LUT-Based FPGA Technology Mapping under Arbitrary Net-Delay Models

Jason Cong and *Yuzheng Ding*

Department of Computer Science
University of California, Los Angeles, CA 90024, U.S.A.

Tong Gao

Department of Computer Science
University of Illinois, Urbana Champaign, IL 61801, U.S.A.

Kuang-Chien Chen

Fujitsu America, Inc.
3055 Orchard Drive, San Jose, CA 95134, U.S.A.

Abstract

The field programmable gate-array (FPGA) has become an important technology in VLSI ASIC designs. Most existing algorithms for performance-driven technology mapping for Lookup-table (LUT) based FPGA designs are based on the unit-delay model. In this paper we study the technology mapping problem under arbitrary net-delay models. We show that if the net delay can be determined or estimated before mapping, the problem can be *optimally* solved in polynomial time based on efficient network flow computation. We have implemented the algorithm and tested it on a number of MCNC benchmark examples.

LUT-Based FPGA Technology Mapping under Arbitrary Net-Delay Models

Jason Cong and *Yuzheng Ding*
Department of Computer Science
University of California, Los Angeles, CA 90024, U.S.A.

Tong Gao
Department of Computer Science
University of Illinois, Urbana Champaign, IL 61801, U.S.A.

Kuang-Chien Chen
Fujitsu America, Inc.
3055 Orchard Drive, San Jose, CA 95134, U.S.A.

1. Introduction

The *field programmable gate array* (FPGA) has become an important technology for VLSI designs in recent years. The unique feature of FPGA is its field programmability, that is, users are able to define and modify the functionality of the FPGA chips in the field (i.e. in their office) without going through the fabrication process. There are several advantages of using FPGAs to implement system designs. Because the FPGA chips are generically pre-fabricated, the fabrication cost of system designs is eliminated. Because of its user programmability and reprogrammability, FPGA results in short system turnaround time and the flexibility of accommodating design modifications at no extra cost. These features make FPGA very attractive to application specific integrated circuit (ASIC) designers. FPGAs has been used in fast implementation of customized VLSI circuit such as image processor, graphical accelerator, etc., reconfigurable system designs, rapid system prototyping and low volume production, system emulation, and FPGA-based computing engines. The steady increase in FPGA density and speed

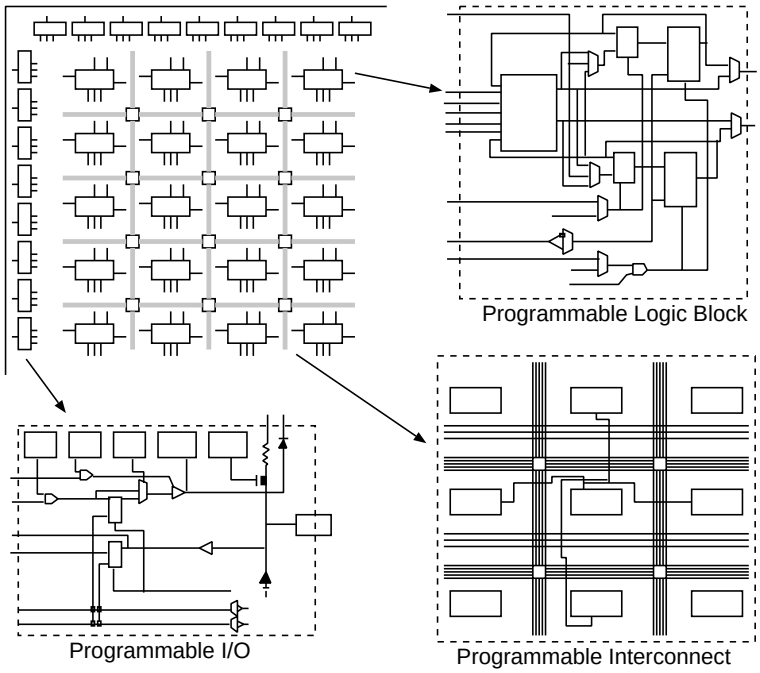

Fig. 1 The Xilinx XC3000 FPGA.

and decrease in FPGA cost have made these applications feasible and economical. In the past several years, the FPGA market has been growing rapidly.

An FPGA architecture consists of the programmable logic blocks, programmable interconnections, and programmable I/O pads. Programmable logic blocks provide the capability of implementing user defined logic functions. Programmable interconnections support flexible connections among the logic blocks. programmable I/O pads provide flexible connections with other chips and devices in the system. All these components can be reprogrammed to accommodate the design change.

The lookup table (LUT) -based FPGA is a popular architecture used by several major FPGA manufacturers, including Xilinx and AT&T [17, 7]. In LUT-based FPGA, the basic programmable logic block contains a K-input lookup table implemented by a 2^K-bit SRAM, which can implement any Boolean function of up to K variables. An example of Xilinx FPGA architecture is illustrated in Fig. 1.

The design process for a FPGA based system consists of the system level design, logic level design, and physical design. The system level design transforms the high level specification of

the system to a logic level representation, which is usually technology independent. The logic level design optimizes the logic representation of the system, and represents the system by logic devices available in the target technology (i.e. FPGA). The physical design determines the physical layout of the logic devices on the FPGA chips.

The logic level design is of particular importance in LUT-based FPGA designs, which consists of the technology independent logic synthesis and technology mapping. Technology independent logic synthesis produces an optimized logic level specification using conventional logic devices like AND and OR gates. It usually optimizes technology independent objectives, such as minimizing the total of literals in the Boolean equations, or the depth of the Boolean network. *Technology mapping* in LUT-based FPGA designs is to transform a general Boolean network into a functionally equivalent network of K-LUTs. The input Boolean network is usually optimized using technology independent logic synthesis techniques in the first stage of the logic level design. Technology mapping optimizes the technology dependent objectives, such as minimizing the number of K-LUTs used in the mapping solution (*area minimization*) [9, 10, 6, 8, 16, 12], minimizing the delay of the mapping solution (*delay minimization*) [11, 5, 2, 3], or maximizing the routability of the mapping solution [1, 14].

