

FPGA Power Reduction by Guarded Evaluation Considering Logic Architecture

Chirag Ravishankar, *Student Member, IEEE*, Jason H. Anderson, *Member, IEEE*, and Andrew Kennings, *Member, IEEE*

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 ASICs; in this paper, 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 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” and “partial non-inverting” paths in a circuit’s AND-inverter graph representation. Experimental results show that guarded evaluation can reduce switching activity on average by as much as 32% and 25% for 6-LUT and 4-LUT architectures, respectively. Dynamic power consumption in the FPGA interconnect is reduced on average by as much as 24% and 22% for 6-LUT and for 4-LUT architectures, respectively. The impact to critical path delay ranges from 1% to 43%, depending on the guarding scenario and the desired power/delay trade-off.

Index Terms—Field-programmable gate arrays, FPGAs, power, optimization, low-power design, logic synthesis, technology mapping.

I. INTRODUCTION

Modern FPGAs are widely used in diverse applications, ranging from communications infrastructure, automotive, to industrial electronics. They enable innovation across a broad spectrum of digital hardware applications, as they reduce product cost, time-to-market, and mitigate risk. 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 [15].

C. Ravishankar and A. Kennings are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada.

J. Anderson is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada.

Copyright (c) 2012 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org.

Recent years have seen intensive research activity on reducing FPGA power through innovations in CAD, architecture, and circuits. In this paper, we attack FPGA dynamic power consumption in the logic synthesis stage of the CAD flow using an approach known as *guarded evaluation*, which has been used successfully in the custom ASIC domain [26]. Recall that dynamic power in a CMOS circuit is defined by: $P_{avg} = \frac{1}{2} \sum_{i \in \text{nets}} C_i \cdot f_i \cdot V^2$, 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. 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 [25], 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 paper, 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 and dynamic power. 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, and we use the available free inputs on LUTs for guarded evaluation. This implies that we do not add in any additional LUTs when implementing 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-based 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) to reduce unnecessary transient switching.

A preliminary version of a portion of this work appeared in [6]. In this extended journal version, we describe an additional form of guarding that provides improved results, namely, guarding opportunities that arise from partial non-inverting paths in the AIG. We also consider the consequences of forcing guarded signals into the logic-1 state, rather than solely forcing to the logic-0 state. Finally, we consider a variety of different FPGA logic block architectures beyond that considered in the conference version. 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.

The remainder of the paper is organized as follows: Section II presents background and related work on technology mapping for FPGAs, power optimization, and describes guarded evaluation in the ASIC context. The proposed approach is described in Section III. An experimental study appears in Section IV. Conclusions and suggestions for future work are offered in Section V.

II. BACKGROUND

A. FPGA Technology Mapping

Here we review the approach used by modern FPGA technology mappers, which are based on finding cuts in Boolean networks [24], [11]. 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).

Fig. 1(a) illustrates cuts for a node x in a circuit graph. A cut for x is a partition, (V, \bar{V}) , of the nodes in the subgraph rooted at x , such that $x \in \bar{V}$. For x ’s cut C_1 in Fig. 1(a), \bar{V} consists of two nodes, x and m . For x ’s cut C_2 in the figure, \bar{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 \bar{V} is less than or equal to K . In the case of cut C_1 , there are 3 nodes that drive nodes in \bar{V} and, the cut is 3-feasible. For a cut $C = (V, \bar{V})$, $\text{Inputs}(C)$

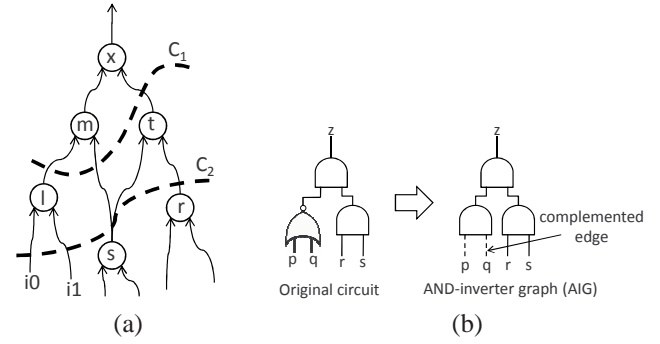


Fig. 1: (a) Cuts in circuit graph; (b) And-Inverter Graph (AIG) example.

represents the nodes in V that drive a node in \bar{V} . For the cut C_1 in Fig. 1(a), $\text{Inputs}(C_1) = \{l, s, t\}$. $\text{Nodes}(C)$ represents the set of nodes, \bar{V} . In Fig. 1(a), $\text{Nodes}(C_1) = \{x, m\}$.

For a K -feasible cut, C , the logic function of the subgraph of nodes, \bar{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 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 cuts for each node in the circuit is accomplished by traversing circuit nodes in topological order from inputs to outputs. As each node is visited, its complete set of K -feasible cuts is generated by merging cuts from its fanin nodes [11], [24].

