Technology Mapping and Architecture Evalution for *k/m*-Macrocell-Based FPGAs

JASON CONG and HUI HUANG
University of California, Los Angeles
and
XIN YUAN
IBM Corporation

In this article, we study the technology mapping problem for a novel field-programmable gate array (FPGA) architecture that is based on k-input single-output programmable logic array- (PLA-) like cells, or, k/m-macrocells. Each cell in this architecture can implement a single output function of up to k inputs and up to m product terms. We develop a very efficient technology mapping algorithm, k_m_flow, for this new type of architecture. The experimental results show that our algorithm can achieve depth-optimality on almost all the testcases in a set of 16 Microelectronics Center of North Carolina (MCNC) benchmarks. Furthermore it is shown that on this set of benchmarks, with only a relatively small number of product terms ($m \leq k+3$), the k/m-macrocell-based FPGAs can achieve the same or similar mapping depth compared with the traditional k-input single-output lookup table- (k-LUT-) based FPGAs. We also investigate the total area and delay of k/m-macrocell-based FPGAs and compare them with those of the commonly used 4-LUT-based FPGAs. The experimental results show that k/m-macrocell-based FPGAs can outperform 4-LUT-based FPGAs in terms of both delay and area after placement and routing by VPR on this set of benchmarks.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*

General Terms: Algorithms, Design

Additional Key Words and Phrases: PLD, FPGA, CPLD, technology mapping

1. INTRODUCTION

The programmable logic devices (PLDs) have been widely used to implement small to medium sized digital circuits. There are two major types of

This work was partially supported by Altera Corp. and Vantis Corp. under the California MICRO program.
X. Yuan was affiliated with the University of California, Los Angeles, Los Angeles, CA, at the time of the research for this article.
Authors' addresses: J. Cong, Department of Computer Science, University of California, Los Angeles, Los Angeles, CA 90095; email: cong@cs.ucla.edu; X. Yuan, System and Technology Group, IBM Corporation, 1000 River St., Mail drop 862F, Essex Junction, VT 05452; email: xinyuan@us.ibm.com.

PLDs—field programmable gate arrays (FPGAs), which usually consist of small programmable logic cells, such as k-input single-output lookup tables (k-LUTs), and complex programmable logic devices (CPLDs), which are based on multiple-input and multiple-output programmable logic array- (PLA-) like logic cells. Both FPGAs and CPLDs have been widely used.

Most commonly used FPGAs are based on k-LUTs. Every k-LUT can implement any function with no more than k inputs. In practice, k is usually small, as the area of a k-LUT grows exponentially with large k. For example, LUTs of four to six inputs are widely used in commercial FPGAs [Altera Corp. 2001; Xilinx Inc. 2001]. On the other hand, CPLDs usually have large basic cells. Each PLA cell can have a large number of inputs (typically between 30 and 40), product terms (typically between 50 to 100), and multiple outputs (16, for example) [Altera Corp. 2000; Cypress Semiconductor Corp. 2000; Lattice Semiconductor Corp. 2000]. As a result, a single PLA cell is able to implement multiple functions with wide inputs.

Rose et al. [1990] showed that among k-LUT cells, the four-input single-output LUT cell yields the smallest FPGA area for a wide range of programming technologies and routing pitches. Most commercially available FPGAs indeed use LUTs with an input size of four or five. Kouloheris and El Gamal [1992] investigated the best granularity for PLA-based CPLDs and found that the total CPLD area is smallest if each basic cell has 8 to 10 inputs, 3 to 4 outputs, and 12 to 13 product terms. The number of product terms is restricted to grow linearly as input size increases [Kouloheris 1993]. In practice, however, most commercially available CPLDs use much larger PLA-like logic cells. Since FPGAs use small programmable cells, they can often offer higher density and capacity, at a price of possibly larger and somewhat unpredictable delay, as the critical path often needs to go through multiple levels of programmable cells connected by the programmable interconnect. On the other hand, CPLDs are usually faster as the programmable cells are much larger, which results in fewer levels of the logic. (The worst-case delay in CPLD also tends to be more predictable as the level of the logic is usually determined by the architecture and can be estimated by the designer.) However, CPLDs usually offer considerably lower logic density. We believe that this is due to two reasons: (a) it is inherently difficult to map logic into multioutput PLA-like programmable cells, as most technology mapping techniques have been developed for single-output logic cells; and (b) the difficulty associated with synthesis/mapping for PLA-based CPLD devices in turn resulted in very limited studies on this topic—the only related works we can find were the DDMap [Kouloheris 1993], a fast heuristic partition method for PLA-based architecture proposed in Hasan et al. [1992], TEMPLA [Anderson and Brown 1998], and PLAmap [Cong et al. 2001]. In the later 1990s, with the introduction of hybrid FPGA families [Kaviani and Brown 1999], a few of mapping algorithms for a hybrid architecture of LUTs and PLAs have been reported [Kaviani 1999; Krishnamoorthy and Tessier 2003]. (In comparison, there have been much more extensive studies on LUT-based FPGAs. A comprehensive survey up to 1997 was reported in Cong and Ding [1996].)

The need to reduce the logic levels (and associated interconnects!) to improve circuit performance, the intention to avoid the mapping problem for

multioutput functions, and the hope to leverage a large amount of research results on synthesis and mapping for LUT-based FPGAs, seem to suggest that we should consider FPGAs with LUTs of much larger numbers of inputs. However, the area of a k-LUT grows expontentially with respect to k. Using k-LUTs with large k may considerably lower chip density. Therefore, we have to explore other alternatives. It has been reported that the functions mapped into large LUTs usually use considerably fewer product terms than the capacity of the lookup table [Kouloheris and Gamal 1992; Kouloheris 1993]. For instance, the utilization of a lookup table cell with K-inputs and N-outputs, which is defined as the ratio of the number of product terms used per cell to the cell capacity, $N \cdot 2^{K-1}$, is less than 0.1 for $K = 7, N = 1$. This leads us to consider an FPGA architecture based on k-input single-output PLA-like logic cells. Each cell can implement a single output function of up to m product terms and up to k inputs. Such a cell is called a *k/m-macrocell* throughout this article. A k/m-macrocell differs from a k-LUT in that each macrocell can implement only a subset of all possible k-input functions. A k/m-macrocell is different from a general PLA-like block used in most CPLDs, as each k/m-macrocell has a single output. If we choose m to be small, k/m-macrocells are much smaller than k-LUTs. Therefore, it is possible to use k/m-macrocells with larger input size in order to get a smaller logic depth and less interconnect without considerably lowering the logic density.

In this article, we develop a very efficient technology mapping algorithm, named *k_m_flow*, for this new type of architecture. The experimental results show that our algorithm can achieve depth-optimality on almost all the testcases in a set of 16 Microelectronics Center of North Carolina (MCNC) benchmarks. Furthermore it is shown that on this set of benchmarks, with only a relatively small number of product terms ($m \leq k+3$), the k/m-macrocell-based FPGAs can achieve the same or a similar mapping depth compared with the traditional k-LUT based FPGAs. We also investigate the total area and delay of k/m-macrocell-based FPGAs and compare them with those of the commonly used 4-LUT-based FPGAs. The experimental results show that k/m-macrocell-based FPGAs can outperform 4-LUT-based FPGAs in terms of both delay and area after placement and routing by versatile place route (VPR) on this set of benchmarks.

