Technology Mapping for *k/m*-macrocell Based FPGAs

Jason Cong, Hui Huang, and Xin Yuan

Department of Computer Science, University of California, Los Angeles, CA 90095

{cong, huanghui, yuanxin}@cs.ucla.edu

Abstract

In this paper, we study the technology mapping problem for a novel FPGA architecture that is based on *k*-input single-output 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 experiment results show our algorithm can achieve depth-optimality in practically all cases. Furthermore it is shown that the *k/m*-macrocell based FPGAs are practically equivalent to the traditional *k*-LUT based FPGAs with only a relatively small number of product terms ($m \leq k+3$). We also investigate the total area and delay of *k/m*-macrocell based FPGAs on various benchmarks to compare it with commonly used 4-LUT based FPGAs. The experimental result shows *k/m*-macrocell based FPGAs can outperform 4-LUT based FPGAs in terms of both delay and area after placement and routing by VPR.

1. Introduction

The Field Programmable Devices (FPDs) have been widely used for implementation of small to medium size digital circuits. There are two major types of FPDs —— Field Programmable Gate Arrays (FPGAs) which usually consist of small programmable logic cells, such as *k*-input single-output lookup tables, and Complex Programmable Logic Devices (CPLDs) which are based on multiple-input and multiple-output PLA-like logic cells. Both of FPGAs and CPLDs have been widely used.

Most commonly used FPGAs are based on *k*-input single-output lookup tables (*k*-LUTs). Every *k*-LUT can implement any function with no more than *k* inputs. In practice, *k* is usually small, for example, 4-LUTs are widely used in commercial FPGAs, as the area of a *k*-LUT grows exponentially with large *k*. On the other hand, PLA based devices usually have large basic cells. Each cell can have a large number of inputs (typically between 30-40). Also, a PLA cell normally has multiple outputs (16, for example). As a result, a single PLA cell is able to implement multiple functions with wide inputs. Unlike lookup table, each cell can only implement functions with no more than *m* product terms.

Rose *et al.* [16] showed 4-input, single-output LUT cell yields the smallest FPGA area of any *k*-LUT cell for a wide range of programming technologies and routing pitches. Most commercially available FPGAs indeed use LUTs of input size of 4 or 5. Kouloheris and El Gamal [15] investigated the best granularity for PLA-based CPLDs and found that the total CPLD area is smallest if each basic cell has 8~10 inputs, 3~4 outputs, and 12~13 product terms. The number of product terms is restricted to grow linearly as input size increases [14]. In practice, however, most commercially available CPLDs use much larger PLA-like logic cells. Since FPGAs use small programmable cells, they often offer high density and high capacity, at a price of possibly larger and somewhat unpredictable delays, as a critical path may need to go through multiple levels of programmable cells connected by 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 in worst-case delay path 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 multi-output PLA-like programmable cells, as most technology mapping techniques are 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 DDMap [14] in early 90's, a fast heuristic partition method for PLA-based architecture proposed in [10], and TEMPLA [14] in 1998. (In comparison, there are much more extensive studies on LUT-based FPGAs, which will be briefly summarized in Section 3.1.)

The need to reduce the logic levels (and associated interconnects!) to improve circuit performance, the intention to avoid the mapping problem for multi-output functions, and the hope to leverage 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 number of inputs. However, the area a *k*-LUTs grows expontentially with respect to *k*. Using *k*-LUTs with large *k* may considerably lowers chip density. Therefore, we have to explore other alternatives. We noticed that the functions mapped into large LUTs usually use considerably fewer product terms than the lookup table capacity [15]. 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 paper. A *k/m*-macrocells differ 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 CPLD devices, too, as each *k/m*-macrocell has 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 use smaller logic depth and less interconnect without lowering the chip capacity considerably.

In this paper, we develop a very efficient technology mapping algorithm, named k_m_flow, for this new type of architecture. The experiment results show our algorithm can achieve depth optimality in practically all cases. Furthermore we show that the *k/m*-macrocell based FPGAs are practically equivalent to the traditional *k*-LUT based FPGAs with only a relatively small number of product terms ($m \le k+3$). We also investigate the total area and delay of *k/m*-macrocell based FPGAs on various benchmarks to compare it with commonly used 4-LUT based FPGAs. The result shows *k/m*-macrocell based FPGAs can outperform 4-LUT based FPGAs in terms of both delay and area after placement and routing by VPR.

