# A Power Optimization Toolbox for Logic Synthesis and Mapping

Stephen Jang Kevin Chung
Xilinx Inc.
{sjang, kevinc}@xilinx.com

Alan Mishchenko Robert Brayton
Department of EECS, University of California, Berkeley
{alanmi, brayton}@eecs.berkeley.edu

## Abstract

*The paper describes several complementary algorithms for power-aware logic optimization:*

- *SimSwitch is an efficient sequential simulator for estimating switching activity of signals in large sequential designs.*
- *PowerMap uses switching activity to make better decisions during power-aware technology mapping.*
- *PowerDC is a resynthesis algorithm that eliminates wires with high switching activity.*

*The proposed simulator draws on new ideas in logic representation and is geared for speed, e.g. it can simulate a 1M-node sequential design using 1000 bit patterns for 100 cycles in about 10 seconds on a typical one-core CPU. Experiments show that, although each technique contributes to the final quality, it is their combination that gives the best results. When applied to large industrial designs in a highly-optimized industrial flow, previous work on sequential synthesis and wire-aware technology mapping led to a 27.6% reduction in switching activity, while the techniques of this paper reduce it additionally by 19.6% without a substantial increase in runtime or degradation of other metrics.*

## 1 Introduction

With the increase of the design size and the reduction of feature size, power minimization becomes an important issue. According to current estimates, about 67% of power dissipation in FPGA devices is due to dynamic power [3]. This high power consumption is an increasing concern for FPGA vendors and their customers. Reducing power is key to lowering packaging and cooling costs, and improving device reliability [13].

Total power in an FPGA (or any semiconductor device) consists of *static power* and *dynamic power*. Static power results primarily from transistor leakage current, which is the small current that travels either from source-to-drain or through the gate oxide when the transistor is logically “off” [33]. Dynamic power dissipation is caused by signal transitions in a circuit and comes from the clock and logic networks. These transitions can be part of normal operation or can be due to glitching. The dynamic power is characterized by the equation:

$$P_{dynamic} = \frac{1}{2} f \cdot V^2 \sum_{i \in signals} C_i \cdot s_i$$

where $f$ is the clock frequency, $V$ the supply voltage, $C_i$ the capacitance switched by signal $i$, and $s_i$ is the probability of signal $i$ making a transition. An overview of leakage reduction techniques used in ASICs and microprocessor is described in [30]. However, the focus of this paper is on reducing the dynamic power consumption due to signal switching in the logic network.

Most power-aware technology mapping [3] and power-aware placement and routing [15] use switching activity information of the netlist to estimate dynamic power dissipation.

Techniques for estimating switching activity can be simulation-based or probability-based (or *vectorless*). This work uses simulation, which is more accurate but slower [16]. Sequential simulation is applied to a Boolean network consisting of logic gates and flip-flops while keeping track of the transitions. Gate-level simulation is a well-studied problem and much effort has been placed on improving its speed [17][30]. Despite many innovations, gate-level simulation has remained slow for large designs.

Logic synthesis and technology mapping transform logic implemenation by choosing among different circuit structures. Given the goal of reducing dynamic power dissipation, it is natural to seek logic structures resulting in a reduction of the total switching activity of their constituent nodes. Thus, it is important to develop a fast and accurate switching activity estimator that can drive power-aware technology mapping and resynthesis.

The present paper contributes to these goals in several ways:

- It describes an implementation of an efficient sequential simulator, *SimSwitch*, for estimating switching activity of all signals in a sequential design. This estimation is used to guide power-aware logic synthesis by evaluating different restructuring and mapping choices.
- It describes two synergistic algorithms, *PowerMap* and *PowerDC*, performing power-aware technology mapping and restructuring of the circuit while trying to minimize the total switching activity.
- It offers an experimental evaluation of these methods and demonstrates that a substantial reduction in the dynamic power is possible without increasing runtime and compromising other parameters such as area and delay.

The contributions of this paper can be extended to work for standard-cell designs. In particular, the same sequential simulator can be used to estimate switching activity of all signals. A cut-based standard-cell mapper [5] can be modified to take switching activity into account, leading to mappings with reduced power consumption. Finally, a don’t-care-based optimization environment for standard-cells can be developed based on the same methodology as [27]. This optimization will rewire a standard-cell design while replacing nets with high switching activity by nets with low switching activity. Although development of such an environment is beyond the scope of this work, our experience with other optimizations applicable to both standard-cell and LUT-based designs suggests that power-reductions for standard-cells are likely to be comparable to those for the LUT-based designs presented in this paper.

The rest of the paper is organized as follows. Section 2 provides necessary background on logic synthesis and technology mapping. Section 3 describes the sequential simulator for power estimation. Section 4 presents the power-aware logic optimization algorithms. Section 5 shows experimental results. Finally, Section 6 concludes the paper and outlines future work.

# 2 Background

## 2.1 Boolean Networks

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. The terms Boolean network and circuit are used interchangeably in this paper. If the network is sequential, the memory elements are assumed to be D-flip-flops with initial states. Terms memory elements, flop-flops, and registers are used interchangeably in this paper.

A node *n* has zero or more *fanins,* i.e. nodes that are driving *n*, and zero or more *fanouts*, i.e. nodes driven by *n*. The *primary inputs* (PIs) are nodes without fanins in the current network. The *primary outputs* (POs) are a subset of nodes of the network. If the network is sequential, it contains registers whose inputs and output are treated as additional PIs/POs in combinational optimization and mapping. It is assumed that each node has a unique integer called its *node ID.*

A *cut C* of node *n*, called *root*, is a set of nodes, called *leaves*, such that each path from a PI to *n* passes through at least one leaf. A cut is *K-feasible* if its cardinality does not exceed *K*. A cut is *dominated* if there is another cut of the same node, contained, set-theoretically, in the given cut. A *fanin* (*fanout*) *cone* of node *n* is the subset of the nodes of the network reachable through the fanin (fanout) edges from *n*.