This paper studies the problem of performance-driven technology mapping for LUT-based FPGAs, i.e. technology mapping with the objective of delay minimization. The speed of FPGA designs is usually slower than the gate array or standard cell designs due to the extra delay introduced by the programmable interconnections on FPGA chips. For example, the programmable interconnections on Xilinx FPGA chips consists of wire segments connected by programmable switches. The switches are implemented using pass transistors controlled by SRAM cells, which induce large parasitic resistance and capacitance, thus significantly slow down the signal propagation. Therefore, it is very important to carry out performance optimization at logic level in FPGA designs in order to get satisfactory system speed.

Previous mapping algorithms for delay minimization include MIS-pga-delay by Murgai et al. [11], Chortle-d by Francis et al. [5], DAG-Map by Chen et al. [2], and FlowMap by Cong and Ding [3]. MIS-pga-delay combines Boolean synthesis with technology mapping and considers the number of levels of the mapping solution as well as the routing delay in its optimization procedure. Chortle-d uses bin-packing heuristic and several postprocesssng operations to

minimize the number of levels in the mapping solution. DAG-Map minimizes the number of levels of the mapping solution based on Lawler's labeling algorithm. None of these algorithms guarantees the optimality of their mapping results. FlowMap for the first time solves the delay-optimal mapping problem in polynomial time based on efficient flow computation. The delay in FlowMap, however, is also measured by the number of levels in the mapping solution, as in Chortle-d and DAG-Map.

If we assume that each level of a network has uniform delay, such a delay model is called the *unit-delay model*. Under the unit-delay model, minimizing delay is equivalent to minimizing the number of levels (or *depth*) of the network. In LUT-based FPGA designs, although the delay of each LUT is the same, the interconnection delay of each net may vary considerably. Therefore, more accurate delay models are needed to allow variable delay values for different nets. Mapping algorithms based on more accurate delay models may produce mapping solutions of better performance.

In this paper we study the problem of LUT-based FPGA technology mapping under *arbitrary net-delay* model. We generalize the idea in FlowMap to develop an efficient algorithm that guarantees delay-optimal mapping solution for general networks if the delay of each net is known prior to mapping. By efficiently computing a *minimum height K-feasible cut* of each node in the network, we are able to compute an optimal mapping for each node, hence obtain the optimal mapping solution for the entire network by dynamic programming. We have implemented our algorithm and tested it on a set of MCNC benchmark circuits under non-unit-delay models.

There are two important reasons to develop an optimal mapping algorithm that can handle arbitrary net-delay models. First, such an algorithm will be more effective in generating good mapping solutions since it is not restricted to any specific net-delay model. Second, such an algorithm can be used as an effective tool to evaluate various net-delay models. Previous study of delay models was sensitive to the choice of the heuristic mapping algorithms due to the lack of the optimal algorithm. using an optimal algorithm under *any* delay model, we can evaluate different delay models more accurately based on the optimal mapping results.

The remainder of this paper is organized as follows. Section 2 gives a precise problem formulation and some preliminaries. Section 3 presents the technology mapping algorithm.

Experimental results are presented in Section 4. Section 5 concludes the paper.

2. Problem Formulation and Preliminaries

A general combinational Boolean network can be represented as a directed acyclic graph (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 primary input (PI) node has no incoming edge and a primary output (PO) node has no outgoing edge. We use $input(v)$ to denote the set of nodes which are fanins of node v, and $output(v)$ to denote the set of nodes which are fanouts of v. Given a subgraph H of the Boolean network, $input(H)$ denotes the set of *distinct* nodes outside H which supply inputs to the gates in H. For a node v in the network, a *K-feasible cone at v*, denoted C_v, is a subgraph consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely in C_v, and $|input(C_v)| \le K$. A Boolean network is *K-bounded* if $|input(v)| \le K$ for each node v.

We assume that each programmable logic block in an FPGA is a K-input lookup-table (K-LUT) that can implement any K-input Boolean function. Thus, each K-LUT can implement any K-feasible cone of a Boolean network. The technology mapping problem for K-LUT based FPGAs is then to cover a given Boolean network with K-feasible cones[1]. Fig. 2 shows an example of mapping a Boolean network into a 3-LUT network. Note that we allow these cones to overlap, which means that the nodes in the overlapped region can be duplicated when generating K-LUTs. In fact, our algorithm is capable of duplicating nodes automatically when necessary, in order to achieve delay optimization. A technology mapping solution S is a DAG

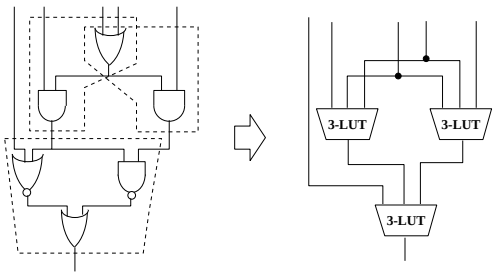

Fig. 2 Mapping a Boolean network to a K-LUT network (K=3).

[1] The PI and PO nodes are not covered.

where each node is a K-feasible cone (equivalently, a K-LUT) and the edge (C_u, C_v) exists if u is in $input(C_v)$.

