Partially-Dependent Functional Decomposition with Applications in FPGA Synthesis and Mapping

Jason Cong and Yean-Yow Hwang

Department of Computer Science
University of California, Los Angeles

Abstract

In this paper, we give a necessary and sufficient condition for the existence of partially-dependent functional decomposition and develop new algorithms to compute such decompositions. We apply our method to the synthesis and mapping for Xilinx XC4000 FPGA's which contain non-uniform sizes of LUT's in its architecture. We develop a new mapping algorithm named PDDMAP which uses CLB's to cover nodes on critical paths for depth minimization and uses LUT's to cover non-critical nodes for area minimization. On average, PDDMAP is able to reduce the depth by 13% with only 1% of increase in area comparing to the results by FlowMap followed by a CLB generation procedure match_4k. We also develop a post-mapping procedure named PDDSYN which resynthesizes mapping solutions to reduce the mapping area. On average, PDDSYN is able to improve PDDMAP mapping solutions by 5% in depth and 7% in CLB count, and achieves 8% smaller depth and 11% fewer CLB count comparing to FlowSyn followed by match_4k.

1. Introduction

Given a function $f(X)=f(x_1,x_2,\cdots,x_n)$ and a bound set $B=\{x_1,\ldots,x_b\}$, the disjoint decomposition of $f(X)$ under the bound set B is a function $g(y_1(B), y_2(B), \cdots, y_t(B), x_{b+1}, \cdots, x_n)=f(X)$ such that $t<b$. When $t=1$, it is called a *simple* disjoint decomposition. Functions $y_1(B)$ to $y_t(B)$ are the *encoding* functions and function g is the *base* function of the decomposition. The necessary and sufficient conditions for a disjoint decomposition to exist were given by Ashenhurst [1], Curtis [10], and Roth and Karp [17] in different forms. Recently, efficient decomposition algorithms [3, 14, 21] have been developed based on the functional representation of reduced ordered binary decision diagrams (ROBDD) [2]. A comprehensive survey of the results on functional decomposition was given in [7].

In general, the encoding functions may depend on all variables in the bound set. A *partially-dependent decomposition* produces encoding functions which are independent of some variables in the bound set. Methods to minimize the support of encoding functions during decomposition were recently proposed in [12, 15] and produce good results. However, we observe some limitations. [12] encodes compatible classes carefully to eliminate variables from the bound set. But because it is a strict encoding, [12] does not guarantee to find a partially-dependent decomposition when there is one. The approach in [15] builds ROBDD's to represent implicitly all assignable encoding functions which are independent of one variable. This approach is capable of searching for partially-dependent encoding functions in a large solution space. However, different encoding functions may partition the compatible classes in the same way. It is unnecessary to compute all encoding functions in order to obtain a decomposition. When there are only few compatible classes, this approach could be inefficient. Besides, neither [12, 15] provides a necessary and sufficient condition to characterize the existence of a partially-dependent decomposition.

In this paper, we shall present a necessary and sufficient condition for testing the existence of a partially-dependent decomposition. Based on this condition, we develop a partition-based method to compute partially-dependent decomposition. We apply our method to technology mapping and synthesis for the Xilinx XC4000 FPGA's. We develop a new mapping algorithm which uses CLB's to cover nodes on critical paths for depth minimization and uses LUT's to cover non-critical nodes for area minimization. We also develop a post-mapping procedure which resynthesizes mapping solutions to reduce area. Both applications are based on the partially-dependent functional decomposition.

The remainder of this paper is organized as follows. Section 2 formulates the problem, introduces theorems and gives decomposition algorithms. Section 3 presents a novel approach for Xilinx XC4000 FPGA technology mapping and a resynthesis procedure for area minimization. Experimental results are presented in Section 4. Section 5 discusses future works.

2. Partially-Dependent Functional Decomposition

2.1. Problem Formulation and Theory

Let $X=\{x_1,x_2,\ldots,x_n\}$ be a set of Boolean variables and $f(X)=f(x_1,x_2,\cdots,x_n)$ be a Boolean function. Given $B=\{x_1,x_2,\ldots,x_b\}$ a subset of X, let $f(B,X-B)$ also represent $f(X)$. The support of $f(X)$ is denoted as

$sup(f)=X$. Let $f_{\bar{x}_i}$ and f_{x_i} represent the cofactors of $f(X)$ with respect to variable x_i. Cofactors of $f(X)$ with respect to multiple variables are defined in a similar way. For example, $f_{x_1\bar{x}_2}=f(X)|_{x_1=1,x_2=0}$. The *cofactor set* of $f(X)$ with respect to B, denoted $cs_B(f)$, is the set of all *distinct* cofactors of $f(X)$ with respect to the variables in B. Clearly $|cs_B(f)|\le 2^{|B|}$. We use ROBDD [2] to represent functions in this paper. For each node u in a ROBDD, let f_u denote the function of the ROBDD rooted at u. Figure 1 shows three partial ROBDD's (top 4 levels). In Figure 1(a), f_{u_0} denotes both $f_{\bar{x}_1x_2}$ and $f_{x_1\bar{x}_2}$. If $B=\{x_1,x_2\}$, $cs_B(f)=\{f_{u_0},f_{u_1},f_{u_4}\}$.

It is clear that $cs_B(f)$ represents the set of distinct columns in the decomposition chart [10] or the set of compatible classes [21]. Let $g(Y(B),X-B)$ represent the function $g(y_1(B), y_2(B), \cdots, y_t(B), x_{b+1}, \cdots, x_n)$ where $Y(B)=\{y_1(B), y_2(B),..., y_t(B)\}$. The condition for a disjoint decomposition $g(Y(B),X-B)$ of $f(X)$ to exist is given by the following theorem.

Theorem 1 [1, 10] There exists a decomposition $g(Y(B),X-B)$ of $f(X)$ under the bound set B if and only if $|cs_B(f)|\le 2^{|Y|}$ where $|Y|$ denotes the number of encoding functions in $Y(B)$.

