A Theory on Partially-Dependent Functional Decomposition with Application in LUT-based FPGA

Jason Cong and Yean-Yow Hwang

Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

In this paper, we present the theory of partially-dependent functional decomposition. A partially-dependent decomposition of function produces subfunctions which do not depend on all variables in the bound set. Non-disjunctive decomposition is derived as a special case while multiple-output decomposition can be reduced to the partially-dependent functional decomposition in our theory. We develop algorithms to compute a maximal set of partially-dependent encoding functions and non-disjunctive variables. We propose a post-mapping algorithm, CutSyn, which resynthesizes LUT subnetworks iteratively to improve technology mapping solutions. Experimental results show an average of 18% reduction in both depth and area for mapping solutions of MCNC benchmarks. Comparing to previous approaches in LUT-based FPGA mapping and synthesis, we can achieve a reduction of 2% to 11% in depth and a reduction of 10% to 39% in area in the final LUT networks.

1. Introduction

Functional decomposition is a core technique in the synthesis of Boolean functions. The idea is to transform a given function into small subfunctions such that these subfunctions altogether compute the function and individually can be implemented by available technologies such as *K*-input lookup table (K-LUT) based field-programmable gate array (FPGA). A K-LUT is a functional cell which can implement an arbitrary function of up to *K* inputs. For functions which depend on more than *K* inputs, they need to be decomposed into subfunctions of no more than *K* inputs.

The number of explicit variables are reduced by functional decomposition. Given a function $f(x_1, \cdots, x_n)$, the decomposition of f produces a function $g = f$ in the form $g(y_1(x_1, \ldots, x_i), \cdots, y_t(x_1, \ldots, x_i), x_j, \cdots, x_n)$ where $t < j \leq i+1$. The function g is called the *base function* and the functions y_i's are called the *encoding functions* of the decomposition. The decomposition is *simple* if $t = 1$ and is *disjunctive* if $j = i+1$. Ashenhurst [AS59] gave a sufficient and necessary condition for the existence of a simple disjunctive decomposition. The condition was generalized to the case of non-simple disjunctive decomposition by Curtis [Cu61]. Both Ashenhurst and Curtis employed the *decomposition chart* (basically a truth table) to represent Boolean functions and consequently limited the application of their works in practice. Roth and

Karp [RoKa62] formulated functional decomposition as a coloring problem of the incompatibility graph and represented the ON-set and the OFF-set of functions in the form of cube covers. Since cube covers are smaller in size than the decomposition chart, a larger set of functions can be decomposed by their method. Recently, the binary decision diagram (BDD) [Br86a] has been proposed and proved to be a compact functional representation. A number of works on single functional decomposition [ChMa92, LaPV93] are based on the BDD representation of functions and produce good results.

In order to produce an area-efficient implementation of function, functional decomposition have been explored along the following extensions. One extension is the *multiple-output* functional decomposition in which the extraction of *common* encoding functions is explored. Karp [Karp63] first proposed a method for two-output functional decomposition. New theories and algorithms are proposed in [LaPP94, WuEA95] for decomposing functions of more than two outputs. Another extension is to explore *support minimization* in functional decomposition to reduce both the area and the routing cost. In [HwJS95] the encoding scheme of compatible classes are carefully designed for two-output LUT architecture to minimize the number of programmable logic blocks (PLB). In [SaSN95] the non-disjunctive decomposition are performed for area minimization and the *don't care* set are exploited to reduce the number of variables in the base function.

Although good results have been reported in previous works, we observe a few limitations. First, the theories of non-disjunctive decomposition, multiple-output decomposition and support minimization are developed as independent theories. In fact, these theories are closely related. Second, the methods in [LaPP94, HwJS95] consider only the strict encoding scheme. A *strict* encoding scheme assigns a single code to each distinct column in the decomposition chart. However, functions may require some columns to receive more than one code in order to generate a successful decomposition. Hence [LaPP94, HwJS95] may fail to find a decomposition when there is one. On the other hand, the methods in [SaSN95] consider support minimization for the base function only. However, the encoding functions are as important as the base function in determining the quality of the decomposition of a function. In this paper, we present the theory of *partially-dependent* functional decomposition for support minimization which includes non-disjunctive decomposition and multiple-output decomposition as special cases. Based on our theory, we propose novel synthesis algorithms to improve the area and the depth of technology mapping solutions.