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 to K -LUTs 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 . A common data structure for this representation is an AND-Inverter Graph (AIG) in which the circuit is represented solely as a network of 2-input AND gates and inverters. An example of an AIG is shown in Fig. 1(b). Inverters are not represented explicitly as nodes in the graph, but rather as properties on graph edges. The AIG has been shown useful for many logic synthesis transformations and as a useful starting point for FPGA technology mapping as exemplified by the ABC tool [22], [21]. We therefore choose to investigate guarded evaluation within the ABC framework and to exploit the properties of AIGs to aid in performing guarded evaluation.

B. Power-Aware Mapping

Power-aware cut-based technology mapping has been studied recently (e.g., [16], [14]). The core approach taken is to keep signals with high switching activity out of the FPGA’s interconnection network (which presents a high capacitive

load). 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 [16], as replication increases signal fanout and capacitance. Replications can therefore be detected and cost accordingly, trading off their power “cost” with their depth “benefit”.

C. Guarded Evaluation

Tiwari et al. [26] first described important techniques for guarded evaluation in ASICs. The key idea is shown in Fig. 2. In Fig. 2(a), a multiplexer is shown receiving its inputs from a shifter and a subtraction unit, depending on the value of select signal *Sel*. Fig. 2(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 [3]. 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 [12]. 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 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 Fig. 2), 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.

D. 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 [7]. 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.

Fig. 3(a) gives an example of a LUT and the corresponding portion of a covered AIG. The logic function implemented by the LUT is: $Z = I \cdot J \cdot \overline{K} \cdot \overline{Q} \cdot \overline{M}$. 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 Fig 3(a), 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 [7], 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 [7], the notions of gating inputs and non-inverting paths were applied to map circuits into a new

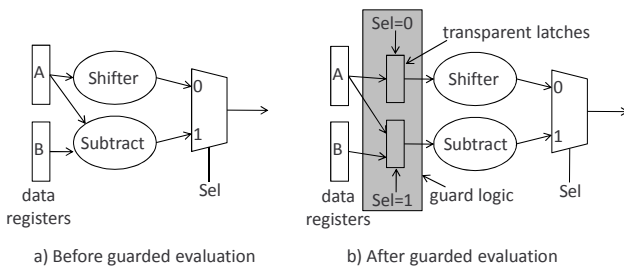


Fig. 2: Guarded evaluation (adapted from [26]).

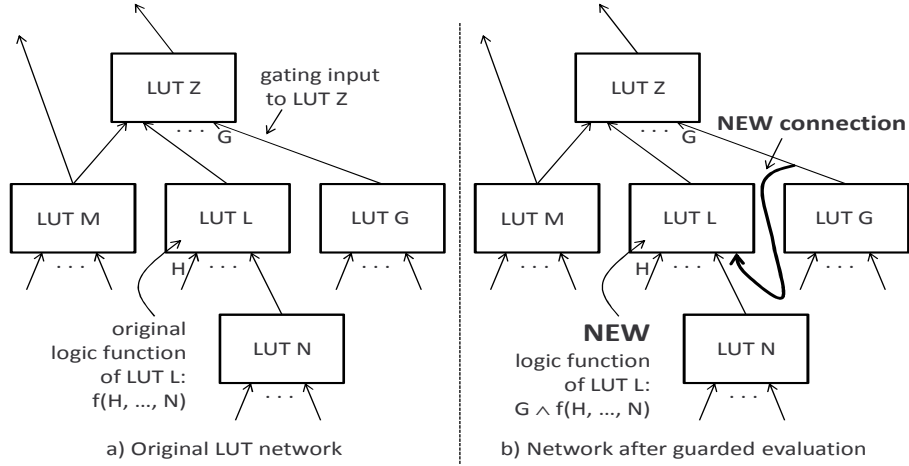


Fig. 4: Guarded evaluation for FPGAs.

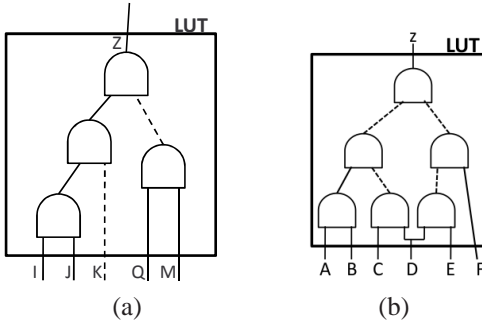


Fig. 3: (a) Identifying gating inputs on LUTs using non-inverting paths; (b) Identifying trimming inputs on LUTs using partial non-inverting paths.

logic block architecture that delivers improved area-efficiency. Here, we apply the ideas for power reduction through guarded evaluation.

