LUT-Based FPGA Technology Mapping for Power Minimization with Optimal Depth

Hao Li, Wai-Kei Mak, and Srinivas Katkoori
Department of Computer Science and Engineering
University of South Florida, Tampa FL 33620
{hli5,wkmak,katkoori}@csee.usf.edu

Abstract

In this paper, we study the technology mapping problem for LUT-based FPGAs targeting at power minimization. We present the PowerMap algorithm to generate a mapping solution to minimize power consumption while keeping the delay optimal. We compute min-height K-feasible cuts for critical nodes to optimize the depth and compute min-weight K-feasible cuts for non-critical nodes to minimize the power consumption of the mapping solution. We have implemented PowerMap in C and tested it on a number of MCNC benchmark circuits. Compared to FlowMap, a delay-optimal mapper, our algorithm reduces the power consumption by 17.8% and uses 9.4% less LUTs without any depth penalty.

1. Introduction

The field programmable gate array (FPGA) is now widely used in VLSI system design and rapid system prototyping. FPGA consists of an array of uncommitted programmable logic blocks, programmable interconnections, and I/O pads. The lookup-table (LUT) based architecture is the most popular architecture in FPGAs. A K-LUT block is a logic block which can be programmed to implement any Boolean function of up to K variables [3].

Previous LUT-based FPGA technology mapping algorithms can be classified into three categories according to their primary optimization objectives: algorithms that minimize area [2, 7, 8, 9], algorithms that minimize delay [4, 5, 10, 11] and algorithms that focus on routability [12, 14]. With the increasing popularity of wireless communications and portable computers, reducing power consumption has become a key concern. However, not much work has been done on minimizing power consumption in LUT-based technology mapping. Farrahi and Sarrafiadzehe [6] proposed a low-power driven mapping algorithm at the expense of increase in depth and the number of LUTs. Wang and Kwan [15] presented another algorithm to reduce the power consumption by generating LUTs rooted at nodes with low transition activities. But the results were only reported on several small circuits and it did not guarantee to find optimal depth mapping solutions either.

We introduce an algorithm named PowerMap, which computes low-power optimal delay mappings for LUT-based FPGAs. Compared to FlowMap, a well-known delay-optimal mapper, PowerMap reduces the power consumption by 17.8% and uses 9.4% less LUTs.

The rest of the paper is organized as follows: Section 2 introduces the power estimation model and reviews the FlowMap algorithm for delay optimization. Section 3 introduces the concept of min-weight K-feasible cut and presents our algorithm for power minimization. Section 4 presents the experimental results and Section 5 gives the conclusion.

2. Problem Formulation

We give the preliminaries including the power model used in our algorithm for power minimization. Furthermore, we briefly review the FlowMap algorithm which is a depth-optimal technology mapping algorithm. We compare our algorithm with FlowMap and it also generates delay optimal solutions.

2.1. Power Dissipation Model

The majority of power consumption in CMOS circuits is due to the dynamic power dissipation [16]. Dynamic power dissipation $P_d$ occurs because of the switching activity (either 0→1 or 1→0 transition) of the circuit, which results in the charging/discharging of the load.
capacitance.

\[ P_d = C_L V_{dd}^2 f \] (1)

where \( C_L \) is the load capacitance of the circuit, \( V_{dd} \) is the supply voltage, and \( f \) is the switching frequency.

For LUT-based FPGAs, this formula still holds except that the transitions only happen at the input/output of each LUT. For a given circuit \( N \) mapped into LUTs, we can estimate the power consumed by a LUT rooted at \( v \), \( \text{LUT}_v \), as follows:

\[ P(v) = K_p \cdot \left[ C_{out} + C_{in} \cdot \text{fanout}(v) \right] \cdot Q(v) \] (2)

where \( K_p = 0.5V_{dd}^2 f \) is a constant, \( C_{out} \) is the gate capacitance of its fanout, \( C_{in} \) denotes the internal capacitance of the LUT, and \( Q(v) \) is the output transition probability of \( \text{LUT}_v \). The term \( C_{out} + C_{in} \cdot \text{fanout}(v) \) accounts for the equivalent load capacitance at the output of \( \text{LUT}_v \) as shown in Figure 1.