The rest of this paper 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 and discuss the future work in Section 5.

Throughout this paper, the letter *k* is used to denote the input size of a macrocell, or the input size of a LUT in FPGA. The letter *m* is used to represent the maximum number of product terms that one macrocell can implement.

2. Definitions and Problem Formulation

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 *2-bounded*, that is, for each node v in the network, $|\, input(v)\,| \le 2$. Any network can be fully decomposed into *2-bounded* network without deteriorating the mapping quality [6].

A *cone* at 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.

A cone C_v is said to be *k-feasible* if and only if $|\, input(C_v)\,| \le k$. 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 to 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.

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 as 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 both a *k-feasible* cut and an *m-packable* one. 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 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 2-bounded Boolean network with k/m-feasible cones.* Note that we allow these cones to overlap, that is, it is *not* a duplication-free mapping problem here. Due to the relationship between cuts and cones, the technology mapping problem for *k/m*-macrocell based FPGAs can be converted to finding *k/m-feasible* cuts for every node. The *k/m-feasible* cones that cover the whole network can be derived from *k/m-feasible* cuts.

We use two delay and area models to evaluate the quality of mapping solution. 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 the total number of macrocells. Such simplification is reasonable because the layout information is not available yet. For architecture comparison in Section 4, however, we will use more accurate delay and area models with consideration of the interconnect, as we use a well-known FPGA placement and routing tool (VPR [2]) 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 unit delay and unit area model.

3. Technology Mapping for *k/m*-macrocells

3.1 Overview

A *k/m*-macrocell can be considered 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 few years.

Currently, there are three major approaches to LUT-based FPGA mapping, tree-based mapping (e.g. Chortle-crf, Chortle-d [8]&[9]), flow-based mapping (e.g. FlowMap [3]) and cut-enumeration-based mapping (4]). See [5] for a more comprehensive survey. 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. They are usually fast heuristic algorithms. Flow-based algorithm is based on the theorem of max-flow-min-cut and the computation of network flow. It can generate depth optimal mapping solution in polynomial time. However, flow-based algorithms lack of flexibility as they find only one or two depth optimal min-cuts for every node. On the other hand, cut-enumeration-based approaches will 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 computation and cut enumeration. We try to find a *k/m-feasible* cut for every node first by flow computation. If failed, we turn to cut enumeration.

3.2 Algorithm

The k_m_flow algorithm consists of two phases ---- labeling the network and mapping the network into macrocells. The labeling phase is trying to finds a *k/m-feasible* cut for every node for depth minimization. The mapping phase generates *k/m*-macrocells in the mapping solution according to the labels and cuts obtained in the labeling phase.

3.2.1 Labeling Phase

For every node *v*, let N_v be the fanin network consisting of node *v* and all its predecessors. 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. It finds a *k/m-feasible* cut for every node *v* and compute a *label* for *v* to minimize 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 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 non-monotone properties of the clustering constraints and the optimal labels as presented in the next subsection.

3.2.1.1 Non-monotone Clustering Constraints and Optimal Mapping Depths

The fundamental difficulty of *k/m*-macrocell based FPGA mapping is that the constraint on the number of inputs and the number of product terms of a *k/m*-macrocell are not *monotone* clustering constraints. That is, a cone C_v is *k-**in**feasible* (or, *m-**un**packable*) does not guarantee that all its super-cones (i.e. those cones root at *v* and include C_v) are *k-**in**feasible* (or, *m-**un**packable*). As a result, a *k-**in**feasible* (or, *m-**un**packable*) cone C_v could become *k-feasible* (or, *m-packable*) by including more nodes into it. (See Figure 1)

In addition, the optimal *k/m*-macrocell mapping depth is not monotone either. The optimal mapping depth is monotone if *label**(*v*)≥*label**(*u*) as long as *u* is an input to *v*. Figure 1 shows that the optimal mapping depth is not monotone. In Figure 1, *label**(*f*) = 1 < 2 = *label**(f_1). Note that for LUT mapping problem, it was shown in [3] that the optimal mapping depth is monotone.

