0.091	-	0.056

4.3.1 Guarding Post Packing and Placement

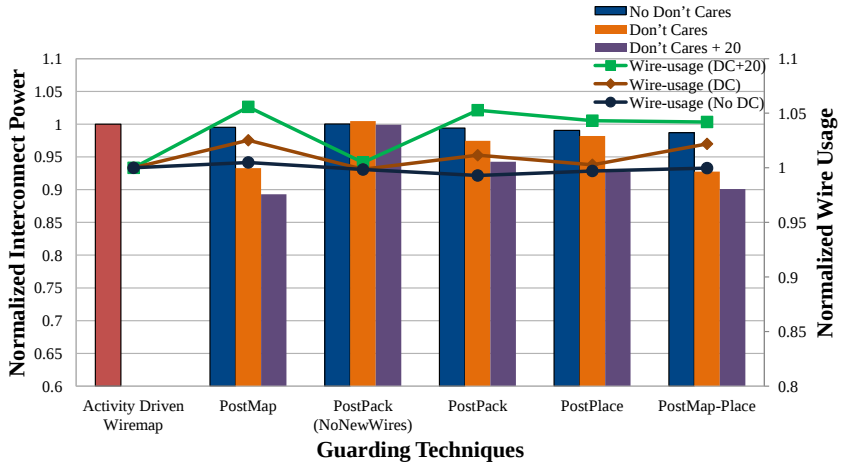
The guarding flows are evaluated on the 4x6 architecture, which gives a good idea of the impact on clustered architectures as seen above. Figure 4.8 shows the average reduction in interconnect power for both area and depth-oriented mappings. For area-oriented, *PostPack*, *PostPlace* and *PostMap-Place* flows have comparable reductions to *PostMap*. Note that this is a slightly different picture than what we saw in the switching activity results. This is because the circuit has been routed and is a complete netlist that can be implemented on an FPGA. As *PostPack* and *PostPlace* attempt to impact the final netlist, the reductions are more visible. Wire usage for the different guarding strategies are also reported. Note that *PostMap* has the most wire-usage, hence the most impact on routing resources. For area-oriented, *PostPlace* is able to achieve 88% of the *PostMap* reduction by inserting only about 35% of the guards. For depth-oriented with relaxation, *PostPlace* is

able to achieve 67% of the reduction through *PostMap*, while inserting 25% of the guards.

This is an important result because it means that a conservative approach to guarding can achieve high power reductions with minimal impact to routability. The next subsection describes the impact to critical path delay which is also an important metric to evaluate the performance of the final netlist.



(a) Area-Oriented Mapping



(b) Depth Oriented Mapping

Figure 4.8: Average reduction in interconnect power for 4x6 architectures

4.3.2 Critical Path Delay

We report the impact of guarded evaluation on post-routed critical path delay (as reported by VPR [9]) to understand the impact of guarding on performance. Table 4.2 shows the geometric mean (across all circuits) of critical path delay for different guarding strategies. Results are presented only for 6-LUT architectures and different cluster sizes; 4-LUT results are similar and are omitted for brevity. Table 4.2(a) shows results for area-oriented mappings. With respect to activity-driven WireMap, the critical path delay is increased, on average, by ∼18% to 20% when using gating inputs, depending on the cluster size. This increases to ∼23% to 30% when using gating and trimming inputs. When non-obvious don't cares can be exploited, critical path delay is further increased with respect to activity-driven WireMap – anywhere from ∼31% to 43% depending on the cluster size. Hence, although guarded evaluation is very effective when applied to area-oriented mapping without any concern for circuit depth, a large performance penalty is incurred.

Table 4.2(b) gives results for several key depth-oriented guarding strategies. When a depth constraint is enforced during guard insertion, the critical path increases only slightly by ∼1% to 3%, on average, with compared to activity-driven WireMap. Some small perturbation is to be expected due to the extra connections added into the network due to the guarding. With depth relaxation (of up to 20%), the critical path increased, on average, anywhere from ∼11% to 17%.

Table 4.3 compares the critical path delay for the different guarding flows described. Results presented are evaluated for Trimming (DC) for area-oriented and Trimming (DC + 20%) for depth-oriented mappings on 4x6 architectures. These strategies were chosen because they have the most impact to critical path delay, as shown in table 4.2. *Post-Pack(NoNewWires)* has virtually no impact on critical path since there are very few new

Guarding Strategy	Normalized Critical Path Delay		
	1x6	4x6	10x6
Activity-Driven WireMap	1.00	1.00	1.00
Gating	1.18	1.20	1.20
Trimming	1.23	1.27	1.30
Gating (DC)	1.31	1.37	1.42
Trimming (DC)	1.31	1.39	1.43

(a) Area-oriented Mapping

Guarding Strategy	Normalized Critical Path Delay		
	1x6	4x6	10x6
Activity-Driven WireMap	1.00	1.00	1.00
Gating	1.01	1.00	1.01
Trimming	1.01	1.00	1.01
Gating (DC)	1.02	1.01	1.02
Trimming (DC)	1.03	1.02	1.02
Gating (DC +20%)	1.14	1.16	1.17
Trimming (DC +20%)	1.11	1.13	1.15

(b) Depth-Oriented Mapping

Table 4.2: Critical path delays for several guarding strategies

wires inserted. *PostPack* has about 26% increase for area-oriented and 11% increase for depth-oriented. There are fewer new wires due to the added feasibility constraint, however since the circuit is placed after the guarding step, impact on the critical path is caused due to the placement solution where some guards may potentially span long distances, which is detrimental to critical path. Since *PostPlace* attempts to encourage new connections that are closer, the impact to critical path is minimized.