In general, encoding functions depend on all variables in the bound set B. An encoding function $y_i(B)$ is *partially-dependent* on B if $|sup(y_i)|<|B|$. The functional decomposition which produces partially-dependent encoding functions is called *partially-dependent (functional) decomposition*. We formulate the following problem and give a necessary and sufficient condition for the existence of a partially-dependent encoding function in a decomposition.

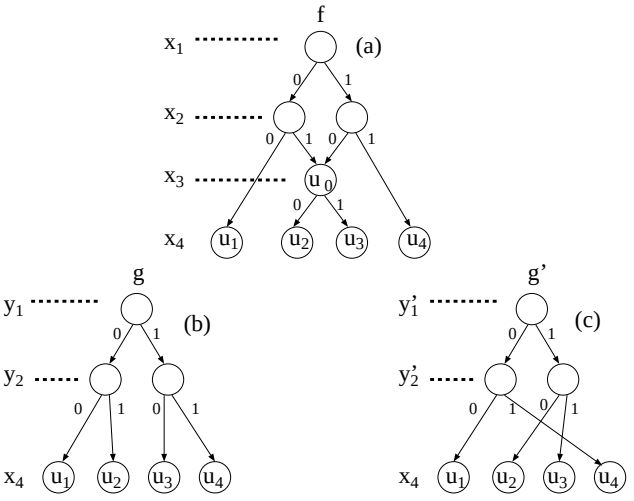

Figure 1 (a) The function f (top 4 levels). (b) A decomposition g of f where $sup(y_1)=B$. (c) A partially-dependent decomposition g' of f where $x_3\notin sup(y'_1)$.

Partially-Dependent Decomposition (PDD) Problem Given a function $f(X)$ and a bound set B, construct a partially-dependent decomposition $g(Y(B),X-B)$ of $f(X)$ such that $|sup(y_1)|$ is minimum.

Theorem 2 There exists a partially-dependent decomposition $g(Y(B),X-B)$ of $f(X)$ under the bound set B with $y_1(B)=y_1(x_1,\cdots,x_m)$ if and only if $cs_{\{x_1,\ldots,x_m\}}(f)$ can be partitioned into two sets cs_1 and cs_2 such that

$$\Big|\bigcup_{h\in cs_1} cs_B(h)\Big|\le 2^{|Y|-1} \quad \text{and} \quad \Big|\bigcup_{h\in cs_2} cs_B(h)\Big|\le 2^{|Y|-1}.$$

Please refer to [8] for a detail proof. The methods for computing single and multiple partially-dependent encoding functions are discussed in Section 2.2.

We illustrate the theorem by an example. Consider a function $f(X)$ of which the ROBDD is in Figure 1(a). Let $B=\{x_1,x_2,x_3\}$. Since $cs_B(f)=\{f_{u_1},f_{u_2},f_{u_3},f_{u_4}\}$, according to Theorem 1, there exists a decomposition of $f(X)$ which requires two encoding functions. Let $g(Y(B),X-B)$ be a decomposition of $f(X)$ such that $g_{\bar{y}_1\bar{y}_2}=f_{u_1}$, $g_{\bar{y}_1y_2}=f_{u_2}$, $g_{y_1\bar{y}_2}=f_{u_3}$ and $g_{y_1y_2}=f_{u_4}$. The ROBDD of $g(Y,X-B)$ is shown in Figure 1(b). In sum of product form, we compute $y_1(B)=x_1x_2+x_2x_3+x_1x_3$ and $y_2(B)=x_1x_2+x_1\bar{x}_3+x_2\bar{x}_3$. Note that both $y_1(B)$ and $y_2(B)$ depend on all variables in the bound set B. Let's compute the following cofactor sets of $f(X)$: $cs_{\{x_1,x_2\}}(f)=\{f_{u_0},f_{u_1},f_{u_4}\}$, $cs_B(f_{u_0})=\{f_{u_2},f_{u_3}\}$, $cs_B(f_{u_1})=\{f_{u_1}\}$, and $cs_B(f_{u_4})=\{f_{u_4}\}$. We can partition $cs_{\{x_1,x_2\}}(f)$ into two sets $\{f_{u_0}\}$ and $\{f_{u_1},f_{u_4}\}$, such that $|cs_B(f_{u_0})|=2$ and $|cs_B(f_{u_1})\cup cs_B(f_{u_4})|=2$. According to Theorem 2, there exists a decomposition $g'(Y'(B),X-B)$ of $f(X)$ such that $y'_1(B)=y'_1(x_1,x_2)$. We construct $g'(Y'(B),X-B)$ by defining $g'_{\bar{y}'_1\bar{y}'_2}=f_{u_1}$, $g'_{\bar{y}'_1y'_2}=f_{u_4}$, $g'_{y'_1\bar{y}'_2}=f_{u_2}$ and $g'_{y'_1y'_2}=f_{u_3}$. The ROBDD of $g'(Y',X-B)$ is shown in Figure 1(c). In sum of product form, we compute $y'_1(B)=\bar{x}_1x_2+x_1\bar{x}_2$ and $y'_2(B)=x_1x_2+x_2x_3+x_1x_3$. Note that $y'_1(B)$ does not depend on the variable x_3. As a result we have found a partially-dependent decomposition $g'(Y'(B),X-B)$ of $f(X)$.

A special case of the partially-dependent decomposition is the non-disjoint decomposition where $B\cap sup(g)\neq\emptyset$. $B\cap sup(g)$ is the set of non-disjoint variables which appear as inputs to both the base function and the encoding functions. [14] gave a necessary and sufficient condition for non-disjoint decomposition and an algorithm to compute it. However, the approach does not relate disjoint and non-disjoint decompositions and is not easy to apply. Later, [15] pointed out that selecting encoding functions of single variable leads to a non-disjoint decomposition so that non-disjoint decomposition becomes a special case of disjoint decomposition with support minimization. If we let $m=1$ in Theorem 2, we obtain a condition for $y_1(B)$ to be a single variable encoding function, i.e., a non-disjoint variable x_1. For the existence of a general non-disjoint decomposition, we have the following corallary.