The rest of this article is organized as follows. Section 2 formulates the problem. Section 3 introduces a technology mapping algorithm for k/m-macrocell-based FPGAs. Section 4 further investigates the area and delay of k/m-macrocell-based architecture. We draw our conclusions based on experimental results in Section 5. An extended abstract of this work was presented at the FPGA Symposium in 2000 [Cong et al. 2000].

2. DEFINITIONS AND PROBLEM FORMULATION

We first review some terminologies defined in Cong and Ding [1994]. A 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 gate v. We assume the network is k-bounded, that is, for each node v in the network, $|input(v)| \leq k$.[1] A cone at node v, denoted as 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. The notation of $input(C_v)$ is also used to represent the set of distinct nodes outside C_v which supply inputs to the gates in C_v. A maximum cone at v, also the fanin network of v, denoted as N_v, is a cone consisting of v and all of its predecessors. The *level* of a node v is the length of the longest path from any PI node to v. The level of a PI is zero. The *depth* of a network is the largest node level in the network.

A cone C_v is said to be *k-feasible* if and only if $|input(C_v)| \leq k$. Similarly, C_v is said to be *m-packable* if and only if its function has a sum-of-product representation with no more than m product terms. C_v is said to be *k/m-feasible* if it is both *k-feasible* and *m-packable*. Please note that the word *feasible* usually refers to the number of inputs of a macrocell, and *packable* refers to the number of product terms. The only exception is *k/m-feasible*, which is a shortened version of *k-feasible and m-packable*. It is obvious that a *k/m-feasible* cone can be implemented by a k/m-macrocell. A network is called *m-packable* if the function of any node in the network has a sum-of-product representation with no more than m product terms.

Several concepts about cuts in a network will be used in our discussion. Given a network N with a source s and a sink t, a cut (X, X') is a partition of the nodes in the network such that $s \in X, t \in X'$, and no nodes in X' provide input to any node in X. Clearly X' may be considered a cone at t inside network N. Therefore we can apply the previous definitions on k/m-feasibility to cuts. A cut (X, X') is said to be *k-feasible* if and only if X' is a k-feasible cone. The cut is said to be *m-packable* if and only if X' is an m-packable cone. A *k/m-feasible* cut is one that is both k-feasible and m-packable. For every node v and its fanin network N_v, a cut (X, X') in N_v is a partition of the nodes such that all the PI nodes belong to X and v belong to X'. It is clear that every cone rooted at v corresponds to a cut in N_v.

The technology mapping problem for k/m-macrocell-based FPGAs is to cover a given k-bounded m-packable Boolean network with a set of k/m-feasible cones. Note that we allow these cones to overlap, that is, nodes may be duplicated in the mapping solutions. This problem is NP-Hard as the two level minimization problem is NP-Hard.

Throughout the discussion on the technology mapping algorithm (Section 3), unit delay and unit area models are used. That is, variation of interconnection delay and routing area is not directly considered during technology mapping of the original network. Each k/m-macrocell contributes a constant delay independent of the function it implements. Each cell is counted as a unit when we evaluate the area; hence the total area of the mapping solution equals to

[1]Any network can be fully decomposed into a two-bounded network without deteriorating the mapping quality [Cong and Hwang 1996].

the total number of macrocells. Such simplification is reasonable because the layout information is not available yet. For architecture evaluation in Section 4, however, we will use more accurate delay and area models with consideration of the interconnect. We use a well-known FPGA placement and routing tool (VPR [Betz et al. 1999]) to get the total area and critical path delay after layout for comparison. To avoid confusion, we use *depth* and *number of macrocells* in Section 3 to refer to the delay and area under the unit delay and the unit area model.

3. TECHNOLOGY MAPPING FOR *k/m*-MACROCELLS

3.1 Overview

A k/m-macrocell can be regarded as a k-LUT with an additional restriction that it can only implement logic functions with no more than m product terms. Therefore, it is natural to start with the k-LUT mapping problem since it has been intensively studied in the past decade.

Currently, there are three major approaches to LUT-based FPGA mapping: tree-based mapping (e.g., Chortle-crf [Francis et al. 1991a], Chortle-d [Francis et al. 1991b]), flow-based mapping (e.g., FlowMap [Cong and Ding 1994]), and cut-enumeration-based mapping (e.g., PREATOR [Cong et al. 1999]). A more comprehensive survey is available in Cong and Ding [1996]. Tree-based mapping algorithms partition the network into trees and handle each tree separately. Each individual tree can be mapped optimally but a prior tree partitioning often compromises the mapping quality. Usually they are fast heuristic algorithms. Flow-based mapping algorithms are based on the theorem of max-flow-min-cut and the computation of network flow. It can generate a depth optimal mapping solution in polynomial time. However, flow-based algorithms lack flexibility as they find only one or two depth-optimal min-cuts for every node. On the other hand, cut-enumeration-based approaches can find out many, if not all, possible cuts for every node. They offer high flexibility and can achieve optimality with more constraints, but they are considerably slower than tree-based or flow-based methods.

The approach we present here, called *k_m_flow*, is a hybrid of flow-based and cut enumeration-based methods. We try to find a k/m-feasible cut for every node first by flow computation. If that fails, we turn to cut enumeration.

The k_m_flow algorithm consists of two phases—(i) labeling the network and (ii) mapping the network into macrocells. The labeling phase is trying to find a k/m-feasible cut for each node for depth minimization. The mapping phase generates k/m-macrocells in the mapping solution according to the labels and cuts computed in the labeling phase.

3.2 Nonmonotone Properties

For every node v, let N_v be the fanin network consisting of node v and all its predecessors. We assume that there is a given label $label(v)$ associated with each node v. We also define $label^*(v)$, the optimal mapping depth of v, to be the

minimum depth of the k/m-macrocell mapping solution for N_v. The labeling phase for k/m-macrocell mapping is similar to that in the FlowMap algorithm [Cong and Ding 1994]. It finds a k/m-feasible cut for every node v and computes a label for v to minimize the depth of the k/m-macrocell implementing node v in the mapping solution. Ideally, we would like the computed label to be equal to the optimal mapping depth, that is, $label(v) = label^*(v)$ for every node v in the network, as is in the case with the FlowMap algorithm for k-LUT mapping. However, it is more difficult to do so for the k/m-macrocell-based mapping due to the nonmonotone property of the clustering constraints and the non-monotone property of the optimal depths.

3.2.1 *Nonmonotone Property of Clustering Constraints.* The fundamental difficulty of k/m-macrocell-based FPGA mapping is that the constraints on the number of inputs and the number of product terms of a k/m-macrocell are not monotone clustering constraints. That is, if a cone C_v is k-infeasible (or m-unpackable), it does not guarantee that all its supercones (i.e., those cones including C_v) are k-infeasible (or m-unpackable). As a result, a k-infeasible (or m-unpackable) cone C_v could become k-feasible (or m-packable) by including more nodes into it. Note that the LUT mapping problem also has this non-monotone property.

3.2.2 *Nonmonotone Property of Optimal Mapping Depths.* In addition, the optimal k/m-macrocell mapping depth is not monotone either. The optimal mapping depth is monotone if $label^*(v) \geq label^*(u)$ as long as u is an input to v.