A *maximum fanout free cone* (MFFC) of node *n* is a subset of the fanin cone, such that every path from a node in the subset to the POs passes through *n*. Informally, the MFFC of a node contains all the logic used exclusively to generate the output function of the node. When a node is removed due to redundancy, the logic in its MFFC can also be removed.

*Merging* node *n* onto node *m* is a structural transformation of a network that transfers the fanouts of *n* to *m* and removes *n* and its MFFC. Merging is often applied to a set of nodes that are proved to be equivalent. In this case, one node is denoted as the *representative* of an equivalence class, and all other nodes of the class are merged onto the representative. The representative can be any node if its fanin cone does not contain any other node of the same class. In this work, the representative is the node of the class that appears first in a topological order.

## 2.2 And-Inverter Graphs

A combinational *And-Inverter Graph* (AIG) [22] is a directed acyclic graph (DAG), in which a node has either 0 or 2 incoming edges. A node with no incoming edges is a primary input (PI). A node with 2 incoming edges is a two-input AND gate. An edge is either complemented or not. A complemented edge indicates the inversion of the signal. Certain nodes are marked as primary outputs (POs). The combinational logic of an arbitrary Boolean network can be factored [4] and transformed into an AIG using DeMorgan's rule.

*Structural hashing* of AIGs ensures that, for each pair of nodes, all constants are propagated and there is at most one AND node having them as fanins (up to permutation). Structural hashing is performed by one hash-table lookup when AND nodes are created and added to an AIG manager. When an AIG is incrementally rehashed, the changes are propagated to the fanouts, which may lead to rehashing large portions of AIG nodes.

The *size* (area) of an AIG is the number of its nodes; the *depth* (delay) is the number of nodes on the longest path from the PIs to the POs. The goal of AIG optimization by local transformations of an AIG is to reduce both area and delay.

*Sequential AIGs* add registers to the logic structure of combinational AIGs. The PIs and register outputs are called *combinational inputs* (CIs) and the POs and register inputs are called *combinational outputs* (COs). Although mostly representing the combinational logic, simplified sequential AIGs are still suitable for sequential transformations.

## 2.3 Technology Mapping with Structural Choices

A typical procedure for structural technology mapping consists of the following step: 1) cut enumeration, 2) delay-optimal mapping, 3) area recovery using heuristics, 4) producing the resulting LUT network.

A detailed description of these steps is given in [7]. The main drawback of the structural approaches to technology mapping is their dependence on the circuit structure given to the mapper (also known as the problem of "structural bias"). If the structure is bad, neither heuristics nor iterative recovery procedures will lead to a good result of mapping.

To overcome the bias of a single netlist structure, mapping with structural choices (lossless synthesis) is explored in [5] and [24]. The structural choices are constructed by recording equivalent circuit structures at a subset of nodes (called "choice" nodes). These alternative structures come from synthesis transformations, such as, AIG rewriting/balancing or co-factoring. As the mapper traverses over these "choice" nodes in the subject graph, it can explore the multiple alternative implementations simultaneously. It is found that using structural choices over multiple passes of LUT mapping yields significantly better results, compared to mapping without choices or running only a single iteration of mapping with choices [24]. Mapping with structural choices is used in the experiments results section.

## 2.4 Sequential Synthesis

Combinational synthesis involves changing the combinational logic of the circuit with no knowledge of its reachable states. As a result, the Boolean functions of the POs and register inputs are preserved for any state of the registers.

In contrast, sequential synthesis can modify the circuit while preserving behavior on the reachable states and allowing arbitrary changes on the unreachable states. Thus after sequential synthesis, the POs and register inputs can be changed combinational functions of the register outputs and PIs, but the resulting circuit is sequentially-equivalent to the original.

A verifiable sequential synthesis is described in [27]. This is based on identifying pairs of sequentially-equivalent nodes/registers (i.e., signals having the same or opposite values in all reachable states). Such equivalent nodes/registers can be merged without changing the sequential behavior of the circuit, leading to substantial reductions, e.g. some pieces of logic can be discarded because they no longer affect the POs. The sequential synthesis technique is used in the experimental results section.

# 3 Sequential Simulation (*SimSwitch*)

A sequential simulator applies sequences of values to the primary inputs. It is assumed that the initial state of the design is known and used to simulate the first frame. In subsequent frames, the state derived at the previous iteration is used. Input information (such as the probability of a transition occurring) can be given by the user or produced by another tool (for example, if an input trace is known, it may be applied to simulate the design).

In this work, sequential simulation is used to estimate switching activity of registers and internal nodes. This is used later to guide heuristic power-aware optimization. The design is sequentially simulated for a fixed number of timeframes. The random input patterns are generated to have a 0.5 probability of the probability of the node changing its value (toggle rate) is fixed. Simulation is performed from the initial state for a fixed number of cycles (typically, 64), of which the first few (typically, 16) are skipped as not representative of normal operation; the remaining ones are used to accumulate the switching activity.

Previous work [6][8][12][18] concluded that the main problem with sequential simulation for switching-activity estimation is the prohibitive runtime of the simulator. In this work, we address this problem with *SimSwitch*, based on new logic representation and manipulationmethods. The salient features of the new simulator are described below.

In general, the smaller the memory footprint of the simulator, the faster it runs. This is due to a CPU having typically a local cache ranging in size from 2Mb to 16Mb. If an application requires more memory than fits into the cache, repeated cache misses cause the runtime to degrade. Therefore, the challenge is to design the simulator that uses minimalistic data-structures without compromising the computation speed.