The remainder of this paper is organized as follows. Section 2 summarizes the classical theory of functional decomposition, presents the theory of partially-dependent functional decomposition, reconsider the non-disjunctive and multiple-output decompositions from the view point of our theory and give algorithms to compute partially-dependent encoding functions. A new resynthesis algorithm, CutSyn, is proposed in Section 3. Experimental results are presented in Section 4. Section 5 concludes the paper and points out future works.

2. Formulation and Theory of Partially-Dependent Functional Decomposition

In this section, we first introduce notations and results of classical functional decomposition. We then present our formulation and theory of partially-dependent functional decomposition and show how to formulate non-disjunctive decomposition and multiple-output decomposition as special cases of partially-dependent functional decomposition. Finally we give functional decomposition algorithms and summarize this section.

2.1. Notations

Let $X = \{x_1, x_2, \ldots, x_n\}$ be an ordered set of Boolean variables. A *prefix* subset of X is an ordered set $\{x_1, x_2, \cdots, x_i\}$ where $i \le n$. A *suffix* subset of X is an ordered set $\{x_i, x_{i+1}, \cdots, x_n\}$ where $1 \le i$. For convenience, we denote a Boolean function $f(x_1, x_2, \cdots, x_n)$ by $f(X)$. If $X_1, X_2, \cdots, X_k$ are disjoint subsets of X such that $X_1 \cup \cdots \cup X_k = X$, $f(X_1, X_2, \cdots, X_k)$ can also denote $f(X)$. We define $arg(f(X)) = X$. The two cofactors of $f(X)$ with respect to variable x_i, denoted $f_{\overline{x}_i}$ and f_{x_i}, are $f(X)|_{x_i=0}$ and $f(X)|_{x_i=1}$, respectively. Cofactors of f with respect to multiple variables are defined similarly, e.g., $f_{x_1\overline{x}_2} = f(X)|_{x_1=1,x_2=0}$. The *cofactor set* of f with respect to a set $B \subseteq X$, denoted $cs_B(f)$, is the set of all *distinct* cofactors of $f(X)$ with respect to the variables in B. For example, if $f_{\overline{x}_1,x_2} = f_{x_1,\overline{x}_2}$ and $B = \{x_1, x_2\}$, then $cs_B(f) = \{f_{\overline{x}_1,\overline{x}_2}, f_{x_1,\overline{x}_2}, f_{x_1,x_2}\}$ and $cs_B(f_{x_1}) = \{f_{x_1,\overline{x}_2}, f_{x_1,x_2}\}$. Clearly $|cs_B(f)| \le 2^{|B|}$. The Shannon expansion of $f(X)$ with respect to x_i is $f(X) = \overline{x}_i f_{\overline{x}_i} + x_i f_{x_i}$.

In this paper, we shall present our theory using the binary decision diagram (BDD) [Br86a] representation of Boolean functions. The BDD of a function $f(X)$ is a rooted directed acyclic graph $D_f = (V(D_f), E(D_f))$ in which nodes are grouped into $n+1$ levels (1 to $n+1$). Each level i is associated with the variable x_i (i.e., nodes are ordered into levels according to X). The root node is the only node on level 1 and denotes $f(X)$. Nodes on level i denote distinct cofactors of $f(X)$ with respect to the variables $x_1, \cdots, x_{i-1}$. There will be exact two nodes on level $n+1$ which denote Boolean constants 0 and 1 respectively. Given a node $u \in V(D_f)$, the cofactor function associated with u is denoted by f_u and the level that u belongs to is $level(u)$. Every edge $(u,v) \in E(D_f)$ is associated with a value $b(u,v) \in \{0,1\}$. If $level(u) = i$, $level(v) = i+1$, and $(u,v) \in E(D_f)$, then $f_v = f_u|_{x_i = b(u,v)}$. If every node in BDD denotes a *distinct* cofactor function, the BDD is a *reduced* BDD. It has been shown [Br86a] that the reduced BDD of $f(X)$ is a canonical representation of $f(X)$. Figure 1(a) is an example of BDD (top 4 levels) of a function $f(X)$. In this BDD, f_{u_0} denotes $f(X)$ and f_{u_2} denotes $f_{\overline{x}_1\overline{x}_2x_3}$. If $B = \{x_1, x_2, x_3\}$, then $cs_B(f) = \{f_{u_1}, f_{u_2}, f_{u_3}\}$.