Corollary 2.1 There exists a non-disjoint decomposition $g(Y(B),S,X-B)$ of $f(X)$ under the bound set B with $S \subset B$ being the set of non-disjoint variables, if and only if $|cs_B(h)| \le 2^{|Y|}$ for every cofactor $h \in cs_S(f)$.

The proof is omitted and can be found in [8]. Note that we do not treat non-disjoint variables as encoding functions in this corallary. Efficient computation for non-disjoint decompositions are introduced in the next section.

2.2. Algorithms for the PDD Problem

Based on Theorem 2, we can compute a partially-dependent encoding function y_1 of minimum support for the PDD problem. The algorithm is listed in Figure 2. Let $m = |sup(y_1)|$. We start with testing $m = 1$. For each variable $x_i \in B$, we reorder the ROBDD of $f(X)$ such that x_i is the first variable in the ordering. Then we compute $cs_B(f_{\bar{x}_i})$ and $cs_B(f_{x_i})$. Note that $cs_{\{x_i\}}(f)$ can only be partitioned into $\{f_{\bar{x}_i}\}$ and $\{f_{x_i}\}$. An encoding function $y_1(x_i)$ exists if and only if $|cs_B(f_{\bar{x}_i})| \le 2^{|Y|-1}$ and $|cs_B(f_{x_i})| \le 2^{|Y|-1}$. In this case, $y_1(x_i)$ is a non-disjoint variable. It is interesting to notice that searching for a non-disjoint variable is the simplest case in our approach, while the method in [15] is most efficient in computing encoding functions of $|B|-1$ variables. In general, the existence of non-disjoint variables (i.e. non-disjoint decomposition) can be tested efficiently in our approach. For example, for testing x_1, x_2 as non-disjoint variables, we shall examine $|cs_B(f_{\bar{x}_1\bar{x}_2})|$, $|cs_B(f_{\bar{x}_1 x_2})|$, $|cs_B(f_{x_1\bar{x}_2})|$ and $|cs_B(f_{x_1 x_2})|$, respectively. In Figure 2, if we fail the test of $m = 1$, we try $m = 2$ by generating all 2-variable subsets of B. The procedure terminates when all $(|B|-1)$-variable subsets of B have been tested.

In the worst case, the algorithm will try $2^{|B|}-2$ subsets of B as the support set of y_1. For each subset $\{z_1, \ldots, z_m\} \subset B$, it partitions the cofactor set $cs_{\{z_1,\ldots,z_m\}}(f)$ into cs_1 and cs_2 to satisfy Theorem 2. For a cofactor set of k members, there are 2^k possible partitions. Hence the complexity of our algorithm is exponential. In order to compute a partition in reasonable time, we use a dynamic programming algorithm similar to that for the number partitioning problem [11]. Details can be found in [8]. In the application to FPGA synthesis, we usually have $|B| = 5$. For $m \le 3$ our method is quite efficient. When $m = 4$ and a large number of partitions which satisfy Theorem 2 are detected, we limit m to 3 in our implementation for partially-dependent decomposition.

```
function Min_Support_Y1 ( f , X , B )
/* f (X) = input function, B = bound set, return y1 */
for m = 1 to |B| − 1 do
  for each set {z1, . . . , zm} ⊂ B do
    Compute cs{z1,...,zm}(f)
    for each partition cs1 and cs2 of cs{z1,...,zm}(f) do
      if cs1 and cs2 satisfies Theorem 2 then
        for each cofactor f (b1, ···, bm, X−B) ∈ cs1 do
          y1(b1, ···, bm) = 1
        for each cofactor f (b1, ···, bm, X−B) ∈ cs2 do
          y1(b1, ···, bm) = 0
        return y1(z1, ···, zm)
```

Figure 2 Computing an encoding function of min. support.

To complete a decomposition, we need to compute $|Y|$ encoding functions. An encoding function $y_i(B)$ is said *assignable* to another encoding function $y_j(B)$ if they together partition $cs_B(f)$ into four subsets such that each subset contains at most $2^{|Y|-2}$ cofactors [21]. After obtaining y_1 by *Min_Support_Y1*, we can compute a minimal-support encoding functions y_2 assignable to y_1 in a similar way, and repeat the process for remaining encoding functions. Or, we can modify the algorithm in Figure 2 to compute *all* partially-dependent encoding functions and select a maximum number of *mutually assignable* encoding functions from the set to construct a decomposition of as many partially-dependent encoding functions as possible. Details can be found in [8].

3. Applications of Partially-Dependent Functional Decomposition

Before we describe our applications, we introduce some notations. Given a network N, let N_v denote the subnetwork consisting of node v and all the predecessors of v. Given a subnetwork H, let $input(H)$ denote the set of nodes outside H which supply inputs to H. A cut in N_v is a partition $(X_v, \bar{X}_v)$ of nodes in N_v such that primary inputs (PI's) are in X_v and $v \in \bar{X}_v$. The cutset of the cut, denoted $n(X_v, \bar{X}_v)$, is $input(X_v)$. The cut is *K-feasible* if $|n(X_v, \bar{X}_v)| \le K$. Let $f_v(X_v, \bar{X}_v)$ denotes the function of v with respect to the cut. The height *height*(v) of a node v is the number of edges on the longest path from any PI to v. The height of a cut, denoted *height*$(X_v, \bar{X}_v)$, is the maximum height of nodes in the cutset. A min-cut of height p in N_v can be computed using the max-flow min-cut algorithm [5]. A fanout-free cone (FFC) at v, denoted FFC_v, is a cone rooted at v such that for every $u \ne v \in FFC_v$, $output(u) \subseteq FFC_v$ [6]. The MFFC at v, denoted $MFFC_v$, is the FFC at v which contains a maximal number of nodes. An important property of MFFC is that two MFFC's are either disjoint or one contains another [6]. As a result, any network can be uniquely partitioned into a set of MFFC's.

