FPGA Power Reduction by Guarded Evaluation

Jason H. Anderson
Dept. of Electrical and Computer Engineering
University of Toronto
janders@eecg.toronto.edu

Chirag Ravishankar
Dept. of Electrical and Computer Engineering
University of Toronto
chirag.ravishankar@utoronto.ca

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 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 "non-inverting paths" in the circuit's `AND`-inverter graph representation. Experimental results show that guarded evaluation can reduce switching activity by 22%, on average, and can reduce power consumption in the FPGA interconnect by 14%.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids

General Terms

Design, Algorithms

Keywords

Field-programmable gate arrays, FPGAs, power, optimization, low-power design, logic synthesis, technology mapping

1. INTRODUCTION

Modern field-programmable gate arrays (FPGAs) are "innovation enablers" across a broad spectrum of digital hardware applications, as they reduce product cost, time-to-market, and mitigate risk. However, their use in hand-held battery powered devices remains elusive, due primarily 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 custom ASICs for implementing a given function [13]. As iPhones, Blackberrys and other mobile devices gain an ever greater penetration in today's society, FPGAs remain wholly absent in such devices. Power consumption stands as the key barrier preventing FPGAs from cracking into the lucrative mobile electronics market.

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 [24]. Recall that dynamic power in a CMOS circuit is defined by: $P_{avg} = \frac{1}{2} \sum_{i \in nets} C_i \cdot f_i \cdot V^2$, 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 netlist. In particular, the approach taken is to eliminate toggles on certain internal signals of a circuit when such toggles are known 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 [23], due primarily to long metal wire segments and the parasitic capacitance of used and unused programmable routing switches.

At a high-level, 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 produces 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.

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 functions

FPGA'10, February 21–23, 2010, Monterey, California, USA.

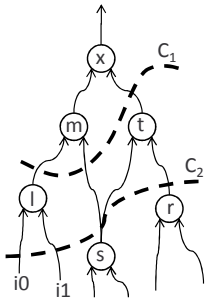

Figure 1: Cuts in circuit graph.

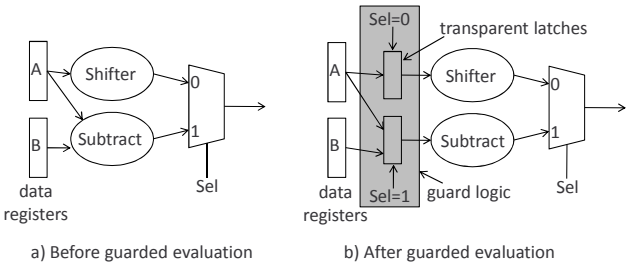

Figure 2: Guarded evaluation (adapted from [24]).

and connectivity to incorporate guards, reducing switching activity and dynamic power. In our approach, identifying the conditions under which a given signal can be guarded is accomplished by analyzing properties of the logic synthesis netlist, which is an AND-inverter graph (AIG). In particular, we show that the presence of non-inverting paths in the AIG can be used to drive the discovery of guarding opportunities. Moreover, 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 free 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. The remainder of the paper is organized as follows: Section 2 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 3. An experimental study appears in Section 4. Conclusions and suggestions for future work are offered in Section 5.

2. BACKGROUND

2.1 FPGA Technology Mapping

Here we review the approach used by modern FPGA technology mappers, which are based on finding cuts in Boolean networks [22, 8]. 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 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 Fig. 1, $\overline{V}$ 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 Fig. 1, $Inputs(C_1) = \{l, s, t\}$. $Nodes(C)$ represents the set of nodes, $\overline{V}$. In Fig. 1, $Nodes(C_1) = \{x, m\}$.

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 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 [8, 22].

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.

2.2 Power-Aware Mapping

Power-aware cut-based technology mapping has been studied recently (e.g., [14, 12]). 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 is generally negative from the power angle [14], as replication increases signal fanout and capacitance. Replications can therefore be detected and cost accordingly, trading off their power "cost" with their depth "benefit".

