GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches *

Lei Cheng,
CS Dept & Coordinated Science Lab
University of Illinois at Urbana-Champaign
lcheng1@uiuc.edu

Deming Chen, Martin D.F. Wong
ECE Dept & Coordinated Science Lab
University of Illinois at Urbana-Champaign
{dchen,mdfwong}@uiuc.edu

ABSTRACT

In 90-nm technology, dynamic power is still the largest power source in FPGAs [1], and signal glitches contribute a large portion of the dynamic power consumption. Previous power-aware technology mapping algorithms for FPGAs have not taken into account the glitch power reduction. In this paper, we present a dynamic power estimation model and a new technology mapping algorithm considering glitches. To the best of our knowledge, this is the first work that explicitly reduces glitch power during technology mapping for FPGAs. Experiments show that our algorithm, named *GlitchMap*, is able to reduce dynamic power by 18.7% compared to a previous state-of-the-art power-aware algorithm, EMap [2].

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer Aided Design

General Terms

Algorithm, Design, Experimentation

Keywords

FPGA technology mapping, dynamic power, glitch

1. INTRODUCTION

FPGAs are not power efficient. It is estimated that an FPGA design consumes 10 times more power than a functionally equivalent ASIC design [3]. Many FPGA vendors report that power dissipation is one of the primary concerns of their customers. The LUT-based FPGA architecture dominates the existing programmable chip industry, in which the basic programmable logic element is a K-input lookup table. A K-input LUT (K-LUT) can implement any Boolean functions of up to K variables. FPGA technology mapping converts a given Boolean circuit into a functionally equivalent network comprised only of LUTs. The technology mapping process has a significant impact on the area, performance, and power for FPGA designs. In this paper, we present an FPGA technology mapping algorithm targeting both delay and power minimization.

FPGA technology mapping algorithms can be classified into the following categories: area minimization algorithms, including Chortle-crf [4] and MIS-pga [5]; delay minimization algorithms, including FlowMap [6] and EdgeMap [7]; area minimization algorithms under optimum delay, including DAOmap [8] and the mapping algorithm based on lossless synthesis [9]; and power minimization algorithm, including mapping algorithms in [10] and [11], PowerMap [12], PowerMinMap [13], Emap [2], and DVmap [14]. Techniques used in these algorithms include bin packing, dynamic programming, greedy algorithm, binate covering, network flow algorithm, and cut-enumeration algorithm, etc.

There are three power sources in FPGAs: switching power, short-circuit power, and static power. The first two types of power together are called *dynamic power*, and they can only occur when a signal transition takes place. The third type of power, static power, is the power consumed when there is no signal transition in a gate. In 90-nm technology, dynamic power is still the dominating power source for FPGAs. For example, the core dynamic power takes about 67% of the total power consumption in Altera's Stratix II FPGAs [1] (note that if we count I/O power as dynamic power as well, this percentage becomes 78%).

There are two types of signal transitions. One is the signal transition necessary to perform the required logic function between two consecutive clock ticks, and it is called *functional transition*. The other is the unnecessary signal transition due to the unbalanced path delays to the inputs of a gate, and it is called *spurious transition* or *glitch*. Glitch power can be a significant portion of the dynamic power. Based on the study in [15], glitch power can be 60% of the dynamic power consumed in the logic. In the datapath of some data-flow intensive designs, glitch transitions can be 4-5X more than functional transitions [16]. Therefore, it is very important to reduce glitches for total power reduction. However, we are not aware of any FPGA technology mapping algorithms that consider glitches in the literature.

In this paper, we present a switching activity estimation model considering glitches for FPGAs, and develop our technology mapping algorithm based on this model. The model is based on the concept of transition density [17], which is the average number of transitions per unit time. Signal probability, which is the ratio of the time the signal is in logic 1 to the total observation time, is very important to compute the transition density. We provide a new algorithm to compute the signal probability, which is 2X more accurate compared to previous works. Our mapping algorithm employs the cut-enumeration method, and uses a novel delay-relaxation-based cost propagation technique to improve the mapping quality. Compared to a previous state-of-the-art power-aware technology mapping algorithm, EMap, our algorithm is 18.7% better for dynamic power reduction on average using a commercial FPGA power estimator.

The rest of this paper is organized as follows. We provide related definitions in Section 2. In Section 3, we introduce some related techniques for signal probability estimation, signal transition estimation, and cut enumeration. Section 4 presents our detailed algorithm. The results are shown in Section 5, and we conclude this paper in Section 6.

2. DEFINITIONS

A Boolean network can be represented by a DAG where each node represents a logic gate, and a directed edge (i, j) exists if the output of gate i is an input of gate j. A PI node has no incoming edges and a PO node has no outgoing edges. We treat the flip-flop outputs as special PIs and the

*This work is partially sponsored by Altera Corporation through a research grant. We used machines donated by Intel.