3.2.1.2 Depth Optimal Mapping Algorithm

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'*), i.e.

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

(It is assumed that every node in *input*(*X'*) 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'*)≤*h*(*Y, Y'*), where (*Y, Y'*) is any other *k/m-feasible* cut.

Lemma *label**(*v*)=*h* (*X, X'*)+1, if (*X, X'*) 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.

Lemma 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.

Theorem 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 this result, a depth optimal mapping algorithm works as follows. It finds the *min-height k/m-feasible* cut for each node in 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-*

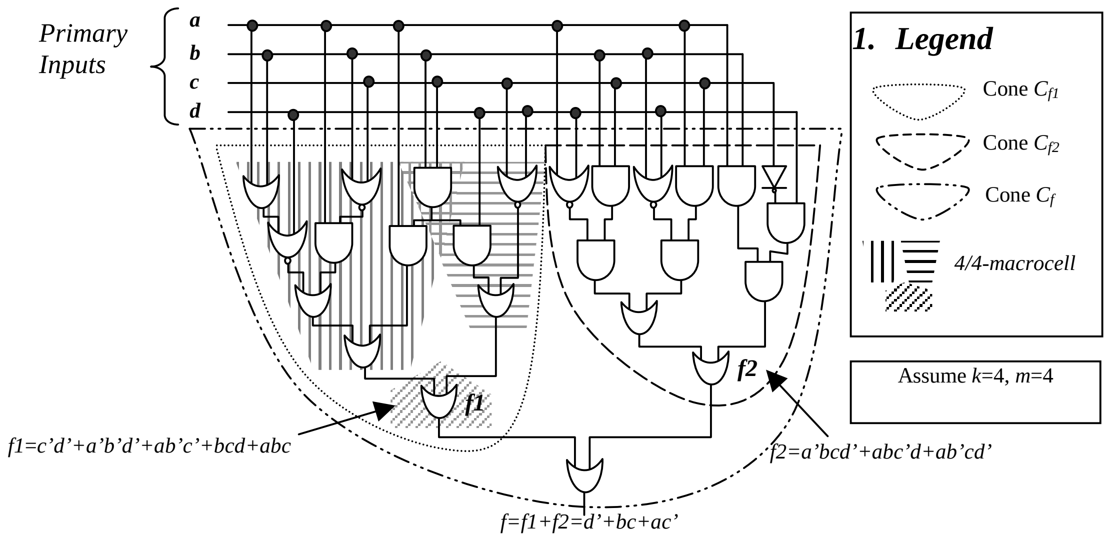

Figure 1 Constraint on the number of product terms and optimal depth of macrocell is not monotone --- Assuming *k*=4 and *m*=4, cone C_{f1} is not *4-packable* while a larger cone C_f is both *4-feasible* and *4-packable*. The optimal depth to implement C_{f1} is 2 with 3 *4/4-macrocell* (as shown in the shaded area). However, C_f can be implemented with only 1 *4/4-macrocell* and therefore the optimal mapping depth for C_f is 1.

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 then can 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 large *k*. In theory, the number of *k*-cuts in a cone of node *s* is in the order of $O(n^k)$. Since we are interested in large *k* with values *k*=6~10, we move to develop more efficient heuristic algorithms.

3.2.1.3 The k_m_flow Algorithm ——A Heuristic Approach

First, we assume that node labels will increase monotonely. 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*)≥*mlevel*. In order to test if we can set *label*(*v*) to *mlevel*, we collapse all the 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.

Based on the max-volume-min-cut theorem, we can find two min-cuts in N'_v easily: 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'*)| ≤ |*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'*⊆*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.

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'*. 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 *max-volume-min-cut*, *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 (*Y, Y'*) is not *m-packable*, a cut outside or across it may not likely to be *k/m-feasible*, since it will have more fanins and tends 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 *Y'* 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*:

$$Cut(v) = (\{(C_x - x, x)\} \cup Cut(x)) \otimes_k (\{(C_y - y, y)\} \cup Cut(y))[4]$$

Cut(*x*) is the set of *k-feasible* cuts for node *x*. Notation "$(C_x - x, x)$" refers to the cut that cuts off the single node *x*. "$\otimes_k$" is a merging operator defined on two cut sets; "$S_1 \otimes_k S_2$" is to 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 pseudo code for labeling phase is shown in Figure 2.