We found three orthogonal ways of reducing the memory requirements of the simulator, without impacting its performance.

### 3.1 Compacting Logic Representation

Sequential designs are represented as AIGs. A typical AIG package uses 32 or more bytes to represent one AIG object, i.e. an internal AND node or a combinational input/output, which includes flop outputs/inputs. We developed a special AIG package that requires only 12 bytes per object. In the case of an internal node, two integer fields, four bytes each, are used to store the fanin IDs. The third integer field is used temporarily, during duplication to store the ID of the object copy, or during structural hashing to store the ID of the next node in the hash table. The new representation takes about 12Mb for a typical AIG with 1M objects.

### 3.2 Recycling Simulation Memory

When simulation is applied to a large sequential design, storing simulated values for all nodes in each timeframe requires a lot of memory. One way of saving memory, is to use the simulation information as soon as it is computed and recycle the memory when it is not needed. For example, to estimate switching activity, we are only interested in counting the number of transitions seen at each node. For this, an integer counter can be used, thereby adding four bytes per object to the AIG package memory requirements, while the simulation information does not have to be stored.

Additionally, there is no need to allocate simulation memory for each object of the AIG. At any time during simulation, we only need to store simulation values for each combinational input/output and *some* internal nodes. For industrial designs, the number of internal nodes where simulation information should be stored is typically 1-10% of the total. These are the nodes on the *simulation frontier*, which is defined, at any time during simulation, as all the nodes whose fanins are already simulated but at least one fanout is not yet simulated.

The notion of simulation frontier has been used also to reduce memory requirements for the representation of priority cuts [23].

### 3.3 Bit-Parallel Simulation of Two Time Frames

A naïve approach to estimate the transition probability for each AIG node would be to store simulation patterns in two consecutive timeframes. This information can be compared (using bitwise XOR) and the number of ones in the bitwise representation is accumulated while simulating the timeframes. However, saving simulation information at each node for two consecutive timeframes may require too much memory. For example, an AIG with 1M objects requires 80Mb to store the simulation information for two timeframes, assuming 10 machine words (40 bytes) per object.

This increase in memory can be avoided by simultaneously simulating data belonging to two consecutive timeframes. In this case, comparison across the timeframes can be made immediately, without memorizing the result of the previous computation. This leads to duplicating the computation effort by simulating every pattern twice, one using the previous state value and the other using the current state value. However, the speedup due to not having to traverse the additional memory greatly outweighs the disadvantage of the re-computation.

## 4 Approaches to Power Minimization

In this section, two orthogonal ways of using switching activity are described. The first (*PowerMap*) uses switching activity to make better decisions during technology mapping while the second (*PowerDC*) does power-aware re-synthesis after mapping.

### 4.1 Technology Mapping (PowerMap)

LUT-based mappers [7][19][23] have evolved from the first technology mapping solutions for FPGAs [11][9], to those that have reduced runtime, improved quality of results, and allow users to select cost-functions employed during decision making, as in [21]. In particular, a recent technology mapper [23] was successfully modified to accommodate heuristics for reducing wire-length and improving design routability (*WireMap*) [12][14].

We propose a similar modification of the priority-cut based mapper [23] geared towards reducing switching activity, resulting in an algorithm called *PowerMap*. Since the basic mapping algorithm is described in detail in [23] and [14], here we only describe the changes in the new cost function which use switching activity to prioritize the cuts during technology mapping.

The intuition behind power-aware technology mapping is to try to cover the nodes with high switching activity (or *hot* nodes) so they are hidden inside a LUT. Since the capacitance inside a LUT is very small, power consumption will be reduced.

The following three metrics are compared below:

- area flow [10][19][23],
- edge flow (*WireMap*) [14],
- switching flow (*PowerMap*) (this paper).

*Area flow* (*AF*) is defined as:

$$AF(n) = Area(n) + \sum_i \frac{AF(Leaf_i(n))}{NumFanout(Leaf_i(n))}, \qquad (1)$$

where $Area(n)$ is the area of the LUT used to map node $n$, $Leaf_i(n)$ is the $i$-th leaf of the representative cut of node $n$, and $NumFanouts(Leaf_i(n))$ is the number of fanouts of node $Leaf_i(n)$ in the currently selected mapping.

*Edge flow* (*EF*) is defined as:

$$EF(n) = Edge(n) + \sum_i \frac{EF(Leaf_i(n))}{NumFanout(Leaf_i(n))}, \qquad (2)$$

where $Edge(n)$ is the total number of fanin edges (or wires) of the LUT used to map the current representative cut of node $n$, while other notations are the same as in (1).

*Switching flow* is defined as:

$$SwitchFlow(n) = Switch(n) + \sum_i \frac{SwitchFlow(Leaf_i(n))}{NumFanout(Leaf_i(n))}, \qquad (3)$$

where *Switch*(*n*) is the total switching activity at the output of node *n*, computed using sequential simulation (from Section 3). Intuitively, each of these flows is an estimate of a resource (area, wiring, switching) associated with the logic rooted at node *n*.

The main idea is to compute for each cut three different metrics (flows) capturing the global view of the netlist during mapping: area flow, edge flow, and switching flow. When computing these metrics, each cut is characterized by 1) its own resources used and 2) the resources used by its fanins. The value of a resource of a fanin is divided by the number of the fanin's fanouts.

Consider node *n*1 in Figure 1. The direct resources of *n*1 are: 1) the node itself; 2) the three input edges; 3) the total resources for the three edges. For nodes *n*2 and *n*4, half of the resources is used by node *n*1 while the other half is used by other fanouts. For node *n*3, all resources are used by node *n*1.