An example of the nonmonotone properties is shown in Figure 1. It is a portion of a two-bounded network and we do not show the complete network (for example, output f^1, f^2 can be drivers to other gates that are not shown in Figure 1). Assuming $k = 4$ and $m = 4$, cone C_{f1} is not *4-packable* while a larger cone C_f is both *4-feasible* and *4-packable*. The optimal depth to implement C_{f1} is 2 with 3 4/4-macrocells (as shown in the shaded regions). However, C_f can be implemented with only 1 4/4-macrocell and therefore the optimal mapping depth for C_f is 1, that is, for node f and its input node f_1, $label^*(f) = 1 < 2 = label^*(f_1)$. Note that for the LUT mapping problem, it was shown in Cong and Ding [1994] that the optimal mapping depth is monotone.

3.3 Depth-Optimal Mapping Algorithm for *k/m*-macrocell

Nevertheless, we can have a depth-optimal mapping algorithm for k/m-macrocell. Given a cut (X, X') in N_v, the height of the cut, denoted as $h(X, X')$, is the maximum label in $input(X')$, that is,

$$h(X, X') = \max\{label(v) | v \in input(X')\}$$

It is assumed that every node in $input(X')$ already has a label. A min-height k/m-feasible cut (X, X') in a network is a k/m-feasible cut such that $h(X, X') \leq h(Y, Y')$, where (Y, Y') is any other k/m-feasible cut. Based on the definition of optimal mapping depth of v $label^*(v)$, we have $label^*(v) = h(X, X')+1$, if (X, X')

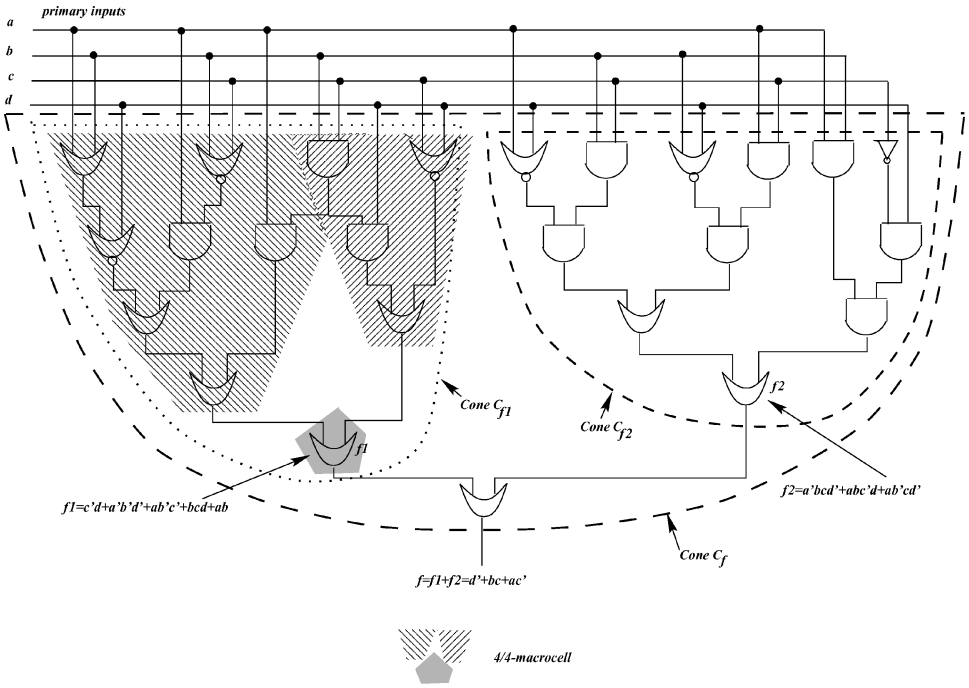

Fig. 1. An example showing the nonmonotone properties of the k/m-macrocell mapping problem.

is the min-height k/m-feasible cut in C_v and $label(u) = label^*(u)$ for any node u other than v in cone C_v. Therefore, following a similar argument as in Cong and Ding [1994], we can conclude that (i) a mapping algorithm can label every node v such that $label(v) = label^*(v)$ if it can find the min-height k/m-feasible cut for every node. And (ii) a mapping algorithm can find the depth optimal mapping solution for k/m-macrocell-based FPGAs if it can find the min-height k/m-feasible cut for every node.

Based on the above observation, a depth optimal mapping algorithm works as follows. It finds the min-height k/m-feasible cut for each node in the topological order from PIs to POs. It then can label each node v such that $label(v) = h(X, X') + 1 = label^*(v)$, where (X, X') is the min-height k/m-feasible cut for v. After labeling the whole network, it can use the min-height k/m-feasible cuts to generate the k/m-macrocells in the mapping solution. The mapping result has the optimal depth.

In order to find the min-height k/m-feasible cut for every node, we can exhaustively enumerate all k-feasible cuts and test if they are m-packable. The enumeration algorithm can then pick the k/m-feasible cut with minimum height for every node. However, such an algorithm is impractical to use due to the high complexity of exhaustive cut enumeration for a large k. In theory, the number of k-cuts of a node in a cone with n nodes can be as large as $O(n^k)$. Since we are interested in large k with values $k = 6$ to 10, we move to develop a more efficient heuristic algorithm.

3.4 The k_m_flow Algorithm—a Heuristic Approach

3.4.1 *Labeling Phase.* The k_m_flow algorithm also computes a label for each node from PIs to POs in topological order. At the beginning, every PI node will receive a label of 0. Then for every node v, suppose *mlevel* is the largest label among v's fanins, it is assumed that $label(v) \geq mlevel$. In order to test if we can set $label(v)$ to *mlevel*, we collapse each node u in N_v with $label(u) = mlevel$ into node v to form an induced network N_v' and test if we can find a k/m-feasible cut in N_v'. Obviously we assume that node labels will increase monotonically.

Based on the max-flow-min-cut theorem, we can easily find two min-cuts in N_v': the max-volume-min-cut (X, X'), which is a min-cut with the largest $|X'|$, and the min-volume-min-cut (Y, Y'), which is a min-cut with the smallest $|Y'|$. Please note both (X, X') and (Y, Y') are min-cuts, which implies that $|input(X')| = |input(Y')| \leq |input(Z')|$ where (Z, Z') is any other cut in N_v'. Also, note that max-volume-min-cut and min-volume-min-cut are unique and $Y' \subseteq X'$. The max-volume-min-cut and min-volume-min-cut can be found in $O(ke)$ time, where e is the number of edges and k is the value of the maximum flow. There are three cases on whether the max-volume and min-volume min-cuts are k/m-feasible. m-packable is checked after calling *Espresso* [Brayton et al. 1984] on the functions in X' and Y' collapsed at v.

Case 1. Neither max-volume nor min-volume min-cut is k-feasible.

Because any k/m-feasible cut must be k-feasible too, this condition implies that no k/m-feasible cut exists in N_v'. In this case, node v can be simply labeled as $mlevel + 1$.

Case 2. Either max-volume or min-volume min-cut is k/m-feasible.

Suppose (X, X') is the k/m-feasible min-cut (either the max-volume or min-volume one). We can create a k/m-macrocell for node v, denoted as $map\_node(v)$, to implement the function of X'. The depth of $map\_node(v)$ in the mapping solution cannot be larger than *mlevel*. Therefore, we assign $label(v) = mlevel$.

Case 3. Both max-volume and min-volume min-cut are k-feasible but not m-packable.

Note that this condition does not guarantee that there is no k/m-feasible cut existing in N_v'. Therefore, we try to do a local cut enumeration in hope of finding a k/m-feasible cut.