It is important to recognize that many FPGA designs do not need to run at the max-

Guarding Flows	Normalized Critical Path Delay	
	Area-oriented	**Depth-oriented**
Activity-Driven Wiremap	1.00	1.00
PostMap	1.39	1.13
PostPack(NoNewWires)	1.00	1.03
PostPack	1.26	1.11
PostPlace	1.10	1.08
PostMap-Place	1.30	1.13

Table 4.3: Critical Path Delay for various flows

imum possible device performance. Despite the reduction in maximum achievable circuit speed, guarded evaluation does indeed produce implementations having lower power. We believe that guarded evaluation is an important power reduction strategy that will be useful in many applications where power consumption is a top tier concern.

In summary, in the 10×6 architecture which aligns closely with the logic block granularity of the the Xilinx Virtex-6 FPGA and Altera Stratix IV FPGA, the "Trimming (DC)" flow with delay-driven mapping provides about 11% reduction in interconnect power, with just 1% increase in critical path delay, on average. Alternatively, the "Trimming (DC + 20%)" flow can be used to achieve 15% power reduction, with a higher, 15% increase in critical path delay. The *PostPack* and *PostPlace* guarding flows had lesser impact on the routing resources and the critical path, while having comparable interconnect power reductions. The different flavors of guarded evaluation thus provide the user with a range of implementation options within the power/speed design space.

4.4 Discussion

There were several interesting results seen during experimentation of the guarded evaluation approach and these results are further investigated in this section.

4.4.1 Use of Gating and Trimming Inputs

The previous numerical results demonstrated that the use of trimming inputs found through the consideration of partial non-inverting paths significantly improved results. Table 4.4 shows results on a per design basis when targeting depth-oriented mapping and 6-LUT architectures. In particular, Table 4.4 shows the number of guarding options computed and the number of actual inserted guards for: 1) guarding using only gating inputs; and 2) guarding using gated and trimming inputs. It can clearly be seen that the use of trimming inputs found through partial non-inverting paths significantly improve the number of guarding candidates and, in turn, results in the insertion of more guards leading to a larger improvement in total switching activity and power dissipation. Although not shown, the results are even more pronounced when non-obvious don't cares are taken into account during guarding.

4.4.2 Use of `OR` Gates as Guard Logic

An apparently counter-intuitive result was observed when attempting to account for the static probability of a signal when inserting guards; recall the intention was to insert either an `AND` gate or an `OR` where appropriate to avoid unnecessary toggles. Counterinituitively, it did not prove effective to use `OR` gates, as demonstrated by the numerical results previously presented. Analysis demonstrated that the insertion of an `OR` gate (when appropriate)

Table 4.4: Number of guarding options and inserted guards for two different guarding strategies.

Design	Gating		Trimming	
	Options	**Inserted**	**Options**	**Inserted**
alu4	3880	86	4071	95
apex2	6720	34	6873	36
apex4	1862	8	1866	10
bigkey	0	0	0	0
clma	5965	243	10403	843
des	153	6	184	9
diffeq	8	1	17	3
dsip	224	1	224	1
elliptic	658	49	658	49
ex1010	12125	15	12125	15
ex5p	812	15	812	15
frisc	641	15	662	18
misex3	3221	48	3270	54
pdc	9169	48	9312	72
s298	1252	18	1527	36
s38417	656	71	767	84
s38584.1	356	36	584	68
seq	2027	27	2111	34
spla	7192	51	7334	86
tseng	11	4	21	9
Average	2846.6	38.8	3141.1	76.9

based on static probability was having a positive effect, but *only on a local level*. In other words, the selection of either an `AND` gate or an `OR` based on static probability resulted in reduced switching activity for the *current signal being guarded*.

Figure 4.9 shows the results of an experiment where the static probability of the guarded LUT was considered and if $P_l > 0.5$, it was guarded with an `OR` gate and activity of the

signal was examined. Subsequently, it was guarded with an AND gate. The experiment was performed across all 20 MCNC circuits, where only one guard was considered to localize the effect of guarding. Guarding with consideration of static probability was able to reduce the corresponding signal activity by 29%, while without consideration, it was able to reduce activity by 27%. This 2% additional reduction occurs because the static probability of guarded LUTs were an average of 69% (i.e. signals are logic-1 more often than they are logic-0). Notice that by guarding with an AND gate, this reduces to 28%, while guarding with an OR gate increased it to 78%. These results intuitively make sense.