![Figure 1: Load capacitance at the output of \( \text{LUT}_v \).](image)

The power consumed by a primary input (PI) node \( v \) can be computed as:

\[ P(v) = K_p \cdot C_{in} \cdot \text{fanout}(v) \cdot Q(v) \] (3)

We use the same approach for computing signal transition probabilities as proposed in [13]. The signal transition probability at the output of a node is dependent only on the Boolean function at the output in terms of the primary inputs. The term \( Q(v) \) shows the tendency of a fanout node \( v \) to change its status either from 0 to 1 or vice versa. This transition leads to the charging/discharging of the load capacitance in which a lot of the circuit is consumed. We use \( p^0 \) and \( p^1 \) to denote the probabilities for the output of node \( v \) to be at logic 0 and 1, respectively. We ignore the temporal correlation on the node. Thus, the transition probability of a signal changing from 0 to 1 or from 1 to 0 is equal to \( p^0 \cdot (1-p^0) \), or equivalently \( p^1 \cdot (1-p^1) \).

Then we have:

\[ Q(v) = p^0 \cdot (1-p^0) + p^1 \cdot (1-p^1) = 2 \cdot p^0 \cdot p^1 \] (4)

Intuitively, we should generate as many LUTs as possible that are rooted at nodes with low transition activity to minimize the total power consumption. But through our experiment implementing this idea, we observed that the greedy approach does not always reduce the power consumption. So we propose the idea of \textit{min-weight} \textit{K-feasible} cut in Section 3 which reduces the total power consumption efficiently.

### 2.2. Review of FlowMap Algorithm

Given a general Boolean network, we can represent it as a directed acyclic graph (DAG). Each node in the DAG represents a logic gate, and a directed edge \((u,v)\) exists only when the output of node \( u \) is in the fanin of node \( v \). For a node \( v \), the term \text{input}(v)\) denotes the set of fanin nodes of node \( v \). A cone at node \( v \) is a subgraph \( C_v \) consisting of \( v \) and its predecessors such that any path connecting a node in \( C_v \) and \( v \) lies entirely within \( C_v \). A cone \( C_v \) is \textit{K-feasible} if the number of nodes feeding \( C_v \) is less than or equal to \( K \). A Boolean network \( N \) is \textit{K-bounded} if \( |\text{input}(v)| \leq K \) for every node \( v \) in \( N \). The technology mapping problem for \( K \)-LUTs is to cover a given \( K \)-bounded Boolean network with \( K \)-input LUTs yielding an equivalent \( K \)-LUT network. The \textit{level} of a node \( v \) in a network is the length of the longest path between \( v \) and some PI node. The levels for all PI nodes are zero. The \textit{depth} of a given network is the largest node level in it. Given a \( K \)-bounded Boolean network, let \( N_v \) denote the subnetwork consisting of node \( v \) and all its predecessors. A cut \((X_v, \bar{X}_v)\) for node \( v \) is a partition of the nodes in \( N_v \) such that \( v \) is in \( X_v \). The \textit{node cut-size} is defined as the number of nodes in \( X \) that are adjacent to some nodes in \( \bar{X}_v \). If the cut size is no larger than \( K \), it is called a \textit{K-feasible} cut. Figure 2 shows a 3-feasible cut \((X_v, \bar{X}_v)\) for \( v \) in \( N_v \).