E. 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 $Z = (A \cdot B \cdot \overline{C \cdot D}) \cdot (\overline{D \cdot E} \cdot F)$ illustrated in Fig. 3(a). There is no non-inverting path from any LUT input to the LUT output. 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 [8]. Recall that any n -variable logic function f can be cofactored with respect to variable x_k as follows:

$$f = x_k \cdot f(x_0, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n) + \overline{x_k} \cdot f(x_0, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n). \quad (1)$$

Here, $f(x_0, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$ is the 1-cofactor of f with respect to x_k and $f(x_0, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ is the 0-cofactor of f with respect to variable x_k . Each cofactor is itself a logic function with at most $n - 1$ variables. In [8], a trimming input was defined as an input to a n -variable function in which the Shannon decomposition produced a cofactor having strictly less than $n - 1$ inputs. In the case of a gating input, the Shannon decomposition produces a decomposition in which one of the cofactors is logic-0. Hence, with respect to [8], the use of non-inverting paths and partial non-inverting paths are structural techniques to identify gating and trimming inputs, respectively.

III. 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.

A. Overview

Fig. 4(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 .

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





Guard	Static Probability	
	$P \leq 0.5$	$P > 0.5$
Logic-0		
Logic-1		

Fig. 5: Inserting guards based on static probability.

occur in computing L 's output when G is logic-0 are an unnecessary waste of dynamic power.

In Fig. 4(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 Fig. 4(b). The question is how to modify L 's logic functionality. In [6], 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. 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 Fig. 4(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). Fig. 5 provides an illustration of the type of guarding used based on the static probability and the guarding value¹. No additional LUTs are required to perform guarding since we are modifying the function of the guarded LUT, which is logic entirely internal to the LUT. After guarding, switching activity on L 's output signal may be reduced, lowering the power consumed by the signal. Note, however, that guarding must 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.

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 Fig. 4(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 the Fig. 4(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 the next section. 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. [26], specifically, the problem of determining which signals can be used to guard which gates.

While we can guard L with G in Fig. 4, 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 III-D, 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 Fig. 6(a). 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.

It is worthwhile to highlight an important difference between our approach and the prior ASIC approach, shown in Fig. 2. In Fig. 2, 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

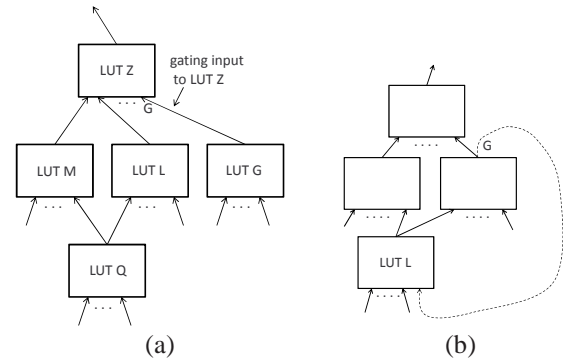


Fig. 6: (a) Guarding with reconvergent fanout; (b) Illustration of how guarding can create a combinational loop.

¹We 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.

guarding opportunities. Nonetheless, our results show that a significant number of guards were inserted effectively reducing dynamic power. Moreover, our method does not add LUTs to the circuit. The only overhead is for additional wires 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 [5], [27], 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 [13]. A similar observation was made earlier in [17] 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.

B. 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 II-D. 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:

$$GuardCost(C) = \frac{1 + \sum_{i \in Inputs(C)} \alpha(i)}{1 + |Gating[Inputs(C)]|} \quad (2)$$

where $\alpha(i)$ represents the switching activity on LUT input i . The numerator of (2) 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 [16], [14]. The denominator of (2) 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 III-A. Cuts with higher numbers of such non-inverting path inputs will have lower values of (2).

C. 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). We score guarding option O as follows:

$$Score(O) = \frac{|Outputs(L)| \cdot \alpha(L) \cdot P(L, G)}{1 + \alpha(G)} \quad (3)$$

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 (3) 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 (3) 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 (3) 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 (3). 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 III-D below).

A few of the conditions warrant further discussion. Rule #2 is illustrated in the LUT network of Fig. 6(b). 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.

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 Fig 7. 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

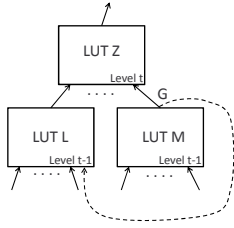


Fig. 7: Example showing how guarding can increase network depth.

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, we periodically update activity and probability values during guarding. This is akin to invoking regular timing analysis passes during routing (e.g., as done in [20]). 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 (3), 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 runtime 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.

D. 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 [14], 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 [21].

As noted in Section III-B, 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 Fig 4(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 Fig. 4(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. In Section III-A above, we described the process by which we identify guarding opportunities, 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. We employ the same procedure 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, our aim in 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 [21], 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. The next section presents results both with and without leveraging non-obvious don’t cares in guarded evaluation.

IV. EXPERIMENTAL RESULTS

A. Methodology

We implemented guarded evaluation within 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 [22] (the `if` command in ABC), 2) WireMap [13], 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. In all cases, prior to mapping, we execute the ABC `choice` command [10] 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 Eq. (2) 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 routing².