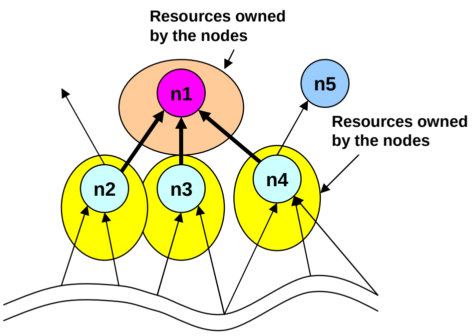


**Figure 1:** Metrics used in PowerMap.

The use of the cost function (3) during power-aware technology mapping is similar to that of its use in (2) in wire-aware mapping [14]. The main differences are as follows. When comparing two cuts, their area flows are compared first, while their switching flows are used as the first tie-breaker, i.e. if the area flows of two cuts are equal up to a certain delta (say, 0.001), the decision of what cut to use is made based on the switching flow. Similarly, edge flow is used as the second tie-breaker.

Similarly, in the second phase of technology mapping when local areas of cuts are compared [14], local switching is used as the first tie-breaker and edge flow is used as the second tie-breaker. The local area of a cut is the sum of the areas of the LUTs in the MFFC of the cut. Similarly, the local switching of a cut is the sum of switching activities of the LUTs in the cut's MFFC.

Formulas (1)-(3) provide different applications of the notion of a flow when applied to technology mapping. The cost functions measure increasingly complex netlist metrics, starting from area (LUT count) to the number of fanins (routing wires) to the total switching activity of fanins (power estimate), while the notion of a flow captures the fact that each incoming flow is shared by the fanouts of the fanins, which gives a global view of each cost function during technology mapping.

## 4.2 SAT-Based Resynthesis (PowerDC)

An efficient approach to resynthesizing logic networks after technology mapping is given in [27]. Its features are listed below:

- substantial optimization power, due to the use of internal don't-cares;
- scalable local computation, due to the use of windowing;
- computation speed, due to the use of Boolean satisfiability for functional manipulation, and
- ability to accommodate various optimization objectives.

*Resubstitution* is a logic restructuring technique that modifies the local space of a mapped node by adding or removing its fanins. When applied to a Boolean network, resubstitution can be iteratively applied, aiming at minimizing a given cost function, such as area, delay, wire-length, etc.

The main idea of the power-aware flavor of the resynthesis algorithm, called *PowerDC*, is summarized below. For details on the basic algorithm and its implementation, refer to [20][27].

Given a mapped network, it is first transformed into an AIG by applying Boolean decomposition of the logical functions of the LUTs into two-input gates. Then sequential simulation is applied to the AIG, resulting in the transition probabilities for all the AIG nodes. This information is back-annotated to the mapped network to provide the switching activity for all nodes in the mapping.

Next, a subset of wires, called *hot wires*, is identified. Based on the experiments, if the toggle rate for primary inputs is 0.5, hot wires are those with switching probability greater than 0.4. Note that switching computed as a transition probability may be higher than 0.5 (but cannot exceed 1.0). Another way of defining hot wires might be a fixed ratio (say, 10%) of the wires with the highest switching activity.

Based on an experiment described in Section 5, about 13.9% of wires are hot and these contribute about 86% of the total power. 82.87% of wires are cool but these only contribute 1.74% of the total power. A detailed distribution of wires (and nodes) by switching activity and their power contribution are shown in Figure 2. Since the vast majority of dynamic power is dissipated in a small percentage of hot wires, power reduction techniques targeting hot wires are very effective.

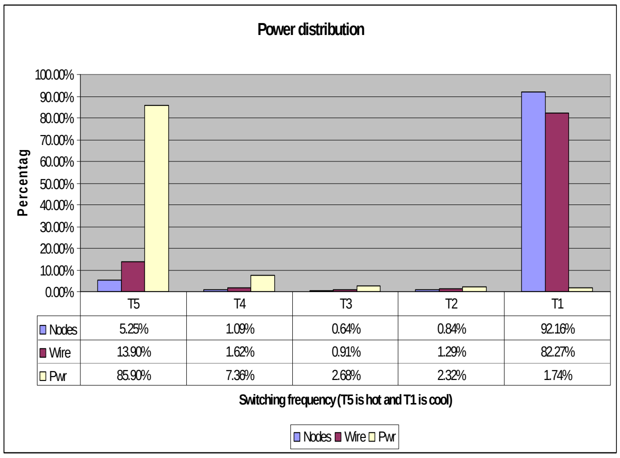


**Figure 2:** Distribution of wires and nodes by toggle rate.

After hot nodes are found, SAT-based resubstitution [27] is applied targeting hot nodes as fanins of other mapped nodes. The goal is to remove or replace them by cooler fanins, as shown in Figure 3. In this figure, Node *n*2 is a hot node, *n*8 is a cool node, and Wire $n2 \rightarrow n9$ is the target to be either removed or replaced by a cooler wire, such as $n8 \rightarrow n9$ in this case. This transformation tends to reduce the total switching of the mapping, which, in turn, leads to reduced dynamic power after place-and-route.

To save runtime, switching activity is not updated after individual steps of resubstitution. This is reasonable because, although Boolean functions of the nodes after resynthesis with observability don't-cares may be changed, their switching activity does not change substantially. Even when it changes, the percentage of nodes whose hot/cool status is modified during resynthesis, is typically negligible.