FlowMap [5] is a LUT-based FPGA technology mapping algorithm which generates depth-optimal \( K \)-LUT mapping solution for \( K \)-bounded Boolean networks. The label of node \( v \), denoted by \text{label}(v)\), is defined as the optimal depth for a \( K \)-LUT mapping of \( N_v \). For any PI node \( v \), \text{label}(v)=0. In the first phase of FlowMap, the nodes are processed in topological order from PI nodes to primary output (PO) nodes and the labels for all nodes are computed. Assume \( L \) is the maximum label of the predecessors of node \( v \). To compute \text{label}(v), FlowMap first merges all nodes \( w \) with \text{label}(w)=L into \( v \) and computes a min-cut in \( N_v \). If the cut size is no larger than \( K \), we set \text{label}(v)\) to \( L \) and store the cut. Otherwise, we set \text{label}(v)\) to \( L+1 \) and store the cut \((N_v - \{v\}, \{v\})\). Note that \((N_v - \{v\}, \{v\})\) is guaranteed to be a \( K \)-feasible cut because the original network
is $K$-bounded. In either case, the cut stored is called a min-height $K$-feasible cut for node $v$.

In the second phase, FlowMap generates a mapping solution. Let $(X_v, \overline{X}_v)$ be the cut stored for node $v$ in the first phase where $v \in \overline{X}_v$. $Q$ is a queue that contains all PO nodes initially. The following operation is repeated until $Q$ contains only PI nodes: Remove the head node $v$ from $Q$ and generate a LUT to cover all the nodes in $\overline{X}_v$; Insert into $Q$ the nodes that provide inputs to the LUT just generated, if they have not been inserted yet.

It has been proved in [5] that FlowMap always generates a depth-optimal mapping solution for LUT-based FPGAs.

3. PowerMap Algorithm

PowerMap is an efficient algorithm to compute a low power depth-optimal mapping solution. The first phase of PowerMap is similar to that of FlowMap. We compute the label and store the min-height $K$-feasible cut for each non-PI node. In addition, for each node $v$, we also compute the output transition probability $Q(v)$, and estimate the power consumption of a $K$-LUT mapping in subnetwork $N_v$.

The second phase of PowerMap differs from that of FlowMap: Before generating a $K$-LUT for a node $v$, we determine if the depth of this LUT can be relaxed from the value $label(v)$ without increasing the depth of the overall mapping solution. If so, we will compute a min-weight $K$-feasible cut for node $v$ aimed at reducing the power consumption of the final mapping solution. We then generate a $K$-LUT for node $v$ according to the min-weight $K$-feasible cut instead of the min-height $K$-feasible cut computed in the first phase.

We can determine if the depth of a LUT for a node $v$ can be relaxed by computing its slack value.

$$\text{slack}(v) = D - \text{label}(v) - D_v$$

where $D$ is the optimal depth of a $K$-LUT mapping solution of the whole network and $D_v$ is the maximum number of LUTs on a path from some child of $v$ to some PO node.

Note that $D$ is known after phase 1 since $D = \max \{ \text{label}(u) : u \in \text{set of PO nodes} \}$. When we generate a LUT for node $v$ in phase 2, a partial mapping solution covering all the successors of node $v$ should have been generated. So $D_v$ is also known by the time we generate a LUT for node $v$. If $\text{slack}(v) > 0$, then the depth of a LUT for node $v$ can be relaxed and $v$ is called a non-critical node. Otherwise, the depth of a LUT for node $v$ cannot be relaxed and $v$ is called a critical node. For example, consider computing the slack value for node $r$ and $u$ after $LUT_v$ is generated to cover node $v$ in Figure 3. The optimal depth $D$ for $N_v$ is 4, $\text{label}(r) = 2$, $\text{label}(u) = 3$, and $D_r = D_u = 1$. We can compute the slack for $r$ and $u$ according to Eq. (5): $\text{slack}(r) = 4 - 2 - 1 = 1$, and $\text{slack}(u) = 4 - 3 - 1 = 0$. So node $r$ is a non-critical node but node $u$ is a critical node.