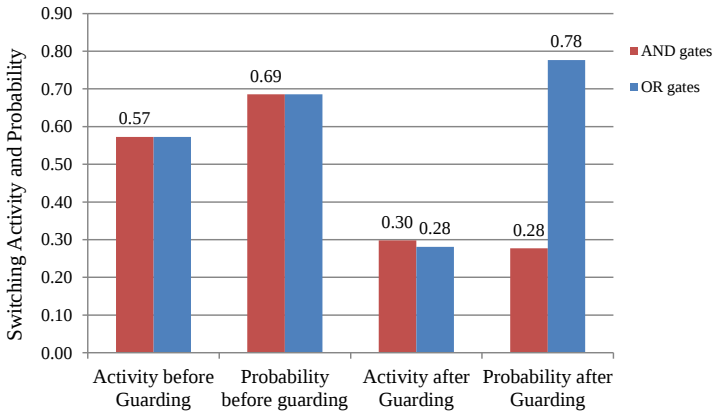


Figure 4.9: Comparing AND gating and OR gating for local signals.

However, further investigation showed that the use of OR gates resulted in fewer total inserted guards. Table 4.5 shows the number of guards inserted when signal probabilities are taken into account; these results are presented for 6-LUT architectures and depth-oriented mappings. In almost all cases, accounting for signal probabilities and choosing an appropriate gate type (AND or OR) resulted in fewer inserted guards when compared to simply inserting an AND gate and forcing a signal to logic-0. In the course of the algorithm, we observed that the insertion of OR gates was creating a different ranking of guarding

Table 4.5: Comparison of the number of inserted guards using gating and trimming inputs when using only `AND` gates versus using `AND` gates or `OR` gates to guard.

Design	**Trimming**	**Trimming with OR**	**Trimming (DC)**	**Trimming with OR (DC)**
alu4	95	75	187	153
apex2	36	23	115	117
apex4	10	9	47	40
bigkey	0	0	0	0
clma	843	505	790	491
des	9	5	37	28
diffeq	3	3	81	46
dsip	1	1	1	1
elliptic	49	49	83	50
ex1010	15	95	95	148
ex5p	15	11	64	62
frisc	18	196	125	252
misex3	54	43	173	158
pdc	72	93	213	210
s298	36	69	96	155
s38417	84	88	128	142
s38584.1	68	69	105	131
seq	34	23	105	111
spla	86	87	209	170
tseng	9	2	64	24
Average	76.9	72.3	136	124.5

options resulting in a different order in which guards were inserted and free LUT inputs were "used up". It is a possibility that a different benchmark suite or a different scoring function for guarding candidates would have resulted in a different outcome.

4.4.3 Architectural Analysis

Typically, clustered architectures tended to benefit more from the application of guarded evaluation (c.f. Figures 4.6 and 4.7). To better understand this phenomenon, we conducted an additional investigation. Despite not impacting the LUT count of the mapped network, guarded evaluation does require the insertion of additional signals. Consequently, guarded evaluation results in networks with additional connections that require routing. For example, if we refer to Table 4.4, we see that guarding using only gating inputs (and depth-oriented mapping) required, on average, an additional 38.8 connections per design for 6-LUT architectures. Although not shown, guarding using only gating inputs (and depth-oriented mapping) required, on average, an additional 45.3 connections per design when considering 4-LUT architectures.

We considered *how* these additional connections were routed. In an architecture with a cluster size of 1 (i.e., flat), these connections must be routed through global interconnect; i.e., inter-cluster routing. However, in a clustered architecture, these additional signals could be *absorbed* into the clusters; i.e., only intra-cluster routing is required. Clearly, when inter-cluster routing is required, the additional power consumption of these additional connections could out-weigh the benefits of guarding due to the increased capacitance of inter-cluster versus intra-cluster routing.

Table 4.6 shows the average number of additional connections added to the networks due to guarding (using only gating inputs) after clustering is performed and LUTs are packed into LBs. In other words, Table 4.6 shows the actual number of additional connections which require inter-cluster routing. Clearly, for flat architectures, the average number of extra inter-cluster signals is equal to the average number of guards inserted. But, for clustered architectures, the average number of inter-cluster signals is reduced. Referring

Table 4.6: Increase of the average number of additional connections that require inter-cluster routing due to guarding (depth-oriented mapping using only gating inputs for guarding).

Architecture	**Avg Num (Guards)**	**Avg Num (Inter-cluster) Signals**
1x6	38.8	38.8
4x6	38.8	33.5
10x6	38.8	33.1
1x4	45.3	45.3
4x4	45.3	11.6
10x4	45.3	12.1

back to figures 4.6 and 4.7 we can observe this trend; when those connections inserted due to guarding are effectively absorbed into the LBs, more significant power reductions are observed when compared to a flat architecture.