2.2. Classical Results

Let $f(X)$ and $g(Y,S,X-B)$ be two functions where B is a prefix subset of X and S is a suffix subset of B. If there is a way to define every variable $y_i \in Y$ as a function $y_i(B)$ such that $f(X) = g(Y(B),S,X-B)$, we say the function $g(Y,S,X-B)$ together with the functions $y_1(B), \cdots, y_{|Y|}(B)$ constitute a *decomposition* of $f(X)$ under the bound set B, or simply $g(Y(B),S,X-B)$ is a decomposition of $f(X)$ in short. The decomposition is *non-trivial* if $|arg(g)| < |arg(f)|$ (i.e. $|Y| + |S| < |B|$). The decomposition is *disjunctive* if $S = \emptyset$, otherwise it is *non-disjunctive*. The decomposition is *simple* if $|Y| + |S| = 1$, otherwise it is *non-simple*. The function g is called the *base function* and the functions $y_i(B)$'s are called *encoding functions*. The variables in S are called *non-disjunctive variables*. The fundamental theorem of the disjunctive decomposition is stated in Theorem 1.

Theorem 1 [AS59, Cu61] There exists a disjunctive decomposition $g(Y(B),X-B)$ of $f(X)$ under the bound set B if and only if $|cs_B(f)| \le 2^{|Y|}$.

Without loss of generality, we impose two restrictions on the functional decomposition. First, $cs_B(f)$ satisfies $2^{|Y|-1} < |cs_B(f)|$. If $|cs_B(f)| \le 2^{|Y|-1}$, according to Theorem 1, there is a more effective decomposition g' of f such that $|arg(g')| < |arg(g)|$. In this case we shall use g' instead of g for decomposition. Second, $f(X)$ and $g(Y(B),X-B)$ satisfy $cs_B(f) = cs_Y(g)$. In general, $cs_B(f)$ might be a proper subset of $cs_Y(g)$, which means $g(Y(B),X-B)$ is a partial function (i.e. g may have don't care). If this is the case, ''extra'' functionality are introduced (into g) during the decomposition of f. Although the don't care is generally a resource for optimization, we consider such a decomposition as ''imprecise''. Therefore, we impose the restriction $cs_B(f) = cs_Y(g)$ in this paper.

Take the function $f(X)$ in Figure 1(a) as an example. Consider the bound set $B = \{x_1, x_2, x_3\}$. In Table 1, the first column shows all possible vertices (x_1, x_2, x_3) and the second column shows the corresponding cofactor node for each vertex in the BDD of $f(X)$. Since $cs_B(f) = \{f_{u_1}, f_{u_2}, f_{u_3}\}$, according to Theorem 1, there exists a decomposition of $f(X)$ which requires two encoding functions. Let $g_1(Y(B),X-B)$ be a decomposition of $f(X)$ such that $(g_1)_{\bar{y}_1\bar{y}_2} = f_{u_1}$, $(g_1)_{\bar{y}_1 y_2} = f_{u_2}$ and $(g_1)_{y_1\bar{y}_2} = (g_1)_{y_1 y_2} = f_{u_3}$. The corresponding encoding functions $y_1(B)$ and $y_2(B)$ in $g_1(Y(B),X-B)$ are specified in the third and fourth columns of Table 1, respectively. The BDD of $g_1(Y,X-B)$ is shown in Figure 1(b). Note that $f(X)$ and $g_1(Y(B),X-B)$ satisfies $cs_B(f) = cs_Y(g_1)$. We might construct another decomposition $g_2(Y(B),X-B)$ of $f(X)$ such that the BDD of $g_2(Y,X-B)$ is shown in Figure 1(c) and the corresponding encoding functions $y_1(B)$ and $y_2(B)$ in $g_2(Y(B),X-B)$ are specified in columns 5 and 6 of Table 1, respectively. Note that $g_2(Y(B),X-B)$ is a partial function since $(g_2)_{y_1 y_2} = f_{u_4}$ is undefined and $cs_B(f) \subset cs_Y(g_2)$. Such a decomposition is not considered since we emphasize on the support minimization of encoding functions in this paper.