2.3 Guarded Evaluation

A highly cited work on guarded evaluation in the ASIC context is by Tiwari et al. [24]. The key idea is shown in Fig. 2. Part (a) of the figure shows a multiplexer 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 netlists, 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. Recently, Howland and Tessier studied guarded evaluation at the RTL level for FPGAs [10]. Their approach produced encouraging power reduction results, however, its application is limited to using select signals on multiplexers as guarding signals, and it therefore only applies to specific types of circuits.

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 netlist. 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 by using existing yet unused FPGA circuitry.

2.4 ABC Logic Synthesis

Recently, a new publicly available framework for logic synthesis, called ABC, has been developed and has spurred a renewed focus on synthesis research [1]. In ABC (developed primarily by Alan Mishchenko at UC Berkeley), the key data structure 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 Fig. 3. 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., [18, 15]). AIGs have shown value in LUT mapping as well [19]. The best published results today for area-oriented FPGA mapping were produced with ABC [19, 17]. We therefore choose to investigate guarded evaluation within the ABC framework.

2.5 Gating Inputs and Non-Inverting AIG Paths

Technology mapping covers the circuit AIG with LUTs – each LUT in the mapped network implements a portion

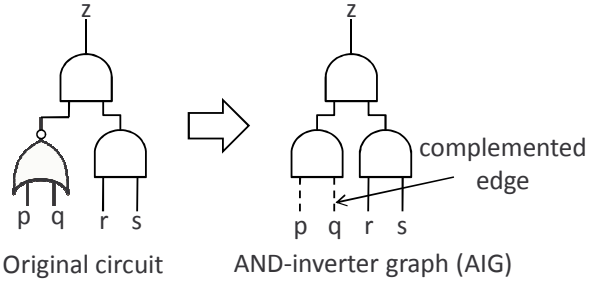

Figure 3: AND-inverter graph (AIG) example.

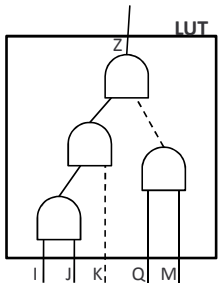

Figure 4: Identifying gating inputs on LUTs using non-inverting AIG paths.

of the underlying AIG logic functionality. Our recent work used properties of the AIG to discover *gating inputs* to LUTs [5]. 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 our activity reduction approach and therefore briefly review the concept here.

Fig. 4 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 Fig 4, 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 [5], we generalized the gating input idea and 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 a LUT's underlying AIG. In [5], the notions of gating inputs and non-inverting paths were applied to map circuits into a new logic block

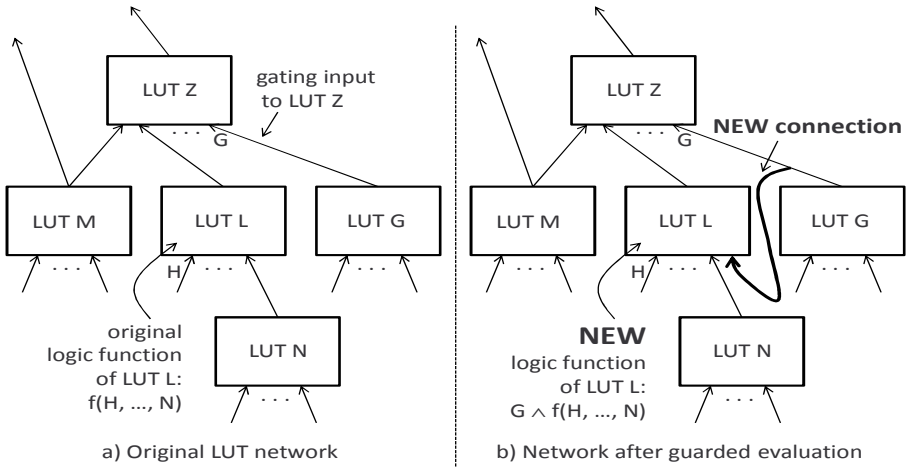

Figure 5: Guarded evaluation for FPGAs.

architecture that delivers improved area-efficiency. Here, we apply the ideas for a different purpose, namely, power reduction through guarded evaluation.

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.

3.1 Overview

Fig. 5(a) illustrates how gating 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. Now, consider the case of LUT L which drives LUT Z. Since L's single fanout is to Z, any transitions on L's output will not affect overall circuit outputs when G is logic-0. Toggles that occur on L's output when G is logic-0 are an unnecessary waste of dynamic power.