3.2.2 Mapping Phase

The second phase of our algorithm is to generate the *k/m*-macrocells in the mapping solution. For every node *v*, if in the labeling phase we found a *k/m-feasible* cut (*X, X'*), then 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'*). If no *k/m-feasible* cut was found during the labeling phase (may occur in case 1 and 3), we can create a *k/m*-macrocell to implement the function of single node *v*. After generating macrocells for every node, we need to remove redundant cells that do not fan out to any other macrocell. Using a list to keep track of "visible" nodes and only generate macrocells for "visible" nodes can optimize this procedure. The detailed algorithm is shown is Figure 2.

3.2.3 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*), then it can 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 $d_1 \leq d_2 \leq d_3$.

3.3 Area Enhancement

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

For every *k/m*-macrocell *v*, we try to pack as many 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

```
algorithm 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 (Y,Y') for v;
  if (no cut exists) /* $N_v$ contains a single node */
  or if (Y,Y') is not k-feasible
     label(v) := mlevel+1;
  else{
    if (Y' is m-packable)
       label(v) := mlevel;
    else{
    /* check if min-vol-min-cut is m-packable */
     find the min-vol-min-cut (X,X') in for v;
     if (X' is m-packable)
       label(v) := mlevel;
     else{
       mark all the inputs to cone Y' as "pseudo-PIs";
       do a local cut enumeration starting from "pseudo-PIs";
       if (found 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(v');  }
```

Figure 2 Pseudo code of k_m_flow

```
algorithm k_m_pack;
modified:=true;
while (modified) 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;    }  }  }  }
```

Figure 3 Pseudo code of k_m_pack

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*)∪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.

3.4 Experiment Result

Our algorithm, k_m_flow, has been implemented in C language within the Berkeley SIS and UCLA RASP [7] framework. We chose a set of 16 MCNC benchmarks to test k_m_flow on a Sun Ultra II workstation with 512M memory. Table 1 shows the size of the 16 benchmark circuits before mapping (all are 2-bounded networks).

circuit	frg2	c2670	apex6	i8
#node	695	1300	714	917
#PI	143	233	135	133
#PO	139	64	99	81
level	12	26	16	12
circuit	rot	x3	ex4p	mm30a
#node	696	768	699	1500
#PI	135	135	128	124
#PO	107	99	28	121
level	22	12	11	108
circuit	s5378	apex5	pair	alu4
#node	1322	828	1556	2347
#PI	196	117	173	14
#PO	210	88	137	8
level	24	11	18	14
circuit	des	c499	c3540	duke2
#node	3026	398	2097	382
#PI	256	41	50	22
#PO	245	32	22	29
level	15	19	42	10

Table 1 Description of 16 benchmark circuits

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 exhaustive cut enumeration on the entire network, as proposed in Section 3.2.1.2. We would like to point out that k_m_enumerate is impractical to use for large *k*. We use it only to collect data to analyze the depth optimality of the result of k_m_flow.

In Table 2, we list the mapping depth generated by k_m_flow and k_m_enumerate under 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; "*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 unknown yet because of the extremely long runtime and large memory requirement of k_m_enumerate

circuit	k=6				k=8				k=10			
	m=6	*m=7*	*m=8*	*m=9*	*m=8*	*m=9*	*m=10*	*m=11*	*m=10*	*m=11*	*m=12*	*m=13*
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
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
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
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
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
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
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
mm30a	21/21	21/21	21/21	21/21	20/20	20/20	17/17	15/15	19/?	15/?	15/?	15/?
s5378	7/7	7/7	7/7	7/7	6/6	5/5	5/5	5/5	5/?	5/?	4/4	4/4
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
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
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/?
c499	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/?
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

Table 2 Experiment results show that the k_m_flow algorithm achieve depth optimal mapping in practice

	k_m_flow				FlowMap
	m=k	*m=k+1*	*m=k+2*	*m=k+3*	
k=6	99	99	96	96	95
k=7	90	86	88	87	81
k=8	84	84	82	78	75
k=9	74	73	71	68	65
k=10	73	69	67	67	62

Table 3 *Total* mapping depth of *k/m*-macrocell vs. *k*-LUT on 16 MCNC benchmarks

	k_m_flow				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

Table 4 *Total* number of *k/m*-macrocells vs. *total* number of *k*-LUTs on 16 MCNC benchmarks

for large *k*. From Table 2, 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.