The delay of an FPGA circuit is determined by two factors: the delay in K-LUTs and the delay in the interconnection paths. Each K-LUT contributes a constant delay (the SRAM access time) independent of the function it implements. The output of a K-LUT *T* can be the input of one or more K-LUTs which are the *fanouts* of *T*. We call the interconnection net that connects *T* to its fanouts the *fanout net* of *T*. The delay of the fanout net of a K-LUT *T* is determined by several factors, including the parasitic resistance and capacitance of the interconnection wires and programmable switches, as well as the total load capacitance of the fanout K-LUTs that are driven by *T*. Therefore, the delay of a fanout net usually varies from net to net. The *unit-delay model* ignores the difference among net delays by assuming they are a constant. The *net-delay model* allows different nets to have different delay values. Although different terminals of the same net may also have different delay values, such difference is insignificant compared with the difference between nets. Therefore, a net-delay model assumes that signals arrive all load terminals in the same net at the same time. Several net-delay models have been proposed to capture variable net delay, including the *nominal delay model* [13] which assumes that the fanout net delay is proportional to the number of fanouts.

The delay of a path from a PI node to a PO node in a K-LUT network is the sum of the delay of all the K-LUTs along the path and the designs of their fanout nets (including the fanout nets of the PIs). The delay of a K-LUT network is the largest delay of any path from a PI to a PO

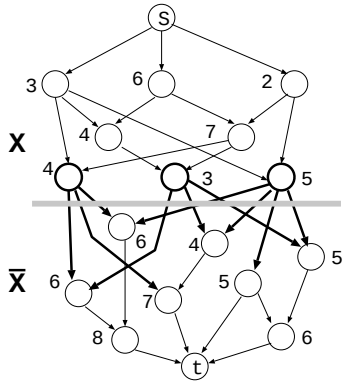

Fig. 3 A 3-feasible cut of edge cut-size 10, node cut-size 3, and height 5.

in the network. We say that a mapping solution is *optimal* if its delay is minimum under the given delay assignment. The objective of our algorithm is to find an optimal mapping solution.

Given a network $N = (V(N), E(N))$ with a source s and a sink t, a *cut* $(X, \bar{X})$ is a partition of the nodes in $V(N)$ such that $s \in X$ and $t \in \bar{X}$. The *node cut-set* of $(X, \bar{X})$, denoted as $C(X, \bar{X})$, is the set of nodes in X that are adjacent to some node in $\bar{X}$, i.e.

$$C(X, \bar{X}) = \{x : (x, y) \in E(N), x \in X \text{ and } y \in \bar{X}\}$$

The *node cut-size* of $(X, \bar{X})$, denoted as $n(X, \bar{X})$, is the number of nodes in $C(X, \bar{X})$. A cut $(X, \bar{X})$ is *K-feasible* if its node cut-size is no more than K, i.e., $n(X, \bar{X}) \leq K$. Assuming that each edge (u, v) has a non-negative capacity $c(u, v)$. Then, the *edge cut-size* of $(X, \bar{X})$, denoted $e(X, \bar{X})$, is the sum of the capacities of the edges that go from X to $\bar{X}$, i.e.

$$e(X, \bar{X}) = \sum_{u \in X,\ v \in \bar{X}} c(u, v)$$

Moreover, assuming that there is a given label $l(v)$ associated with each node v. Then, the *height* of a cut $(X, \bar{X})$, denoted $h(X, \bar{X})$, is defined to be the maximum label of the nodes in $C(X, \bar{X})$. Fig. 3 shows a cut $(X, \bar{X})$ in a network with given node labels. Assuming that each edge has unit capacity, we have $n(X, \bar{X}) = 3$, $e(X, \bar{X}) = 10$, and $h(X, \bar{X}) = 5$. The highlighted nodes (edges) form the node cut-set (edge cut-set).

3. Delay Optimal Technology Mapping Algorithm for LUT-Based FPGAs

In this section we present our delay optimal technology mapping algorithm. We assume that prior to the mapping, the delay of each net is known and fixed. In the final mapping solution, if all the terminals of the net are in a single K-LUT, its delay will become zero since the net is not visible in the mapping solution; otherwise its delay will be the pre-assigned value. More discussion on this restriction will be given at the end of this section.

Our algorithm is applicable to any *K-bounded* Boolean network. Given a general Boolean network as input, if it is not K-bounded, we first transform it into a 2-input simple gate network using the DMIG algorithm discussed in [2]. Note that the optimality of our algorithm holds not only for such 2-input simple gate networks, but also for any K-bounded general Boolean network.

The mapping algorithm presented in this paper is a generalization of the FlowMap algorithm [3] which produces optimal mapping solution under unit-delay model. The algorithm has two phases. In the first phase, it computes a label for each node which reflects the delay of the K-LUT implementing that node in an optimal mapping solution. In the second phase, it generates the K-LUT mapping solution based on the node labels computed in the first phase. The details are discussed in the following subsections.

3.1. The Labeling Phase

Given a K-bounded Boolean network N, let N_t denote the subnetwork consisting of node t and all the predecessors of t. We define the *label* of t, denoted as $l(t)$, to be the delay of the optimal K-LUT mapping solution of N_t. Clearly, the delay at the K-LUT containing t, which is the maximum delay for an input signal to reach the output pin of that K-LUT, is at least $l(t)$ in an optimal mapping solution, and the maximum label of the POs of N determines the delay of the optimal mapping solution of N. The first phase of our algorithm computes such labels of all the nodes in N, according to the topological order starting from the PIs.

Denote the delay of a K-LUT as D_T. For each node u other than the PO nodes, denote the fanout net delay of u as $D(u)$. For the simplicity of the algorithm, instead of computing the $l(u)$ directly, we compute $L(u) = l(u) + D(u)$ which can be trivially converted to $l(u)$. We shall call $L(u)$ the *label* of u in the remaining of this subsection.