In Fig. 5(a), L is a candidate for guarded evaluation by signal G. If LUT L has a free input, we modify the mapped netlist by attaching G to L, and then modifying L's logic functionality as shown in Fig.5(b). The new logic function for L is set equal to the logical AND of its previous logic function and signal G. 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 uphill (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. 5(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. 5(a) were a gating 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 inputs (via finding non-inverting paths) circumvents one of the key difficulties encounted by Tiwari et al. [24], specifically, the problem of determining which signals can be used to guard which gates.

While we can guard L with G in Fig. 5, 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 3.4, 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. 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 of an existing LUT function with the guarding signal, making the LUT output logic-0 while guarded. Consider that at the instant prior to guarding, the LUT's output could conceivably have been at logic-1. In our case, therefore, guarding can potentially induce additional transitions on LUT outputs, versus the technique that uses transparent latches. Nevertheless, results below show that despite this "weakness", our method is effective in power reduction. Moreover, our method has the advantage of imposing little hardware overhead.

Since our guarding approach relies on there being free inputs on LUTs, it is also worth mentioning that LUTs in today's commercial FPGAs have 6 inputs [4, 25], 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 [11]. 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 netlist, 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, as well as we seek to maintain 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 2.5. 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 power of the mapped netlist:

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

where $\alpha(i)$ represents the switching activity on LUT input i. The numerator of (1) tallies the switching activities on cut inputs, minimizing activity on inter-LUT connections in the mapped netlist. 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 [14, 12]. The denominator of (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 (1).

3.3 Post-Mapping Guarded Evaluation

Following mapping, the circuit is represented as a network of LUTs. Consider a guarding option, O, comprising

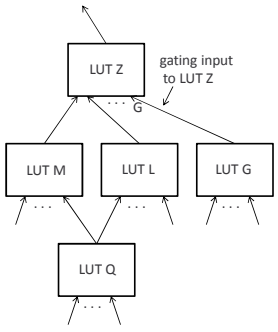

Figure 6: Guarding with reconvergent fanout.

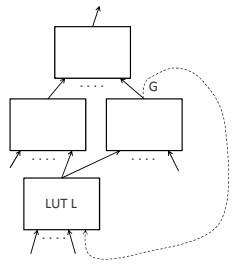

Figure 7: Illustration of how guarding can create a combinational loop.

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:

$$Score(O) = |Outputs(L)| \cdot \alpha(L) \cdot P(G) - \alpha(G) \quad (2)$$

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, $P(G)$ is the static probability of G, which is the fraction of time that G spends in the "gating state" under typical input vectors. Static probability is a property of logic signals widely used in the power estimation domain [20]. The first term of (2) represents the benefit of guarding, which increases in proportion to L's fanout, its activity and the fraction of time G is in the gating state. The more time that G spends in the gating state, the higher the likely activity reduction on L. The second term of (2) represents the cost of guarding, which is an increase 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 (2) are associated with what we expect will be better guarding candidates.

For a mapped netlist, we capture all possible guarding options in an array and sort the array in descending order of each option's score, as computed through (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 3.4 below).

A few of the conditions warrant further discussion. Rule #2 is illustrated in the LUT network of Fig. 7. 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

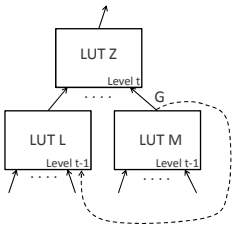

Figure 8: Example showing how guarding can increase network depth.

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 8. 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. This is akin to timing slacks becoming stale and needing periodic updates in FPGA placement and routing (e.g., as done in [16]). To deal with this, we periodically update activity and probability values during guarding. 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 (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.

3.4 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 [12], 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 [17].

As noted in Section 3.2, gating inputs on LUTs can be identified though 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 5(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. 5(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 3.1 above, we described the process by which we identify guarding opportunities, namely, by identifying a gating input, G, to a LUT, Z, and then walking the mapped network uphill 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.

We use simulation and combinational logic verification (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 [17], can be incorporated to mitigate run-times for large industrial circuits[1]. 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.