![Figure 3: Labels computed for a Boolean network assuming $K = 3$.](image)

3.1. Min-Weight $K$-Feasible Cut

While generating a LUT to cover node $v$, intuitively we would like to minimize the sum of the power consumption of the inputs. One trivial solution is to enumerate all $K$-feasible cuts, which is clearly inefficient.
So, we introduce the notion of min-weight $K$-feasible cut for computing low-power mapping solutions, which is proved to be a good and an efficient approximation.

Let $LUT_v$ denote a $K$-LUT covering node $v$ in the mapping solution. We estimate the total power consumption of the $K$-LUT subnetwork covering $N_v$ as follows. For every node $u$ that provides input to $LUT_v$, the estimated power consumption $ep(u)$ of the $K$-LUT subnetwork covering $N_u$ is computed assuming the fanout of the LUT for $u$ is 1 and the covering is computed using min-height $K$-feasible cut. For a PI node $u$, $ep(u)$ is taken to be $K_p \cdot C_{in} \cdot Q(u)$. For a non-PI node $u$, $ep(u)$ can be computed recursively. For example, if $K=3$, for node $j$ and $l$ in Figure 4,

$$ep(j) = K_p \cdot (C_{out} + C_{in}) \cdot Q(j) + ep(a) + ep(b) + ep(c),$$

$$ep(l) = K_p \cdot (C_{out} + C_{in}) \cdot Q(l) + ep(j) + ep(k)$$

Figure 4: A subnetwork rooted at $l$. (Nodes $a$, $b$, $c$, $d$, and $e$ are PI nodes.)

A min-weight $K$-feasible cut $(X_v, \overline{X_v})$ for node $v$ is a $K$-feasible cut such that the maximum weight of the nodes in $X_v$ that provide inputs to the nodes in $\overline{X_v}$ is minimized. In phase 2, we compute a min-weight $K$-feasible cut and generate a K-LUT for a non-critical node. The weight of a node $u$, $weight(u)$, is set to $ep(u)$ initially. However, a LUT rooted at node $a$ may provide input to multiple LUTs. In this case, the estimated power consumption $ep(u)$ of the LUT-subnetwork covering $N_u$ should be counted only once. So after generating $LUT_v$ for node $v$, $weight(u)$ of any node $u$ that provides input to $LUT_v$ is reset to $K_p \cdot C_{in} \cdot Q(u)$ which is equal to the increase in power consumption of the LUT-subnetwork covering $N_u$ if the fanout of the LUT for $u$ is increased by one. The min-weight $K$-feasible cut can be found using a binary search strategy in $\log n$ steps, where $n$ is the number of nodes in the subnetwork rooted at $v$. In each step, we fix a weight threshold $T$. For nodes having weights larger than $T$, we assign infinite capacity to them. And other nodes are assigned unit capacity. Then we compute the max-flow according to this assignment. If the max-flow computed is no larger than $K$, we will try a smaller value for $T$ and repeat the assignment. Otherwise, a larger value for $T$ will be tried. Clearly, by using binary search, we can find the min-weight $K$-feasible cut in $\log n$ steps. The complete algorithm of PowerMap is shown in Figure 5.

![Algorithm PowerMap](image)

Figure 5. Pseudocode of PowerMap algorithm
3.2. Time Complexity of PowerMap

**Theorem 1:** A min-weight $K$-feasible cut in $N_v$ can be computed in $O(K m \log n)$ time, where $n$ and $m$ are the number of nodes and edges in $N_v$, respectively.

**Proof:** Since finding an augmenting path in a residual graph for $N_v$ takes $O(m)$ time [1], we can determine whether there exists a $K$-feasible cut or not for $N_v$ in $O(K m)$ time. Because a min-weight $K$-feasible cut $(X_v, \bar{X}_v)$ is a $K$-feasible cut such that the minimum weight of the nodes in $X_v$ that provide inputs to the nodes in $\bar{X}_v$ is minimized, we can use a binary search strategy on the weights of the nodes in $N_v$ in order to find a min-weight $K$-feasible cut. This can be done in no more than $\log n$ passes where each pass checks for the existence of a $K$-feasible cut for $N_v$ when cutting at nodes with weights greater than a threshold is prohibited. Therefore, it takes $O(K m \cdot \log n) = O(K m n \log n)$ time to compute a min-weight $K$-feasible cut for $N_v$.