3.1. XC4000 CLB Functional Capability

The Xilinx XC4000-families configurable logic block (XC4000 CLB) contains three LUT's F, G and H as shown in Figure 3 [22]. Four independent inputs (F1-F4 and G1-G4) are provided for each of two 4-input LUT's F and G. The LUT H has inputs from F, G and a third input from outside the block (H1). With this design, a XC4000 CLB can be used to implement (i) any two functions of up to four variables, or (ii) any single function of five variables, or (iii)

any function of four variables together with some function of five variables, or (iv) some function of up to nine variables. The flexibility provided by XC4000 CLB, however, also creates a great deal of difficulty to have efficient use of all resources.

To implement a Boolean network on XC4000 FPGA's, the common approach is to first map the network into a K-LUT network and then pack the K-LUT network into XC4000 CLB's. The number K can be set to either 4 or 5. If we set $K=4$, every two 4-LUT's in the mapping solution can be implemented by the F and G LUT's in a XC4000 CLB, but the H LUT is left unused. Alternatively, the RASP system [9] set $K=5$ to generate 5-LUT mapping solutions followed by a routine *match_4k* to decompose each 5-LUT into either (i) a 4-LUT feeding into a 2-LUT (which we call a 4-2 decomposition). or (ii) two 4-LUT's feeding into a 3-LUT (which we call a 4-4-3 decomposition). In a 4-2 decomposition, only the F and H LUT's of a XC4000 CLB are used. As a result, the G LUT is saved for another 4-LUT. Our experiments show that over 90% of 5-LUT's in MCNC benchmark mapping solutions have 4-2 decomposition. By matching a 5-LUT (with 4-2 decomposition) with a 4-LUT, the XC4000 CLB can be used efficiently.

Although both approaches take advantage of XC4000 CLB structure to some extent, they do not exploit the possibility of using a single CLB to implement a function of more than five variables (*wide* functions). At first it may seem unlikely. There are 2^{64} distinct functions of six variables. A XC4000 CLB can implement at most 2^{40} functions since it has only $32+8=40$ memory bits for logic functions. That is only $1/2^{24}$ of all 6-variable functions. For $K>6$, the percentage becomes even lower. However, our experiments show 99% of 6-input cones and 65% of 7-input cones in all MCNC benchmarks can be implemented by XC4000 CLB's (see Table 1 for more details).

From a functional decomposition point of view, a XC4000 CLB can implement any function of the form $g(y_1(X_1), y_2(X_2), x_n)$ where $X=\{x_1,x_2,\ldots,x_n\}$, $n\le 9$, $X_1, X_2 \subset X$, $|X_1|\le 4$ and $|X_2|\le 4$. We call this form the *XC4000-CLB form*. A function $f(X)$ can be implemented by a XC4000 CLB if $f(X)$ can be transformed into the XC4000-CLB form. For example, when $n=5$, the Shannon expansion $f(X)=\bar{x}_5 f_{\bar{x}_5}+x_5 f_{x_5}$ transforms $f(X)$ into the XC4000-CLB form, where $f_{\bar{x}_5}$ and f_{x_5} correspond to y_1 and y_2 respectively and $g(y_1, y_2, x_5)$ has the functionality of a 2-input multiplexer. For $n>5$, we consider the following two categories of functions.

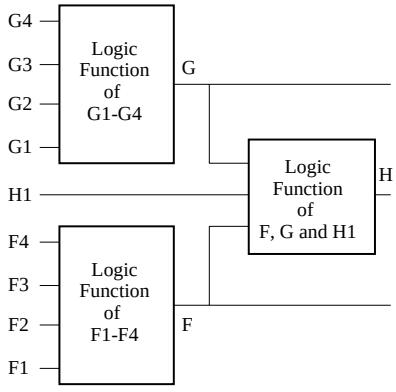

Figure 3 Simplified Block Diagram of XC4000 CLB

Category I A function $f(X)$ belongs to this category if $|X|=6$ and it can be decomposed into the form $g(y_1(x_1,x_2,x_3,x_4), x_5, x_6)$. In a XC4000 CLB, the F LUT will implement $y_1(x_1,x_2,x_3,x_4)$ and the H LUT will implement $g(y_1, x_5, x_6)$. Note that if x_5 is the output of a fanout-free LUT L of no more than four inputs, the G LUT in the CLB can be used for L. Otherwise, the G LUT is used by x_5.

Category II A function $f(X)$ belongs to this category if $5<|X|\le 9$ and it can be decomposed into the form $g(y_1(B_1), y_2(B_2), x_n)$ where $B_1, B_2 \subset X-\{x_n\}$, $|B_1|\le 4$, $|B_2|\le 4$ and $2<|cs_{X-\{x_n\}}(f)|\le 4$, In a XC4000 CLB, the F and G LUT's will implement $y_1(B_1)$ and $y_2(B_2)$ respectively and the H LUT will implement $g(y_1, y_2, x_n)$.

In the description for Category I, it is easy to see $g(y_1(B), x_5, x_6)$ is in fact a simple disjoint decomposition of $f(X)$ under the bound set $B=\{x_1,x_2,x_3,x_4\}$. We have the following observation for Category I.

Observation 1 A function $f(x_1,x_2,\cdots,x_6)$ belongs to Category I if and only if $|cs_B(f)|=2$ where $B=\{x_1,x_2,x_3,x_4\}$.

In the description for Category II, it is easy to see $g(y_1(B_1), y_2(B_2), x_n)$ is a partially-dependent decomposition of $f(X)$ under the bound set $B=X-\{x_n\}$. The condition $2<|cs_B(f)|\le 4$ is to ensure *two* encoding functions (each has no more than four variables) instead of *one* encoding function of $n-1$ variables (when $|cs_B(f)|=2$). We have the following observation for Category II.