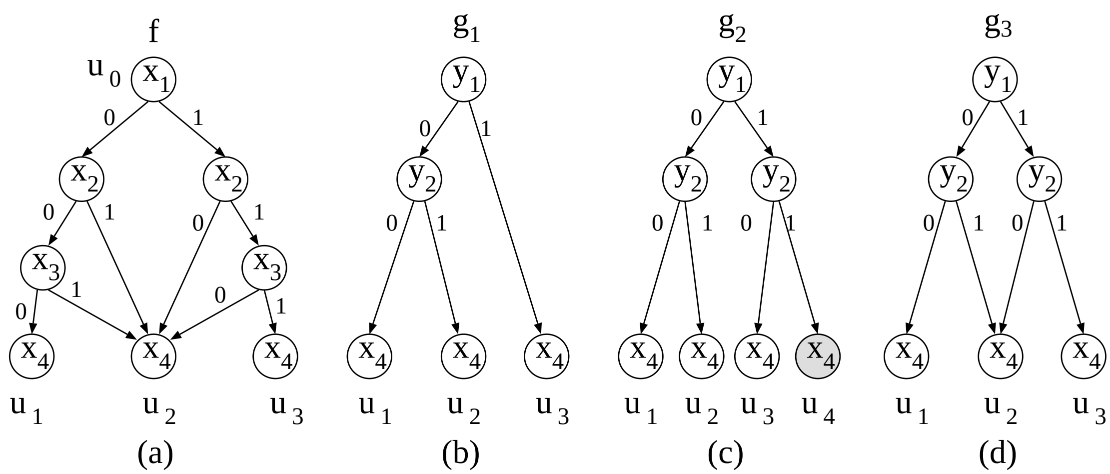

Figure 1 (a) The original function *f*. (b) The base function g_1 is a total function. (c) The base function g_2 is a partial function. (d) A non-disjunctive decomposition g_3 of *f*.

f		g_1		g_2		g_3	
(x_1,x_2,x_3)	node	y_1	y_2	y_1	y_2	y_1	y_2
(0,0,0)	u_1	0	0	0	0	0	0
(0,0,1)	u_2	0	1	0	1	0	1
(0,1,0)	u_2	0	1	0	1	0	1
(0,1,1)	u_2	0	1	0	1	0	1
(1,0,0)	u_2	0	1	0	1	1	0
(1,0,1)	u_2	0	1	0	1	1	0
(1,1,0)	u_2	0	1	0	1	1	0
(1,1,1)	u_3	1	-	1	0	1	1

Table 1 Encoding functions $y_1(B)$ and $y_2(B)$ in decompositions g_1, g_2 and g_3 of *f*.

2.3. Partially Dependent Decomposition

The complexity of a function can be measured by the number of terms or literals in two-level logic design. For LUT-based FPGA architectures, a more reasonable measure would be the number of variables in the function, since this number determines whether a function can be implemented by a single LUT. In general, every encoding function $y_i(B)$ may depend on *all* the variables in the bound set B. A function $y(B)$ is said to be *partially-dependent* on B if $y_{\bar{x}_i} = y_{x_i}$ for some $x_i \in B$, i.e., if $y(B)$ does not depend on x_i. If an encoding function $y_i(B)$ in a decomposition is partially-dependent on the bound set B, it is called a *partially-dependent encoding function*. For example, if $B = \{x_1, x_2, x_3\}$ and $y_i(B) = y_i(x_1, x_2)$, then $y_i(B)$ is a partially-dependent encoding function since it is independent of x_3. The functional decomposition which generates partially-dependent encoding functions is called *partially-dependent functional decomposition*. For a given function, we want to compute a decomposition such that the encoding functions depend on as few variables as possible. The best possibility would be the case where an encoding function depends on only one variable in the bound set. In general, a partially-dependent encoding function depends on variables in a proper subset of the bound set. We now give a sufficient and necessary condition for a (partially-dependent) decomposition $g(Y(B), X-B)$ of $f(X)$ such that $y_1(B) = y_1(x_1, x_2)$. For the general case where $y_1(B) = y_1(x_1, x_2, \cdots, x_m)$ and $m < |B|$, the condition is similar.