We also compare the *k/m*-macrocell mapping solution generated by k_m_flow with *k*-LUT mapping solution generated by FlowMap. Table 3 shows the total mapping depth of *k/m*-macrocell vs. *k*-LUT on 16 benchmarks. Table 4 shows the total number of macrocells vs. the total number of *k*-LUTs on 16 benchmarks. FlowMap is the depth optimal *k*-LUT mapping algorithm based on flow computation. Since *k/m*-macrocells can be considered as *k*-LUT with additional *m*-product-term constraints, the optimal depth of *k*-LUT mapping solution is the lower bound of the optimal depth of *k/m*-macrocell mapping solution.

From the above tables, the mapping results of k_m_flow (both depth and number of macrocells) are close to the *k*-LUT mapping results when *m=k+2* for smaller *k* or *m=k+3* for larger *k*. 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 [1] (pp 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.

k_m_flow is a hybrid of flow computation and cut enumeration. So the complexity of k_m_flow is somewhere between the complexity of flow computation $O(n^2)$ (FlowMap [3]) and the complexity of cut enumeration, depending on the frequency of performing cut enumeration and the size of the cone to perform cut enumeration.

The complexity of cut enumeration can be estimated as $O(npq^2)$, where n is the number of nodes in the network, p is *max{size of max-volume-k-feasible cone}*, 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 2-bounded.

The percentage of node where the max-volume or min-volume min-cut returned by flow computation being *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 ,...,10 and *m=k, k*+1,...,*k*+3 of the 16 benchmarks. Detailed data is shown in Table 5. 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 to create macrocells, resulting in the similar mapping depth.

From our observations on the range *k=6~10*, the cones need to perform cut enumeration are usually small, with less than 50

	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%

Table 5 *Quick success rate* of flow computation

nodes inside. An exhaustive cut enumeration on a small network with no more 50 nodes usually runs very fast. Therefore, k_m_flow algorithm shall be an efficient algorithm to generate the *k/m*-macrocell mapping solution for medium *k*. For large *k*, the cone may be large and even the local cut enumeration may take a long time to finish. Table 6 shows the total CPU time (in seconds**)** needed to generate all the mapping solutions for 16 benchmarks. Since the *quick success rate* is usually very high, in practice, skipping local enumeration will cause little impact on the mapping quality but will save the runtime.

	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

Table 6 CPU Runtime

4. Investigation of *k/m*-macrocell Based Architectures

In section 3 we use unit area and 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 architecture study conclusion, we use VPR[2], an FPGA placement and routing tool developed in 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.

Figure 4 shows the schematic diagram of the logic block used in our *k/m*-macrocell-based architecture (we call it *k/m* logic block), and Figure 5 shows the logic block used in 4-LUT-based architecture (we call it 4-LUT logic block) [2]. Since the area of a logic block is greatly effected by the total number of I/O pins of the block[1] and the number of transistors in the block, we use the geometric mean of the ratio of number of I/O pins and the ratio of number of transistors to estimate the ratio of the area of two logic blocks (i.e., *k/m* logic block vs. 4-LUT logic block). The total number of pins of a *k/m* logic block is *k*+3 and the total number of pins of a 4-LUT logic block is 7, as shown in Figure 4 and 5. Our *k/m*-macrocell consists of *k* inverters, *km* 3:1 MUXs, *km* 2-bit-SRAMs, *m* 1-bit-SRAMs, *m* 2:1 MUXs , *m k*-input AND blocks and one *m*-input OR block as Figure 6 shows, while the 4-LUT consists of 16 1-bit-SRAMs and 15 2:1 MUXs as Figure 7 shows [11]. Every 1-bit-SRAM can be implemented by 6 transistors. A 3:1 MUX needs 8 transistors and a 2:1 MUX needs 4 transistors. *k*-input AND blocks and *m*-input OR block are implemented by two-level NAND gates and NOR gates. The total number of transistors used in a 4-LUT cell is 164 and the numbers of transistors of *k/m*-macrocell are shown in Table 7. Therefore we can estimate that the area of a *k/m* logic block is 4~6 times large as the area of a 4-LUT logic block for *k=7~10, m=10~13*. As we have not done any simulation on the *k/m* logic block, we do not have the accurate delay for the *k/m* logic block. A rough estimation on the delay of *k/m* logic block is that it is 2 times slower than a 4-LUT for *k* between 7 and 10 based on the observation that the number of transistors in the longest path a signal would pass in the *k/m* logic block is about 3 times of that in a 4 -LUT logic block.