To search for a k/m-feasible cut, perhaps the most natural way is to do a local cut enumeration within the cone defined by the max-volume-k-feasible-cut. However, unlike a max-volume-min-cut, the max-volume-k-feasible-cut may not be unique. Moreover, there is no good algorithm to find the max-volume-k-feasible-cut. Furthermore, it is intuitive to think that if the max-volume-min-cut (X, X') is not m-packable, a cut outside or across it may not likely be k/m-feasible, since it will have more fanins, tending to require more product terms in its sum-of-product representation. Therefore, in order to search for a k/m-feasible cut under Case 3, we only enumerate cuts inside X' to see if they are k/m-feasible.

To do a local cut enumeration in a cone C_v, first we mark all inputs to C_v as *pseudo-PIs* and then go through all nodes inside C_v in topological order from *pseudo-PIs* and enumerate all k-feasible cuts for every node v by the following equation, where x and y are the fanins of v[2]:

$$Cut(v) = (\{(C_x - x, x)\} \cup Cut(x)) \bigotimes_k (\{(C_y - y, y)\} \cup Cut(y))$$ [Cong et al. 1999].

$Cut(x)$ is the set of k-feasible cuts for node x. The notation $(C_x - x, x)$ refers to the cut that cuts off the single node x. $\bigotimes_k$ is a merging operator defined on two cut sets; $S_1 \bigotimes_k S_2$ means merge every cut cut_1 in S_1 with every cut cut_2 in S_2 and only keep the k-feasible cuts in the result.

After the enumeration process, we check $Cut(v)$ to see if there is an m-packable cut. If there exists a k/m-feasible cut, node v can be labeled as $mlevel$; otherwise, it will be labeled as $mlevel + 1$.

The pseudocode for the labeling phase is shown in Figure 2. For all the nodes, we record the k/m-feasible cuts which correspond to their labels.

3.4.2 *Mapping Phase.* The second phase of our algorithm is to generate the k/m-macrocells in the mapping solution. For every node v, basd on the k/m-feasible cut (X, X') found in labeling phase, we can create a k/m-macrocell $map\_node(v)$ for v to implement the function of X' and $input(map\_node(v)) = input(X')$. The mapping phase is very similar to the mapping phase in FlowMap [Cong and Ding 1994]. The detailed description is shown in Figure 2.

3.4.3 *Properties of the k_m_flow Algorithm.* We can prove the following properties for the algorithm discussed above:

(1) If a node v is labeled as $label(v)$, it can then be implemented with a depth no more than $label(v)$. That is, $label(v)$ is the upper bound estimation of the depth of v in the mapping solution.
(2) If Case 3 never happens when mapping a specific circuit, then the mapping solution is delay optimal. Indeed, it is just the same as k-LUT mapping.
(3) For any certain circuit, if the optimal depth for k-LUT-based mapping is d_1, the optimal depth for k/m-macrocell-based mapping is d_2 and the depth of k_m_flow mapping result is d_3, then $d_1 \leq d_2 \leq d_3$.

3.5 Area Enhancement

After obtaining a k/m-macrocell mapping solution, we want to further reduce the number of k/m-macrocells in the mapping solution without increasing its depth.

[2]Although in problem formulation the given network is k-bounded, our algorithm is implemented for a two-bounded given network as it is proved in Cong and Hwang [1996] that any network can be fully decomposed into a two-bounded network without deteriorating the mapping quality. Therefore the cut enumeration equation provided here is for a two-bounded network. Of course it can be generalized to a k-bounded network at the cost of higher computation complexity.

```
Algorithm k_m_flow
BEGIN-k_m_flow
/* phase 1: labeling network */
for each PI node v do
  label(v) := 0;
for each node v
{
  mlevel := max { label(n) — n is a fanin of v };
  collapse all nodes with label = mlevel into v in N_v;
  find the max-vol-min-cut (X, X') for v;
  if (no cut exists) /* N_v contains a single node */
  or if (X, X') is not k-feasible
    label(v) := mlevel + 1;
  else
  {
    if (X' is m-packable)
      label(v) := mlevel;
    else/* check if min-vol-min-cut is m-packable */
    {
      find the min-vol-min-cut (Y, Y') for v;
      if (Y' is m-packable)
        label(v) := mlevel;
      else
      {
        mark all the inputs to cone X' as "pseudo-PIs";
        do a local cut enumeration starting from "pseudo-PIs";
        if find a k/m-feasible cut (Z, Z') through enumeration
          label(v) := mlevel;
        else label(v) := mlevel + 1;
      }     }
  }
}
/* phase 2: generating k/m-macrocells */
L := list of PO nodes;
while L contains non-PI nodes do{
  remove a non-PI node v from L;
  let (X, X') be the cut generated by the labeling phase for node v;
  if X' contains only one node v {
    generate a k/m-macrocell map_node to implement the function of the single node v; }
  else{
    generate a k/m-macrocell map_node to implement the
    function of X' such that input(map_node) = input(X'); }
    L := (L − {v}) ∪ input(X'); }
END-k_m_flow
```

Fig. 2. Pseudocode of k_m_flow algorithm.

For every k/m-macrocell v, we try to pack as many of its predecessors with it as possible into a single k/m-macrocell. Clearly we need to guarantee the condition that the new k/m-macrocell is still k/m-feasible. In order to do so, we try to combine v with any of its fanins and then check if they can be packed into one k/m-macrocell. After v and one of its fanins have been successfully packed together, a new node v' will be formed to replace v in the mapping solution. It could be that some of the fanins of v' ($input(v') = input(v) \cup input(fanin)$) can still be packed together with v'. Therefore, the above greedy packing process will be repeated until no more nodes can be packed. The detailed packing algorithm, k_m_pack, is shown in Figure 3. On average, the total number of macrocells in the mapping solution may be reduced by a factor of 6% after the above packing process.

```
Algorithm k_m_pack
BEGIN-k_m_pack
modified:=true;
while modified==true do {
  modified:=false;
  for each fanin of v do{
    if |input(fanin) ∪ input(v)| ≤ k then {
      v' :=collapse fanin into v;
      if v' has less than m product terms
        then {
          replace v with v';
          modified:=true;
          break; } } } }
END-k_m_pack
```

Fig. 3. Pseudocode of k_m_pack algorithm.

Table I. Description of 16 Benchmark Circuits

Circuit	duke2	C499	frg2	rot	ex4p	apex6	x3	apex5
#node	382	398	695	696	699	714	768	827
#PI	22	41	143	135	128	135	135	117
#PO	29	32	139	107	28	99	99	88
level	10	19	12	22	11	16	12	11
Circuit	i8	C2670	s5378	mm30a	pair	C3540	alu4	des
#node	917	1299	1317	1500	1556	2097	2347	3026
#PI	133	233	196	124	173	50	14	256
#PO	81	64	210	121	137	22	8	245
level	12	26	24	108	18	42	14	15

3.6 Experiment Result

Our algorithm, k_m_flow, has been implemented in C language within the Berkeley SIS and UCLA RASP [Cong et al. 1996] framework. We chose a set of 16 MCNC benchmarks to test k_m_flow on a Sun Ultra II workstation with 512M memory. Table I shows the size of the 16 benchmark circuits before mapping (all are two-bounded networks).

In order to find out the optimal mapping depth for each benchmark and compare it with the k_m_flow mapping solution, we implemented an algorithm called *k_m_enumerate*. The k_m_enumerate algorithm can find the depth-optimal mapping solution by performing an exhaustive cut enumeration in the entire network, as proposed in Section 3.3. We would like to point out that k_m_enumerate is impractical to use for a large k. We use it only to collect data to analyze the optimality of k_m_flow algorithm.