4. EXPERIMENTAL STUDY

We implemented guarded evaluation within the ABC logic synthesis framework and target 6-LUTs. We compare the guarded mappings with several different baseline mappings: 1) LUT mapping based on priority cuts [19] (the `if` command in ABC), 2) WireMap [11], and 3) activity-driven

[1]The scalable verification work in [17] has not been released publicly.

WireMap. WireMap is a technique that reduces the number of inter-LUT connections, which is likely beneficial for power. For activity-driven WireMap, we altered the WireMap cut selection cost function to break ties using the sum of switching activity on LUT (cut) inputs. In all cases, prior to mapping, we execute the `choice` command in ABC to perform technology independent optimization. Consequently, our technology mapping executes with "choices" [7], 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 (1). Combinational equivalence after guarding was verified using the `cec` command in ABC.

We measure the impact of guarded evaluation using two power metrics: 1) switching activity, and 2) power dissipated in the FPGA interconnect, considering interconnect capacitance. For switching activity, we sum the activity across all nets of a circuit. For power, we target an architecture with length-4 wire segments and logic blocks containing four 6-LUT/FF pairs per block. We use the power-aware packing, placement and routing framework described in [14], which is integrated with the power model of [21]. Our power numbers therefore account for post-routed interconnect capacitance. We do not allow the number of routing tracks per channel (W) to "float" during our runs. Instead, for each circuit, we compute the minimum W needed for the priority cuts mapping. We then increase this W by 30% and use the resulting W for the circuit across all runs. The routing fabric is therefore invariant for a given circuit across all different mappings, allowing us to fairly evaluate the impact of the mapping.

To generate switching activities, we used the simulator built-in to ABC. Each combinational input (primary input or register output) was first 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 was used for each circuit across all runs.

Lastly, since the core mapper in ABC (based on priority cuts) can operate in depth or area mode, we consider the consequences of guarding on both area and depth-oriented mappings, and for the case of depth, we consider the tradeoffs between power and depth.

4.1 Results

Fig. 9 shows results for switching activity, averaged across 20 benchmark circuits[2]. Part (a) of the figure gives results for area-oriented mappings (`if -a` command in ABC). Part (b) of the figure shows results for depth-oriented mappings. Focusing first on the area-oriented mappings, the left-most bar shows switching activity values for mapping based on priority cuts [19]. The second bar shows activity values for WireMap [11]. Observe that WireMap reduces switching activity by 10% on average, versus mapping based on priority cuts. The third bar shows results for activity-driven WireMap; activity is reduced by an additional 4% relative to the priority cuts mapping, on average.

The fourth and fifth bars in Fig. 9(a) show results for guarding without and with consideration of the non-obvious

[2]Reported averages are geometric means.

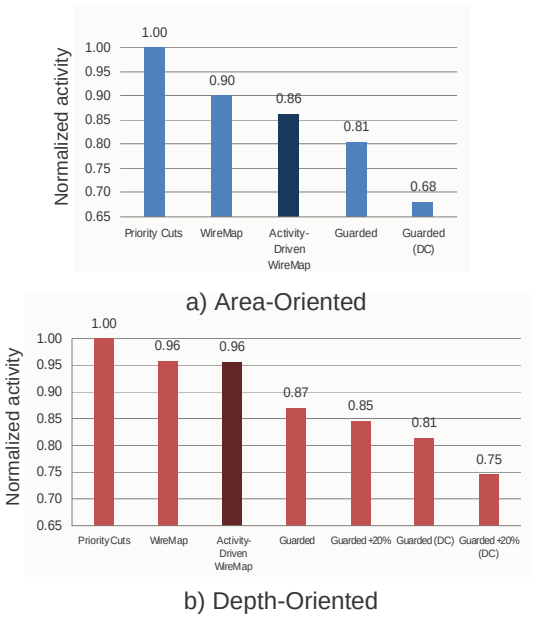

Figure 9: Switching activity reduction results.