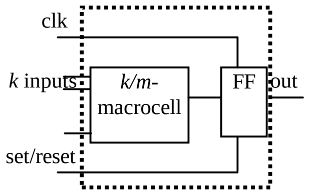

Figure 4 *k/m* logic block

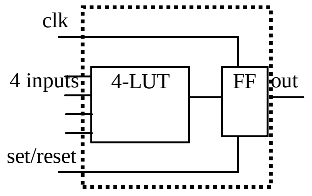

Figure 5 4-LUT logic block

The total area is the sum of routing area and logic block area; the critical path delay is the sum of interconnect delay and logic block delay. The routing area and interconnect delay is estimated by VPR.

	k=7 *m*=9	*k*=8 *m*=10	*k*=9 *m*=12	*k*=10 *m*=13
#trans. of *k/m*-macrocell	1568	1964	2616	3148
#trans. of *k/m*-macrocell vs. #trans. of 4-LUT cell	9.6	12.0	16.0	19.2
#pins of *k/m* logic block vs. #pins of 4-LUT logic block	10/7	11/7	12/7	13/7
assumed area ratio of *k/m* logic block vs. 4-LUT logic block = $\sqrt{\text{pin\_ratio} \times \text{logic\_area\_ratio}}$	3.7	4.3	5.2	6.0

Table 7 *k/m* logic block area estimation

1.