In Table II, we list the mapping depth generated by k_m_flow and k_m_enumerate under a different k and m. The data is in the form of x/y, where x is the depth of mapping solution generated by k_m_flow and y is the optimal mapping depth obtained by k_m_enumerate under the specified k and m. A question mark ? means the optimal depth is still unknown because of the extremely long runtime and large memory requirement of k_m_enumerate for a large k. From Table II, we can see that although k_m_flow cannot guarantee delay optimality in theory, in practice it is almost always able to find out the depth-optimal mapping solution.

Table II. Experimental Results Show that the k_m_flow Algorithm Can Achieve Depth-Optimal Mapping in Practice

	$k=6$				$k=8$				$k=10$			
Circuit	$m=6$	$m=7$	$m=8$	$m=9$	$m=8$	$m=9$	$m=10$	$m=11$	$m=10$	$m=11$	$m=12$	$m=13$
duke2	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
C499	5/5	5/5	4/4	4/4	4/?	4/?	4/?	4/?	3/3	3/3	3/3	3/3
frg2	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
rot	6/6	6/6	6/6	6/6	5/5	5/5	5/5	5/5	4/4	4/4	4/4	4/4
ex4p	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
apex6	4/4	4/4	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3
x3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	2/2	2/2	2/2	2/2
apex5	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
i8	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
C2670	7/7	7/7	6/6	6/6	6/6	6/6	6/6	6/6	4/4	4/4	4/4	4/4
s5378	7/7	7/7	7/7	7/7	6/6	5/5	5/5	5/5	5/?	5/?	4/4	4/4
mm30a	21/21	21/21	21/21	21/21	20/20	20/20	20/20	17/?	19/?	15/?	15/?	15/?
pair	5/5	5/5	5/5	5/5	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3
C3540	11/11	11/11	10/10	10/10	9/?	9/?	8/8	8/8	8/?	8/?	7/?	7/?
alu4	6/6	6/6	6/6	6/6	5/5	5/5	5/5	5/5	4/4	4/4	4/4	4/4
des	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/?	3/?	3/?	3/?

Table III. Total Mapping Depth of k/m-Macrocell Versus k-LUT on 16 MCNC Benchmarks

	Total depth for k/m-macrocell by k_m_flow				Total depth for k-LUT by FlowMap
	$m=k$	$m=k+1$	$m=k+2$	$m=k+3$	
$k=6$	99	99	96	96	95
$k=7$	90	88	87	87	81
$k=8$	84	83	82	78	75
$k=9$	74	73	71	68	65
$k=10$	73	70	67	67	62

Table IV. Total Number of k/m-Macrocells Versus Total Number of k-LUTs on 16 MCNC Benchmarks

	Total # of k/m-macrocells by k_m_flow				Total # of k-LUTs by FlowMap
	$m=k$	$m=k+1$	$m=k+2$	$m=k+3$	
$k=6$	7872	7728	7607	7554	7419
$k=7$	6742	6574	6528	6496	6349
$k=8$	6045	5971	5908	5909	5646
$k=9$	5628	5526	5535	5513	5275
$k=10$	5148	5128	5105	5075	4789

We also compare the k/m-macrocell mapping solution generated by k_m_flow with k-LUT mapping solution generated by FlowMap. Table III shows the total mapping depth of k/m-macrocell vs. k-LUT on the 16 benchmarks. Table IV shows the total number of macrocells vs. the total number of k-LUTs on 16 benchmarks. FlowMap is a depth-optimal k-LUT mapping algorithm based on flow computation. Since k/m-macrocells can be regarded as k-LUTs with additional m-product-term constraints, the optimal depth of a k-LUT mapping solution is the lower bound of the optimal depth of a k/m-macrocell mapping solution.

Table V. Quick Success Rate of Flow Computation in k_m_flow on 16 Benchmarks

	$m = k$	$m = k + 1$	$m = k + 2$	$m = k + 3$
$k = 6$	99%	99%	99.6%	99.8%
$k = 7$	97%	98%	99%	99%
$k = 8$	97%	98%	98%	99%
$k = 9$	96%	96%	97%	97%
$k = 10$	96%	96%	97%	97%

We can see that the mapping results of k_m_flow (both depth and number of macrocells) are close to the k-LUT mapping results when $m = k + 3$. It implies that the k/m-macrocell is almost equivalent to a k-LUT if m is slightly larger than k. This observation is consistent with the results reported by Kouloheris [1993 (p. 75)] where the author claimed the number of product terms needed to implement the function of a k-LUT grows almost linearly with k. Increasing the flexibility of the macrocell by allowing more product terms to be implemented will not significantly improve the performance.

The k_m_flow algorithm is a hybrid of flow computation and cut enumeration. The complexity of flow computation is $O(n^2)$ [Cong and Ding 1994]. The complexity of cut enumeration can be estimated as $O(npq^2)$,[3] where n is the number of nodes in the network, and p is *max*{*size of max-volume-k-feasible cone*}, and q is the max number of k-feasible cuts inside any cone. The equation assumes we will do a cut enumeration for every node in a cone of size p. The conservative estimation on the complexity to enumerate all k-feasible cuts for a single node is $O(q^2)$ because the input network is two-bounded. In order to get a polynomial complexity, we can bound the number of cuts enumerated by a constant number. Therefore, the complexity of k_m_flow is bounded by the complexity of flow computation $O(n^2)$.

The percentage of nodes where the max-volume or min-volume k-feasible cut returned by flow computation is m-packable is called *quick success rate*. *Quick success rate* is a characteristic of individual network and may differ from network to network. Fortunately, the quick success rate is on average 98% for $k = 6, 7, \ldots, 10$ and $m = k, k + 1, \ldots, k + 3$ of the 16 benchmarks. Detailed data is shown in Table V. It implies that k_m_flow has a complexity close to $O(n^2)$ in practice. It is understandable that due to this high success rate, k_m_flow will use almost the same cuts as FlowMap finds to create macrocells, resulting in the similar mapping depth.

From our observations on the range $k = 6$ to 10, the cones in which cut enumeration is performed are usually small, with less than 50 nodes inside. An exhaustive cut enumeration on a small network with no more than 50 nodes usually runs very fast. Therefore, the k_m_flow algorithm shall be an efficient algorithm to generate the k/m-macrocell mapping solution for medium k. For a large k, the cone may be large and even the local cut enumeration may take a long time to finish. Table VI shows the total CPU time (in seconds) needed to generate all the mapping solutions for 16 benchmarks. Since the *quick success*

[3]Assume cut enumertation is done on a two-bounded network.

Table VI. Total CPU Runtime of k_m_flow Algorithm on 16 Benchmarks

	$m = k$	$m = k + 1$	$m = k + 2$	$m = k + 3$
$k = 6$	105.9	101.7	96.8	96.1
$k = 7$	149.1	106.0	106.4	105.3
$k = 8$	119.0	119.0	118.7	119.3
$k = 9$	140.0	137.8	138.4	138.7
$k = 10$	210.5	207.8	206.3	207.6

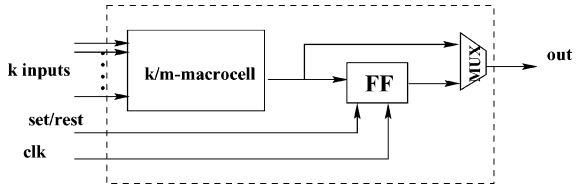

Fig. 4. k/m logic block.