Observation 2 A function $f(X)$ belongs to Category II if and only if there is a partially-dependent decomposition of $f(X)$ under the bound set $B=X-\{x_n\}$ such that $|sup(y_1)|\le 4$ and $|sup(y_2)|\le 4$.

Note that Category I and II do not include all functions that a XC4000 CLB can implement. For example, $f(X)=g(y_1(X_1), y_2(X_2))$ can be implemented by a XC4000 CLB (H1 stuck at 0 or 1) but $f(X)$ is not in either category. However, with only these two categories of functions, we can implement a large percentage of 6- and 7-input cones in technology mapping for XC4000 FPGA's.

3.2. Technology Mapping for XC4000 FPGA's

Given XC4000 CLB architecture, we can set $K=9$ in technology mapping for XC4000 FPGA's and apply Observation 1 and 2. But considering the success rate of

mapping wide functions to XC4000 CLB and the computing time, we select $K=7$ in our implementation. For $K=7$, we try to cover networks by CLB's (not only LUT's) directly. In the FlowMap algorithm [5] the label of a node v is defined to be the minimum level of v in any LUT mapping solutions. We call it the LUT label of v, $lut\_label(v)$. Similarly, the minimum level of node v when the network is covered by CLB's is called the CLB label of v, $clb\_label(v)$. In general, $clb\_label(v) \le lut\_label(v)$. The largest CLB label in a network N is called the CLB depth $CLB\_depth(N)$ of the network. The algorithm which computes CLB labels is listed in Figure 4. Note that if we try all possible cuts of height $max\_label-1$ in lines 9 to 19, the algorithm will compute the minimum CLB label for each node. However, we only consider two cuts (max. and min. volume min-cuts) in our implementation. The cut for each node is saved during the CLB label computation.

We design a new mapping algorithm, PDDMAP, to map an input network to a network of CLB's and LUT's. The use of CLB's for wide functions in general will decrease the depth of mapping solution. However, the use of 5-LUT's is more area efficient because most of them have 4-2 decompositions. PDDMAP first computes LUT labels and CLB labels for all nodes in the network. A primary output v is a *critical* primary output if $LUT\_label(v) \ge CLB\_depth(N)$. PDDMAP covers critical primary outputs and their fanin networks using CLB's to obtain minimal CLB depth in the CLB mapping phase. For the remaining non-critical primary outputs and their fanin networks, PDDMAP will use 5-LUT's or smaller LUT's to cover them for area minimization in the LUT mapping phase. Finally, PDDMAP performs a matching on LUT's and packs matched LUT's into a minimum number of CLB's using the *match_4k* procedure developed in [9]. The PDDMAP mapping algorithm is outlined in Figure 5.

```
1   function Compute_CLB_Label ( N , K )
2   /* N = input network, K = max. CLB input size */
3   /* output = clb_label(v) for all v ∈ N */
4   for each node v ∈ N in topological order do
5     if v is a primary input
6       clb_label(v) = 0
7     else
8       max_label = max { clb_label(u) | (u,v) ∈ N }
9       Compute a cut (X_v, X̄_v) of height max_label − 1
10      if |n(X_v, X̄_v)| ≤ 5
11        clb_label(v) = max_label
12      else if |n(X_v, X_v)| ≤ K
13        Compute function f_v(X_v, X̄_v)
14        if f_v(X_v, X̄_v) belongs to Category I or II
15          clb_label(v) = max_label
16        else
17          clb_label(v) = max_label + 1
18      else if |n(X_v, X_v)| > K
19        clb_label(v) = max_label + 1
```

Figure 4 Computing CLB labels for network nodes.

In order to find the best cut $(X_v, \bar{X}_v)$ in N_v when mapping node v for area minimization in the LUT mapping phase, PDDMAP computes a set $C(v)$ of all 5-feasible cuts for every node v (from PI's to PO's in topological order) based on the suggestion of [18]. For a node v, this set $C(v)$ can be computed by merging cuts from all sets $C(u_i)$ where $(u_i, v) \in N$ and accepting only 5-feasible cuts in the merged set. Although there are $O(n^5)$ 5-feasible cuts in the worst case, [18] pointed out that average number of cuts per node is usually smaller than 100 for a 2-input network.

After computing the set of all 5-feasible cuts for every node $v \in N$, PDDMAP estimates the mapping area (counted in number of CLB's) for N_v rooted at v from PI's to PO's in topological order as follows. If v is a PI node, the mapping area for N_v is zero. For an internal node v and a cut $(X_v, \bar{X}_v) \in C(v)$, the estimated mapping area for N_v, denoted $CLB\_count_v(X_v, \bar{X}_v)$, is computed by the following formula

$$CLB\_count_v(X_v, \bar{X}_v) = \sum_{u_i \in S_1} \frac{CLB\_count(u_i)}{|output(u_i)|} + \sum_{u_i \in S_2} \varepsilon + cost_v(X_v, \bar{X}_v)$$

where $S_1 \subseteq n(X_v, \bar{X}_v)$ is the set of nodes not yet mapped as CLB or LUT, $S_2 \subseteq n(X_v, \bar{X}_v)$ is the set of nodes already being mapped as CLB or LUT, ε is a small value, and

```
function PDDMAP ( N )
/* input  = Boolean network N   */
/* output = mapped network of CLB's and LUT's */
Compute lut_label(v) for every v ∈ N
Compute clb_label(v) for every v ∈ N and save the cut
    which achieves the label
CLB_depth(N) = max{ CLB_label(v) | v ∈ N }
/*--- CLB Mapping Phase ---*/
for each PO v of LUT_label(v) ≥ CLB_depth(N) do
  Get saved cut (X_v, X̄_v) of height clb_label(v) − 1
  Map X̄_v to a CLB
  Map every u ∈ n(X_v, X̄_v) recursively using CLB's
/*--- LUT Mapping Phase ---*/
Compute C(v) of all 5-feasible cuts for every v ∈ N
for each remaining PO v do
  Compute CLB_count_u(X_u, X̄_u) for all u ∈ N_v and
      (X_u, X̄_u) ∈ C(u) from PI's
  Select (X_v, X̄_v) of minimum CLB_count_v(X_v, X̄_v)
  Map X̄_v to a LUT
  Map every u ∈ n(X_v, X̄_v) recursively using LUT's
Collapse N to a network of CLB's and LUT's
Match LUT's using match_4k in [8]
Pack matched LUT's into CLB's and return
```

Figure 5 The PDDMAP algorithm.