For total switching activity, we sum the activity across all nets of a circuit. 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 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 [16], which is based on VPR4.3, and integrates the FPGA power model of [23]³. 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 (25 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 paper 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. 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 [4]. Finally, 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

²Prior work has shown that interconnect comprises $\sim 2/3$ of dynamic power in FPGAs [25].

³Newer versions of VPR are available [18], [19], but these newer versions do not include a power model which is required for our investigations.

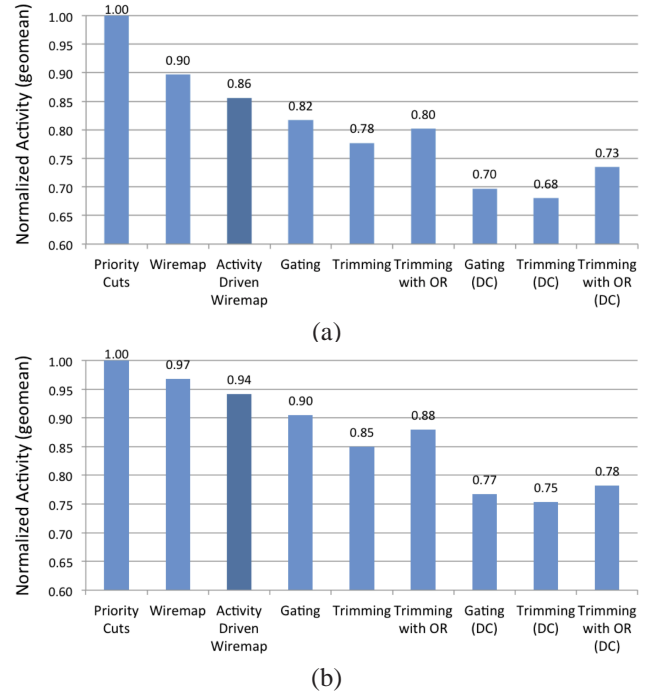


Fig. 8: Average reduction in switching activity (normalized) across a benchmark suite of 20 designs for area-oriented mapping: (a) 6-LUT architectures; (b) 4-LUT architectures.

irregardless of the mapping algorithm used. This allows for a fair comparison in terms of dissipated power. In this paper, 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.

Finally, for benchmarks, we use the larger designs from the MCNC suite [28] 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.

B. Switching Activity Results

The reduction in total switching activity for area-oriented mapping (using all different mapping techniques) is shown in Fig. 8(a) and Fig. 8(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 Fig. 8(a) and (b), the left-most bar shows total switching activity for priority cut-based mapping [22] and represents the baseline result. The second bar shows activity values for WireMap [13]. 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 Fig. 8 shows results for guarding with only gating inputs (c.f. Section II-D) without any consideration of trimming inputs (c.f. Section II-E) or non-obvious don't cares (c.f. Section III-D). Further, the fourth bar does not consider the guard insertion based on static probabilities, but only inserts AND gates (c.f. Section III-A) to force 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 Fig. 8(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 III-A). 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 Fig. 8(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 IV-D.

Finally, the last three bars (bars 7 through 9) in Fig. 8(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.

Fig. 9(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 Fig. 9(a) and (b) show the depth-oriented results for the different mappers in the same order as presented previously in Fig. 8(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,

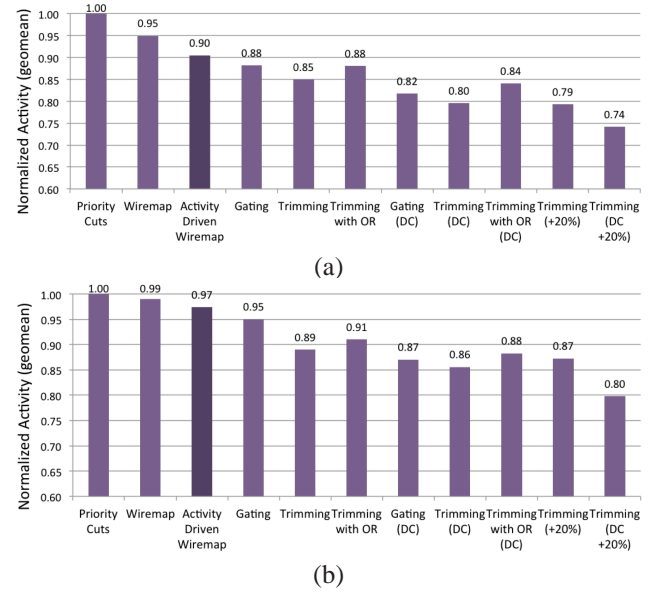


Fig. 9: Average reduction in switching activity (normalized) across a benchmark suite of 20 designs for depth-oriented and depth-relaxed mapping: (a) 6-LUT architectures; (b) 4-LUT architectures.

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 depth⁴. 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 Fig. 9(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 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, Table I presents the circuit-by-circuit results for the 6-LUT architectures depth-oriented mapping, respectively⁵. The last two rows in Table I shows the ratio

⁴That 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.