First, for each PI node t, we know that

$$L(t) = D(t). \tag{1}$$

Suppose t is the current node being processed, and t is not a PI or PO. Then, for each node $u \neq t$ in N_t, the label $L(u)$ have been computed. By including in N_t an auxiliary node s and connecting s to all the PI nodes in N_t, we obtain a network with s as the source and t as the sink. For simplicity we still denote it as N_t. Let $LUT(t)$ be the K-LUT that implements node t in a K-LUT mapping solution of N_t, and let $\bar{X}$ denote the set of nodes in $LUT(t)$ and X denote the remaining nodes in N_t. It is easy to see that $(X, \bar{X})$ forms a K-feasible cut between s and t in N_t, because the number of input nodes of $LUT(t)$ is no more than K. Moreover, let u be a node in $C(X, \bar{X})$. Clearly, u must be implemented by a K-LUT $LUT(u)$ in this mapping solution of N_t. Since $l(u)$ is the minimum delay that any K-LUT implementing u can have, the delay at $LUT(u)$ is at least

$l(u)$, which implies that the delay of $LUT(t)$ is at least $L(u) + D_T$. More precisely,

$$L(t) \geq \max \{L(u) : u \in C(X, \bar{X}) \} + D_T + D(t) = h(X, \bar{X}) + D_T + D(t). \quad (2)$$

Since we want to find the minimum label for t that corresponds to the minimum possible delay of any mapping solution of N_t, we need to minimize the righthand side of (2), which is equivalent to minimizing the *height* of the cut $(X, \bar{X})$. Therefore, we need to compute a *minimum height K-feasible cut* [2] $(X^*, \overline{X^*})$ in N_t, which will give t the label

$$L(t) = H(X^*, \overline{X^*}) + D_T + D(t). \quad (3)$$

When t is a PO node, since t does not have to be covered by any K-LUT and it has no fanout, we have

$$L(t) = l(t) = \max \{L(u) : u \in input(t) \}. \quad (4)$$

Based on the above discussion, we have

Lemma 1. Let $L(t)$ be the label computed by Eqs. (1), (3), and (4), then $l(t) = L(t) - D(t)$ gives the minimum delay of any mapping solution of N_t. □

Clearly, the unit-delay model is a special case where $D(u)$ is always a constant for each fanout net. By including it into D_T, we can assume $D(u) = 0$, so $l(u) = L(u)$.

The computation of a minimum height K-feasible cut in the case of arbitrary net-delay model is more complicated than that under the unit-delay model, since $L(t)$ may not be monotonic along a path from PI to PO due to the arbitrary net delay $D(u)$. However, we can still identify the possible range of $L(t)$.

Lemma 2. Let t be a node in a network N, t is not a PI or PO. Then,

$$L(t) \leq \max \{L(u) : u \in input(t) \} + D_T + D(t), \quad (5)$$

$$L(t) \geq \max \{D(u) : u \in N_t, u \text{ is a PI} \} + D_T + D(t), \quad (6)$$

---

[2] We exclude the cuts $(X, \bar{X})$ where $\bar{X}$ contains a PI node. Our algorithm to be shown later on guarantees that such kind of cuts are not generated.

$$L(t) \geq \max \{L(u) - D(u) : u \in N_t,\ u \text{ is not a PI}\} + D(t). \quad \textbf{(7)}$$

Proof Since $(N_t - \{ t \}, \{ t \})$ is a K-feasible cut in N_t, (5) is true. Since a PI node cannot be included in any K-LUT, (6) is true. To show (7) is true, consider the two cases of u in an optimal mapping solution of N_t.

Case 1: u is contained in $LUT(t)$. In this case, the cut $(X, \bar{X})$ defined by $LUT(t) = \bar{X}$ also induces a K-feasible cut $(Y, \bar{Y})$ in N_u, where $C(Y, \bar{Y}) = C(X, \bar{X}) \cap N_u$. Thus, $h(Y, \bar{Y}) \leq h(X, \bar{X})$, which implies that the minimum height of a K-feasible cut in N_u is no more than $h(X,\bar{X})$. Therefore, $L(u) - D_T - D(u) \leq L(t) - D_T - D(t)$, i.e. $L(t) \geq L(u) - D(u) + D(t)$.

Case 2: u is outside of $LUT(t)$. In this case, either u is implemented by a K-LUT in the mapping solution, which implies that $L(t) \geq L(u) + D_T + D(t) > L(u) - D(u) + D(t)$; or u is covered by the K-LUT that implements another node v outside $LUT(t)$, which implies that $L(v) \geq L(u) - D(u) + D(v)$ (according to the proof of Case 1), and $L(t) \geq L(v) + D_T + D(t)$. Thus, $L(t) \geq L(u) - D(u) + D(v) + D_T + D(t) > L(u) - D(u) + D(t)$. Therefore, in either case we have $L(t) \geq L(u) - D(u) + D(t)$, so (7) is true. □

Note that (7) means that $l(t) \geq l(u)$, so $l(t)$ is monotonic. However, due to the existence of $D(t)$, $L(t)$ is in general not monotonic.

Given the upper bound in (5) and the lower bounds in (6) and (7) of $L(t)$, we can derive the range of the minimum height of a K-feasible cut in N_t. If we can determine whether N_t has a K-feasible cut of height H or not, we can perform a binary search over the possible height values to determine the minimum height $H_{\min}$ and hence the label $L(t)$.

To compute a K-feasible cut of height H in N_t, we apply the following transformation on N_t to convert the problem into a standard edge-cut computation problem. Specifically, we obtain a network N'_t as the following: N'_t has the source s and sink t of N_t, and for each of the other nodes u in N_t, N'_t contains two nodes u' and u'', where u' inherits all the incoming edges of u and u'' inherits all the outgoing edges of u. In other words, u is *split* into two nodes. Moreover, u' and u'' are connected by an edge $<u', u''>$ which is called a *bridge edge*, and labeled as u. Finally, we assign capacities to the edges as follows. For each bridge edge $<u', u''>$ labeled u satisfying $L(u) \leq H$ in N_t, the capacity of $<u', u''>$ is one; for all other edges, the capacity is infinite.