$cost_v(X_v,\bar{X}_v)$ is the cost of mapping v using a XC4000 CLB with $n(X_v,\bar{X}_v)$ as inputs. The first term is the area contributed by implementing nodes in S_1 with consideration of possible sharing of each $u_i \in S_1$ among its fanouts. In the second term, ε is set to be very small as $u_i \in S_2$ have been implemented as CLB or LUT but we still want to minimize the fanin size of the current CLB. The third term is the area cost of mapping node v itself. If $|n(X_v,\bar{X}_v)| \le 4$, $cost_v(X_v,\bar{X}_v)$ is 0.4 CLB as it can be implemented by the F or G LUT in a CLB. If $|n(X_v,\bar{X}_v)| = 5$ and $f_v(X_v,\bar{X}_v)$ has a 4-2 decomposition, $cost_v(X_v,\bar{X}_v)$ is 0.6 CLB as it can be implemented by the F or G LUT plus the H LUT. Otherwise, $cost_v(X_v,\bar{X}_v)$ is 1.0 CLB. The estimated mapping area in CLB count for N_v would be the minimum $CLB\_count_v(X_v,\bar{X}_v)$ for all cuts $(X_v,\bar{X}_v) \in C(v)$.

3.3. Resynthesis of Mapping Solutions

Direct synthesis for LUT-based FPGA produces excellent results in both area and depth for MCNC benchmarks [13, 21, 16]. The ROBDD's of functions are computed first, and multiple output functional decomposition and support minimization techniques are applied to produce LUT networks. However, it might be difficult to compute ROBDD's for large circuits. Alternatively, [19, 4] integrate synthesis into technology mapping. For example, the FlowSyn algorithm [4] synthesizes subnetworks to obtain depth below the combinatorial limit of technology mapping. Since synthesis is applied locally, this approach can be used for large circuits.

We propose a new approach named PDDSYN which resynthesizes any mapping solutions for area minimization. The resynthesis is based on partially-dependent decomposition for small encoding functions (less than five variables). Alternatively, we can select a bound set of up to nine variables and implement encoding functions by XC4000 CLB's. But our experiments show very few (6%) six variable encoding functions ($|B| = 6$) can fit XC4000 CLB's. So we only consider $|B| \le 5$. Our post-mapping resynthesis has two advantages over synthesis during mapping. First, we may resynthesize the network iteratively for continuous decrease of network area. Second, we can calculate the area gain precisely during post-mapping resynthesis. Such gain is difficult to estimate during the mapping phase since the final mapped network is not constructed yet. We outline the PDDSYN algorithm in Figure 6.

PDDSYN is based on resynthesis of nodes (either CLB or LUT) with respect to a cut. Let N be a CLB/LUT network, v be a node in N, and $(X_v,\bar{X}_v)$ be a cut in N_v. If $|n(X_v,\bar{X}_v)| \le 7$, PDDSYN will first try to replace $\bar{X}_v$ by a single CLB. If it cannot be done or $|n(X_v,\bar{X}_v)| > 7$, the ROBDD of $f_v(X_v,\bar{X}_v)$ is computed and a LUT network R_v with $n(X_v,\bar{X}_v)$ as inputs will be synthesized through a series of partially-dependent decomposition. The area gain is the CLB count of $MFFC_v$ (not $\bar{X}_v$) minus the CLB count of R_v

```
function PDDSYN( N )
/* input  = CLB/LUT network N */
/* output = resynthesized CLB/LUT network N' */
repeat
   /*--- MFFC RESYNTHESIS ---*/
   Partition N into MFFC's
   for each MFFC MFFC_v do
      Resynthesize MFFC_v into R_v
      if R_v < MFFC_v in CLB count
         Replace MFFC_v by R_v
   /*--- MIN-CUT RESYNTHESIS ---*/
   for each node v ∈ N do
      for p = height(v) − 1 to 0 do
         Compute a min-cut (X_v, X̄_v) of height p
         Resynthesize X̄_v into R_p(X_v, X̄_v)
      Let R_v be the R_p(X_v, X̄_v) of min. area for all p
   for each node v ∈ N in reverse topological order do
      if R_v < MFFC_v in CLB count
         Replace MFFC_v by R_v
until no further improvement
return resynthesized N'
```

Figure 6 The PDDSYN algorithm.

(since some LUT in $\bar{X}_v$ may fanout to LUT's outside $\bar{X}_v$). The $MFFC_v$ will be replaced if the area gain is positive.

Two kinds of resynthesis are performed in PDDSYN: the MFFC resynthesis and the min-cut resynthesis. In MFFC resynthesis, the input network N is partitioned into MFFC's and each MFFC is resynthesized for a smaller area. This process terminates when every MFFC has been tried for a resynthesis. The min-cut resynthesis includes two phases. In phase I, PDDSYN computes the *best* alternative network R_v rooted at LUT $v \in N$ as follows. First, the min-cuts of height p in N_v, $0 \le p < height(v)$, are computed. For each min-cut $(X_v,\bar{X}_v)$ of height p, a LUT network and its area gain is computed. PDDSYN chooses the one of largest positive gain as R_v for v. In phase II, PDDSYN traverses N from PO's to PI's and replaces each LUT v of positive area gain by R_v if LUT v still fanouts. The LUT's with zero fanout will be removed from N in the traverse.