Furthermore, *PostPack* and *PostPlace* flows would benefit more with heavily clustered architectures since the increased number of intra-cluster signals would result in an increased number of feasible guards. Hence, more guarding can be done, while adding minimal inter-cluster wires in the *PostPack(NoNewWires)*, *PostPack* and *PostPlace* flows.

Chapter 5

Conclusions and Future Work

Guarded evaluation reduces dynamic power by identifying sub-circuits whose inputs can be held constant at certain times during circuit operation, eliminating toggles within the sub-circuits. We have proposed the adaption of guarded evaluation to make it suitable for FPGAs. Specifically, we have shown that guarding can be applied after technology mapping without any increase to the overall area (measured in terms of the number of LUTs) of the network; it is only necessary to add extra connections into the network in order to perform the guarding. Increases in area are avoided by exploiting the availability of unused inputs on LUTs and the existing circuitry inside the LUTs to perform guarding. Numerical results demonstrate the efficacy of our proposed techniques and show that guarded evaluation is effective for FPGA designs.

Additionally, we have proposed a *structural* technique to identify guarding candidates based on the ideas of *non-inverting* and *partial non-inverting paths*; the use of partial non-inverting paths was demonstrated to significantly improve the availability of guarding options and, in turn, improve the reduction in both total reduction in total switching

activity and reduction in total dynamic power dissipation. Finally, we considered the impact of post-mapped guarded evaluation on different FPGA architectures. We discovered that, more often than not, guarded evaluation was most effective for clustered architectures. Analysis demonstrated that this was due to the guarding signals being "absorbed" into the logic block clusters.

Guarded evaluation was implemented after various steps in the traditional FPGA CAD flow to leverage the additional information. Guarding after the packing step allowed for the algorithm to recognize the difference between inter-cluster and intra-cluster guards, which was used to encourage guards that have minimal impact on routing resources. Guarding after the placement step provided additional information, which was used to encourage inter-cluster signals of smaller distances. Results show that while fewer guards are inserted, there is comparable interconnect power reduction with minimal impact to routing resources.

5.1 Future Work

Possible directions for future work would include the consideration of alternative scoring schemes to improve the guard selection. For example, the *PostPlace* scoring function is currently used to rank the guarding options such that shorter guards are encouraged. This can be modified such that options that do not meet a certain *PlaceScore* threshold are discarded. Although the structural identification of guards is extremely fast, the scalability of the algorithm — both without and with don't cares — can be improved. Looking for additional techniques for easily identifying guarding candidates beyond the use of non-inverting and partial non-inverting paths such as simulation or boolean satisfiability checking [18]. More guarding candidates, for example, might serve to improve guarding results during depth-oriented mapping when many candidates are discarded due to violation of depth

constraints.

Lastly, it would be valuable to consider a newer version of VPR [31, 32] once power calculation tools are available for these versions of VPR. This would enable an investigation of guarded evaluation on even more realistic FPGA architectures (such as those supporting fracturable LUTs) as well as enable larger sets of realistic designs to be considered. Various architectural factors might require additional modifications to the scoring function and the decision as to whether or not a guard should be inserted.

References

[1] Berkeley logic synthesis and verification group, ABC – a system for sequential synthesis and verification. *http://www.eecs.berkeley.edu/∼alanmi/abc/*, 2009.

[2] A. Abdollahi, M. Pedram, F. Fallah, and I. Ghosh. Precomputation-based guarding for dynamic and leakage power reduction. In *IEEE Int'l Conf. on Computer Design*, pages 90–97, 2003.

[3] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicro FPGA performance and density. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 85–94, Monterey, CA, 2002.

[4] Altera, Corp., San Jose, CA. *Stratix-III FPGA Family Data Sheet*, 2008.

[5] J. Anderson and C. Ravishankar. FPGA power reduction by guarded evaluation. In *Proc. FPGA*, pages 157–166, 2010.

[6] J. Anderson and Q. Wang. Improving logic density through synthesis-inspired architecture. In *IEEE International Conference on Field Programmable Logic and Applications*, pages 105 – 111, Prague, Czech Republic, 2009.

[7] J. Anderson and Q. Wang. Area-efficient FPGA logic elements: Architecture and synthesis. In *IEEE/ACM Asia and South Pacific Design Automation Conference*, pages 369–375, 2011.

[8] J. H. Anderson and F.N. Najm. Power-aware technology mapping for LUT-based fpgas. In *IEEE International Conference on Field-Programmable Technology*, pages 211–218, Hong Kong, 2002.

[9] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Int'l Workshop on Field Programmable Logic and Applications*, pages 213–222, 1997.

[10] S.D. Brown. An overview of technology, architecture and cad tools for programmable logic devices. In *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994*, pages 69 –76, may 1994.

[11] B.H. Calhoun, A. Wang, and A. Chandrakasan. Device sizing for minimum energy operation in subthreshold circuits. In *IEEE Custom Integrated Circuits Conference*, pages 95–98, Orlando, FL, 2004.