[1] Private communication with Prof. J. Rose of University of Toronto

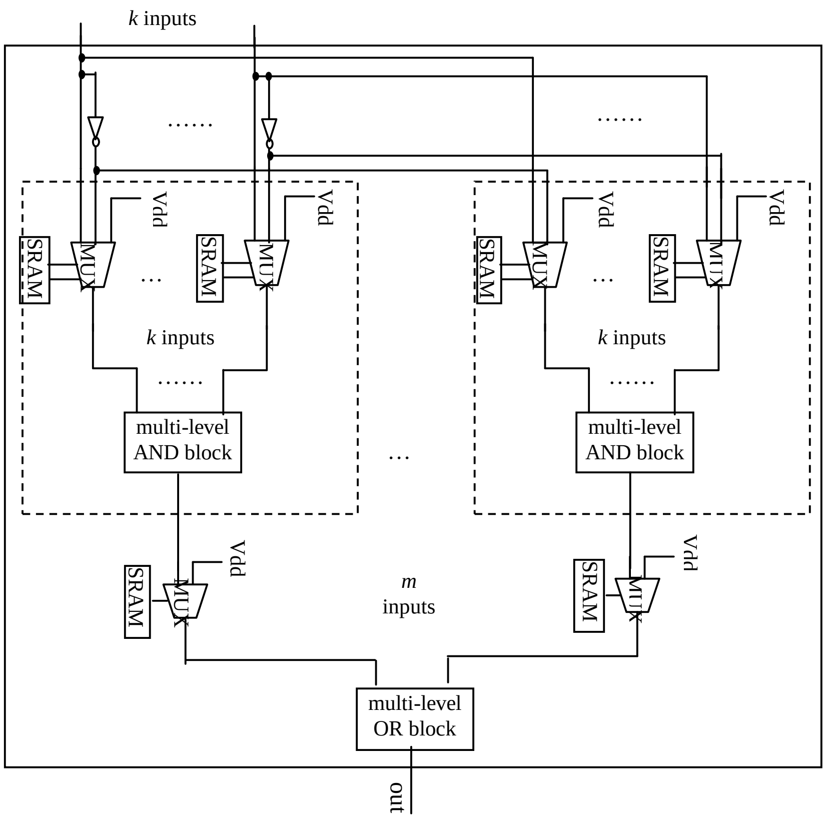

Figure 6 *k/m*-macrocell

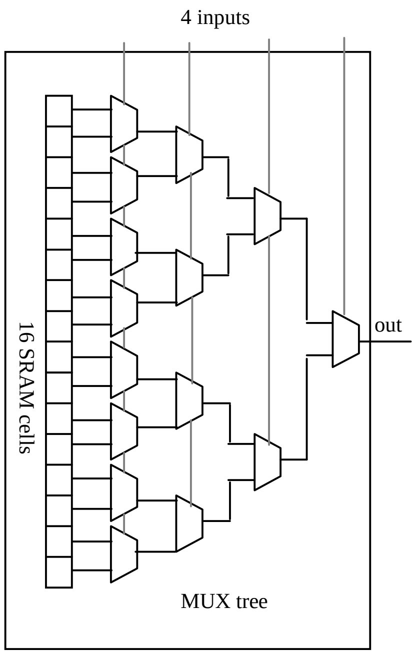

Figure 7 4-LUT cell
(This figure is taken from [11] pp10)

4.2 Experimental Setting of VPR

The authors of VPR did lots of studies on area/delay trade-off for 4-LUT and cluster-based logic block. They proposed a detailed 4-LUT-based FPGA architecture under TSMC's 0.35 μm, 3.3V process [2]. The 4-LUT logic block they proposed is exactly the same as what Figure 5 shows. We compare our *k/m*-macrocell-based architecture with their 4-LUT-based architecture by only changing the area and delay of logic block in the architecture file. VPR reports routing area in number of min-width transistors and the delay of critical path in seconds. We add up the logic block area to the routing area and get the total area of each mapping solution.

4.3 Experimental Result

We compared *k/m*-macrocell based architecture with 4-LUT-based architecture by running VPR on the two kinds of mapping solutions of the 16 MCNC benchmarks under the experimental settings mentioned above. The *k/m*-macrocell mapping solutions are 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 are obtained by running FlowMap followed by greedy-pack. Average area and delay are showed in Table 8. $A_{k/m}$ refers to the area of one *k/m* logic block and $A_{4\text{-}LUT}$ refers to the area of one 4-LUT logic block. $D_{k/m}$ refers to the delay of one *k/m* logic block and $D_{4\text{-}LUT}$ refers to the delay of one 4-LUT logic block. The area and delay of *k/m*-macrocell-based architecture are normalized with 4-LUT's =1.

From the above results, we can see that *k/m*-macrocell architecture can implement the same function as 4-LUT based architecture with less 25% area and 37% delay. 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. Since the area of *k/m*-macrocell blocks does not grow exponentially as *k*-LUT does, the total area decreases. Since the logic depth and routing area decrease, the total delay decreases.

	k=7 *m=9*	*k=8* *m=10*	*k=9* *m=12*	*k=10* *m=13*
$A_{k/m} / A_{4\text{-}LUT}$	4	4	5	6
$D_{k/m} / D_{4\text{-}LUT}$	3	3	3	3
area	0.78	0.78	0.75	0.78
delay	0.75	0.70	0.63	0.63

Table 8 Normalized area and delay of *k/m*-macrocell based architecture

5. Conclusions and Future Work

We have studied a novel FPGA architecture based on *k/m*-macrocells through this paper and proposed a *k/m*-macrocell technology mapping algorithm, named k_m_flow, which produces optimal mapping depths in most cases. Using this algorithm, we showed that *k/m*-macrocell based FPGAs are similar to *k*-LUT based FPGAs in terms of the mapping depths and number of macrocells being used. The high *quick success rate* (Table 5) suggests that *k/m*-macrocell can provide similar flexibility as lookup table while each *k/m*-macrocell is much smaller than *k*-LUT. We have analyzed the delays and areas of *k/m*-macrocell based FPGAs using VPR. We compared the results with those of traditional 4-LUT based FPGAs. Our comparison showed convincingly that *k/m*-macrocell based FPGAs can significantly outperform 4-LUT based FPGAs both in delay and area when the delay of a *k/m* logic block is no more than 3 times and area is no more than 6 times worse than those of a 4-LUT logic block.