⁵Circuit-by-circuit results are omitted for 4-LUT architectures and area-oriented mapping for sake of brevity.

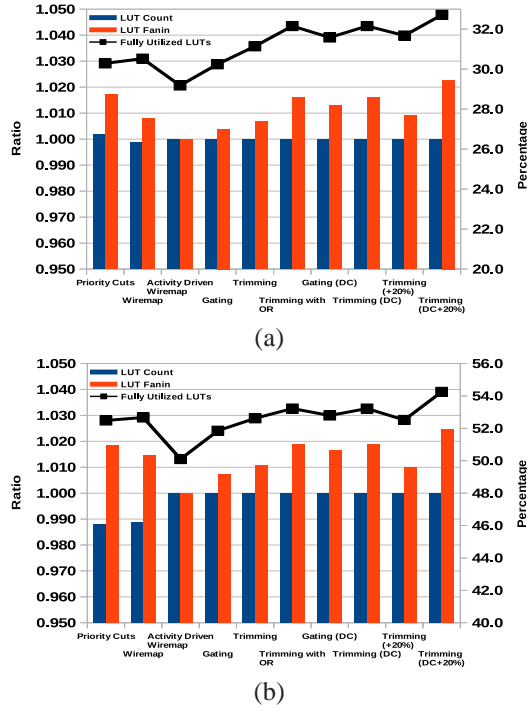


Fig. 10: Network statistics to judge the impact of guarding for depth-oriented mapping: (a) 6-LUT architectures; (b) 4-LUT architectures. Results show (i) LUT counts (normalized), (ii) average LUT fanin (normalized), and (iii) percentage of fully utilized LUTs.

(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.

Fig. 10 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 LUT fanin, normalized to the activity-driven WireMap scenario. 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).

Fig. 11 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 activity-driven WireMap. The line shows average LB fanin (# of used LB inputs) for the architecture having 4 LUTs/LB (LB fanin for other LB sizes is omitted for brevity). Observe that for both 6-LUTs and 4-LUTs, guarding does not have an appreciable

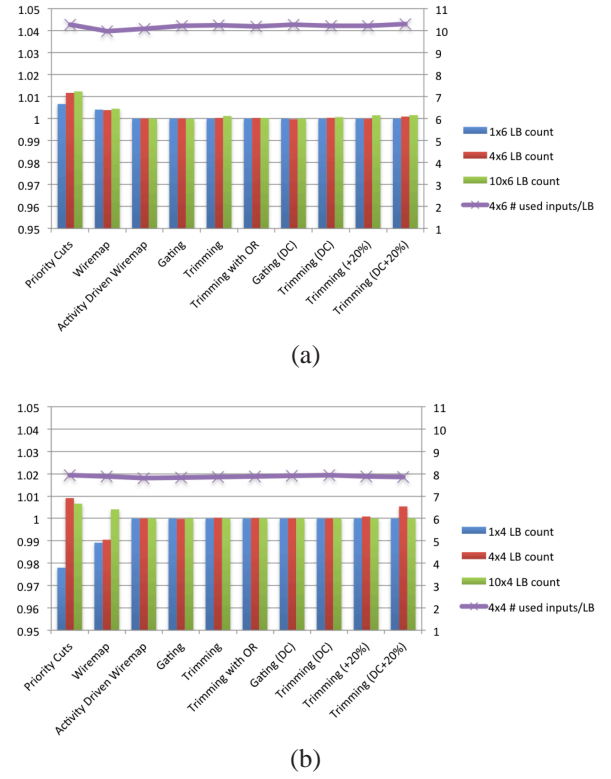


Fig. 11: Post-packing statistics on LB count and fanin for depth-oriented mapping: (a) 6-LUT architectures; (b) 4-LUT architectures. Results show (i) LB counts (normalized), (ii) average LB fanin for the case of 4 LUTs/LB.

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 seconds⁶. 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

⁶The 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.

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.

C. 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.

Fig. 12(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, Fig. 12(a) shows an average improvement of 20%, 24% and 22% for cluster sizes of 1, 4 and 10, respectively, relative to priority cuts-based mapping. For 4-LUT architectures, Fig. 12(b) shows an average improvement of 14%, 22% and 20% 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 IV-D.

Fig. 13(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, Fig. 13(a) shows reductions of 16%, 17% and 14% for clusters sizes of 1, 4 and 10, respectively, relative to priority cuts-based mapping. With depth-relaxation, these results improved to 18%, 21% and 19% for cluster sizes of 1, 4 and 10, respectively. For 4-LUT architectures, interconnect power was reduced by 11%, 15% and 13% for cluster sizes of 1, 4 and 10, respectively. Similar to the 6-LUT result, further improvements were obtained using depth relaxation; reductions of 12%, 18%, and 17% were obtained for clusters sizes of 1, 4 and 10, respectively.