Lemma 1 Let B be a prefix subset of X and $|B| \geq 2$. There exists a disjunctive decomposition $g(Y(B), X-B)$ of $f(X)$ under the bound set B such that $y_1(B) = y_1(x_1, x_2)$ if and only if $cs_{\{x_1, x_2\}}(f)$ can be partitioned into two sets cs_1 and cs_2 such that

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

Proof (**only if**) Suppose $g(Y(B), X-B)$ is a disjunctive decomposition of $f(X)$ under the bound set B such that $y_1(B) = y_1(x_1, x_2)$. Let $cs_1 = \{ f(b_1, b_2, x_3, \cdots, x_n) \mid y_1(b_1, b_2) = 1 \}$. Then

$$\bigcup_{h \in cs_1} cs_B(h) = \{ f(b_1, b_2, \cdots, b_{|B|}, x_{|B|+1}, \cdots, x_n) \mid y_1(b_1, b_2) = 1 \}$$

$$= \{ g(1, y_2(B), \cdots, y_{|Y|}(B), x_{|B|+1}, \cdots, x_n) \mid y_1(x_1, x_2) = 1 \}$$

$$\subseteq \{ g(1, c_2, \cdots, c_{|Y|}, x_{|B|+1}, \cdots, x_n) \mid c_2, \cdots, c_{|Y|} \in \{0, 1\} \}$$

$$= cs_Y(g_{y_1}).$$

Since $|cs_Y(g_{y_1})| \leq 2^{|Y|-1}$, we have $|\bigcup_{h \in cs_1} cs_B(h)| \leq 2^{|Y|-1}$. Following similar steps, we define $cs_2 = \{ f(b_1, b_2, x_3, \cdots, x_n) \mid y_1(b_1, b_2) = 0 \}$ and we can derive $|\bigcup_{h \in cs_2} cs_B(h)| \leq 2^{|Y|-1}$. Since cs_1 and cs_2 form a partition of $cs_{\{x_1, x_2\}}(f)$, the (only if) condition is proved.

(**if**) Let $Y = \{ y_1, \ldots, y_t \}$ and assume $cs_{\{x_1, x_2\}}(f)$ can be partitioned into two sets cs_1 and cs_2 such that $|\bigcup_{h \in cs_1} cs_B(h)| \leq 2^{|Y|-1}$ and $|\bigcup_{h \in cs_2} cs_B(h)| \leq 2^{|Y|-1}$. Define $y_1(B) = y_1(x_1, x_2)$ as follows:

$$y_1(b_1, b_2) = \begin{cases} 1 & \text{if } f(b_1, b_2, x_3, \cdots, x_n) \in cs_1 \\ 0 & \text{if } f(b_1, b_2, x_3, \cdots, x_n) \in cs_2 \end{cases}$$

Let $f_1(X)$ be the partial function such that $f_1(X) = f(X)$ if $y_1(x_1, x_2) = 1$ and $f_1(X)$ is undefined if $y_1(x_1, x_2) = 0$. Then $\bigcup_{h \in cs_1} cs_B(h) = cs_B(f_1)$, which implies $|cs_B(f_1)| \leq 2^{|Y|-1}$. According to Theorem 1, there exists a decomposition $g_1(W(B), X-B)$ of $f_1(X)$ where $W = \{ w_2, \ldots, w_{|Y|} \}$. Note that each $w_i(B)$ is a partial function defined on the domain $y_1^{-1}(1)$. Similarly, let $f_0(X)$ be the partial function such that $f_0(X) = f(X)$ if $y_1(x_1, x_2) = 0$ and $f_0(X)$ is undefined if

$y_1(x_1,x_2)=1$. Then $\bigcup_{h \in cs_2} cs_B(h)=cs_B(f_0)$, which implies $|cs_B(f_0)| \le 2^{|Y|-1}$. According to Theorem 1, there exists a decomposition $g_0(Z(B),X-B)$ of $f_0(X)$ where $Z=\{z_2, \ldots, z_{|Y|}\}$. Each $z_i(B)$ is a partial function defined on the domain $y_1^{-1}(0)$. Let $\hat{Y}=Y-\{y_1\}$. Define $y_i(y_1,w_i,z_i)=y_1w_i+\bar{y}_1z_i$ for $2 \le i \le |Y|$ and $g(Y,X-B)=y_1g_1(\hat{Y},X-B)+\bar{y}_1g_0(\hat{Y},X-B)$. We claim $g(Y(B),X-B)$ is a decomposition of $f(X)$ under the bound set B. To see this, note that $y_i=w_i$ when $y_1=1$ and $y_i=z_i$ when $y_1=0$ for $2 \le i \le |Y|$, and

$$
\begin{aligned}
f(X) &= y_1(x_1,x_2)f_1(X)+\overline{y_1(x_1,x_2)}f_0(X) \\
&= y_1(x_1,x_2)g_1(W(B),X-B)+\overline{y_1(x_1,x_2)}g_0(Z(B),X-B) \\
&= y_1(x_1,x_2)g_1(\hat{Y}(B),X-B)+\overline{y_1(x_1,x_2)}g_0(\hat{Y}(B),X-B) \\
&= g(Y(B),X-B)
\end{aligned}
$$