DAC 2007, June 4–8, 2007, San Diego, California, USA.

flip-flop inputs as special POs, and make no distinction in terms of notation. We use $input(v)$ to denote the set of nodes which are fanins of gate v. Given a Boolean network N, we use O_v to denote a *cone* rooted on node v in N. O_v is a subnetwork of N consisting of v and some of its predecessors, such that for any node $w \in O_v$, there is a path from w to v that lies entirely in O_v. The maximum cone of v, consisting of all the PI predecessors of v, is called a *fanin cone* of v, denoted as F_v. We use $input(O_v)$ to denote the set of distinct nodes outside O_v which supply inputs to the gates in O_v. A *cut* is a partitioning (X, X') of a cone O_v such that X' is a cone of v. The *cut-set* of the cut, denoted $V(X, X')$, consists of the inputs of cone X', or $input(X')$. A cut is *K-feasible* if X' is a K-feasible cone. In other words, the cardinality of the cut-set is $\leq K$. We use $f(K, v)$ to denote all the K-feasible cuts rooted at node v. The level of a node v is the length of the longest path from any PI node to v. The level of a PI node is zero. The depth of a network is the largest node level in the network. A Boolean network is l-bounded if $|input(v)| \leq l$ for each node v. In this work, all initial networks are 2-bounded. If a network is not 2-bounded, we can transfer it into a 2-bounded network using gate decomposition.

We use a widely accepted unit delay model [2, 6, 8], where each LUT on a path contributes one unit delay. The largest optimal delay of the mapped circuit is called the *optimal mapping depth* of the circuit. The mapping problem for depth-optimal power optimization of FPGA is to cover a given l-bounded Boolean network with K-feasible cones, or equivalently, K-LUTs in such a way that the total power consumption after mapping is minimized while the optimal mapping depth is guaranteed under the unit delay model.

3. RELATED WORKS

3.1 Signal Probability

Signal probabilities can be calculated exactly using the famous Parker-McCluskey algorithm [18] or binary decision diagrams (BDDs) based algorithm [17]. However, it was proved in [19] that computing signal probabilities is an NP-Complete problem, which means none of the above exact algorithms is guaranteed to run in polynomial time. One class of heuristic techniques computes signal probability bounds (upper and lower), and the exact probability is guaranteed to lie within the computed bounds. One such technique is the cutting algorithm by [20], which cuts fanout lines in the circuit to make the circuit a forest, and assigns a bound $[0, 1]$ to the cut lines, then propagates the probability bounds to the primary outputs. Another technique for computing signal probability bound is to use ordered partial decision diagrams (OPDDs) as described in [21]. In our work, we are interested in the probability value instead of the probability bound. To estimate signal probabilities, one can use the weighted averaging algorithm [22] or the possibilistic algorithm [23].

3.2 Switching Activity and Dynamic Power

The dynamic power can be computed by

$$P = \frac{1}{2} \sum_{n \in nets} s_n \cdot C_n \cdot f \cdot V^2 \tag{1}$$

where C_n is the load capacitance of a net n, V is the supply voltage, s_n is the average switching activity (or transition density) of net n, and f is the frequency of the circuit. Therefore, we can reduce power consumption by reducing s_n and C_n. The term C_n can be roughly estimated by the number of pins on the net n. The term s_n can not be easily estimated, especially when we consider glitches.

In [24], the authors presented a pre-layout activity prediction method for FPGAs. They defined a formula to predict the switching activity on a node y: $predict(y) = \alpha \cdot gen(y) + \beta \cdot prop(y) + \phi$, where $prop(y)$ represents the activities propagated from one of y's inputs, and $gen(y)$ represents activities generated by y itself. One drawback of this approach is that the parameters α, β, and ϕ are heavily dependent on the characteristics of a circuit. The parameters working for one circuit may not work well for another circuit. Symbolic simulation [25] can also be used to estimate switching activities. However, one concern of this method is its long runtime.

Work [17] introduced another method to compute switching activities. For a node y with fanin nodes $x_1, x_2, \ldots, x_n$, given the switching activity $s(x_i)$ of each fanin node x_i, the switching activity of node y, $s(y)$ can be computed by the Boolean difference of y with respect to x_i [17]. However, this method does not take simultaneous switching into account. The authors of [26] extend this method to handle simultaneous switching. Let y be a Boolean expression, $y(t)$ be its value at time t, $P(y)$ be the signal probability of y, and $s(y)$ be the switching activity of y. Obviously, $s(y) = P(y(t)\overline{y(t+T)}) + P(\overline{y(t)}y(t+T))$, that is, $s(y)$ is the probability of y having different values at time t and $t+T$, where T is a unit time period. Since every falling transition is followed by a rising transition, and vice versa, $P(y(t)\overline{y(t+T)}) = P(\overline{y(t)}y(t+T))$. We have $P(y(t)\overline{y(t+T)}) = P(\overline{y(t)}y(t+T)) = 1/2 \cdot s(y)$. Since $P(y(t)) = P(y(t)y(t+T)) + P(y(t)\overline{y(t+T)})$, we have [26]

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

The term $P(y(t)y(t+T))$ can be calculated from the probabilities and switching activities of fanin nodes of y using the procedure in [26]. We use this method as a base to build our switching activity estimation model for efficiency and accuracy.