"don't cares" (described in Section 3.4), respectively. It is most relevant to compare these data points with activity-driven WireMap, shown as a bolded bar in Fig. 9 (i.e. we compare guarded evaluation to a good quality activity-driven baseline mapping). Observe that without non-obvious don't cares, guarded evaluation reduces switching activity by 6%, on average, versus activity-driven WireMap. Using the non-obvious don't cares, activity is reduced by 21%, on average. There is a strong benefit to activity reduction when the full flexibility of don't cares can be exploited for guarded evaluation. Table 1 shows the activity results on a circuit-by-circuit basis. The left side of the table shows results for area-oriented mappings. For some circuits, e.g. *bigkey*, guarded evaluation provided little benefit. Further analysis of *bigkey* showed very few LUTs with free inputs and therefore limited guarding opportunities.

Fig. 9(b) gives activity results for depth-oriented mappings. The first, second and third data points in the figure are for optimal-depth mappings created using flows analogous to the first, second and third data points in Fig. 9(a). Observe that WireMap and activity-driven WireMap produce less benefit in switching activity reduction than in the area-oriented mappings. On average, these mappings have 4% less activity compared with priority cuts mapping. The "Guarded" data point in Fig. 9(b), gives the activity for mappings subjected to guarded evaluation that are optimal depth. The "Guarded + 20%" data point, gives the activity for mappings whose depth was allowed to grow by 20% during guarded evaluation, in terms of the number of logic levels[3]. This permits us to study depth/power tradeoffs in guarded evaluation. Guarding can reduce activity by 9%, while maintaining optimal depth, versus activity-driven WireMap. Further switching activity reductions are possible (12%) when depth is permitted to grow by 20%.

The two right-most data points in Fig. 9(b) give results when non-obvious don't cares may be exploited. In this

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

Table 1: Switching activity reduction results.

	Area-Oriented					Depth-Oriented						
Circuit	Priority Cuts	WireMap	Activity-Driven WireMap	Guarded	Guarded (DC)	Priority Cuts	WireMap	Activity-Driven WireMap	Guarded	Guarded +20%	Guarded (DC)	Guarded +20% (DC)
alu4	133.79	130.54	122.7	105.3	81.12	146.29	143.66	141.32	109.86	104.76	104.81	92.41
apex2	62.76	54.49	50.76	40.48	28.95	59.54	60.77	60.25	51.03	48.89	43.06	36.25
apex4	75.39	54.66	50.08	47.57	41.17	94.09	54.14	54.09	48.55	47.77	48.29	43.26
bigkey	223.82	223.82	223.82	223.82	223.82	223.82	223.82	223.82	223.82	223.82	223.82	223.82
clma	760.03	739.81	713.43	564.11	303.63	751.69	750.8	716.61	570.32	561.24	361.13	322.4
des	225.44	244.18	252.83	248.68	247.49	267.16	261	270.59	255.87	255.19	255.63	253.58
diffeq	239.4	243.19	245.73	246.08	243.91	246.04	251.23	259.52	260.65	260.34	259	258.67
dsip	265.57	265.57	264.67	264.67	264.67	268.53	248.58	264.67	247.64	247.66	247.64	247.66
elliptic	685.17	673.49	681.28	681.27	680.67	686.36	696.56	699.7	701.68	701.63	700.91	700.64
ex1010	239.79	122.78	107.82	105.79	69.41	163.28	123.45	131.64	119.24	107.05	113.65	78.92
ex5p	118.39	94.86	86.19	83.42	69.85	121.25	116.77	106.35	106.58	102	102.14	88.3
frisc	493.32	499.38	485.5	479.81	477.49	497.9	509.5	492.65	491.16	491.16	487.71	487.43
misex3	90.86	90.59	82.55	71.91	60.36	102.56	101.85	100.84	87.3	81.87	81.91	69.31
pdc	159.31	107.76	93.94	85.16	56.92	140.91	140.36	139.69	102.47	97.42	89.23	74.6
s298	69.47	69.56	63.65	52.78	45.99	76.97	79.92	74.65	62.07	60.46	58.46	57.23
s38417	772.89	781.37	782.69	783.55	780.89	774.97	783.08	797.22	801.3	800.82	799.12	798.04
s38584.1	686.22	681.83	677.46	675.28	674.43	692.97	686.66	689.59	682.28	681.86	681.21	678.79
seq	88.78	78.3	78.1	69.11	48.36	90.28	87.87	97.72	90.2	83.89	77.88	65.69
spla	199.02	145.77	136.37	122.77	93.22	194.86	193.98	180.22	146.08	138.31	131.64	109.83
tseng	237.79	247.7	251.62	251.61	250.32	240.02	250.68	253.89	254.79	254.71	253.56	253.44
Geomean:	211.86	190.75	182.96	170.61	144.40	214.25	205.12	204.97	186.39	181.35	174.60	159.84
Ratio:	**1.00**	0.90	0.86	0.81	0.68	**1.00**	0.96	0.96	0.87	0.85	0.81	0.75
Ratio:			**1.00**	0.93	0.79			**1.00**	0.91	0.88	0.85	0.78