[12] S. Chatterjee, A. Mishcenko, R. Brayton, X. Wang, and T. Kam. Reducing structural bias in technology mapping. In *Int'l Workshop on Logic Synthesis*, 2005.

[13] J. Cong, C. Wu, and E. Ding. Cut ranking and pruning: Enabling A general and efficient FPGA mapping solution. In *Int'l Symp. on Field-Programmable Gate Arrays*, pages 29–35, 1999.

[14] M.E. Dehkordi and S.D. Brown. The effect of cluster packing and node duplication control in delay driven clustering. In *IEEE International Conference on Field-Programmable Technology*, pages 227–233, Hong Kong, 2002.

[15] J. Greene, E. Hamdy, and S. Beal. Antifuse field programmable gate arrays. *Proceedings of the IEEE*, 81(7):1042 –1056, jul 1993.

[16] D. Howland and R. Tessier. RTL dynamic power optimization for FPGAs. In *IEEE Midwest Symp. on Circuits and Systems*, pages 714–717, 2008.

[17] H.-C. Hsieh, K. Dong, J.Y. Ja, R. Kanazawa, L.T. Ngo, L.G. Tinkey, W.S. Carter, and R.H. Freeman. A 9000-gate user-programmable gate array. In *Custom Integrated Circuits Conference, 1988., Proceedings of the IEEE 1988*, pages 15.3/1 –15.3/7, may 1988.

[18] Aaron P. Hurst. *Sequential Optimization for Low Power Digital Design.* PhD thesis, EECS Department, University of California, Berkeley, May 2008.

[19] M. Hutton, J. Schleicher, D. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, and et. el. Improving FPGA performance and area using an adaptive logic module. In *International Conference on Field-Programmable Logic and Applications*, pages 135–144, Antwerp, Belgium, 2004.

[20] H.Veendrick. *Deep Submicron CMOS ICs.* Kluwer Academic, 1998.

[21] S. Jang, B. Chan, K. Chung, and A. Mishchenko. Wiremap: FPGA technology mapping for improved routability and enhanced LUT merging. *ACM Trans. on Reconfigurable Technology and Systems*, 2(2):1–24, 2009.

[22] S. Jang, K. Chung, A. Mishchenko, and R. Brayton. A power optimization toolbox for logic synthesis and mapping. In *IEEE International Workshop on Logic Synthesis*, San Francisco, CA, 2009.

[23] M. Ketkar and S. S. Sapatnekar. Standby power optimization via transistor sizing and dual threshold voltage assignment. In *IEEE International Conference on Computer-Aided Design*, pages 375–378, San Jose, CA, 2002.

[24] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Trans. On CAD*, 26(2):203–215, February 2007.

[25] J. Lamoureux, G. Lemieux, and S. Wilton. GlitchLess: an active glitch minimization technique in FPGAs. In *ACM/SIGDA Int'l Symposium on Field Programmable Gate Arrays*, pages 156–165, Monterey, CA, 2007.

[26] J. Lamoureux and S.J.E. Wilton. On the interaction between power-aware FPGA CAD algorithms. In *IEEE/ACM Int'l Conf. on Computer-Aided Design*, pages 701–708, 2003.

[27] D. Lee and D. Blaauw. Static leakage reduction through simultaneous threshold voltage and state assignment. In *ACM/IEEE Design Automation Conference*, pages 191–194, Anaheim, CA, 2003.

[28] Frank Thomson Leighton. *Complexity issues in VLSI: optimal layouts for the shuffle-exchange graph and other networks*. MIT Press, Cambridge, MA, USA, 1983.

[29] David Lewis and et al. The stratix II logic and routing architecture. In *ACM Int'l Symp. on FPGAs*, pages 14–20, 2005.

[30] A. Ling, J. Zhu, and S. Brown. Delay driven AIG restructuring using slack budget management. In *ACM/IEEE Great Lakes Symp. on VLSI*, pages 163–166, 2008.

[31] J. Luu, J. Anderson, and J. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *FPGA*, pages 227–236, 2011.

[32] J. Luu and et al. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *FPGA*, pages 133–142, 2009.

[33] A. Marquardt, V. Betz, and J. Rose. Using cluster based logic blocks and timing-driven packing to improve FPGA speed and density. In *International Symposium on Field-Programmable Gate Arrays*, pages 37–46, Monterey, CA, 1999.

[34] A. Marquardt, V. Betz, and J. Rose. Speed and area tradeoffs in cluster-based fpga architectures. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(1):84 –93, feb. 2000.

[35] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *ACM Int'l Symp. on Field-Programmable Gate Arrays*, pages 203–213, 2000.

[36] S.M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *IEEE International Conference on Computer-Aided Design*, pages 721–725, San Jose, CA, 2002.

[37] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. Scalable don't-care-based logic optimization and resynthesis. In *Proc. FPGA*, pages 151–160, 2009.

[38] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *ACM/IEEE Design Automation Conf.*, pages 532–536, 2006.