3.3 Cut Enumeration Based Mapping

Cut enumeration is an effective method for finding all the possible ways of the K-feasible cones rooted on a node. The cut-enumeration process will combine each subcut (or the fanin node itself) on one of the fanin nodes with each counterpart from the other fanin node to form new cuts for the node [8, 27]. If the input of the new cut exceeds K, the cut is discarded. Experiments turn out that cut enumeration is very efficient for small K's. The average numbers of cuts on each node are 6, 13, and 35 for $K = 4$, 5, and 6, respectively [8]. The depth of a node u, $D(u)$, is the smallest level of the node of all possible mapping solutions. For a PI node, $D(u) = 0$; otherwise, $D(u)$ is computed by the following formula from PIs to POs in the topological order

$$D(u) = \underset{\forall C \text{ on } u}{MIN} \quad \underset{v \in input(C)}{\{MAX\{D(v)\} + 1\}} \tag{3}$$

where C represents every cut generated for u through cut enumeration. The depth of a cut C, $D(C)$, is defined as the maximum depth of the nodes in its cut-set plus 1, that is, $D(C) = MAX\{D(v) + 1 \mid v \in input(C)\}$.

4. ALGORITHM DESCRIPTION

We will provide an new algorithm for computing signal probabilities in Section 4.1, a switching activity estimation model considering glitches in Section 4.2, and a delay-relaxation-based cost propagation scheme for technology mapping in Section 4.3.

4.1 Computing Signal Probability

Previous algorithms either collapse the whole network to compute exact signal probabilities, or compute a node signal probability only using the probabilities of its fanin nodes. The former algorithms are not practical due to their long runtime, while the later algorithms are not effective because their calculations are based on limited local information. The main difficulty of these algorithms is how to deal with reconvergence paths in the circuits, and still get good estimation [22, 23].

In our algorithm, we use a cone of each node to compute its signal probability. The advantage of this approach is that the reconvergence paths inside of the cone have no

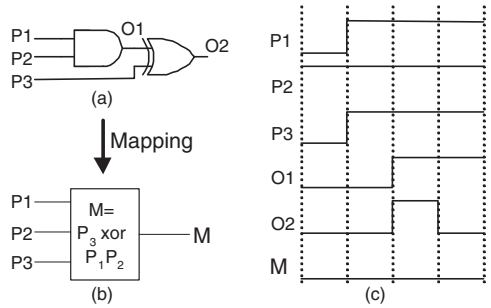

Figure 1: (a) A simple circuit consisting of an AND gate and an XOR gate. (b) The LUT to implement (a). (c) The timing diagram of signals in (a) and (b) assuming unit delay for each gate. Obviously, the switching activity of M is different from $O2$ since one glitch of $O2$ is eliminated by LUT M. This example is only for illustration, and the timing may not be exact.

effect on the signal probability. Any cone of the node can be used, but we prefer those cones covering a large number of reconvergencs. If the cone covers all the reconvergences affecting the node, we get the exact signal probability for this node (Theorem. 1). For a node v, we choose one of its cuts, $C = (X, X')$, in such a way that the cone X' covers the largest number of nodes. Then, we collapse the cone X' into a single node v'. The node v' is called the *collapsed node* of the cut C, while its function is also called the *function of the cut* C, denoted as f_C. We compute the signal probability of node v according to function f_C using the weighted averaging algorithm [22]. To use the weighted averaging algorithm, every gate should be an AND, OR or INV gate. We process the function f_C so that it can be expressed by the following sum of product (SOP) form:

$$f_C = g_1 + g_2 + \cdots + g_n, \text{ where } g_i \cdot g_j = 0 \text{ for all } i \neq j \quad (4)$$

In the above formula, every function g_i can be represented by an AND gate (unless it is a literal), so $P(g_i)$ can be computed using the weighted averaging algorithm, and we have $P(f_C) = \sum_{i=1}^{n} P(g_i)$.

THEOREM 1. *If a cone O_v of a node v covers all the reconvergence paths in the finin cone, F_v, of v, our signal probability estimating algorithm produces the exact signal probability for node v using cone O_v.*

We will use an example to illustrate how we calculate cut functions during cut enumeration. Even though we only need to compute the function of one cut for the signal probability of a node, we still compute the functions of all cuts, because these functions are used later for estimating switching activities. Let v_1 and v_2 denote the fanin nodes of a two-input node u, and C_1 and C_2 are two feasible cuts of v_1 and v_2, respectively. Let f_u denote the function of the node u. When we form a new cut C for u using C_1 and C_2, we first set $f_C = f_u$, then replace the inputs of f_C from v_1 and v_2 to f_{C_1} and f_{C_2} (they are computed previously already), and collapse the node of f_C with the nodes of f_{C_1} and f_{C_2} to get the function of the cut C. Since the fanin number of all functions are limited by K, the above collapsing operation is very fast, and will not increase the runtime of the cut enumeration significantly. Note that any cut of a node can be used to compute its signal probability, we use $K-$feasible cuts in our algorithm to reach a balance of runtime and quality. Even though we present this method in the context of FPGA technology mapping, it can be used to compute signal probabilities for ASIC designs. The experiments show that our algorithm produces much better results compared to previous algorithms.