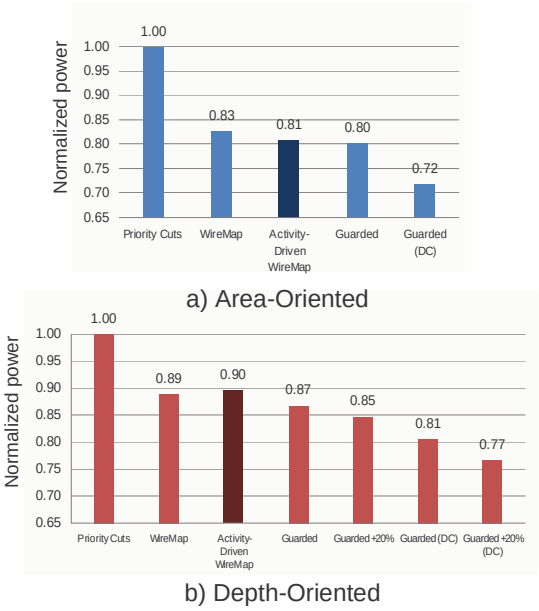

Figure 10: Interconnect power reduction results (reported by [14, 21]).

case, guarded evaluation can reduce activity by 15%, on average, without compromising depth, and 22% when mapped depth is allowed to increase by 20% (compared with activity-driven WireMap). Results again demonstrate the benefit of leveraging non-obvious don't cares, and also show the trade-off between power and depth in guarded evaluation. The right side of Table 1 gives circuit-by-circuit results for depth-oriented mappings and guarded evaluation.

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 netlist, likely increasing their capacitance and power. Consequently, it is not adequate to focus solely on activity reduction to evaluate the power benefit of the technique. Fig. 10 gives the average power consumed in the FPGA interconnect, considering post-routing interconnect capacitance. Table 2 gives the same results on a circuit-by-circuit basis. Recall that dynamic power in the interconnect comprises ∼60% of total power in commercial FPGAs [23]. The power numbers across all columns reflect a single clock frequency for each circuit. Thus, the data allows us to evaluate the power consumption of different implementations of each circuit clocked at a specific frequency. Below, we will discuss the impact of guarded evaluation on critical path delay. The data points in Fig. 10 are analogous to those in Fig. 9; the columns in Table 2 are analogous to those in Table 1.

Looking first at the area-oriented mappings (Fig. 10(a)), observe that WireMap and activity-driven WireMap provide significant power benefits over mapping with priority cuts. Power is reduced by 17%, on average, with WireMap, and 19% with activity-driven WireMap. The fourth and fifth bars in Fig. 10(a) give power numbers for guarded evaluation. Without non-obvious don't cares, guarded evaluation provides only 1% power reduction versus activity-driven WireMap. With the additional don't cares, however, considerable power reductions are possible – power is reduced by ∼11%, on average, relative to activity-driven WireMap. Compared with priority cuts mapping, guarded evaluation and WireMap reduce power by ∼20% and 28%, without and with non-obvious don't cares, respectively. We consider this to be a promising result that should keenly interest power-sensitive FPGA vendors and customers.

Fig 10(b) gives results for depth-oriented mappings. First, observe that WireMap and activity-driven WireMap provide

Table 2: Interconnect power reduction results (power given in Watts reported by [14, 21]).