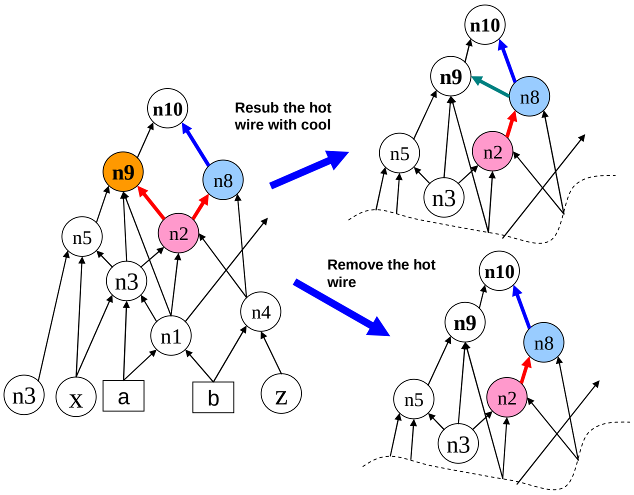


**Figure 3:** Two ways to "cool down" hot wires.

## 5 Experimental Results

The algorithms of this paper are implemented in ABC [1]. Experimental evaluation targeting 6-LUT implementations were performed using a suite of 20 large industrial designs ranging in size from 12K to 165K 6-LUTs. The experiments were run on an Intel Xeon 2-CPU 4-core computer with 8GB of RAM. The resulting networks were verified using combinational equivalence checker in ABC (command *cec*).

The power consumption of a circuit includes the sum of the powers consumed by each wire, which is the product of the capacitance and switching activity for all wires. Because the netlists were not placed yet, the capacitance of wires were not known. A unit-capacitance model is used in our experiments so the focus on only reducing total switching activity for all wires.

The sequential simulator for estimating the switching activity was based on a new AIG package, called "gia". The following commands in ABC were extended to be power-aware. In each case, new switch "-p" enables switching activity minimization:

- Technology mapping for FPGAs [23] (command *if –p).*
- SAT-based resubstitution [27] (command *mfs –p).*
- Switching activity report (command *ps –p).*

Several other commands, including sequential synthesis and mapping with structural choices, were used in this experiment:

- Structural sequential cleanup [26] (command *scl*).
- Partitioned register-correspondence computation using simple induction [26] (command *lcorr*).
- Partitioned signal-correspondence computation using *k*-step induction [26] (command *scorr*).
- Computation of structural choices [3] (command *dch*).

### 5.1 Simulation Runtime

As mentioned, one of the main issues with sequential simulation is potentially long runtime. In this section, we perform an experiment to show that running *SimSwitch* offers affordable runtime for large designs.

Four industrial designs ranging from 304K to 1.3M AIG nodes were simulated with different numbers of simulation patterns, ranging from 2,560 to 20,480. The input toggle rate was assumed to be 0.5. The results are shown in Table 1. Columns "AIG" and "FF" show the number of AIG nodes and registers. The runtimes for different sizes of inputs patterns are shown in the last columns. Note that the runtimes are quite affordable even for Design C4 with 1.3M AIG nodes. In most cases, 2,560 patterns are sufficient for node switching activity rates to converge to a steady state.

**Table 1: Runtime of SimSwitch.**

| Design | AIG | FF | Runtime for inputs patterns (seconds) | | | |
|---|---|---|---|---|---|---|
| | | | 2560 | 5120 | 10240 | 20480 |
| C1 | 304K | 1585 | 0.1 | 0.2 | 0.2 | 0.4 |
| C2 | 362K | 27514 | 2.7 | 2.9 | 4.1 | 6.6 |
| C3 | 842K | 58322 | 7.4 | 7.6 | 10.2 | 18.2 |
| C4 | 1306K | 87157 | 12.1 | 15.4 | 15.7 | 24.2 |

### 5.2 Power-Aware Optimizations

To evaluate the contribution of power-aware mapping and resynthesis, three experiments were performed, denoted "Baseline", "FullOpt", and "PowerMap".

- "Baseline" corresponds to two runs of (*dch; if –e*). It performs priority-cut based technology mapping with structure choices. WireMap [14] is not used for "Baseline" because WireMap is known to reduce power dissipation as a by-product of wire count minimization.
- "FullOpt" is the complete flow including high-effort sequential and combinational logic synthesis. Sequential synthesis (*scl; lcorr; scorr*) [26] is run first to remove sequentially equivalent nodes and registers. Then, two iterations of combinational synthesis and mapping with structural choices are performed (*dch; if*). WireMap is used in this flow. It reduces 10% of the wires on top of "Baseline" [14].
- In the "PowerMap" run, "FullOpt" is used as the input netlist followed by two runs (*dch; if –p*), which is the power-aware technology mapping developed in this paper.
- In the "PowerDC" run, the results of "PowerMap" are used as input followed by two iterations of power-aware resynthesis (*mfs –p*).

The sequential simulator *SimSwitch* is used to drive the power-aware logic synthesis. In sequential simulation, 2,560 random input patterns with a toggle rate of 0.5 were used. The results are reported in Table 4. Columns "PI", "PO", "FF", and "LUT" show the number of primary inputs, primary outputs, flip-flops, and 6-input LUTs. Columns "Lv" and "Pwr" show the number of logic levels and the total switching activity of the network.

The experimental results lead to the following observations:

- "FullOpt" improved on "Baseline" in the number of registers, LUTs, and logic levels by 14.8%, 12.2%, and 3%, respectively. Power is reduced by 27%.
- "FullOpt", "PowerMap", and "PowerDC" have the same number of registers because only combinational synthesis is used in "PowerMap" and "PowerDC".
- "PowerMap" produces an additional 10.5% power reduction on top of "FullOpt".
- "PowerDC" produces an additional 10.1% power reduction on top of "PowerMap".
- The overall power reduction for "PowerDC" vs. "FullOpt" is 19.6%.
- The overall power reduction for "PowerDC" vs. "Baseline" is 41.9%.

To investigate robustness of the algorithm, we performed the same experiments and changed the toggle rate of the inputs from 0.5 to 0.25. The results are summarized in Table 2.