4.2 Estimating Switching Activity with Glitches

In [2], Monte-Carlo simulation is first used to produce a switching activity file for a circuit, and this activity file is fed into the technology mapping algorithm to produce a low power mapping solution for the circuit. The drawback of this approach is that the switching activity of a node before mapping is generally different from that after mapping (see Fig. 1). Moreover, different cuts of the same node produce different switching activities. Before we carry out the technology mapping procedure, we do not know which cut will be used for a node, which means we can not use Monte-Carlo simulation to get the switching activity of a node after it is mapped. To tackle this problem, we use formula (2) to compute the switching activity of a cut, which is the switching activity of its collapsed node. The switching activity of a node is equal to the switching activity of its best cut (the best cut of a node is the cut picked for implementing the node).

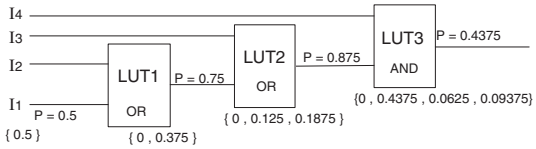

Figure 2: LUT1 and LUT2 implement an OR function, and LUT3 implements an AND function. The signal probabilities of $I1$, $I2$, $I3$, and $I4$ are all 0.5, and their switching activity arrays are all $\{0.5\}$. $P(LUT1) = 0.75$ and $S_A(LUT1) = \{0, 0.375\}$; $P(LUT2) = 0.875$ and $S_A(LUT2) = \{0, 0.125, 0.1875\}$; $P(LUT3) = 0.4375$ and $S_A(LUT3) = \{0, 0.4375, 0.0625, 0.09375\}$. The effective switching activities for LUT1, LUT2, and LUT3 are 0.375, 0.3125, and 0.59375, respectively.

Under the unit delay model, signal transition happens only at the discrete time. For a cut C with depth $D(C)$, the value of the output of C (i.e., the functional value of cone X' due to C) may change at time $1, 2, \ldots, D(C)$. The transition at time $D(C)$ is the functional transition, while the transitions at time $1, 2, \ldots, D(C) - 1$ are possibly glitches. To model glitches, we use an array of $D(C) + 1$ elements to represent the switching activities of a cut (or a node) at time $0, 1, \ldots, D(C)$ (only PIs transit at time 0). For a given cut C (or node u), let $S_A(C)$ (or $S_A(u)$) denote the *switching activity array* of C (or u). The *effective switching activity*, $s_e(C)$ (or $s_e(u)$), of a cut C (a node u) is the summation of all the elements in its switching activity array, that is, $s_e(C) = \sum_{i=0}^{D(C)} S_A(C)[i]$ ($s_e(u) = \sum_{i=0}^{D(u)} S_A(u)[i]$). For a primary input, its switching activity array is $\{\alpha\}$ (an array with only one element), where α is the switching activity of a PI. For internal nodes, the switching activity arrays are calculated in a topological order from PIs to POs. Given an internal node u, we first calculate switching activity arrays of all its cuts. The effective switching activity of a cut C, $s_e(C)$, is used to compute the cost of the cut C. The cut with the smallest cost will be picked to implement the node u, and the switching activity array of u is equal to the switching activity array of its picked cut. The switching activity array of a cut C, $S_A(C)$, is calculated using its function, f_C, by formula (2). The ith ($i >= 1$) element of C's switching activity array, $S_A(C)[i]$, is calculated from the $(i-1)$th elements of switching activity arrays of the nodes in $input(C)$ (the cut-set of C).

We will use the circuit in Fig. 2 to illustrate how to propagate switching activity arrays. In Fig. 2, each PI node has signal probability of 0.5, and the switching activity of a PI is also 0.5 (these are just example values). In the example, LUT1 and LUT2 implement a Boolean OR operation, and LUT3 implements a Boolean AND operation. For an OR operation $y = x_1 + x_2$, we can derive $s(y) = [1-P(x_1)]s(x_2)+[1-P(x_2)]s(x_1)-1/2 \cdot s(x_1)s(x_2)$ according to formula (2), [1] and for an AND operation $y = x_1 x_2$, $s(y) = P(x_1)s(x_2)+P(x_2)s(x_1)-1/2 \cdot s(x_1)s(x_2)$. The switching ac-

[1]In formula (2), $P(y(t)y(t+T))$ can be expressed in terms of probabilities and switching activities of y's fanin nodes.

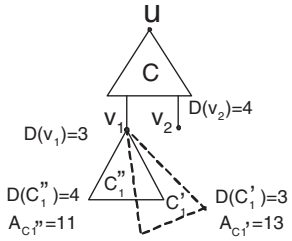

Figure 3: In this example, the cost of v_1 is $A_{v_1} = A_{C_1'} = 13$. If we relax v_1's depth to 4, we can use $A_{C_1''} = 11$ as the cost of A_{v_1}, which produces more accurate cost estimation for cut C.

tivity arrays of PIs are all $\{0.5\}$. For LUT2, $S_A(LUT2)[0] = 0$, since the output of LUT2 does not change at time 0. At time 1, the switching activity of LUT2 is calculated from the switching activities of $I3$ and LUT1 at time 0, which are 0.5 and 0, respectively. Since $P(I3) = 0.5$ and $P(LUT1) = 0.75$, $S_A(LUT2)[1] = 0.5 \cdot 0 + 0.25 \cdot 0.5 - 1/2 \cdot 0.5 \cdot 0 = 0.125$. Similarly, $S_A(LUT2)[2] = 0.5 \cdot 0.375 + 0.25 \cdot 0 - 1/2 \cdot 0 \cdot 0.375 = 0.1875$. We get the switching activity arrays of LUT1 and LUT3 in a similar way. Note that the signal probability value stays the same for each cell for different discrete time slots under the unit delay model (the proof of this statement is omitted due to space limit).