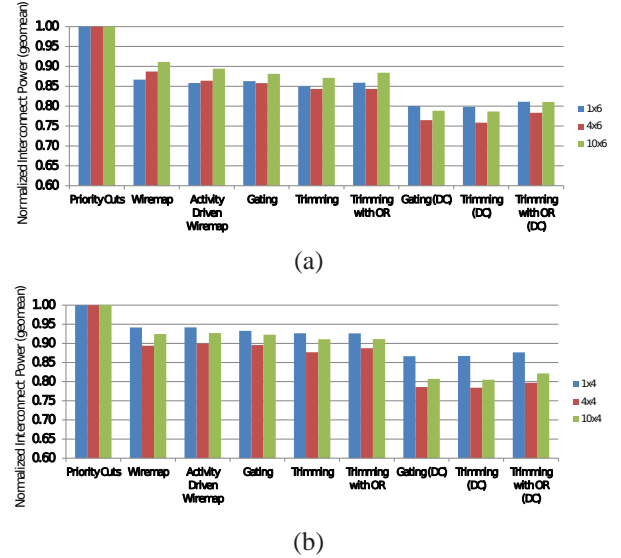


Fig. 12: Average reduction in interconnect power (normalized) across a benchmark suite of 20 designs for area-oriented mapping: (a) 6-LUT architectures; (b) 4-LUT architectures.

For reference, Table II 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.

Lastly, we report the impact of guarded evaluation on post-routed critical path delay (as reported by VPR [9]). Fig. 14 shows the geometric mean (across all circuits) of critical path delay for the several of the key mapping techniques. Results are presented only for 6-LUT architectures and different cluster sizes; 4-LUT results are similar and are omitted for brevity. Fig. 14(a) shows results for area-oriented mappings. With respect to activity-driven WireMap, the critical path delay is increased, on average, by $\sim 18\%$ to 20% when using gating inputs, depending on the cluster size. This increases to $\sim 23\%$ 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 activity-driven WireMap – anywhere from $\sim 31\%$ 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.

Fig. 14(b) gives results for several key depth-oriented mappings. When a depth constraint is enforced during guard

TABLE I: 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)	Trimming (+20%)	Trimming (DC) (+20%)
alu4	144.35	147.12	124.59	121.93	119.37	122.28	115.07	111.09	113.57	107.32	96.27
apex2	56.26	54.53	47.46	45.39	44.84	47.55	37.46	35.32	40.95	36.31	30.32
apex4	98.73	58.62	54.37	53.28	52.95	58.40	52.59	52.46	55.16	51.48	46.94
bigkey	219.98	223.58	223.82	223.82	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	379.43	292.68
des	272.98	262.79	256.77	255.28	255.22	256.22	255.85	253.35	254.37	252.65	251.91
diffeq	249.60	249.35	259.82	259.82	254.98	254.70	247.78	245.25	253.04	242.94	241.33
dsip	260.17	261.92	261.92	261.92	253.13	253.13	261.92	261.92	261.92	261.89	261.89
elliptic	692.56	697.87	708.61	707.54	705.60	705.49	706.44	705.85	706.50	706.28	706.01
ex1010	127.09	91.32	96.42	94.69	91.49	94.09	90.74	90.65	94.87	90.33	76.49
ex5p	102.07	98.67	85.08	82.76	71.70	73.54	78.38	78.38	81.54	81.46	72.10
frisc	562.33	512.07	489.53	486.60	482.34	468.51	477.42	477.21	456.48	485.61	476.69
misex3	92.17	91.00	83.21	80.99	80.31	82.63	75.20	73.19	76.93	68.96	67.53
pd	138.38	129.19	111.29	105.19	96.71	112.42	93.69	85.81	99.18	78.83	77.02
s298	80.72	82.78	76.67	75.22	74.85	74.55	73.98	67.21	69.54	67.06	65.37
s38417	759.56	806.74	819.83	817.04	816.31	811.65	813.64	813.01	808.31	813.43	808.91
s38584.1	751.54	681.06	687.20	683.54	685.97	680.99	683.11	678.26	667.94	678.87	676.17
seq	84.08	86.28	85.07	82.40	82.22	87.72	69.42	64.59	72.64	58.67	52.01
spla	174.35	179.97	144.87	136.12	130.72	142.85	115.45	107.47	135.55	109.27	91.38
tseng	249.21	247.25	253.87	252.97	237.17	237.18	251.09	250.73	252.24	251.56	250.77
Geomean	208.12	197.57	188.22	183.57	176.26	182.93	170.05	165.67	174.87	165.02	154.33
Ratio	1.00	0.95	0.90	0.88	0.85	0.88	0.82	0.80	0.84	0.79	0.74
Ratio			1.00	0.98	0.94	0.97	0.90	0.88	0.93	0.88	0.82

TABLE II: Power reduction results for different architectures (power given in Watts reported by [16], [23]).