Lemma 3. N_t has a K-feasible cut of height no more than H if and only if N'_t has a cut whose edge cut-size is no more than K.

Proof Suppose N_t has a K-feasible cut $(X', \bar{X}')$ of height H. Then, the size of $C(X, \bar{X})$ is no more than K, and the label of any node in $C(X, \bar{X})$ is no more than H. Therefore, the set of bridging edges in N'_t corresponding to the nodes in $C(X, \bar{X})$ all have capacity one, which form an edge cut-set of size no more than K in N'_t.

On the other hand, if N'_t has a cut $(X', \bar{X}')$ such that $e(X', \bar{X}') \leq K$, clearly the edge cut-set only contains the bridging edges that correspond to the nodes in N_t with labels no more than H, since the capacities of all other edges are infinite. Because this is an edge cut-set, the corresponding node set in N_t forms a node cut-set $C(X, \bar{X})$ with $n(X, \bar{X}) = e(X', \bar{X}')$. Therefore, $(X, \bar{X})$ is a K-feasible cut in N_t of height no more than H. □

This proof also shows how to convert a edge cut in N'_t to a node cut in N_t. Fig. 4 shows the entire transformation. In Fig. 4(b), all the edges whose capacities are not specified have infinite capacity. Fig. 4(c) shows the cut computed from (b), and in (d) it is converted back to a cut in the original network (a).

According to the Max-flow Min-cut Theorem [4], N'_t has a cut whose edge cut-size is no more than K if and only if the maximum flow between s and t in N'_t has value no more than K. Since we are only interested in testing if the maximum flow is of value K or smaller, we apply the augmenting path algorithm in N'_t to compute a maximum flow. (For details of the augmenting path algorithm, see [15].) From the construction of N'_t, it can be seen that each augmenting path in the flow residual graph of N'_t from s to t increases the flow by either one or infinity. If we can find a path of infinite capacity, or $K + 1$ augmenting paths, then the maximum flow in N'_t has value more than K, and we can conclude that N'_t does not have a cut $(X', \bar{X}')$ with $e(X', \bar{X}') \leq K$. Otherwise, the residual graph is disconnected before we find the $(K + 1)$-th augmenting path, and the disconnected residual graph induces a cut of edge cut-size no more than K. Moreover, we can find such a cut $(X', \bar{X}')$ by performing a depth first search starting at the source s, and including in X' all the nodes which are reachable from s. Since finding an augmenting path takes $O(m)$ time, where m is the number of edges in the residual graph of N'_t (which is in the same order as the number of edges in N_t), we can determine in $O(Km)$ time whether N'_t has a cut of edge cut-size

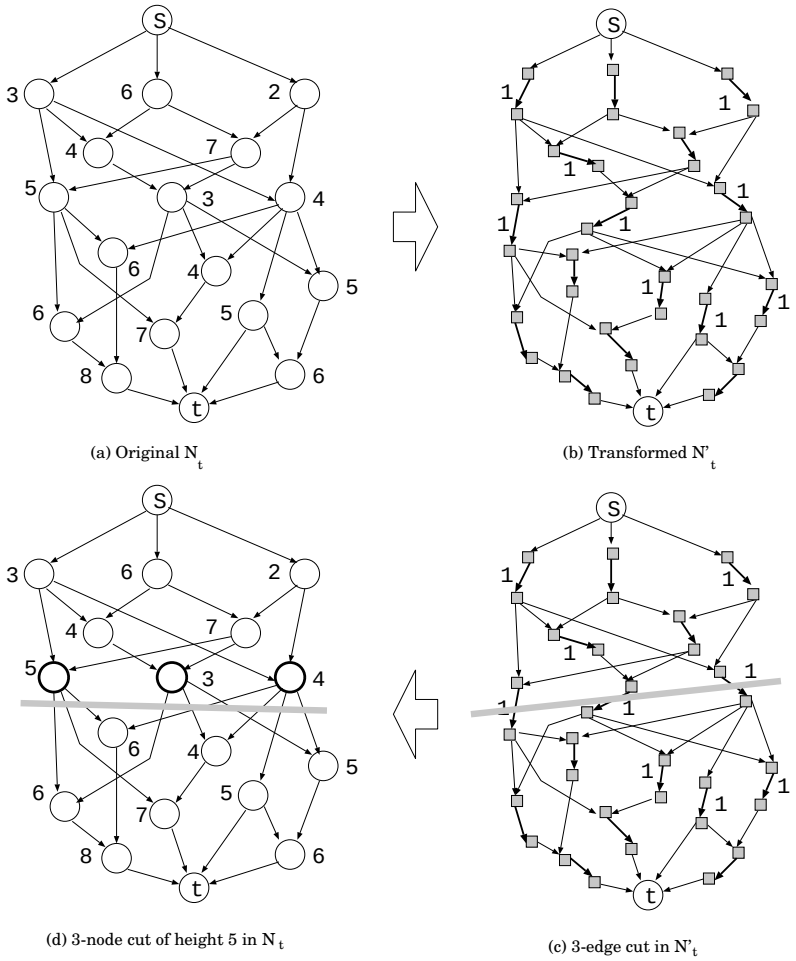

Fig. 4 Network transformation for computing a K-feasible cut of height H in N_t ($H = 5,\ K = 3$).

no more than K and find one if such a cut exists. Such a cut $(X', \bar{X}')$ in N'_t induces a K-feasible cut $(X,\ \bar{X})$ of height no more than H in N_t.[3]

In practice, we can simplify N'_t as the following. For any node u in N'_t other than s and t, if u has a single outgoing edge $<u,\ v>$ of infinite capacity, it cannot be in any edge cut-set, and any s-t path that reaches u must go through v. In this case, u can be collapsed into v and the edge $<u,\ v>$ can be removed. Similarly, if u has a single incoming edge $<v,\ u>$ of infinite capacity, u can also be collapsed into v. This may to reduce the size of the network significantly. For the example shown in Fig. 4(b), such collapsing will reduce the number of nodes from 34 to 14, and reduce the number of edges from 48 to 25.