4.3 Delay Relaxation Based Cost Propagation

In previous cut-enumeration-based algorithms [8, 27], there is only one cost associated with each node, which is the smallest cost of all cuts having the same depth as the node. Those cuts with larger depths are not used for cost propagation during cut enumeration process (while these cuts are still considered during the cut selection stage after the cut enumeration). This approach does not provide very accurate cost estimation. For example (see Fig. 3), assume node u has a cut C with the cut-set $\{v_1, v_2\}$, the depths of v_1 and v_2 are 3 and 4, respectively. Node v_1 has two cuts C_1' and C_1'' with depths 3 and 4, respectively. Previously, only cut C_1' is used to calculate the cost of v_1. However, for cut C, the cut C_1'' can also be used to calculate the cost of v_1, since the depth of v_1 can be relaxed to 4 for cut C, and considering C_1'' may produce better cost for v_1.

Instead of one single cost per node (or cut), there is a cost array for each node (cut) in our algorithm. For a node u (a cut C), the cost $A_u[d]$ (or $A_C[d]$) is the smallest cost when the node u (cut C) is allowed to have a depth no larger than d, $\forall d.D(u) \le d \le D$ ($\forall d.D(C) \le d \le D$), where D is the optimum mapping depth of the network. If $d < D(u)$ (or $D(C)$), $A_u[d]$ (or $A_C[d]$) is not defined. If $d \ge D(u)$ (or $D(C)$), $A_u[d]$ (or $A_C[d]$) can be calculated using the following formulae:

$$A_u[d] = MIN\{A_C[d] \mid C \text{ is a cut of } u, \text{ and } d(C) \le d\}, \quad \text{for } D(u) \le d \le D \tag{5}$$

$$A_C[d] = \sum_{v \in input(C)} [A_v[d-1]/f_o(v)] + U_C[d], \quad \text{for } D(C) \le d \le D \tag{6}$$

where $f_o(v)$ is the fanout number of node v, and $U_C[d]$ ($\forall d.D(C) \le d \le D$) is the unit cost of cut C when C is allowed to have a depth no larger than d. The unit cost $U_C[d]$ is the cost of the cut itself. We use $U_C[d] = s_e(C[d]) \cdot (1 + f_o(C))$ to estimate the dynamic power consumed by the LUT corresponding to cut C, where $s_e(C[d])$ is the effective switching activity of C at depth d. The cost $A_u[d]$ is the total estimated cost of the mapped circuit of F_u. Therefore, $A_u[d]$ is the total estimated power consumed by the mapped circuit of F_u when the mapping depth of F_u is less than or equal to d. If we use cost array in Fig. 3, we have $A_{v_1}[3] = A_{C_1'}[3] = 13$, and $A_{v_1}[4] = A_{C_1''}[4] = 11$ (assume $A_{C_1'}[4] > 11$), and we use the value of $A_{v_1}[4]$ to compute the cost $A_C[5]$ of cut C.

Algorithm 1: GlitchMap algorithm

```
Input: a circuit to be mapped
       K : LUT size
Output: mapped circuit
/* cut enumeration */
for each node u in topological order do
    if u is a primary input then
        f(K,u) = {u};
    else
        v1, v2 ⇐ fanins of u;
        f(K,u) = {K-feasible combination of C1, C2
        where C1 ∈ f(K,v1), C2 ∈ f(K,v2)};
    end
end
D = optimum mapping depth;
/* signal probability estimation */
for each node u in topological order do
    if u is a primary input then
        P(u) = {α};
    else
        P(u)= probability computed according to
        Section 4.1;
    end
end
/* cost propagation */
for each node u in topological order do
    if u is a primary input then
        for i = 0 to D do
            S_A(u[i]) = {β};
            A_u[i] = 0;
        end
    else
        ComputeNodeCost(u, D, K);
    end
end
/* cut selection */
Push all PO nodes into a queue qu;
while qu is not empty do
    Pop u from qu;
    Pick the best cut C for u based on cut costs;
    for each v ∈ input(C) do
        if v has not been pushed into qu then
            Push v into qu;
    end
end
```

THEOREM 2. *The cost of a larger depth is better than or equal to the cost of a smaller depth, $A_u[d+1] \le A_u[d]$ (or $A_C[d+1] \le A_C[d]$). The size of the cost array is bounded by the optimum depth of the network without losing optimality.*

Note that, in this paper, each node is associated with an array of costs, and each cost is associated with an effective switching activity which is a summation over a switching activity array. There are $D - D(u) + 1$ costs for a node u, and there are $D - D(u) + 1$ switching activity arrays for u, one for each cost.