	Area-Oriented					Depth-Oriented						
Circuit	Priority Cuts	WireMap	Activity-Driven WireMap	Guarded	Guarded (DC)	Priority Cuts	WireMap	Activity-Driven WireMap	Guarded	Guarded +20%	Guarded (DC)	Guarded +20% (DC)
alu4	0.084	0.080	0.080	0.079	0.072	0.083	0.085	0.085	0.079	0.080	0.079	0.075
apex2	0.085	0.061	0.060	0.057	0.047	0.091	0.076	0.079	0.073	0.072	0.066	0.061
apex4	0.061	0.046	0.046	0.045	0.044	0.062	0.048	0.047	0.046	0.046	0.046	0.044
bigkey	0.119	0.119	0.119	0.119	0.119	0.119	0.119	0.119	0.119	0.119	0.119	0.119
clma	0.359	0.303	0.292	0.287	0.157	0.421	0.353	0.364	0.315	0.313	0.225	0.190
des	0.168	0.170	0.175	0.175	0.170	0.191	0.182	0.182	0.177	0.183	0.184	0.179
diffeq	0.077	0.071	0.068	0.071	0.064	0.083	0.074	0.079	0.079	0.080	0.073	0.073
dsip	0.139	0.139	0.139	0.139	0.139	0.142	0.130	0.139	0.125	0.125	0.125	0.125
elliptic	0.291	0.286	0.283	0.281	0.252	0.305	0.290	0.276	0.281	0.282	0.246	0.245
ex1010	0.248	0.127	0.119	0.118	0.106	0.199	0.146	0.153	0.143	0.142	0.141	0.132
ex5p	0.050	0.038	0.036	0.036	0.033	0.052	0.046	0.044	0.044	0.044	0.043	0.040
frisc	0.279	0.236	0.241	0.246	0.239	0.288	0.251	0.256	0.257	0.260	0.245	0.241
misex3	0.076	0.069	0.067	0.068	0.062	0.078	0.076	0.078	0.075	0.074	0.073	0.070
pdc	0.175	0.100	0.089	0.087	0.083	0.183	0.142	0.134	0.115	0.113	0.111	0.096
s298	0.057	0.058	0.059	0.058	0.054	0.061	0.062	0.061	0.060	0.059	0.058	0.058
s38417	0.277	0.267	0.269	0.268	0.270	0.281	0.274	0.278	0.418	0.276	0.276	0.278
s38584.1	0.300	0.282	0.285	0.287	0.282	0.305	0.301	0.299	0.301	0.299	0.300	0.295
seq	0.085	0.065	0.065	0.062	0.040	0.086	0.077	0.078	0.076	0.075	0.068	0.059
spla	0.172	0.088	0.080	0.077	0.069	0.186	0.142	0.144	0.102	0.104	0.097	0.087
tseng	0.068	0.061	0.060	0.059	0.057	0.073	0.062	0.062	0.061	0.061	0.058	0.058
Geomean:	0.131	0.108	0.106	0.105	0.094	0.135	0.121	0.121	0.117	0.115	0.109	0.104
Ratio:	**1.000**	0.827	0.810	0.803	0.723	**1.000**	0.891	0.896	0.867	0.848	0.807	0.769
Ratio:			**1.000**	0.991	0.892			**1.000**	0.968	0.946	0.900	0.858

a smaller power benefit (versus priority cuts mapping) in comparison with the area-oriented runs. Both WireMap and activity-driven WireMap yield a ∼10-11% power reduction, on average. Turning to the guarded evaluation results, the "Guarded" data point shows that power is reduced by 3%, with optimal depth and without non-obvious don't cares. The "Guarded +20%" data point shows that power is reduced by 5% when circuit depth can be increased. The last two data points in Fig. 10(b) give the power reduction results when the larger set of don't cares can be utilized. Power reductions of 10% over activity-driven WireMap are possible without any increase in mapped circuit depth. Guarded evaluation can reduce power by 14% when depth can be increased by 20%. Compared with priority cuts mapping, which is not power aware, the combination of WireMap and guarded evaluation reduces power by 13-23%, depending on the don't cares model, and speed performance. Again, the power reduction results are encouraging and exhibit a clear trade-off between power and depth.