Mapping Flow	Architecture and Mapping Objective											
	1x4		4x4		10x4		1x6		4x6		10x6	
	Area	Depth	Area	Depth	Area	Depth	Area	Depth	Area	Depth	Area	Depth
Priority Cuts	0.356	0.373	0.110	0.115	0.072	0.075	0.321	0.335	0.103	0.108	0.069	0.076
WireMap	0.335	0.352	0.099	0.108	0.067	0.073	0.279	0.301	0.092	0.099	0.063	0.071
Activity Driven WireMap	0.335	0.349	0.099	0.107	0.067	0.071	0.276	0.298	0.089	0.096	0.062	0.072
Gating	0.332	0.349	0.099	0.107	0.066	0.071	0.277	0.293	0.089	0.096	0.061	0.070
Trimming	0.330	0.349	0.097	0.106	0.066	0.071	0.273	0.292	0.087	0.096	0.060	0.070
Trimming with OR	0.330	0.349	0.098	0.106	0.066	0.071	0.276	0.292	0.087	0.096	0.061	0.069
Gating (DC)	0.308	0.336	0.087	0.099	0.058	0.066	0.257	0.282	0.079	0.090	0.055	0.066
Trimming (DC)	0.309	0.334	0.086	0.098	0.058	0.065	0.257	0.282	0.078	0.090	0.055	0.065
Trimming with OR (DC)	0.312	0.337	0.088	0.100	0.059	0.064	0.261	0.287	0.081	0.091	0.056	0.066
Trimming (+20%)	-	0.339	-	0.100	-	0.066	-	0.282	-	0.090	-	0.065
Trimming (DC +20%)	-	0.327	-	0.094	-	0.062	-	0.274	-	0.086	-	0.062

insertion, the critical path increases only slightly by $\sim 1\%$ 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 $\sim 11\%$ to 17% .

It is important to recognize that many FPGA designs do not need to run at the maximum 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 14% 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 19% power reduction, with a higher,

15% increase in critical path delay. The different flavors of guarded evaluation thus provide the user with a range of implementation options within the power/speed design space.

D. Discussion

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

1) *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. Counterintuitively, 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) 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 or based on static probability

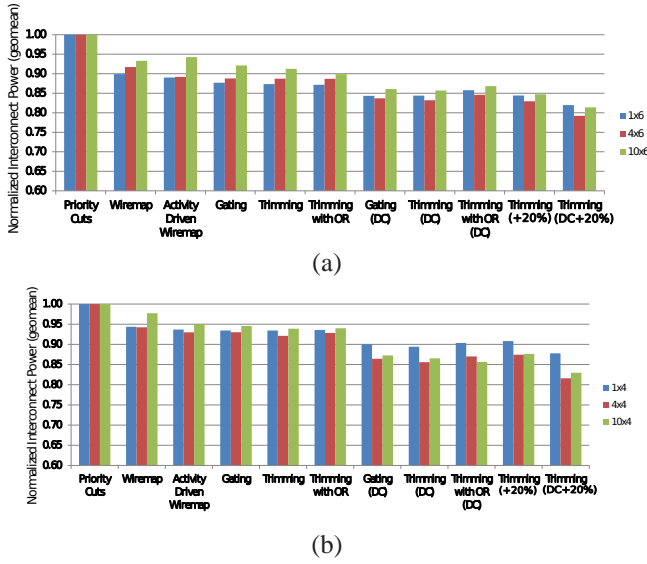


Fig. 13: Average reduction in interconnect power (normalized) across a benchmark suite of 20 designs for depth-oriented mapping: (a) 6-LUT architectures; (b) 4-LUT architectures.

Mapping Flow	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)

Mapping Flow	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)

Fig. 14: Critical path delays for several key mapping techniques to understand the impact of guarding on performance: (a) area-oriented mapping; (b) delay-oriented mapping.

resulted in reduced switching activity for the *current signal being guarded*.

However, further investigation showed that the use of OR gates resulted in fewer total inserted guards. Table III 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 options resulting in a different order in which guards were inserted and free LUT inputs were “used up”. It

TABLE III: Comparison of the number of inserted guards using gating+trimming inputs when using only AND gates versus using AND gates + 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

is a possibility that a different benchmark suite or a different scoring function for guarding candidates would have resulted in a different outcome.

2) *Architectural Analysis*: Clustered architectures tended to benefit more from guarded evaluation (c.f. Figs. 12 and 13); this tendency is more pronounced for 4-LUT architectures in our particular experiments. For flat architectures (i.e., 1 LUT/LB), the added guarding connections require inter-cluster routing; the additional power consumed by the routing of these connections could out-weigh the benefits of reduced switching activity due to guarding. Conversely, in clustered architectures, it was often the case that many guarding connections were internal to the LBs and, consequently, did not require inter-cluster routing; these intra-cluster signals do not consume as much power and are less likely to out-weigh the benefit of the inserted guards. This observation motivates future work in which guarding is applied after clustering to have a better estimation of the impact on inter-cluster routing.