4.4 Overall Algorithm

Our algorithm can be summarized by algorithm 1. In the algorithm, α and β are default signal probability and switching activity for PIs. At the beginning, we enumerate all feasible cuts for every node in the topological order. During cut enumeration, we compute the depth of every node

Algorithm 2: ComputeNodeCost

Input: u is a node, D is the optimum mapping depth of the network, and K is the LUT size
Output: cost array and switching activity arrays of u

The elements of A_u are initialized to large numbers;
for *every cut* $C \in f(K,u)$ **do**
  for $d = D(C)$ **to** D **do**
    Compute $S_A(C[d])$ with $S_A(v[d-1])$ for $v \in input(C)$ using formula (2);
    $s_e(C[d]) = \sum_{i=0}^{d} S_A(C[d])[i]$;
    $U_C[d] = s_e(C[d]) \cdot (1 + f_o(u))$;
    $A_C[d] = \sum_{v \in input(C)} [A_v[d-1]/f_o(v)] + U_C[d]$;
    if $A_C[d] < A_u[d]$ **then**
      $A_u[d] = A_C[d]$;
      $S_A(u[d]) = S_A(C[d])$;
  end
end

and every cut, and cut functions. At the end of the cut enumeration, the optimum mapping depth of the circuit is available. In the second stage, we compute signal probabilities for all nodes using the approach of Section 4.1. In the third stage, we propagate switching activity arrays and costs from PIs to POs. Finally, we carry out the cut selection procedure in a similar way used in most cut-enumeration-based algorithm [8, 27] for the final mapping.

Algorithm 2 shows how to process a node in the third stage of algorithm 1. In the algorithm, $A_u[d]$, $A_C[d]$ and $U_C[d]$ are defined in Section 4.3. The term $S_A(u[d])$ (or $S_A(C[d])$) is the switching activity array of the node u (cut C) when u (or C) is allowed to have a depth no larger than d. $S_A(u[d])[i]$ (or $S_A(C[d])[i]$) is the ith element of the switching activity array $S_A(u[d])$ (or $S_A(C[d])$). For a cut C, when we compute its cost at depth d, we first compute its switching activity array $S_A(C[d])$ according to the switching activity arrays of its input nodes at depth $d-1$. Then we are able to compute $A_C[d]$ as shown in algorithm 2. The unit cost of the cut C at depth d, $U_C[d] = s_e(C[d]) \cdot (1 + f_o(u))$, where $s_e(C[d])$ is the effective switching activity of C at depth d and $f_o(u)$ is the fanout number of node u. From formula (1), we know the dynamic power of a node is proportional to the production of its switching activity and fanout load capacitance. For FPGAs, the capacitance of a net is positively correlated with the number of pins on the net, so we use $1+f_o(u)$ to estimate the LUT output load capacitance.

4.5 Complexity Analysis

The runtime of our algorithm is dominated by the cost propagation stage. Let n denote the total number of nodes, K denote the input size of a LUT, D denote the optimum mapping depth, and n_c denote the average number of cuts on each node. We have the following theorem with regard to complexity. The term $K2^K$ in the theorem is related to processing a node function with at most K inputs, and this limit is seldom reached. In practice, n_c, D, and K are all small numbers, so the algorithm is efficient.

THEOREM 3. *The runtime of the algorithm is* $O(nn_cKD^22^K)$, *and the memory needed is* $O(nn_cD + nn_cK2^K + nD^2)$.

5. EXPERIMENTAL RESULTS

Our experiments are carried out on a desktop PC with a 2.4 GHz Intel(R) Xeon(TM) CPU. The OS is Red Hat Linux 8.0, and we use gcc to compile our program. We use the 20 largest MCNC benchmark circuits in our experiment. We compare our algorithm, GlitchMap, with EMap [2]. We use $K = 5$ for all circuits. The signal probabilities and switching activities of primary inputs are both 0.5 in our experiment. The mapped circuits are in Berkeley logic interchange format (BLIF). Figure 4 shows our experimental flow. For EMap, we follow its own design flow, which uses the activity estimator ACE [28] to generate switching activity information for all circuits, then runs EMap with the switching activity file. In order to use a gate-level power estimator, PowerPlay Analyzer, available from Altera's Quartus II software, we write a converter from BLIF format to Verilog Quartus Mapping (VQM) format. We build a project for each VQM file, and we set TRUE_WYSIWYG_FLOW to ON so that Quartus II will not optimize and remap our circuits. We use Stratix II device family, and let Quartus II software to decide which device to use, which is the default setting. Then we run Quartus II commands quartus_map, [2] quartus_fit, quartus_asm, and quartus_tan to synthesize, place, and route the circuits. Next, we generate random input vectors, in Vector Waveform Format (VWF), for each circuit. The input vectors are generated in such a way that the signal probability and the switching activity of each primary input are both 0.5. These files are used for both EMap and our mapper, GlitchMap. With the VWF files, we run quartus_sim command to simulate the circuits, and generate a switching activity file (SAF) for each circuit. With the SAF files, we run PowerPlay Analyzer for each circuit. We use 1000 input vectors for each circuit, and PowerPlay Analyzer reports high confidence power estimation for all circuits. Table 1 shows the dynamic power consumed by each circuit reported by PowerPlay Analyzer. From the table, GlitchMap is able to reduce dynamic power consumption by 18.7% compared to EMap. We have not shown the total power consumed by each circuit, because the LUT numbers of the mapped circuits are much smaller than the total number of LUTs available in the Stratix II devices, which means most power is consumed by unused LUTs statically, and the total power is not able to compare our algorithm with EMap fairly. It was reported in [1] that dynamic power takes about 67% of total power consumption for real designs, so we can project that our algorithm is able to reduce the total power consumption by 13% if we assume the static power values are the same for both GlitchMap and EMap. Note that EMap only works to reduce dynamic power as well.