Since $y_1(B)=y_1(x_1,x_2)$, we proved the (if) condition. □

Theorem 2 Suppose B is a prefix subset of X and $|B| > m$. There exists a decomposition $g(Y(B),X-B)$ of $f(X)$ under the bound set B such that $y_1(B)=y_1(x_1, \cdots, x_m)$ if and only if $cs_{\{x_1, \cdots, x_m\}}(f)$ can be partitioned into two sets cs_1 and cs_2 such that

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

Proof Similar to the proof of Lemma 1 and omitted.

We illustrate the theorem by the following example. Consider the function $f(X)$ in Figure 2(a). Assume $B=\{x_1,x_2,x_3\}$. The first two columns of Table 2 show the vertices (x_1,x_2,x_3) and their corresponding cofactor nodes in the BDD of $f(X)$. Since $cs_B(f)=\{f_{u_0},f_{u_1},f_{u_2},f_{u_4}\}$, according to Theorem 1, there exists a decomposition of $f(X)$ which requires two encoding functions. Let $h_1(Y(B),X-B)$ be a decomposition of $f(X)$ such that $(h_1)_{\bar{y}_1\bar{y}_2}=f_{u_1}$, $(h_1)_{\bar{y}_1y_2}=f_{u_2}$, $(h_1)_{y_1\bar{y}_2}=f_{u_3}$ and $(h_1)_{y_1y_2}=f_{u_4}$. The BDD of $h_1(Y,X-B)$ is shown in Figure 2(b). The corresponding encoding functions $y_1(B)$ and $y_2(B)$ in $h_1(Y(B),X-B)$ are specified in the third and fourth columns of Table 2, respectively. We observe that both $y_1(B)$ and $y_2(B)$ depend on every variable in the bound set B. We now 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 partition $cs_{\{x_1,x_2\}}(f)$ into two sets $\{f_{u_0}\}$ and $\{f_{u_1},f_{u_4}\}$. Then we have $|cs_B(f_{u_0})|=2$ and $|cs_B(f_{u_1}) \cup cs_B(f_{u_4})|=2$. According to Lemma 1, there exists a decomposition of $f(X)$ such that $y_1(B)=y_1(x_1,x_2)$. We construct such a decomposition $h_2(Y(B),X-B)$ of $f(X)$ by defining $(h_2)_{\bar{y}_1\bar{y}_2}=f_{u_1}$, $(h_2)_{\bar{y}_1y_2}=f_{u_4}$, $(h_2)_{y_1\bar{y}_2}=f_{u_2}$ and $(h_2)_{y_1y_2}=f_{u_3}$. The BDD of $h_2(Y,X-B)$ is shown in Figure 2(c). The corresponding encoding functions $y_1(B)$ and $y_2(B)$ in $h_2(Y(B),X-B)$ are specified in columns 5 and 6 of Table 2, respectively. We observe that for $h_2(Y(B),X-B)$,

$y_1(x_1,x_2,0)=y_1(x_1,x_2,1)$. In other words $y_1(B)=y_1(x_1,x_2)$. Therefore $h_2(Y,X-B)$ is a partially-dependent decomposition of $f(X)$.

It might appear there is no advantage in considering the partially-dependent decomposition since it still costs one LUT to implement a partially-dependent encoding function (except the simplest case which will be discussed in the next section). However, the synthesis of LUT networks are benefited from the partially-dependent decomposition for the following reasons. First, many FPGA architectures allow multiple LUTs to be placed into one programmable logic block (PLB) in an FPGA with certain input constraints. Since the partially-dependent encoding functions depend on fewer variables, the number of PLBs can be reduced. Second, LUT packing algorithms, such as FlowPack [ChCD92], can further decrease the number of LUTs by packing small LUTs into their fanout LUTs. Therefore, use of partially-dependent encoding functions can