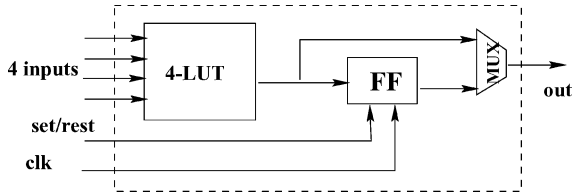

Fig. 5. 4-LUT logic block.

rate is usually very high, in practice, skipping local enumeration may cause little impact on the mapping quality and may save the runtime.

4. INVESTIGATION OF *k*/*m*-MACROCELL-BASED ARCHITECTURES

In Section 3 we used the unit area and the unit delay model to evaluate the quality of our k/m-macrocell mapping algorithm. In order to collect more accurate delay and area information to draw an architectural study conclusion, we use VPR [Betz et al. 1999], an FPGA placement and routing tool developed at the University of Toronto, to do placement and routing for our k/m-macrocell-based architecture and compare this architecture with the traditional 4-LUT-based architecture in terms of total area and critical path delay.

4.1 Area and Delay Models

Figure 4 shows the schematic diagram of the logic block used in our k/m-macrocell-based architecture (we call it the k/m the *logic block*). Figure 5 shows the logic block used in 4-LUT-based architecture [Betz et al. 1999] (we call it the 4-LUT *logic block*). Since the area of a logic block is greatly affected by the number of transistors in the basic cell, we use the ratio between the number of transistors in k/m-macrocell and that in 4-LUT to estimate the ratio of the area of the two logic blocks (i.e., k/m logic block vs. 4-LUT logic block). For the

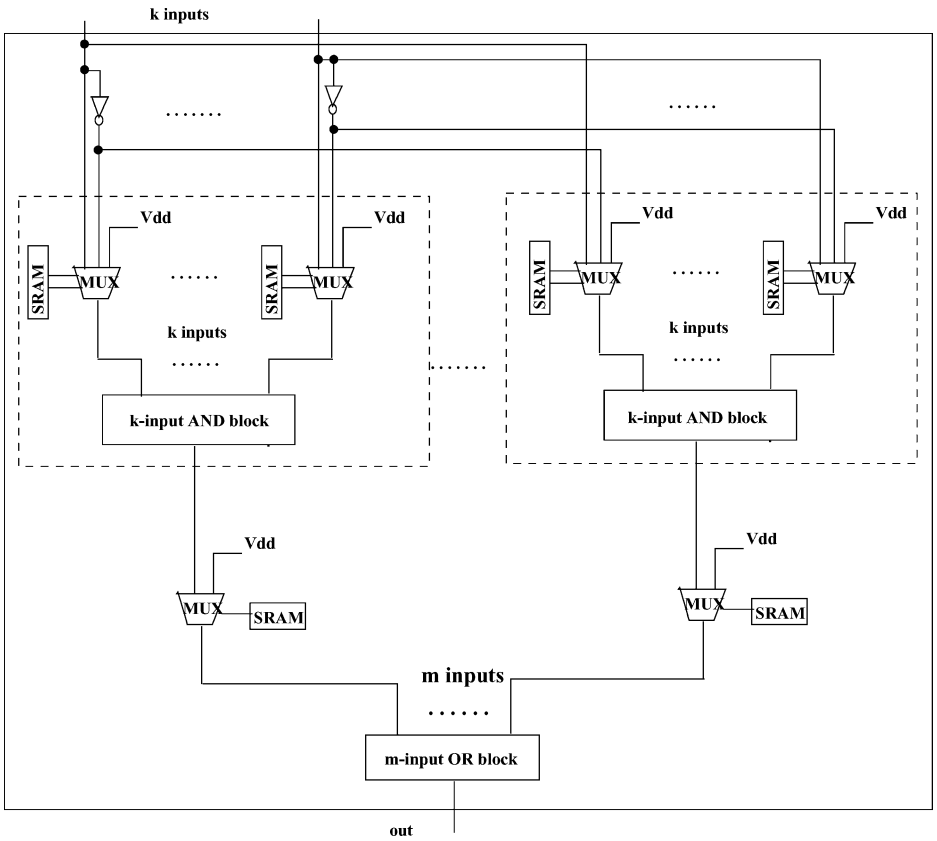

Fig. 6. Schematic diagram of k/m-macrocell.

logic block delay, we assume that it is proportional to the number of transistors on the longest path of the basic cell.

4.1.1 *Detailed Description of k/m-Macrocell Implementation.* Our k/m-macrocell consists of k inverters, km 3:1 MUXs, km 2-bit-SRAMs, m 1-bit-SRAM, km 2:1 MUXs, m k-input AND blocks, and one m-input OR block shown in Figure 6. The 3:1 MUX in the product-term block is used to choose x, x', and 1 for each input, so its logic function is $f = xab + yab' + a' = yb' + xb + a' = p' + a' = (pa)'$, $p = yb + xb'$. Therefore it can be implemented by the circuit shown in Figure 7(a), costing eight transistors. (We assume that SRAM can provide both data and its complementary, i.e., b' is provided.) There are three transistors on the longest path a signal would pass (one pass transistor and two n-transistors). The 2:1 MUX can be implemented by the circuit shown in Figure 7(b). It costs four transistors and there is one pass transistor in the longest path. The 1-bit-SRAM needs six transistors (see Betz et al. [1999], pp. 208, for the schematic diagram) and 2-bit-SRAM needs 12 transistors. The x-input AND block and y-input OR block can be implemented by multiple-level NAND gates and NOR gates. Here we adopt two-level gates to implement them

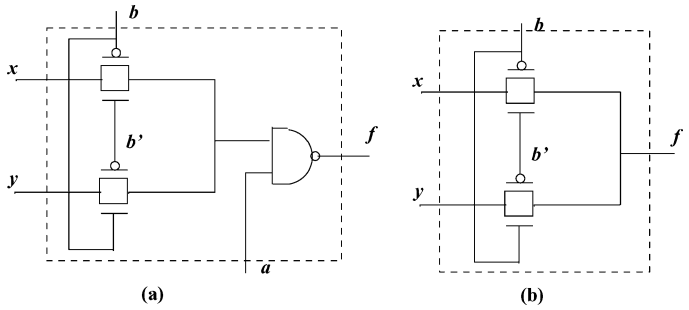

Fig. 7. 3:1 MUX and 2:1 MUX.

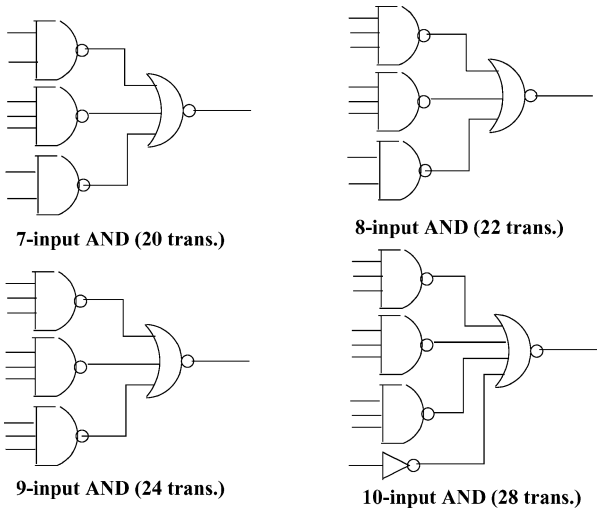

Fig. 8. x-input AND blocks.

as shown in Figures 8 and 9. The number of transistors used in the block is listed in the figures. For a z-input NAND (NOR) gate, there are z n-transistors (p-transistors) on the longest path; thus the number of transistors on the longest path of each x-input AND block and y-input OR block are listed as follows:

7-input AND block: $3+3=6$;	10-input OR block: $3+3=6$;
8-input AND block: $3+3=6$;	11-input OR block: $4+3=7$;
9-input AND block: $3+3=6$;	12-input OR block: $4+3=7$;
10-input AND block: $3+4=7$;	13-input OR block: $4+4=8$.