- "PowerMap" produces an additional 9.8% power reduction on top of "FullOpt"

- "PowerDC" produces an additional 8.7% power reduction on top of "PowerMap".
- The overall power reduction for "PowerDC" vs. "FullOpt" is 17.7%.
- The overall power reduction for "PowerDC" vs. "Baseline" is 39.7%

**Table 2:** Comparison of power-optimization algorithms (inputs toggle rate is 0.25).

| Power | BaseLine | FullOpt | PwrMap | PwrDC |
|---|---|---|---|---|
| Geomean | 17312.8 | 12687.5 | 11445.7 | 10445.8 |
| Ratio | 1 | 0.733 | 0.661 | 0.603 |
| Ratio | | 1 | 0.902 | 0.823 |
| Ratio | | | 1 | 0.913 |

## 5.3 Wire Distribution Analysis

For the same suite of 20 designs, the changes in the distribution of wires by their switching activity as a result of resynthesis is analyzed. The results are reported in Table 3 and Figure 3.

Column "T5" is the total number of wires whose switching probability exceeds 0.4. These "hot wires" are targeted by power reduction. Column "T4" is the total number of wires whose switching probability is between 0.3 and 0.4. Columns T3, T2, and T1 are defined similarly. Thus the T1 wires are the cool wires. In general, they don't contribute much to the power because their switching probability is less than 0.1.

The results in Figure 3 lead to the following observations:

- For "PowerMap vs. FullOpt", hot (T5) wires are reduced by 13.22%, and T4 are increased by 4.96%. This indicates that PowerMap reduces the number of hot wires.
- For "PowerDC vs. PowerMap", total wires are reduced by 2.7% but power is reduced by 10.1%. Thus, the resynthesis has removed the "right" wires, i.e., the hot wires. Hot wires are reduced by 11.04%. The cooler wires are reduced also but the reduction is smaller in this case.

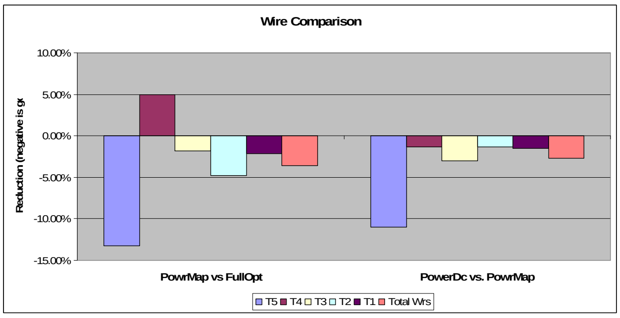


**Figure 3:** The changes in wire ratios after two power-aware transforms.

In addition to wire count reduction for hot wires, the wire distribution in terms of toggle rates is also improved. Figure 4 shows the power distribution before and after optimization. "Wire" and "Wire2" are the percentages of wires, while "Pwr" and "Pwr2" are the percentages of power contributions before and after optimization. The chart leads to the following conclusions:

- After "FullOpt", 13.9% (82.3%) of the wires are hot (cool).
- After "PowerMap", 11.4% (84.6%) of wires are hot (cool).
- After "FullOpt", 85.9% (1.7%) of the power is contributed by hot (cool) wires.
- After "PowerMap", 82.6% (2.0%) of power is contributed by hot (cool) wires.

The above observations show that the power contributed by hot wires is less than "Baseline" while the power contributed by cool wires is more than "Baseline". The conclusion is that the power-aware logic synthesis was able to shift the peak of power consumption to lower-switching wires.

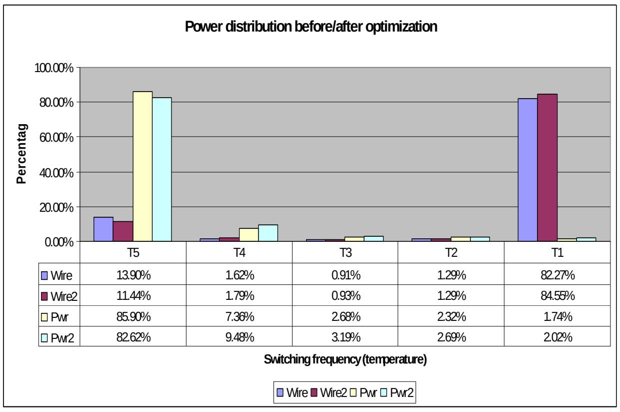


**Figure 4:** Power distribution before/after optimization.

## 6 Conclusions

This paper describes a toolbox for power-aware logic synthesis and mapping. Estimation of power consumption at the early stages of the design flow is based on calculating the probabilities of signal transition during sequential simulation.

The toolbox also includes two techniques for optimizing the design during synthesis and mapping to reduce switching activity and thereby minimize dynamic power consumption:

- **PowerMap**: a power-aware LUT mapper that extends [23] to prioritize cuts based on switching activity of the nodes.
- **PowerDC**: a SAT-based engine for resubstitution with don't-cares that extends [27] to remove signals with high switching activity (so called "hot signals").

Estimation of power dissipation is efficiently performed by a new sequential simulator, *SimSwitch*. The estimation converges for most industrial designs after a reasonable number of simulation cycles. This allows for making targeted heuristic decisions during logic synthesis and technology mapping

Future work will include:

- Speeding up switching activity estimation (it is estimated that the current implementation can be made 50% faster).
- Extending switching activity computation to include glitch estimation.
- Implementing other power-aware commands, such as computing better choices to reduce power and logic restructuring to reduce power.
- Developing sequential techniques for power reduction, such as clock-gating, which uses induction to compute signals that are valid clock gates on the reachable states.

## Acknowledgements

This work is supported in part by SRC contracts 1444.001 and 1875.001, NSF grant CCF-0702668 entitled "Sequentially Transparent Synthesis", and the California MICRO Program with industrial sponsors Actel, Altera, Atrenta, Calypto, IBM, Intel, Intrinsity, Magma, Mentor Graphics, Synopsys, Synplicity (Synopsys), Tabula, Verific, and Xilinx.