By setting the unit cost of a cut C, U_C, to be 1, our mapper targets minimizing the total number of LUTs. Ex-

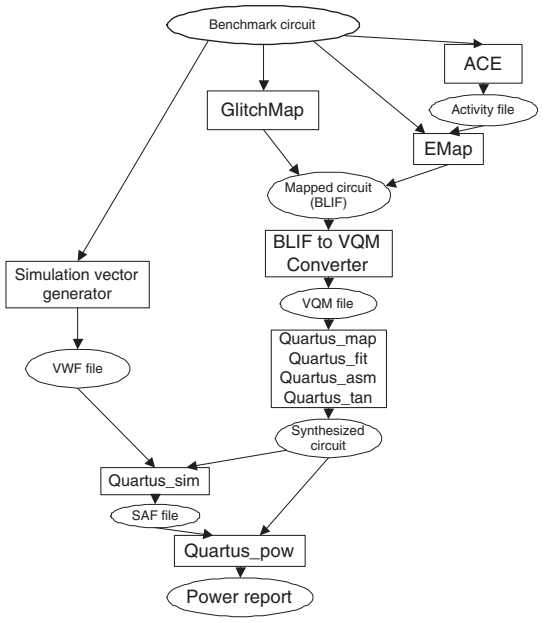

Figure 4: Experimental flow

[2]The command quartus_map is used to prepare the input files for later process. It does not optimize and re-map our circuits since we set TRUE_WYSIWYG_FLOW to ON.

Table 1: Comparison of our algorithm with EMap on 20 largest MCNC benchmark circuits. In this table, the column P_E is the dynamic power of EMap; the column P_G is the dynamic power of our algorithm; the column Time is the runtime of our algorithm; the column Impr is the improvement of our algorithm over EMap, and Impr = (P_E-P_G)/P_E.

Ckt.	$P_E(mW)$	$P_G(mW)$	Time (s)	Impr (%)
alu4	30.33	21.22	2	30
apex2	25.95	17.07	2.6	34.2
apex4	18.9	14.37	2.6	24
bigkey	55.99	53.39	2.5	4.6
clma	62.57	45.35	24	27.5
des	76.1	74.84	2.8	1.66
diffeq	14.29	12.46	2.9	12.8
dsip	44.46	46.18	1.8	-3.87
elliptic	40.65	34.61	8	14.9
ex1010	47.6	27.85	13.5	41.5
ex5p	17.37	18.33	3	-5.53
frisc	29.3	24.07	13.1	17.8
misex3	25.35	18.19	2.4	28.2
pdc	45.07	28.76	14.9	36.2
s298	20.3	16.3	3.7	19.7
s38417	95.62	87.89	21.8	8.08
s38584.1	86.8	83.38	17.6	3.94
seq	25.5	16.52	2.7	35.2
spla	38.29	26.17	14.1	31.7
tseng	15.77	13.96	2.2	11.5
average			7.9	18.7

perimental results show that our mapper for area minimization produces results with 3.5% less area on average compared to DAOmap [8], which shows the effectiveness of our delay-relaxation-based cost propagation technique (the detailed data are omitted due to space limit). We also compare our signal probability estimation algorithm with the weighted averaging algorithm [22] and the possibilistic algorithm [23]. For each algorithm, we compute the root mean square (RMS) of the deviation of the estimated probability from the exact signal probability. The smaller the RMS of deviations, the better the results. Among 20 benchmark circuits, only 14 circuits can be used to run an exact probability algorithm (the exact probability algorithm runs too long to get results for the other 6 circuits.) Therefore, we only show 14 circuits in Table 2. From the table, we know that our signal probability estimating algorithm produce results more than twice better than previous algorithms.

6. CONCLUSIONS

In this paper, we presented an FPGA technology mapping algorithm for power optimization. We developed a new cut-enumeration-based signal probability estimation algorithm, which performs more than 2X better over previous algorithms. Combining this new signal probability model and an effective glitch estimation model, we introduced a new FPGA technology mapping algorithm for low power. We also enhanced cost estimation along cut enumeration. Compared to a previous power-aware mapper, EMap, our algorithm is able to reduce 18.7% more dynamic power.

Table 2: Comparison of our algorithm with the weighted averaging algorithm and the possibilistic algorithm. In this table, RMS_g, RMS_w, and RMS_p are RMS of deviations of our algorithm, weighted averaging algorithm, and possibilistic algorithm, respectively. R_{gw} =RMS_w/RMS_g, and R_{gp} =RMS_p/RMS_g.