**Theorem 2:** Given a general Boolean network $N$, PowerMap generates a depth-optimal low-power mapping solution in $O(K m n \log n)$ time, where $n$ and $m$ are the number of nodes and edges in $N$, respectively.

**Proof:** According to [5], the labeling phase for all nodes in a general Boolean network $N$ can be done in $O(K m n)$ time. Besides, it takes $O(1)$ time to compute $e_p(u)$ and $Q(u)$ for each node $u$. So the time needed in the first phase of PowerMap is $O(K m n)$. In the second phase, the number of nodes of each subnetwork is bounded by $n$ and it needs $O(1)$ time to generate the mapping for critical nodes. Therefore, phase 2 takes $n_1 \cdot O(1) + n_2 \cdot O(K m \log n) = O(K m n \log n)$ time, where $n_1$ is the number of LUTs covering critical nodes and $n_2$ is the number of LUTs covering non-critical nodes. The total time of both phases is $O(K m n) + O(K m n \log n)$, so the total computation complexity for PowerMap is $O(K m n \log n)$.

4. Experimental Results

We have implemented the PowerMap algorithm using C language and experimented with a number of MCNC benchmark circuits on a Sun Ultra workstation. Given a general Boolean network, we first decompose it into a 2-bounded network using 'dmig' within SIS. Then we apply our algorithm on the decomposed network to compute a low power depth-optimal mapping solution.

First, we compared our algorithm with FlowMap in terms of depth, power consumption, and number of LUTs. We assume that $V_{dd} = 5V$ and all PI nodes have 0.5 switching activities. For $C_{out}$ and $C_{in}$, we use the same value assumed in [6], which was set to be 10 pF apiece. We run our algorithm for mapping into 5-input LUTs and estimate the power consumption according to Eq. 2 and 3. The results are shown in Table 1. On an average, PowerMap reduces the power consumption due to the logic nodes by 17.3% and uses 9.4% less LUTs than FlowMap. The maximum improvement in power consumption is 42.5%. For some circuits the improvement is not that significant, for instance, '9symml' only decrease the power consumption by 3.7%. Through our observation, the reason is that either most nodes in the benchmark circuit have zero slack value or the circuit is too small so that there is not much space for our algorithm to relax the depth and hence reduce the total power consumption. Due to the unavailability of the algorithm proposed by Farrahi and Sarrafzadeh in [6], we could not compare our results with it directly. But from the results reported in Table 1 and Table 2 in that paper, which compared their power minimization against FlowMap, we can see that our algorithm is better in both power consumption and depth.

In addition, we noticed that previous works on reducing the dynamic power for either library-based or LUT-based technology mapping seldom take the variation of transition probability of the PI nodes into account. In practice, the transition probability of PI nodes is not uniform. So we also did some experiments using
randomly generated transition probabilities for the PI nodes and the results are also delay optimal. On an average, the improvement of power consumption and number of LUTs are similar to the result shown in Table 1, when all PI nodes have 0.5 transition probability.

5. Conclusions

In this paper, we studied the technology mapping problem for LUT-based FPGAs targeting at power and depth minimization. We presented an efficient PowerMap algorithm to generate low-power depth-optimal mapping. The experimental results showed that our algorithm generated depth-optimal mapping solutions with significant power reduction.

6. Acknowledgments

We would like to thank Dr. Jason Cong and his group members for providing the RASP package and many valuable suggestions and help. We are also very grateful to Dr. Amir H. Farrahi for his helpful information on his former work for low-power technology mapping.

References