## References


[1] Berkeley Logic Synthesis and Verification Group, ABC: A system for sequential synthesis and verification, Release 904xx. http://www.eecs.berkeley.edu/~alanmi/abc/

[2] Altera, "Stratix II vs. Virtex-4 power comparison and estimation accuracy" (white paper). http://www.altera.com/literature/wp/wp_s2v4_pwr_acc.pdf

[3] J. Anderson and F. N. Najm, "Power-aware Technology Mapping for LUT-Based FPGAs," *IEEE Int. Conf. on Field Programmable Technology*, Dec. 2002, pp. 211-218.

[4] R. Brayton and C. McMullen, "The decomposition of logic functions," Proc. *ICCAD* '97, pp. 78-82

[5] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping", *Proc. ICCAD '05*, pp. 519-526

[6] D. Chen, J. Cong, and P. Pan, "FPGA design automation: A survey," *Foundations and Trends in Electronic Design Automation*, Vol. 1(3), November 2006, pp.195-330.

[7] D. Chen and J. Cong. "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," *Proc. ICCAD'04*, pp. 752-757.

[8] L. Cheng, D. Chen, and M.D. Wong, "GlitchMap: FPGA technology mapper for low power considering glitches". *Proc. DAC '07*.

[9] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs", *IEEE TCAD*, Vol. 13(1), 1994, pp. 1-12.

[10] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," *Proc. FPGA'99*, pp. 29-36.

[11] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays", *Proc. DAC '90*, pp. 613-619.

[12] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits" *Proc. DAC'92.*

[13] S. Gupta and J. Anderson, "Optimizing FPGA power with ISE design tools," Issue 60, 2007, pp. 16-19, http://www.xilinx.com/publications/xcellonline/xcell_60/xc_pdf/p16-19_60-gupta.pdf

[14] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "WireMap: FGPA technology mapping for improved routability". *Proc. FPGA '08*.

[15] J. Lamoureux and S.J.E. Wilton, "On the Interaction between Power-Aware FPGA CAD Algorithms,", *Proc. ICCAD*, Nov. 2003.

[16] J. Lamoureux and S.J.E. Wilton, "Activity estimation for Field-Programmable Gate Arrays", *Proc. Intl Conf. Field-Prog. Logic and Applications (FPL)*, 2006, pp. 87-94.

[17] J. N. Kozhaya and F. Najm, "Accurate power estimation for large sequential circuits", *Proc. ICCAD*'97.

[18] J. Lamoureux, "Modeling and reduction of dynamic power in Field-Programmable Gate Arrays", *Ph.D. Thesis*, 2007.

[19] V. Manohara-rajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," *Proc. IWLS '04*, pp. 14-21.

[20] A. Mishchenko and R. Brayton, "SAT-based complete don't-care computation for network optimization", *Proc. DATE '05*.

[21] A. Mishchenko, S. Chatterjee, R. Brayton, and M. Ciesielski, "An integrated technology mapping environment", *Proc. IWLS '05*, pp. 383-390.

[22] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", *Proc. DAC '06*, pp. 532-536.

[23] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts", *Proc. ICCAD '07*, pp. 354-361.

[24] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs", Proc. *FPGA* '06, pp. 41-49

[25] A. Mishchenko, R. K. Brayton, and S. Jang, "Global delay optimization using structural choices", *Proc. IWLS'08*, pp. 1-6.

[26] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", *Proc. ICCAD'08*, pp. 234-241.

[27] A. Mishchenko, R. Brayton, J.-H. R. Jiang, S. Jang, "Scalable don't-care-based logic optimization and resynthesis", *Proc. FPGA '09*.

[28] J. Najm, "Transition density: A new measure of activity in digital circuits", *IEEE Trans. CAD*, Vol. 12(2), 1993, pp. 310-323.

[29] PowerPlay Power Analysis, *Quartus II 9.0 Handbook*, Vol. 3, http://www.altera.com/literature/hb/qts/qts_qii53013.pdf

[30] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits", in *Proceedings of the IEEE*, pages 305-327, 2002

[31] E. Todorovich, M. Gliabert, G. Sutter, S. Lopez-Buedo, and E. Boemo, "A tool for activity estimation in FPGAs", *Proc. Intl. Conf. Field-Prog. Logic (FPL)*, 2002, pp. 340-349.

[32] *Xilinx Power Estimator User Guide*, http://www.xilinx.com/products/design_resources/power_central/ug440.pdf

[33] Xilinx White Paper: *Power Consumption in 65 nm FPGAs* http://www.xilinx.com/support/documentation/white_papers/wp246.pdf


**Table 3:** Wire distribution by toggle rate (T5 stands hot wires).

| | T5 (Hot) | T4 | T3 | T2 | T1 (Cool) | Total Wrs |
|---|---|---|---|---|---|---|
| **FullOpt** | 49992 | 5838 | 3283 | 4642 | 295828 | 359583 |
| **PowrMap** | 43383 | 6127 | 3224 | 4419 | 289494 | 346647 |
| **Reduction** | **-13.22%** | **4.96%** | **-1.80%** | **-4.80%** | **-2.14%** | **-3.60%** |
| **PowrMap** | 43383 | 6127 | 3224 | 4419 | 289494 | 346647 |
| **PowrDC** | 38591 | 6047 | 3126 | 4358 | 285162 | 337285 |
| **Reduction** | **-11.04%** | **-1.31%** | **-3.03%** | **-1.38%** | **-1.50%** | **-2.70%** |