[39] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *Proc. ICCAD*, pages 354–361, 2007.

[40] S.R. Naidu and E.T.A.F Jacobs. Minimizing standby leakage power in static CMOS circuits. In *ACM/IEEE Design Automation and Test in Europe Conference*, pages 370–376, Munich, Germany, 2001.

[41] F. Najm. Transition density: A new measure of activity in digital circuits. *IEEE Trans. on CAD*, 12:310–323, February 1993.

[42] K. Poon, A. Yan, and S. Wilton. A flexible power model for FPGAs. In *Int'l Conf. on Field-Programmable Logic and Applications*, pages 312–321, 2002.

[43] S.T. Rajavel and A. Akoglu. Mo-pack: Many-objective clustering for FPGA CAD. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 818 –823, june 2011.

[44] M. Schlag, J. Kong, and P.K. Chan. Routability-driven technology mapping for lookup table-based FPGAs. *IEEE Trans. on CAD*, 13(1):13–26, 1994.

[45] L. Shang, A. Kaviani, and K. Bathala. Dynamic power consumption of the Virtex-II FPGA family. In *ACM Int'l Symp. on Field-Programmable Gate Arrays*, 2002.

[46] A. Singh and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in fpgas. In *ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2002.

[47] A. Srivastava, D. Sylvester, and D. Blaauw. Power minimization using simultaneous gate sizing, dual-vdd and dual-vth assignment. In *ACM/IEEE Design Automation Conference*, pages 783–787, San Diego, CA, 2004.

[48] V. Tiwari, S. Malik, and P. Ashar. Guarded evaluation: pushing power management to logic synthesis/design. *IEEE Trans. on CAD*, 17(10):1051–1060, October 1998.

[49] Jeffrey D Ullma. *Computational Aspects of VLSI*. W. H. Freeman & Co., New York, NY, USA, 1984.

[50] K. Vorwerk, M. Raman, J. Dunoyer, Y.-C. Hsu, A. Kundu, and A. Kennings. A technique for minimizing power during FPGA placement. In *IEEE International Conference on Field Programmable Logic and Applications*, pages 233–238, Heidelberg, Germany, 2008.

[51] Xilinx, Inc., San Jose, CA. *Virtex-5 FPGA Data Sheet*, 2007.

[52] Xilinx, Inc., San Jose, CA. *Virtex-7 FPGA Data Sheet*, 2012.

[53] S. Yang. Logic synthesis and optimization benchmarks. version 3.0. Technical report, Microelectronics Center of North Carolina, 1991.

[54] Wei Zhao and Yu Cao. New generation of predictive technology model for sub-45 nm early design exploration. *Electron Devices, IEEE Transactions on*, 53(11):2816 –2823, nov. 2006.

APPENDICES

Appendix A

Circuit-by-circuit results for 6-LUT Architectures

Table A.1: Switching activity reduction results for 6-LUT area-oriented mappings.

Circuit	Priority Cuts	WireMap	Activity Driven WireMap	Gating	Trimming	Trimming with OR	Gating (DC)	Trimming (DC)	Trimming with OR (DC)
alu4	136.86	133.87	126.50	106.66	108.55	114.20	86.22	84.52	97.64
apex2	49.46	48.37	45.61	39.51	39.77	44.71	26.63	25.59	30.18
apex4	88.06	55.96	50.49	49.27	49.97	52.73	42.96	42.44	45.05
bigkey	219.98	223.58	223.82	223.82	223.82	223.82	223.82	223.82	223.82
clma	701.8	687.43	666.43	555.99	289.26	382.74	292.82	270.18	391.48
des	210.27	211.95	232.51	232.02	232.36	233.06	230.06	229.30	230.01
diffeq	242.29	250.77	253.89	253.89	248.50	248.50	251.30	238.46	247.84
dsip	260.17	261.92	261.92	261.89	253.10	253.13	261.89	261.89	261.97
elliptic	693.05	678.97	682.80	683.00	677.95	677.95	682.19	682.58	682.30
ex1010	219.88	111.47	91.21	88.95	83.63	73.81	73.38	73.38	76.92
ex5p	100.81	86.07	79.49	76.41	67.95	69.96	67.23	67.23	70.35
frisc	569.32	505.37	488.12	485.20	459.17	452.78	478.12	476.43	454.52
misex3	86.19	82.09	79.98	76.38	77.31	81.09	65.32	62.36	64.32
pdc	164.39	114.66	95.05	87.72	86.78	89.78	53.29	50.79	66.49
s298	80.47	75.91	73.92	72.43	68.05	68.00	62.64	57.90	58.57
s38417	740.14	798.36	802.79	799.78	794.92	787.83	796.99	795.57	791.77
s38584.1	744.16	675.73	674.81	671.86	668.85	662.26	671.46	668.59	660.72
seq	82.02	77.48	71.66	66.55	69.62	74.85	46.32	44.90	51.30
spla	190.19	154.84	131.25	118.63	115.35	130.42	90.67	84.88	113.70
tseng	248.43	248.09	247.98	247.82	238.30	232.23	245.72	245.38	246.21
Geomean	209.27	187.64	179.10	171.02	162.64	167.81	145.85	142.33	153.80
Ratio	1.00	0.90	0.86	0.82	0.78	0.80	0.70	0.68	0.73
Ratio			1.00	0.95	0.91	0.94	0.81	0.79	0.86

Table A.2: Switching activity reduction results for 6-LUT depth-oriented mappings.