Being able to determine if a K-feasible cut of height H exists in N_t, we can easily determine the smallest $H = H_{\min}$ such that a K-feasible cut of height $H_{\min}$ exists in N_t by binary search. Let the K-feasible minimum height cut be $(X, \bar{X})$, then, the implementation of t in an optimal mapping solution of N_t is $LUT(t) = \bar{X}$, and $L(t) = h(X, \bar{X}) + D_T + D(t)$. (In other words, $l(t) = H_{\min} + D_T$). Note that the height of any cut must be a node label according to the definition. Therefore, the range of the binary search is the set of different labels $L(u)$ for $u \in N_t$, restricted by Lemma 2. Since the number of different labels never exceeds the number of nodes in N_t, the binary search takes $O(\log n)$ steps, where n is the number of the nodes in N_t. Based on the above discussions, we have

Theorem 1. A minimum height K-feasible cut in N_t can be found in $O(Km\log n)$ time where m and n is the number of edges and nodes in N_t. □

Note that although a K-feasible cut can also be computed by enumerating all possible combinations of K or less nodes, which has time complexity of $O(n^K)$. Such a brute-force method is significantly worse than our method when n or K is not very small.

Applying Theorem 1 to each node in N in the topological order in the labeling phase, we have

Corollary 1. The labels of all the nodes in N can be computed in $O(Kmn\log n)$ time, where n and m are the number of nodes and edges in N, respectively. □

In current technology, K is usually 4 or 5. Moreover, for a K-bounded network, $m = O(Kn)$. Therefore, the complexity of the labeling phase of our algorithm is $O(n^2 \log n)$ in practice.

3.2. The Mapping Phase

In the labeling phase, for each node u we have computed $LUT(u)$, the K-LUT implementing u in an optimal mapping of N_u. The second phase of our algorithm is to generate the K-LUTs in the optimal mapping solution. Since this phase is the same as that in FlowMap [3], we give only a brief description. We generate K-LUTs only for nodes which have fanouts to

[3] It is clear that for the resulting cut $(X, \bar{X})$ in N_t, $\bar{X}$ does not contain any PI nodes since any outgoing edge of the source s in N'_t has infinite capacity.

POs or other K-LUTs. During the mapping phase, we maintain a list L of nodes which have to be implemented by K-LUTs. Initially, L contains those nodes that have fanouts to the PO nodes. Then, we repeatedly remove a node u from L, create the K-LUT $LUT(u)$ computed in the first phase as the K-LUT implementing u, and add into L all the nodes in $input(LUT(u))$, whose K-LUTs have not yet been created. (The nodes that are never added into L do not need to be implemented since they are completely covered by the K-LUTs implementing other nodes.) The mapping phase ends when L only contains PI nodes. It is easy to see that the resulting network is logically equivalent to the original network, and the entire second phase takes linear time. Moreover, it is not hard to see that the delay from any PI to a K-LUT $LUT(u)$ is no more than $l(u) = L(u) - D(u)$. Therefore, the mapping solution is optimal. (A formal proof of a similar result under unit-delay model can be found in [3]).

In summary, we have

Theorem 2. For any K-bounded Boolean network N, the algorithm produces a K-LUT mapping solution with the minimum delay under the pre-assigned net delay in $O(Kmn\log n)$ time, where n and m are the number of nodes and edges in N. □

3.3. Static versus Dynamic Net-Delay Models

We have shown that our algorithm produces mapping solutions with minimum delay according to the pre-assigned net delay (which can be arbitrary). However, the pre-assigned net delays may not accurately reflect the delays of the corresponding nets in the mapping solution, as the size of some nets may change after mapping. There are two possible causes of net size change. First, two or more fanout nodes of a node v may be packed into a single K-LUT. In this case, the size of the fanout net of v will decrease. Second, a node v may be duplicated so that it can be packed into several K-LUTs. In this case, the size of the fanout net of v is decreased by one when each duplicated copy is introduced and packed. Moreover, when v is duplicated, the fanout net size of each node u in $input(v)$ may be increased, as u now may have to supply input to both u and its duplicated copies.

In our mapping algorithm, we have assumed that the net delay will not change even if the net size is changed in the mapping solution. Such *static net-delay* models are usually not accurate, since in reality net delay is often closely related to net size. For example, the *nominal*

delay model [13] assumes that net delay is always proportional to the net size. Our algorithm does not produce optimal mapping solutions under *dynamic net-delay* models such as the nominal delay model.

Under dynamic delay models, the dependency of net delay on the net size significantly complicates the mapping problem. The difficulty comes from the paradox that the delay of a node v (or any possible mapping solution for it) need to be determined based on the delays of its predecessors in N_v, while its predecessors may have fanouts to the nodes outside N_v which have not been processed. Therefore, the dynamic programming technique used in this paper does not apply anymore. In fact, we have shown that delay-optimal mapping under the nominal delay model is NP-hard.

Nevertheless, even with the inaccuracy caused by net size change, a static net-delay model is usually more accurate than the unit-delay model. Moreover, we can derive static net-delay models on the unmapped network to *predict* the net delay after mapping. Accurate delay models can be used to derive such *net-delay predict* models, which are practically useful in K-LUT mapping.