V. CONCLUSIONS

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-*

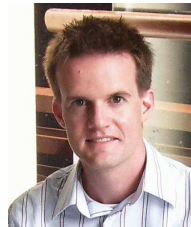
inverting and *partial non-inverting paths*; the use of partial non-inverting paths was demonstrated to significantly improve the availability of guarding options and, in turn, improve the reduction in both total reduction in total switching activity and reduction in total dynamic power dissipation. Finally, we considered the impact 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.

REFERENCES

- [1] ABC – a system for sequential synthesis and verification. <http://www.eecs.berkeley.edu/~alanmi/abc/>, 2009.
- [2] Quartus-II university interface program. <http://www.altera.com/education/univ/research/univ-quip.html>, 2009.
- [3] 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.
- [4] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicro FPGA performance and density. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 85–94, 2002.
- [5] Altera, Corp., San Jose, CA. *Stratix-III FPGA Family Data Sheet*, 2008.
- [6] J. Anderson and C. Ravishankar. FPGA power reduction by guarded evaluation. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 157–166, 2010.
- [7] J. Anderson and Q. Wang. Improving logic density through synthesis-inspired architecture. In *IEEE Int'l Conf. on Field-Programmable Logic and Applications*, pages 105 – 111, Prague, Czech Republic, 2009.
- [8] J. Anderson and Q. Wang. Area-efficient FPGA logic elements: Architecture and synthesis. In *IEEE/ACM Asia and South Pacific Design Automation Conf.*, pages 369–375, 2011.
- [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. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam. Reducing structural bias in technology mapping. In *Int'l Workshop on Logic Synthesis*, 2005.
- [11] 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.
- [12] D. Howland and R. Tessier. RTL dynamic power optimization for FPGAs. In *IEEE Midwest Symp. on Circuits and Systems*, pages 714–717, 2008.
- [13] 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.
- [14] S. Jang, K. Chung, A. Mishchenko, and R. Brayton. A power optimization toolbox for logic synthesis and mapping. In *IEEE Int'l Workshop on Logic Synthesis*, 2009.
- [15] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Trans. On CAD*, 26(2):203–215, February 2007.
- [16] J. Lamoureux and S. Wilton. On the interaction between power-aware FPGA CAD algorithms. In *IEEE/ACM Int'l Conf. on Computer-Aided Design*, pages 701–708, 2003.
- [17] D. Lewis, et al. The Stratix II logic and routing architecture. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 14–20, 2005.
- [18] J. Luu, J. Anderson, and J. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 227–236, 2011.
- [19] J. Luu and et al. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 133–142, 2009.
- [20] 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.
- [21] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. Scalable don't-care-based logic optimization and resynthesis. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, pages 151–160, 2009.
- [22] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *IEEE Int'l Conf. on Computer-Aided Design*, pages 354–361, 2007.
- [23] 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.
- [24] M. Schlag, J. Kong, and P.K. Chan. Routability-driven technology mapping for lookup table-based FPGAs. *IEEE Trans. on Computer-Aided Design*, 13(1):13–26, 1994.
- [25] 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.
- [26] V. Tiwari, S. Malik, and P. Ashar. Guarded evaluation: pushing power management to logic synthesis/design. *IEEE Trans. on Computer-Aided Design*, 17(10):1051–1060, October 1998.
- [27] Xilinx, Inc., San Jose, CA. *Virtex-5 FPGA Data Sheet*, 2007.
- [28] S. Yang. Logic synthesis and optimization benchmarks. version 3.0. Technical report, Microelectronics Center of North Carolina, 1991.



Chirag Ravishankar (S'11) received the B.A.Sc. degree from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada in 2010. He is currently pursuing the M.A.Sc. degree at the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research interests include eld-programmable gate array architectures and CAD algorithms. He also works on dynamic power management algorithms for multi-processor systems on chip (MPSoCs). Mr. Ravishankar was the recipient of the Undergraduate Student Research Award (USRA) from the Natural Sciences and Engineering Research Council (NSERC) of Canada in 2009. He was nominated for the university-wide “Amit and Meena Chakma Award for Exceptional Teaching” in 2011.



Jason H. Anderson (S'96-M'05) received the B.Sc. degree in computer engineering from the University of Manitoba, Winnipeg, MB, Canada, in 1995 and the Ph.D. and M.A.Sc. degrees in electrical and computer engineering from the University of Toronto (U of T), Toronto, ON, Canada, in 2005 and 1997, respectively. He is an Assistant Professor with the Department of Electrical and Computer Engineering (ECE), U of T. In 1997, he joined the field-programmable gate array (FPGA) implementation tools group at Xilinx, Inc., San Jose, CA, working on placement, routing and synthesis. He joined the ECE Department at U of T in 2008. His research interests include all aspects of computer-aided design (CAD) and architecture for FPGAs.



Andrew Kennings (M'10) received the B.A.Sc., M.A.Sc., and Ph.D. degrees in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1992, 1994, and 1997, respectively. Dr. Kennings is an Associate Professor of electrical and computer engineering with the University of Waterloo. His research interests include large scale optimization and physical design, with particular emphasis on synthesis and placement.