Circuit	Priority Cuts	WireMap	Activity Driven WireMap	Gating	Trimming	Trimming with OR	Gating (DC)	Trimming (DC)	Trimming with OR (DC)
alu4	144.35	147.12	124.59	121.93	119.37	122.28	115.07	111.09	113.57
apex2	56.26	54.53	47.46	45.39	44.84	47.55	37.46	35.32	40.95
apex4	98.73	58.62	54.37	53.28	52.95	58.40	52.59	52.46	55.16
bigkey	219.98	223.58	223.82	223.82	223.82	223.82	223.82	223.82	223.82
clma	685.31	688.05	695.13	601.22	432.92	538.79	347.17	333.66	416.01
des	272.98	262.79	256.77	255.28	255.22	256.22	255.85	253.35	254.37
diffeq	249.60	249.35	259.82	259.82	254.98	254.70	247.78	245.25	253.04
dsip	260.17	261.92	261.92	261.92	253.13	253.13	261.92	261.92	261.92
elliptic	692.56	697.87	708.61	707.54	705.60	705.49	706.44	705.85	706.50
ex1010	127.09	91.32	96.42	94.69	91.49	94.09	90.74	90.65	94.87
ex5p	102.07	98.67	85.08	82.76	71.70	73.54	78.38	78.38	81.54
frisc	562.33	512.07	489.53	486.60	482.34	468.51	477.42	477.21	456.48
misex3	92.17	91.00	83.21	80.99	80.31	82.63	75.20	73.19	76.93
pdc	138.38	129.19	111.29	105.19	96.71	112.42	93.69	85.81	99.18
s298	80.72	82.78	76.67	75.22	74.85	74.55	73.98	67.21	69.54
s38417	759.56	806.74	819.83	817.04	816.31	811.65	813.64	813.01	808.31
s38584.1	751.54	681.06	687.20	683.54	685.97	680.99	683.11	678.26	667.94
seq	84.08	86.28	85.07	82.40	82.22	87.72	69.42	64.59	72.64
spla	174.35	179.97	144.87	136.12	130.72	142.85	115.45	107.47	135.55
tseng	249.21	247.25	253.87	252.97	237.17	237.18	251.09	250.73	252.24
Geomean	208.12	197.57	188.22	183.57	176.26	182.93	170.05	165.67	174.87
Ratio	1.00	0.95	0.90	0.88	0.85	0.88	0.82	0.80	0.84
Ratio			1.00	0.98	0.94	0.97	0.90	0.88	0.93

Table A.3: Switching activity reduction results for 6-LUT depth-oriented mappings with depth-relaxation

Circuit	**Trimming** (+20%)	**Trimming (DC)** (+20%)
alu4	107.32	96.27
apex2	36.31	30.32
apex4	51.48	46.94
bigkey	223.82	223.82
clma	379.43	292.68
des	252.65	251.91
diffeq	242.94	241.33
dsip	261.89	261.89
elliptic	706.28	706.01
ex1010	90.33	76.49
ex5p	81.46	72.10
frisc	485.61	476.69
misex3	68.96	67.53
pdc	78.83	77.02
s298	67.06	65.37
s38417	813.43	808.91
s38584.1	678.87	676.17
seq	58.67	52.01
spla	109.27	91.38
tseng	251.56	250.77
Geomean	165.02	154.33
Ratio	0.79	0.74
Ratio	0.88	0.82

Table A.4: Interconnect Power reduction results on the 4x6 architecture for area-oriented mapping