4.1.2 *Transistor Count for k/m-Macrocell Area Estimation.* The total transistor number of a k/m-macrocell is the sum of following items: k inverters ($2k$ transistors), km 2-bit-SRAM ($12km$ transistors), km 3:1 MUX ($8km$ transistors), m 2:1 MUX ($4m$ transistors), m 1-bit-SRAM ($6m$ transistors), m k-input AND block ($m\times$ (#transistors_per_k-input AND)), and 1 m-input OR block (#transistors_per_m-input OR), that is, $2k+20km+10m+m\times$ (#transistors_per_k-input AND)+#transistors_per_m-input OR.

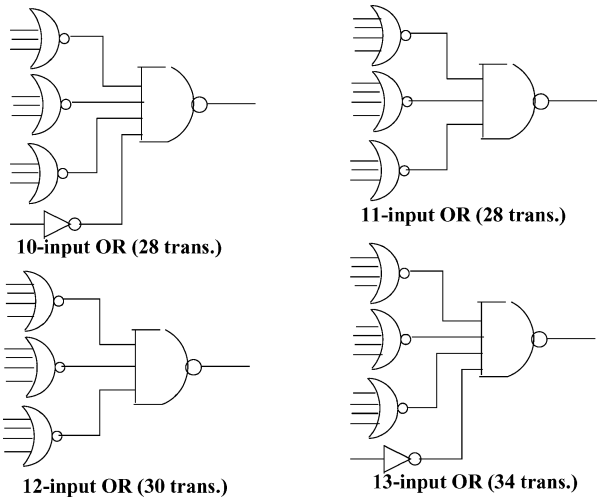

Fig. 9. x-input OR blocks.

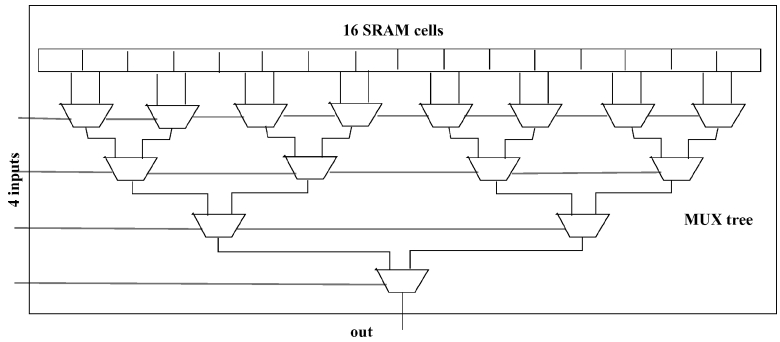

Fig. 10. Schematic diagram of 4-LUT-cell.

4.1.3 *Transistor Count for k/m-Macrocell Delay Estimation.* The longest path a signal would pass in the k/m-macrocell consists of an inverter, a 3:1 MUX, a k-input NAND block, a 2:1 MUX, and an m-input NAND block. Therefore the number of transistors in the longest path of k/m-macrocell is

$$\begin{array}{llll} k=7, & m=10: & 1+3+6+1+6=17; \\ k=8, & m=11: & 1+3+6+1+7=18; \\ k=9, & m=12: & 1+3+6+1+7=18; \\ k=10, & m=13: & 1+3+7+1+8=20. \end{array}$$

4.1.4 *Transistor Count for 4-LUT Area and Delay Estimation.* The 4-LUT cell we compare to is shown in Figure 10. Because we use four transistors to implement the 2:1 MUX, there should be extra four inverters for four inputs to provide their complementaries which are not drawn in this figure. This 4-LUT actually consists of 16 1-bit SRAMs, 4 inverters, and 15 2:1 MUXs. Therefore the total number of transistors used in this 4-LUT cell is $16\times6+15\times4+4\times2=164$.

Table VII. k/m Logic Block Area Estimation

	$k=7, m=10$	$k=8, m=11$	$k=9, m=12$	$k=10, m=13$
#Trans. of k/m-macrocell	1742	2156	2616	3148
#Trans. of k/m-macrocell vs. #Trans. of 4-LUT cell	10.6	13.2	16.0	19.2
Assumed delay ratio of k/m-macrocell vs. 4-LUT cell	3.4	3.6	3.6	4.0

The longest path a signal would pass is from an inverter to each level of 2:1 MUX; thus there are five transistors in the longest path.

Therefore, based on the above transistor counting and the estimation model for the logic block, we have the area and delay ratio of k/m-macrocell versus 4-LUT cell listed in the Table VII.

4.2 Experimental Setting of VPR

We used VPR version 4.22 to do placement and routing for the mapping solution of the k/m-macrocell-based architecture and that of the 4-LUT-based architecture. The authors of VPR extensively studied area/delay tradeoff for 4-LUT-based architecture. They proposed a detailed 4-LUT-based FPGA architecture under TSMC's 0.35-μm, 3.3-V process [Betz et al. 1999]. The 4-LUT logic block they proposed is identical to what Figure 5 shows. Two corresponding routing architecture files are attached with VPR package. In terms of interconnect configuration, one is simple and thus offers less area but a larger delay (we call it *RA1*), and the other is more complicated and offers a smaller delay but at a cost of larger area (we call it *RA2*). RA1 uses only buffer switched wires of length 1, while RA2 uses 50% buffer switched wires of length 4 and 50% pass transistor switched wires of length 4. As we only wanted to compare two architectures rather than getting an absolute result, it was reasonable to scale the area and delay parameters in the architecture file of 4-LUT-based architecture according to the area and delay estimation ratio between k/m logic block and 4-LUT logic block as discussed in Section 4.1 in order to derive the routing architecture file for k/m-macrocell-based architecture, that is, we compared our k/m-macrocell-based architecture with their 4-LUT-based architecture by only changing the parameters related to the area and delay of the logic block in the routing architecture files.

VPR reports routing area in the number of min-width transistors and the delay of critical path in seconds. We added up the logic block area to the routing area and got the total area of each mapping solution.

4.3 Experimental Results

We compared k/m-macrocell-based architecture with 4-LUT-based architecture by running VPR on their mapping solutions for the 16 MCNC benchmarks under the experimental settings mentioned above. The k/m-macrocell mapping solutions were obtained by running k_m_flow algorithm and then performing k_m_pack to further reduce the number of macrocells. The 4-LUT mapping solutions were obtained by running FlowMap followed by greedy-pack.