In the next section, we will present experimental results based on a static net-delay model that tries to predict the nominal delay of the nets in the mapping solution.

4. Experimental Results

We have implemented the above algorithm as an extension of the FlowMap algorithm. The new mapping algorithm is named FlowMap-d, written in C language on SUN SPARC workstations.

We tested the algorithm based on the nominal delay model. Since the nominal delay model is a dynamic net-delay model, we use a static *nominal delay predict* model which considers possible net size changes during mapping to predict the nominal delay of the nets in the mapping solution. Intuitively, for a node v with multiple fanout, the fanout and/or reconvergence of its successors will affect the fanout net size of v after mapping. Therefore, we use this factor to estimate the potential change of the fanout net size of v. To simplify the delay computation, we only considered the immediate fanout nodes of v. In particular, we model the fanout net delay of a node v in the original unmapped network to be

$$D(v) = \alpha \cdot | \, output(v) \, | + \beta \cdot (\, | \bigcup_{w \in output(v)} output(w) \, | - | \, output(v) \, | \,),$$

where α and β are positive constants used to adjust the relative weights of the two terms and the K-LUT delay D_T. When the second term is positive, it means that the nodes in $output(v)$ will fanout to more nodes. In this case there is possibility that they are duplicated during mapping, resulting in the size increase of the fanout net of v. Therefore, we increase the pre-assigned net delay accordingly. If the second term is negative, it means that the nodes in $output(v)$ converge, in which case it is likely that some of them will be packed into one K-LUT, resulting in the size decrease of the fanout net of v. Therefore, we decrease the pre-assigned net delay in this case.

We used this model with $D_T = 100$, $\alpha = 10$, and $\beta = 2$ in the above formula [4]. We First mapped (for K=5) a set of mid-size MCNC benchmark networks using FlowMap-d. Then, We applied several postprocessing operations on the network to minimize area. The postprocessing operations were *predecessor packing* and *gate decomposition*, proposed in [3] and used in FlowMap. It has been shown that these operations will not increase the depth of the network. However, this operations may carry out node duplications, hence may increase the delay under the nominal delay model. Therefore, we used a restricted version such that no node duplication is used. This guarantees that the delay of the mapping solution will not increase. Finally, we went through the physical design step, using Xilinx XACT design system to place and route the mapped circuits onto Xilinx XC3000 FPGA chips, whose logic blocks can be viewed as 5-LUTs [17].

In order to compare the performance, we also carried out the same experiment procedure using the unit-delay model (i.e. $\alpha = \beta = 0$) and the nominal delay model (i.e. $\beta = 0$) to assign the net delays for the mapping. The experimental results are reported in Tables 1, 2, and 3.

Table 1 compares the mapping solutions of FlowMap using the unit-delay model, FlowMap-d using the nominal delay model, and FlowMap-d using the nominal delay predict model, *before* the postprocessing. The delay value in this table were calculated using the nominal delay model on the mapped networks. For each circuit, the best delay among the three solutions is highlighted. As can be seen from the table, in most cases FlowMap-d got mapping solutions

[4] The reason of choosing $\alpha = 10$ was that according to our observation, for Xilinx XC3000 FPGA, on average the worst case net delay increases about 0.7ns for each fanout added to the net, which is one tenth of the 7ns logic block delay.

Circuit	Original network		FlowMap (unit delay)		FlowMap-d (nominal delay)		Flowmap-d (nominal delay predict)	
	#nodes	delay	#nodes	delay	#nodes	delay	#nodes	delay
9sym	200	1770	75	**880**	75	**880**	69	**880**
C880	548	2880	251	1570	286	1540	285	**1440**
alu2	393	3490	173	**1730**	199	1830	199	1820
apex6	779	1390	300	890	301	790	298	**750**
apex7	247	1370	107	880	108	660	107	**640**
count	216	1500	92	780	94	**760**	93	**760**
rot	647	2260	299	1440	304	1260	309	**1140**
vg2	120	990	54	590	60	610	59	**510**

Table 1 Comparison of nominal delay before postprocessing

with smaller delay, and by using nominal delay predict model, we get better solutions than directly using the nominal delay model. On the other hand, since the nominal delay predict model is static, the estimate may be inaccurate. On the circuit *alu2*, FlowMap actually got the best result.

Table 2 shows the mapping solutions of the three algorithms *after* the postprocessing. For FlowMap we used the original postprocessing operations [3] which may duplicate nodes, and for FlowMap-d we restricted to the duplication-free version. For FlowMap, in some circuits the delay increased due to node duplication, while for FlowMap-d the delay never increased. However, since the postprocessing procedure for FlowMap is more aggressive, its area reduction is more significant. In fact, in both Table 1 and Table 2, the FlowMap-d solutions used more K-LUTs (except for one case). The reason is that when computing a minimum height K-feasible cut,

Circuit	FlowMap (unit delay)		FlowMap-d (nominal delay)		Flowmap-d (nominal delay predict)	
	#nodes	delay	#nodes	delay	#nodes	delay
9sym	61	870	70	870	64	870
C880	232	1590	283	1540	280	1440
alu2	162	1670	196	1830	196	1820
apex6	257	890	290	780	286	750
apex7	89	870	105	660	105	640
count	76	760	76	710	76	710
rot	268	1450	298	1260	302	1140
vg2	45	590	56	600	56	510

Table 2 Comparison of nominal delay after postprocessing

FlowMap-d usually has fewer choices than FlowMap due to the larger number of different labels in a network when an arbitrary net-delay model is used. Consequently, FlowMap-d cannot minimize area during the mapping procedure as effectively as FlowMap does [3]. This is a disadvantage of the FlowMap-d algorithm.