**Table 4:** Comparison of power-optimization algorithms. (inputs toggle rate is 0.5).

| Design | Statistics | | Base | | | | FullOpt | | | | PowerMap | | | | PowerDC | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | PI | PO | FF | LUT | Lv | Pwr | FF | LUT | Lv | Pwr | FF | LUT | Lv | Pwr | FF | LUT | Lv | Pwr |
| **D01** | 4725 | 16657 | 43309 | 71956 | 14 | 63592 | 41868 | 70060 | 13 | 53666 | 41868 | 67214 | 12 | 49268 | 41868 | 65693 | 12 | 46139 |
| **D02** | 8561 | 20356 | 65881 | 144295 | 27 | 59306 | 41327 | 90304 | 27 | 34090 | 41327 | 86789 | 25 | 33418 | 41327 | 85798 | 25 | 30497 |
| **D03** | 8879 | 52334 | 41521 | 123845 | 11 | 79928 | 39884 | 122947 | 11 | 66571 | 39884 | 119625 | 10 | 61239 | 39884 | 117946 | 10 | 57068 |
| **D04** | 781 | 5563 | 16205 | 50328 | 10 | 38961 | 15123 | 48392 | 10 | 29901 | 15123 | 47361 | 9 | 26706 | 15123 | 46651 | 9 | 23938 |
| **D05** | 3343 | 5533 | 23740 | 65704 | 17 | 37158 | 18933 | 55512 | 13 | 25415 | 18933 | 52086 | 12 | 20771 | 18933 | 50973 | 12 | 16247 |
| **D06** | 3664 | 21989 | 29947 | 36188 | 8 | 36899 | 27896 | 33733 | 8 | 27844 | 27896 | 33220 | 8 | 25985 | 27896 | 32962 | 8 | 25351 |
| **D07** | 1284 | 2929 | 81328 | 164437 | 10 | 90849 | 74898 | 153317 | 11 | 66515 | 74898 | 145918 | 10 | 56595 | 74898 | 143357 | 10 | 52249 |
| **D08** | 261 | 359 | 4586 | 12412 | 22 | 6480 | 4463 | 12325 | 23 | 4938 | 4463 | 11986 | 21 | 4461 | 4463 | 11844 | 21 | 3326 |
| **D09** | 2561 | 9586 | 23612 | 55639 | 9 | 40244 | 16290 | 37736 | 8 | 21981 | 16290 | 35123 | 7 | 20053 | 16290 | 34103 | 7 | 17538 |
| **D10** | 3765 | 9987 | 37630 | 96677 | 31 | 62304 | 36665 | 95005 | 31 | 50132 | 36665 | 90167 | 30 | 43864 | 36665 | 88191 | 30 | 37483 |
| **D11** | 2418 | 6000 | 34834 | 76798 | 19 | 51724 | 34446 | 75724 | 20 | 43953 | 34446 | 70531 | 18 | 36783 | 34446 | 69117 | 18 | 32231 |
| **D12** | 1134 | 3965 | 8371 | 14939 | 12 | 11632 | 8256 | 15316 | 12 | 10480 | 8256 | 14825 | 12 | 9254 | 8256 | 14623 | 12 | 8624 |
| **D13** | 210 | 299 | 6662 | 15888 | 11 | 5766 | 6591 | 15381 | 11 | 3893 | 6591 | 14960 | 11 | 3265 | 6591 | 14710 | 11 | 2498 |
| **D14** | 2326 | 3713 | 61789 | 109865 | 18 | 22738 | 36887 | 67338 | 19 | 7956 | 36887 | 66681 | 17 | 7582 | 36887 | 66097 | 17 | 7537 |
| **D15** | 2312 | 5523 | 26233 | 49031 | 7 | 35819 | 13575 | 27498 | 6 | 20779 | 13575 | 26234 | 6 | 17972 | 13575 | 25871 | 6 | 16819 |
| **D16** | 5124 | 21571 | 39127 | 146931 | 17 | 18685 | 37772 | 146298 | 16 | 14087 | 37772 | 139226 | 16 | 13095 | 37772 | 135903 | 16 | 12966 |
| **D17** | 2587 | 7025 | 6975 | 12528 | 12 | 8152 | 6429 | 11780 | 12 | 7124 | 6429 | 11957 | 11 | 6904 | 6429 | 11834 | 11 | 6685 |
| **D18** | 3918 | 6110 | 23727 | 35996 | 13 | 24990 | 22260 | 34630 | 12 | 21728 | 22260 | 34215 | 10 | 19339 | 22260 | 33994 | 10 | 18297 |
| **D19** | 4633 | 7540 | 28376 | 43515 | 13 | 31148 | 26241 | 41415 | 12 | 26941 | 26241 | 41120 | 10 | 24176 | 26241 | 40839 | 10 | 22822 |
| **D20** | 6631 | 19368 | 58322 | 158216 | 25 | 101921 | 53581 | 144455 | 25 | 75781 | 53581 | 139174 | 22 | 64332 | 53581 | 136747 | 22 | 54554 |
| **Geom** | 2438 | 6590 | 25656 | 55456 | 14.1 | 31294 | 22118 | 48700 | 13.7 | 22621 | 22118 | 47049 | 12.6 | 20238 | 22118 | 46318 | 12.6 | 18190 |
| **Ratio** | | | **1** | **1** | **1** | **1** | **0.862** | **0.878** | **0.97** | **0.723** | **0.862** | **0.848** | **0.90** | **0.647** | **0.862** | **0.835** | **0.90** | **0.581** |
| **Ratio** | | | | | | | **1** | **1** | **1** | **1** | **1.000** | **0.966** | **0.92** | **0.895** | **1.000** | **0.951** | **0.92** | **0.804** |
| **Ratio** | | | | | | | | | | | **1** | **1** | **1** | **1** | **1.000** | **0.984** | **1.00** | **0.899** |