Ckt.	RMS of Deviations				
	RMS_g	RMS_w	R_{gw}	RMS_p	R_{gp}
alu4	0.098	0.123	1.26	0.128	1.31
apex4	0.0084	0.045	5.36	0.043	5.12
bigkey	0.13	0.14	1.08	0.134	1.03
clma	0.295	0.331	1.12	0.327	1.11
des	0.175	0.185	1.06	0.186	1.06
dsip	0.093	0.086	0.92	0.086	0.92
ex1010	0.015	0.02	1.33	0.02	1.33
ex5p	0.064	0.154	2.41	0.154	2.41
misex3	0.119	0.136	1.14	0.137	1.15
pdc	0.023	0.036	1.57	0.034	1.48
s298	0.034	0.047	1.38	0.048	1.41
s38584.1	0.734	0.745	1.01	0.741	1.01
seq	0.054	0.104	1.94	0.103	1.91
spla	0.0026	0.029	11.15	0.029	11.15
Average			2.34		2.31

7. REFERENCES

[1] Altera and white paper. Stratix II vs. Virtex-4 Power Comparison & Estimation Accuracy White Paper. *http://altera.com/literature/wp/wp_s2v4_pwr_acc.pdf.*

[2] J. Lamoureux and S. J. E. Wilton. On the Interaction between Power-Aware CAD Algorithms for FPGAs. In *ICCAD*, 2003.

[3] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. In *FPGA*, pages 21 – 30, 2006.

[4] R. J. Francis et al. Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs. In *DAC*, 1991.

[5] R. Murgai et al. Improved Logic Synthesis Algorithms for Table Look Up Architectures. In *ICCAD*, Nov. 1991.

[6] J. Cong and Y. Ding. An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. In *ICCAD*, Nov. 1992.

[7] H. Yang and M. D. F. Wong. Edge-map: Optimal Performance Driven Technology Mapping for Iterative LUT based FPGA Designs. In *ICCAD*, Nov. 1994.

[8] D. Chen and J. Cong. DAOmap: A Depth-optimal Area Optimization Mapping Algorithm for FPGA Designs. In *ICCAD*, Nov. 2004.

[9] A. Mishchenko, S. Chatterjee, and R. Brayton. Improvements to Technology Mapping for LUT-Based FPGAs. In *FPGA*, pages 41–49, 2006.

[10] A. H. Farrahi and M. Sarrafzadeh. FPGA Technology Mapping for Power Minimization. In *FPL*, pages 66–77, 1994.

[11] J. Anderson and F. N. Najm. Power-Aware Technology Mapping for LUT-Based FPGAs. In *IEEE Intl. Conf. on Field-Programmable Technology*, 2002.

[12] Z. H. Wang et al. Power Minimization in LUT-Based FPGA Technology Mapping. In *ASPDAC*, 2001.

[13] H. Li, W. Mak, and S. Katkoori. Efficient LUT-Based FPGA Technology Mapping for Power Minimization. In *ASPDAC*, pages 353 – 358, 2003.

[14] D. Chen et al. Low-Power Technology Mapping for FPGA Architectures with Dual Supply Voltages. In *FPGA*, 2004.

[15] F. Li et al. Architecture Evaluation for Power-Efficient FPGAs. In *FPGA*, pages 175–184, 2003.

[16] A. Raghunathan, S. Dey, and N. K. Jha. Register Transfer Level Power Optimization with Emphasis on Glitch Analysis and Reduction. *TCAD*, 18(8):1114–1131, 1999.

[17] F. N. Najm. Transition Density, A New Measure of Activity in Digital Circuits. *TCAD*, 12(2):310–323, 1993.

[18] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE trans. on Computers*, c-24(6):668–670, 1975.

[19] R. Garey and D. S. Johnson. *Computers and Interactability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[20] J. Savir. Improved cutting algorithm. *IBM J. RES. Develop*, 34(2/3):381–388, 1990.

[21] A. Dutta and N. A. Touba. Iterative OPDD Based Signal Probability Calculation. In *VTS*, pages 72–77, 2006.

[22] B. Krishnamurthy and I. G. Tollis. Improved Techniques for Estimating Signal Probabilities. *IEEE Trans. on Computers*, 38(7):1041–1045, 1989.

[23] M. A. Al-Kharji and S. A. Al-Arian. A New Heuristic Algorithm for Estimating Signal and Detection Probabilities. In *GLSVLSI*, pages 26–31, 1997.

[24] J. H. Anderson and F. N. Najm. Switching Activity Analysis and Pre-Layout Activity Prediction for FPGAs. In *SLIP*, pages 15–21, 2003.

[25] J. Monteiro et al. Estimation of Average Switching Activity in Combinational Logic Circuits Using Symbolic Simulation. *TCAD*, 16(1):121–127, 1997.

[26] T. L. Chou and K. Roy. Estimation of activity for static and domino CMOS circuitsconsidering signal correlations and simultaneous switching. *TCAD*, 15(10):1257–1265, 1996.

[27] J. Cong, C. Wu, and E. Ding. Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution. In *FPGA*, pages 29–35, 1999.

[28] ACE : a Switching Activity Estimation. Available[Online]: http://www.ece.ubc.ca/ julienl/activity.htm.