We are extending this work in several directions. First, we plan to perform detailed layout of a *k/m*-macrocell (including necessary transistor sizing) to collect more accurate area and delay information and compare those with a *k*-LUT based logic cell. Such more accurate area and delay models will be fed into VPR for more accurate area and delay results. Second, we plan to compare an FPGA architecture with clusters of *k/m*-macrocell and compare it with an architecture with clusters of *k*-LUTs, as most modern FPGAs use LUT clusters for density and performance enhancement.

Acknowledgement

The authors would like to thank Prof. David Lewis and Dr. Vaughn Betz from Right Track Corp. for their helpful comments and discussion.

This work is partially supported by Altera Corp. and Vantis Corp. under the California MICRO program.

Reference:

[1] J. H. Anderson, S. D. Brown, Technology Mapping for Large Complex PLDs, Proc. 35th ACM/IEEE Design Automation Conference 1998, pp 698-703.

[2] V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, 1999.

[3] J. Cong and Y. Ding, FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs, IEEE Trans. on Computer-Aided Design, Jan. 1994, Vol. 13, No. 1, pp. 1-12.

[4] J. Cong, C. Wu and Y, Ding, Cut Ranking and Pruning: Enabling A General And Efficient FPGA Mapping Solution, Proc. ACM Int'l. Symp. on FPGA, Monterey, CA, Feb. 1999, pp. 29-35.

[5] J. Cong and Y. Ding, Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays, ACM Trans. on Design Automation of Electronic Systems, Vol. 1, No. 2, April, 1996, pp. 145-204.

[6] J. Cong and Y. Hwang, Structural Gate Decomposition for Depth-Optimal Technology Mapping in LUT-based FPGA, Proc. ACM/IEEE the 33rd Design Automation Conference, 1996, pp 726-729.

[7] J. Cong, J. Peck and Y. Ding, RASP: A General Logic Synthesis System for SRAM-based FPGAs, Proc. ACM 4th Int'l Symp. on FPGA, 1996, pp137-143.

[8] R. J. Francis, J. Rose and Z. G. Vranesic, Technology Mapping of Lookup Table-Based FPGAs for Performance, Proc. IEEE International Conference on Computer-Aided Design, 1991, pp 568-571.

[9] R. J. Francis, J. Rose and Z. G. Vranesic, Chortle-crf: Fast technology mapping for lookup table-based FPGAs, Proc. ACM/IEEE Design Automation Conference 1991, pp 227-233.

[10] Z. Hasan, D. Harrison and M. Ciesielski, A Fast Partition Method for PLA-based FPGAs, IEEE Design and Test of Computers, Dec. 1992, pp34-39.

[11] A. S. Kaviani, Novel Architecture and Synthesis Methods for High Capacity Field Programmable Devices, Ph.D. Thesis, University of Toronto, 1999.

[12] J. L. Kouloheris and A. El Gamal, PLA-based FPGA Area versus Cell Granularity, IEEE Custom Integrated Circuits Conference, 1992.

[13] J. L. Kouloheris and A. El Gamal, FPGA Performance vs. Cell Granularity," Proc. Custom Integrated Circuits Conference, 1991, pp. 6.2.1-4.

[14] J. L. Kouloheris, Empirical study of the effect of cell granularity on FPGA density and performance, PhD Thesis, Stanford University 1993.

[15] E. L. Lawler, K.N. Levitt and J. Turner, Module Clustering to Minimize Delay in Digital Networks, IEEE Transactions on Computers, Vol. C-18(1) pp. 47-57, January 1969.

[16] J. Rose, R. J. Francis, D. Lewis, and P. Chow, Architecture of Field Programmable Gate Arrays: The effect of Logic Block Functionality on Area Efficiency," IEEE Journal of Solid State Circuits, Vol. 25, No. 5, October 1990, pp. 1217-1225.