4. Experimental Results

We have implemented PDDMAP and PDDSYN in C language and incorporated our implementation into the SIS [20] and the RASP [9] FPGA logic synthesis package. We test our algorithms using MCNC combinational benchmarks on a SPARC 5 workstation with 96MB of memory. The set of benchmarks are first optimized for area and then decomposed into 2-input simple-gate multi-level networks using standard SIS routines. The same set of benchmarks has been widely used in previous mapping experiments. We shall compare our results with those from FlowMap [5], TechMap-D [19], FlowSyn [4] and BoolMap-D [16].

In our first experiment, we collect statistics of implementing wide functions in technology mapping using XC4000 CLB's. For every node in MCNC benchmarks, we compute all 6-input and 7-input cuts of the node. For each node with respect to each cut, we compute the function and test if it belongs to Category I or II. The results are shown in Table 1. Numbers in columns *All* are the total numbers of cuts. Numbers in columns *I* and *II* are the percentage of cuts with respect to which the node functions are in Category I and II respectively. Functions which have less than six *observable* variables are not shown in the table. In summary, 99% of 6-input cones and 65% of 7-input cones can be covered by XC4000 CLB's in our approach. This high percentage is the reason why PDDMAP can reduce the depth of mapping solutions by using XC4000 CLB's. On the other hand, much lower percentage of wide encoding functions can be transformed into the XC4000-CLB form during post-mapping resynthesis. The last two columns show the numbers of 6-variable encoding functions produced (in column *All*) and those implementable by XC4000 CLB's (in column *CLB*) during functional decomposition ($|B|=6$). Only 6% of encoding functions can be implemented by XC4000 CLB's.

We use PDDMAP to generate mapping solutions for MCNC benchmarks. In order to compare our results with previous results, we use a procedure *match_4k* proposed in the RASP system [9] to transform LUT networks into XC4000 CLB networks. The *match_4k* procedure decomposes every 5-LUT into either a 4-LUT feeding into a 2-LUT (a 4-2 decomposition) or two 4-LUT's and one 3-LUT (a 4-4-3 decomposition) and computes a maximum matching for LUT's which can be packed into one CLB.

	6-input cuts			7-input cuts			6-var. $y(B)$	
Circuit	All	I	II	All	I	II	All	CLB
5xp1	551	66%	32%	1028	0%	73%	10	1
9sym	1283	78%	21%	2398	0%	74%	63	2
9symml	1188	80%	19%	2312	0%	70%	56	0
C499	25429	81%	15%	-	-	-	100	6
C880	9139	87%	10%	23739	1%	74%	187	14
alu2	7383	84%	14%	20407	2%	68%	-	-
alu4	17117	83%	13%	47571	2%	63%	-	-
apex4	32589	92%	8%	76921	0%	58%	-	-
apex6	4570	89%	11%	8708	0%	75%	158	13
apex7	1648	88%	8%	3639	3%	77%	65	5
count	1595	83%	10%	3077	5%	65%	49	3
duke2	6042	91%	6%	13730	3%	67%	141	5
e64	3933	100%	0%	8553	0%	0%	109	0
misex1	516	79%	15%	835	0%	58%	2	0
rd84	2412	81%	18%	5569	0%	71%	30	0
rot	6335	84%	13%	14205	1%	76%	173	15
vg2	1328	85%	15%	2913	0%	74%	27	4
z4ml	368	81%	16%	769	0%	67%	2	0
Total	123426	86%	11%	236374	1%	63%	1172	68

Table 1 Statistics of mapping wide functions by XC4000 CLB's.

Experiments show *match_4k* can reduce the size of mapping solutions (from LUT's to CLB's) significantly. We run FlowMap followed by *match_4k* to produce one set of CLB mapping solutions for comparison. The results are in Table 2. Comparing to FlowMap followed by *match_4k*, PDDMAP is able to decrease CLB depth by 13% with only 1% increase of CLB's.

We tested PDDSYN on CLB networks generated by PDDMAP and compared the resynthesized CLB networks with the results by FlowSyn [4] followed by *match_4k*, and also with the LUT mapping results by TechMap-D [19] and BoolMap-D [16]. Because the two sets of data in [19, 16] are from LUT mapping solutions, only their depths are used for direct comparison. From Table 3 and Table 2, we see PDDSYN is able to improve PDDMAP mapping solutions by 5% in depth and 7% in CLB count. Comparing to the results by FlowSyn and *match_4k*, PDDMAP together with PDDSYN produce results which are 8% smaller in depth and 11% smaller in CLB count. The subtotal in Table 3 is calculated from benchmarks which are tested by TechMap-D. The overall mapping depth by PDDSYN is slightly larger than the depth of BoolMap-D mapping solutions, but is substantially smaller than that of TechMap-D mapping solutions.

5. Future Work

Currently, we are studying the complete functional characterization of the XC4000 CLB based on the partially-dependent decomposition.

	FlowMap			PDDMAP		
Circuit	d	clb	T(s)	d	clb	T(s)
5xp1	3	17	0.3	2	16	6.6
9sym	4	35	0.5	4	36	13.3
9symml	5	37	0.5	4	36	14.2
C499	5	83	6.9	5	81	128.2
C880	8	132	3.6	7	131	52.7
alu2	8	94	2.5	7	99	78.9
alu4	10	149	6.4	7	176	138.6
apex4	6	425	13.1	5	435	122.3
apex6	4	160	2.2	4	138	31.4
apex7	4	46	0.7	3	54	9.9
count	3	44	0.6	3	32	9.7
des	3	676	18.2	4	664	879.6
duke2	4	99	1.6	3	111	24.3
e64	3	84	1.4	3	80	15.9
misex1	2	10	0.2	2	10	3.7
rd84	4	29	0.6	4	26	12.2
rot	6	143	3.0	5	154	38.1
vg2	4	23	0.3	3	23	5.8
z4ml	3	7	0.1	2	5	3.9
Total	89	2293	62.7	77	2307	1589.3
ratio	1	1		87%	101%	

Table 2 Comparison of mapping results of FlowMap and PDDMAP for XC4000 FPGA.