Circuit	Priority Cuts	WireMap	Activity Driven WireMap	Gating	Trimming	Trimming with OR	Gating (DC)	Trimming (DC)	Trimming with OR (DC)
alu4	0.114	0.093	0.089	0.062	0.059	0.059	0.052	0.055	0.057
apex2	0.122	0.083	0.078	0.058	0.056	0.057	0.050	0.048	0.049
apex4	0.100	0.087	0.084	0.076	0.070	0.076	0.058	0.057	0.062
bigkey	0.241	0.246	0.227	0.227	0.241	0.241	0.227	0.227	0.227
clma	0.218	0.195	0.192	0.137	0.119	0.137	0.107	0.103	0.125
des	0.226	0.216	0.223	0.207	0.201	0.204	0.187	0.198	0.216
diffeq	0.226	0.059	0.063	0.062	0.063	0.063	0.061	0.057	0.059
dsip	0.062	0.267	0.271	0.249	0.238	0.260	0.249	0.249	0.249
elliptic	0.179	0.162	0.170	0.158	0.161	0.163	0.159	0.159	0.180
ex1010	0.327	0.253	0.289	0.239	0.252	0.218	0.210	0.210	0.200
ex5p	0.079	0.067	0.064	0.060	0.050	0.047	0.041	0.041	0.041
frisc	0.165	0.133	0.131	0.129	0.131	0.133	0.116	0.115	0.104
misex3	0.094	0.083	0.074	0.056	0.051	0.058	0.042	0.045	0.045
pdc	0.251	0.230	0.224	0.197	0.183	0.160	0.159	0.148	0.156
s298	0.054	0.052	0.050	0.039	0.035	0.034	0.035	0.034	0.031
s38417	0.226	0.233	0.233	0.172	0.169	0.181	0.173	0.168	0.170
s38584.1	0.311	0.278	0.271	0.271	0.275	0.273	0.252	0.254	0.252
seq	0.106	0.078	0.079	0.071	0.067	0.067	0.049	0.042	0.044
spla	0.221	0.177	0.186	0.162	0.151	0.145	0.121	0.125	0.122
tseng	0.073	0.059	0.059	0.058	0.063	0.062	0.053	0.053	0.051
Geomean	0.148	0.131	0.130	0.113	0.108	0.109	0.097	0.096	0.097
Ratio	1.000	0.881	0.875	0.759	0.731	0.736	0.653	0.645	0.657
Ratio			1.000	0.868	0.836	0.841	0.746	0.737	0.751

Table A.5: Interconnect Power reduction results on the 4x6 architecture for depth-oriented mapping

Circuit	Priority Cuts	WireMap	Activity Driven WireMap	Gating	Trimming	Trimming with OR	Gating (DC)	Trimming (DC)	Trimming with OR (DC)
alu4	0.129	0.127	0.106	0.112	0.077	0.084	0.107	0.105	0.104
apex2	0.148	0.126	0.105	0.100	0.103	0.098	0.099	0.096	0.099
apex4	0.130	0.117	0.110	0.111	0.093	0.095	0.108	0.109	0.104
bigkey	0.241	0.246	0.227	0.227	0.241	0.241	0.227	0.227	0.227
clma	0.320	0.267	0.252	0.239	0.244	0.252	0.198	0.198	0.241
des	0.254	0.260	0.255	0.241	0.236	0.234	0.226	0.249	0.244
diffeq	0.101	0.094	0.093	0.089	0.095	0.096	0.084	0.084	0.086
dsip	0.263	0.266	0.271	0.271	0.262	0.262	0.271	0.271	0.271
elliptic	0.227	0.226	0.220	0.233	0.230	0.232	0.193	0.179	0.213
ex1010	0.368	0.357	0.338	0.322	0.328	0.311	0.321	0.320	0.317
ex5p	0.093	0.078	0.073	0.071	0.063	0.062	0.070	0.070	0.071
frisc	0.226	0.198	0.193	0.190	0.199	0.186	0.178	0.180	0.180
misex3	0.134	0.118	0.110	0.107	0.098	0.095	0.098	0.100	0.101
pdc	0.310	0.289	0.260	0.252	0.234	0.225	0.248	0.240	0.258
s298	0.070	0.066	0.064	0.065	0.058	0.057	0.063	0.062	0.060
s38417	0.324	0.310	0.315	0.314	0.321	0.327	0.308	0.294	0.299
s38584.1	0.406	0.377	0.385	0.370	0.360	0.363	0.377	0.364	0.393
seq	0.138	0.116	0.114	0.115	0.109	0.113	0.106	0.108	0.109
spla	0.271	0.252	0.206	0.215	0.197	0.189	0.211	0.209	0.194
tseng	0.092	0.072	0.076	0.077	0.080	0.189	0.072	0.069	0.075
Geomean	0.188	0.173	0.163	0.162	0.155	0.161	0.153	0.153	0.157
Ratio	1.000	0.918	0.869	0.862	0.824	0.856	0.815	0.815	0.836
Ratio			1.000	0.992	0.949	0.986	0.938	0.938	0.962

Table A.6: Interconnect Power reduction results on the 4x6 architecture for depth-oriented mappings with depth-relaxation

Circuit	**Trimming** (+20%)	**Trimming (DC)** (+20%)
alu4	0.088	0.081
apex2	0.089	0.079
apex4	0.101	0.077
bigkey	0.227	0.227
clma	0.187	0.175
des	0.219	0.216
diffeq	0.082	0.075
dsip	0.249	0.249
elliptic	0.193	0.167
ex1010	0.310	0.297
ex5p	0.064	0.059
frisc	0.164	0.170
misex3	0.090	0.084
pdc	0.210	0.216
s298	0.055	0.054
s38417	0.273	0.269
s38584.1	0.317	0.289
seq	0.097	0.087
spla	0.177	0.173
tseng	0.066	0.065
Geomean	0.141	0.133
Ratio	0.750	0.707
Ratio	0.863	0.814