Lastly, we report the impact of guarded evaluation on post-routed critical path delay (as reported by VPR [6]). Table 3 shows the geometric mean (across all circuits) of critical path delay for the key mapping solutions presented above. Part a) of the table shows results for area-oriented mappings. Without the full set of don't cares, guarded evaluation increases critical path delay by 28%, on average, versus activity-driven WireMap. However, when non-obvious don't cares can be exploited, critical path delay is increased by 43% versus activity-driven WireMap, on average. We enforced a hard limit of *at most* a 50% increase in the number of logic levels in area-oriented guarded evaluation. Without the hard limit, the speed performance degradation was considerably worse, with little added power benefit.

Part b) of Table 3 gives results for depth-oriented mappings. Observe that even when the number of logic levels was restricted to remain optimal, critical path delay increased by 6-9%, on average, relative to activity-driven WireMap. This is likely due to routing congestion caused by increases to net fanout. When depth increases of 20% were permitted, critical path delay increased by 8% and 21% (vs. activity-dirven WireMap), without and with considering the non-obvious don't cares, respectively. Although not shown in the table, the depth-oriented activity-driven WireMap solutions have 20% higher performance, on average, versus the area-oriented activity-driven WireMap solutions.

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.

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. In this paper, we proposed guarded evaluation for FPGAs and used non-inverting paths in a circuit's AND-inverter graph to discover guarding opportunities. Our approach leverages unused circuitry that is already part of the FPGA fabric, and thus imposes little area overhead. Experimental results demonstrate that guarded evaluation can reduce switching activity by 7-22%, on average, depending on whether the mapping is area or depth-oriented and whether guarded evaluation can exploit don't cares and can relax mapping depth. Compared with a good power-aware baseline mapping, dynamic power in the FPGA interconnect is reduced by between 1-14%, again depending on the map-

Table 3: Critical path delay results.

a) Area-Oriented	
Mapping Flow	Normalized Critical Path Delay (Geomean)
Activity-Driven WireMap	1.00
Guarded	1.28
Guarded (DC)	1.43

b) Depth-Oriented	
Mapping Flow	Normalized Critical Path Delay (Geomean)
Activity-Driven WireMap	1.00
Guarded	1.06
Guarded +20%	1.08
Guarded (DC)	1.09
Guarded +20% (DC)	1.21

ping conditions and the willingness to trade-off performance for power.

An interesting direction for future work is to target both dynamic and leakage power reduction using guarded evaluation. In this paper, we essentially held outputs of guarded sub-circuits at logic-`0` when the outputs of such sub-circuits did not affect overall circuit outputs (since we altered LUT functions by taking the logical `AND` of the LUT's existing function with the guarding signal). However, we could have equally well held outputs at logic-`1` (a logical `OR` would be applied). Leakage power in CMOS circuits is known to depend strongly on the applied input vector [9]. Consequently, it is possible that holding a sub-circuit's outputs at logic-`1` or even a combination of `0`s and `1`s might improve leakage power. Other directions for future work include integrating this work with the scalable don't care analysis described in [17], and also with the power optimization work of [12]. It is unclear whether the power benefits of [12] are orthogonal to the benefits reported here. Lastly, it would be valuable to study the power benefits of guarded evaluation for a commercial FPGA, perhaps using Altera's QUIP framework [2] to bring our guarded mapping solutions into Altera's commercial tool flow. Beyond the obvious advantage of gauging power for a "real" commercial FPGA, establishing such a flow would permit us access to a more accurate switching activity model, namely one that considers glitches resulting from post-routed interconnect delays.

ACKNOWLEDGEMENTS

The authors thank Dr. Qiang Wang for his helpful comments on the manuscript.

6. REFERENCES

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

[2] Quartus-II university interface program. *Altera Corp.*, 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] Altera, Corp., San Jose, CA. *Stratix-III FPGA Family Data Sheet*, 2008.

[5] 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, 2009.

[6] 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.

[7] 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.

[8] 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.

[9] J.P. Halter and F.N. Najm. A gate-level leakage power reduction method for ultra-low-power CMOS circuits. In *IEEE Custom Integrated Circuits Conf.*, pages 475–478, 1997.

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. Scalable don't-care-based logic optimization and resynthesis. In *ACM Int'l Symp. on Field Programmable Gate Arrays*, pages 151–160, 2009.

[18] 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.

[19] A. Mishchenko, Sungmin Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *IEEE/ACM Int'l Con. on CAD*, 2007.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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