Table VIII. Normalized Area and Delay of k/m-Macrocell-Based Architecture on the 16 Benchmarks

Results under routing architecture RA1				
	$k=7, m=10$	$k=8, m=11$	$k=9, m=12$	$k=10, m=13$
$A_{k/m\_tile}/A_{4-LUT\_tile}$	1.56	1.70	2.16	2.15
Total area	1.15	1.20	1.16	1.27
Crit. path delay	0.79	0.75	0.68	0.67
Routing area percentage	59%	56%	52%	48%
Results under routing architecture RA2				
	$k=7, m=10$	$k=8, m=11$	$k=9, m=12$	$k=10, m=13$
$A_{k/m\_tile}/A_{4-LUT\_tile}$	1.46	1.59	1.47	2.03
Total area	0.85	0.86	0.81	0.83
Crit. path delay	0.86	0.81	0.74	0.73
Routing area percentage	71%	67%	63%	58%

Based on the estimation ratio discussed in Section 4.1, the average total area and critical path delay comparison between these two architectures are shown in Table VIII. A_{k/m_tile} refers to the area of the max-basic tile in k/m-macrocell-based architecture, which is the sum of the area of k/m logic block and the max-routing area per tile among the 16 benchmarks. A_{4-LUT_tile} refers to the area of the max-basic tile in 4-LUT-based architecture, which is the sum of the area of 4-LUT logic block and the max-routing area per tile among the 16 benchmarks. The area and delay of k/m-macrocell-based architecture were normalized with 4-LUT's =1. We compared the total area and critical path delay of the two architectures under the two routing architectures RA1 and RA2 mentioned in Section 4.2. We also reported the routing area percentage in k/m-macrocell-based architecture on the 16 benchmarks after routing.

For LUT-based FPGA, when k is small, most of the area is devoted to routing. With the increase of k, routing area decreases, but the area increase of logic blocks could be more than the decrease of routing area. Because the area of k/m-macrocell blocks does not grow exponentially as k-LUT does, the logic block area is much less than k-LUT-based architecture's; thus the total area decreases. Since the logic depth and routing area decrease, the total critical path delay decreases.

From the results, we can see that k/m-macrocell-based architecture can have 14–31% delay reduction with more area in RA1 and less area in RA2 when compared to 4–LUT-based architecture.

5. CONCLUSIONS AND FUTURE WORKS

We have studied a novel FPGA architecture based on k/m-macrocells through this article and proposed a k/m-macrocell technology mapping algorithm, named *k_m_flow*. In a set of 16 MCNC benchmarks, it produces optimal mapping depths in most test cases. Using this algorithm and this set of benchmarks, we have shown that k/m-macrocell-based FPGAs are similar to k-LUT-based FPGAs in terms of the mapping depths and the number of macrocells being used. The high quick success rate (Table V) suggests that k/m-macrocell can provide similar flexibility to a lookup table while each k/m-macrocell is much

smaller than a k-LUT. We have analyzed the delays and areas of k/m-macrocell-based FPGAs using VPR on this set of benchmarks. We compared the results with those of traditional 4-LUT-based FPGAs. Our comparison has shown that k/m-macrocell-based FPGAs can significantly outperform 4-LUT-based FPGAs in terms of delay.

Future works includes two directions. (i) We can use a packing process to pack mapped k/m-macrocells into multiple-output macrocells in order to support CPLDs mapping. (ii) As the area and delay models we used for k/m-macrocells in Section 4 are primitive, a more accurate and detailed model needs to be studied.

ACKNOWLEDGMENTS

The authors would like to thank Professor David Lewis and Dr. Vaughn Betz for their helpful comments and discussion.

REFERENCES

ALTERA CORP. 2000. *MAX 700B, Programmable Logic Device Family*. San Jose, CA. Web site: www.altera.com.

ALTERA CORP. 2001. *APEX II, Programmable Logic Device Family*. San Jose, CA. Web site: www.altera.com.

ANDERSON, J. H. AND BROWN, S. D. 1998. Technology mapping for large complex PLDs. In *Proceedings of the Design Automation Conference*. 698–703.

BETZ, V., ROSE, J., AND MARQUARDT, A. 1999. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic, Norwell, MA.

BRAYTON, R., SANGIOVANNI-VINCENTELLI, A., HACHTEL, G., AND MCMULLIN, C. 1984. *Logic Minimization Algorithms for Digital Circuits*. Kluwer, Norwell, MA.

CONG, J., CHEN, D., ERCEGOVAC, M., AND HUANG, Z. 2001. Performance-driven mapping for CPLA architecture. In *Proceedings of the ACM International Symposium on FPGA*. 39–47.

CONG, J. AND DING, Y. 1994. An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 13*, 1, 1–12.

CONG, J. AND DING, Y. 1996. Combinational logic synthesis for LUT based field programmable gate arrays. *ACM Trans. Des. Automat. Electron. Syst. 1*, 2, 145–204.

CONG, J., HUANG, H., AND YUAN, X. 2000. Technology mapping for k/m-macrocell based FPGAs. In *Proceedings of the ACM International Symposium on FPGA*. 51–59.

CONG, J. AND HWANG, Y. 1996. Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA. In *Proceedings of the Design Automation Conf*. 726–729.

CONG, J., PECK, J., AND DING, Y. 1996. RASP: A general logic synthesis system for SRAM-based FPGAs. In *Proceedings of the ACM International Symposium on FPGA*. 137–143.

CONG, J., WU, C., AND DING, Y. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the ACM International Symposium on FPGA*. 29–35.

CYPRESS SEMICONDUCTOR CORP. 2000. *The Cypress Data Book*. San Jose, CA. Web site: www.cypress.com.

FRANCIS, R. J., ROSE, J., AND VRANESIC, Z. G. 1991a. Chortle-CRF: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of the Design Automation Conference*. 227–233.

FRANCIS, R. J., ROSE, J., AND VRANESIC, Z. G. 1991b. Technology mapping of lookup table-based FPGAs for performance. In *Proceedings of the International Conference on Computer Aided Design*. 568–571.

HASAN, Z., HARRISON, D., AND CIESIELSKI, M. 1992. A fast partition method for PLA-based FPGAs. *IEEE Des. Test Comput. 9*, 4, 34–39.

KAVIANI, A. AND BROWN, S. 1999. The hybrid field-programmable architecture. *IEEE Des. Test Comput. 16*, 74–83.

KAVIANI, A. S. 1999. Novel architecture and synthesis methods for high capacity field programmable devices. Ph.D. dissertation. University of Toronto, Toronto, Ont., Canada.

KOULOHERIS, J. L. 1993. Empirical study of the effect of cell granularity on FPGA density and performance. Ph.D. dissertation. Stanford University, Stanford, CA.

KOULOHERIS, J. L. AND GAMAL, A. E. 1992. PLA-based FPGA area versus cell granularity. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 4.3/1–4.

KRISHNAMOORTHY, S. AND TESSIER, R. May 2003. Technology mapping algorithms for hybrid FPGAs containing lookup tables and PLAs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 22*, 5, 545–559.

LATTICE SEMICONDUCTOR CORP. 2000. *The Lattice Data Book*. Hillsboro, OR. Web site: `www.latticesemi.com`.

ROSE, J., FRANCIS, R. J., LEWIS, D., AND CHOW, P. Ct. 1990. Architecture of field programmable gate arrays: The effect of logic block functionality on area efficiency. *IEEE J. Solid State Circ. 25*, 5, 1217–1225.

XILINX INC. 2001. *Virtex-II Field-Programmable Gate Arrays*. San Jose, CA. Web site: `www.xilinx.com`.

Received April 2003; revised August 2003; accepted January 2004