Finally, for those circuits that can be implemented using a single FPGA chip of XC3000 series, Table 3 shows the actual delays of the mapping solutions in Table 2 after placement and routing. We used Xilinx **apr** placement/routing tool, which is part of the XACT FPGA design system, to perform the placement and routing and to estimate the actual delay. We always selected the smallest XC3000 FPGA chip that can implement the design for the experiment.

Unfortunately, although the sizes of all the circuits are within the limit of the XC3000 FPGA chip size (the XC3090 FPGA has 320 logic blocks), The circuits *apex6* and *rot* have too many I/O pins (234 and 242 respectively) to be implemented using a single chip (the maximum number of user I/O pins for XC3090 is 144). On the other circuits, the FlowMap-d mapping solutions obtained under the nominal delay predict model outperform the other two sets of solutions consistently. Meanwhile, on the two circuits *alu2* and *vg2*, the mapping solutions obtained directly under the nominal delay model are worse than the FlowMap solutions, which is consistent with Tables 1 and 2.

Circuit	**Device**	**Delay**		
		FlowMap (unit delay)	**FlowMap-d (nominal delay)**	**Flowmap-d (nominal delay predict)**
9sym	3030PC68	101.9	101.1	99.8
C880	3090PQ160	208.5	202.6	200.2
alu2	3064PC84	196.9	199.0	196.6
apex6	******	-	-	-
apex7	3042PP132	104.0	99.9	96.8
count	3030PC68	87.6	81.5	76.2
rot	******	-	-	-
vg2	3020PC68	79.1	81.7	74.6

Table 3 Comparison of actual delay after placement/routing

5. Conclusion

In this paper we have presented an efficient algorithm to carry out performance-driven technology mapping for LUT-based FPGA designs. Our algorithm is applicable to general Boolean networks and guarantees optimal solution under arbitrary pre-assigned net delays. By using static net-delay models that properly predict the net delays after mapping, our algorithm can generate better mapping solutions than unit-delay model based mapping algorithms. We have implemented our algorithm and tested it on a set of MCNC benchmark examples.

The capability of taking arbitrary delay models allows our algorithm to be used as an evaluation tool for the study of delay models. The modeling of FPGA interconnect delay is a very important problem. We have analyzed the inherent difficulty of incorporating a net-delay model, like the nominal delay model, directly into the mapping procedure. We have proposed to use models that predict net delays in the mapping solution, instead of the unmapped network. Our experiments justified this approach.

The nominal delay model considers the driving load as a primary factor that affects the delay. There are many other factors. The placement and global routing result determines the wire length and the number of programmable switches of each net, while the detail routing result determines, in the case of Xilinx 3000 series, the number of pass transistors each connection will go through, which result in considerable capacitance. Accurate prediction of such factors in the mapping stage is a difficult task. An iterative approach that combines mapping with placement and routing will be beneficial. Unlike the Boolean optimization based algorithms, our algorithm is very efficient even for large scale circuits, hence is affordable to be integrated with placement and routing in such an iterative procedure.

The development of accurate delay model is beyond the scope of this paper. The simple nominal delay predict model used in our experiments is only for the purpose of testing our algorithm. Currently, we are working on developing more accurate delay models for FPGA interconnections so that they can be used in the synthesis phase.

6. Acknowledgment

This research is partially supported by a grant from Xilinx Inc. under the State of California MICRO program, a grant from Fujitsu America, and the National Science Foundation Young Investigator Award.

References

[1] Bhat, N. and D. Hill, ‘‘Routable Technology Mapping for FPGAs,’’ *First Int’l ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 143-148, Feb. 1992.

[2] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, ‘‘DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization,’’ *IEEE Design and Test of Computers*, pp. 7-20, Sep. 1992.

[3] Cong, J. and Y. Ding, ‘‘An Optimal Technology Mapping Algorithm fo Delay Optimization in Lookup-Table Based FPGA Designs,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 48-53, Nov. 1992.

[4] Ford, L. R. and D. R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, N.J. (1962).

[5] Francis, R. J., J. Rose, and Z. Vranesic, ‘‘Technology Mapping for Delay Optimization of Lookup Table-Based FPGAs,’’ *MCNC Logic Synthesis Workshop*, 1991.

[6] Francis, R. J., J. Rose, and Z. Vranesic, ‘‘Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1991.

[7] Hill, D., ‘‘A CAD System for the Design of Field Programmable Gate Arrays,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 187-192, June 1991.

[8] Karplus, K., ‘‘Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 240-243, June 1991.

[9] Murgai, R., Y. Nishizaki, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Logic Synthesis Algorithms for Programmable Gate Arrays,’’ *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[10] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Improved Logic Synthesis Algorithms for Table Look Up Architectures ,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 564-567, Nov. 1991.

[11] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,’’ *Proc. IEEE Int’l Conf.*

on Computer-Aided Design, pp. 572-575, Nov. 1991.

[12] Sawkar, P. and D. Thomas, ‘‘Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays,’’ *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 83-88, Feb. 1992.

[13] Schlag, M., P. Chan, and J. Kong, ‘‘Empirical Evaluation of Multilevel Logic Minimization Tools for a Field Programmable Gate Array Technology,’’ *Proc. 1st Int’l Workshop on Field Programmable Logic and Applications*, Sept. 1991.

[14] Schlag, M., J. Kong, and P. K. Chan, ‘‘Routability-Driven Technology Mapping for Lookup Table-Based FPGAs,’’ *Proc. 1992 IEEE International Conference on Computer Design*, pp. 86-90, Oct. 1992.

[15] Tarjan, R. E., *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania (1983).

[16] Woo, N.-S., ‘‘A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 248-251, June 1991.

[17] Xilinx, *The Programmable Gate Array Data Book,* Xilinx, San Jose (1992).