	PDDMAP		FlowSyn		TechMap-D		BoolMap-D	
	PDDSYN		match_4k					
Circuit	d	clb	d	clb	d	lut	d	lut
5xp1	2	12	2	14	2	17	2	13
9sym	3	5	3	7	3	9	3	7
9symml	3	5	4	38	3	9	3	7
C499	5	72	4	56	4	148	4	101
C880	7	131	8	142	7	213	7	146
alu2	7	85	6	122	8	197	4	43
alu4	7	146	8	312	-	-	7	268
apex4	5	433	6	513	-	-	-	-
apex6	4	131	4	178	5	252	4	189
apex7	3	52	4	46	4	86	3	78
count	3	29	3	43	4	71	2	42
des	4	662	3	544	8	1395	3	594
duke2	3	108	4	111	4	175	5	193
e64	3	80	3	83	-	-	-	-
misex1	2	7	2	10	2	18	2	15
rd84	2	8	3	10	3	16	2	10
rot	5	153	6	164	6	315	6	228
vg2	3	23	4	23	4	36	4	30
z4ml	2	4	2	4	2	9	2	5
Total	73	2146	79	2420				
Subtotal	58	1487	62	1512	69	2966	56	1701

Table 3 Comparison of mapping results by PDDSYN, FlowSyn, TechMap-D, and BoolMap-D.

Acknowledgement

The authors would like to thank Dr. Richard Rudell from Synopsys for his valuable insight and discussions on cut computation. This work is partially supported by NSF Young Investigator (NYI) Award MIP-9357582, and grants from Xilinx, Xerox PARC, and Lucent Technologies under NSF NYI and California MICRO programs.

References

[1] Ashenhurst, R. L., ‘‘The Decomposition of Switching Functions,’’ *Proc. Int’l Symp. on Theory of Switching Functions*, 1959.

[2] Bryant, R. E., ‘‘Graph-based Algorithms for Boolean Function Manipulation,’’ *IEEE Trans. on Computers*, Vol. **C-35**, pp. 677-691, Aug. 1986.

[3] Chang, S.-C. and M. Marek-Sadowska, ‘‘Technology Mapping via Transformations of Function Graphs,’’ *Proc. IEEE Int’l Conf. on Computer Design*, pp. 159-162, Oct. 1992.

[4] Cong, J. and Y. Ding, ‘‘Beyond the Combinatorial Limit in Depth Minimization for LUT-Based FPGA Designs,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 110-114, 1993.

[5] Cong, J. and Y. Ding, ‘‘FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs,’’ *IEEE Trans. on Computer-Aided Design*, Vol. **CAD-13**(1) pp. 1-12, Jan. 1994.

[6] Cong, J. and Y. Ding, ‘‘On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,’’ *IEEE Trans. on VLSI Systems*, Vol. **2**, June 1994.

[7] Cong, J. and Y. Ding, ‘‘Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays,’’ *ACM Trans. on Design Automation of Electronic Systems*, Vol. **1, 2**, pp. 145-204, 1996.

[8] Cong, J. and Y.-Y. Hwang, ‘‘A Theory on Partially-Dependent Functional Decomposition with Application in LUT-based FPGA,’’ in *UCLA Computer Science Dept. Tech. Report CSD-950050*, (December 1995).

[9] Cong, J., J. Peck, and Y. Ding, ‘‘RASP: A General Logic Synthesis System for SRAM-based FPGAs,’’ *Proc. ACM 4th Int’l Symp. on FPGA*, Feb. 1996.

[10] Curtis, H. A., ‘‘A Generalized Tree Circuit,’’ *Journal of the ACM*, Vol. **8**(4) pp. 484-496, 1961.

[11] Garey, M. and D. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco (1979).

[12] Huang, J.-D., J.-Y. Jou, and W.-Z. Shen, ‘‘Compatible Class Encoding in Roth-Karp Decomposition for Two-Output LUT Architecture,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 359-363, Nov. 1995.

[13] Lai, Y.-T., K.-R. R. Pan, and M. Pedram, ‘‘FPGA Synthesis using Function Decomposition,’’ *Proc. Int’l Conf. on Computer Design: VLSI in Computers*, pp. 30-35, Oct. 1994.

[14] Lai, Y.-T., M. Pedram, and S. Vrudhula, ‘‘BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 642-647, June 1993.

[15] Legl, C., B. Wurth, and K. Eckl, ‘‘An Implicit Algorithm for Support Minimization during Functional Decomposition,’’ *Proc. European Design and Test Conf.*, March 1996.

[16] Legl, C., B. Wurth, and K. Eckl, ‘‘A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Designs,’’ *Proc. ACM/IEEE 33rd Design Automation Conf.*, pp. 730-733, 1996.

[17] Roth, J. P. and R. M. Karp, ‘‘Minimization Over Boolean Graphs,’’ *IBM Journal of Research and Development*, pp. 227-238, April 1962.

[18] Rudell, R., Private Communication 1996.

[19] Sawkar, P. and D. Thomas, ‘‘Performance Directed Technology Mapping for Look-Up Table Based FPGAs,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 208-212, June 1993.

[20] Sentovich, E., K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephen, R. Brayton, and A. Sangiovanni-Vincentelli, ‘‘SIS: A System for Sequential Circuit Synthesis,’’ *U.C.Berkeley Technical Report UCB/ERL M92/41*, May, 1992.

[21] Wurth, B., K. Eckl, and K. Antreich, ‘‘Functional Multiple-Output Decomposition: Theory and an Implicit Algorithm,’’ *Proc. ACM/IEEE Design Automation Conf.*, pp. 54--59, Jun. 1995.

[22] Xilinx, *The Programmable Logic Data Book*, Xilinx, San Jose, CA (1994).