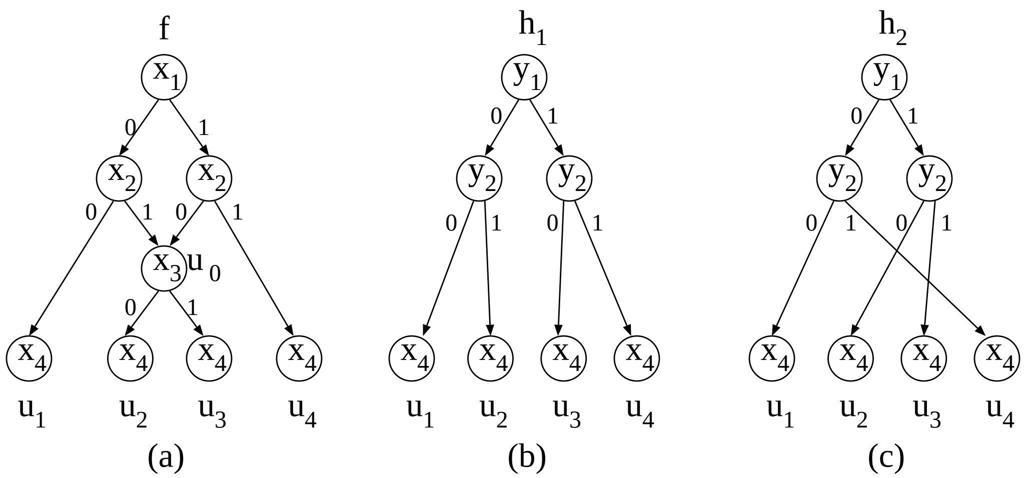

Figure 2 (a) The original function f. (b) A decomposition h_1 where y_1 is fully-dependent on B. (c) A partially-dependent decomposition h_2 where y_1 does not depend on x_3.

f		h_1		h_2	
(x_1,x_2,x_3)	node	y_1	y_2	y_1	y_2
(0,0,0)	u_1	0	0	0	0
(0,0,1)	u_1	0	0	0	0
(0,1,0)	u_2	0	1	1	0
(0,1,1)	u_3	1	0	1	1
(1,0,0)	u_2	0	1	1	0
(1,0,1)	u_3	1	0	1	1
(1,1,0)	u_4	1	1	0	1
(1,1,1)	u_4	1	1	0	1

Table 2 Encoding functions $y_1(B)$ and $y_2(B)$ of decompositions h_1 and h_2 of f. Note that $y_1(B)=y_1(x_1,x_2)$ in h_2.

help those packing algorithms to achieve better results. Third, it enables possible decompositions with oversized bound sets (larger than the LUT input size K). As long as the encoding functions depend on at most K variables, it does not matter how large the bound set is. Hence the partially-dependent functional decomposition allows us to explore a larger solution space when transforming a given network into an LUT network.

Note that Theorem 2 does not require $(\bigcup_{h \in cs_1} cs_B(h)) \cap (\bigcup_{h \in cs_2} cs_B(h)) = \emptyset$ (although in the illustrated example $cs_B(f_{u_0}) \cap (cs_B(f_{u_1}) \cup cs_B(f_{u_4})) = \emptyset$). A decomposition $g(Y(B), X-B)$ of $f(X)$ under the bound set B is said to be *strict* (or *uni-coded*) if $cs_Y(g_{\bar{y}_i}) \cap cs_Y(g_{y_i}) = \emptyset$ for every $y_i \in Y$. In other words, there is exactly one path from the root node to each cofactor node in $cs_Y(g)$ in the BDD of $g(Y, X-B)$. Sometimes a function may have a non-strict partially-dependent decomposition but no strict partially-dependent decomposition. An example is the function f in Figure 1(a) which does not have a strict non-disjunctive decomposition (special case of partially-dependent decomposition as will be discussed in the next section) but do have a non-strict non-disjunctive decomposition g_3 as shown in Figure 1(d). Hence Theorem 2 is a general theorem for the existence of the partially-dependent decomposition of a function.

2.4. Non-Disjunctive Decomposition

The non-disjunctive decomposition is considered more general than the disjunctive decomposition previously. One way to consider the non-disjunctive decomposition is to treat it as a disjunctive decomposition of partial function with additional variables, or simply think of the non-disjunctive variables as encoding functions [RoKa62]. A sufficient and necessary condition for non-disjunctive decomposition was given in [LaPV93] using the non-disjunctive BDD cutset formulation. However, the formulation is complex, difficult to use in practice (computationally inefficient), and provides little intuition. In this section, we analyze non-disjunctive decomposition from the perspective of partially-dependent decomposition and derive a sufficient and necessary condition for it. We first give the following simple but important lemma.

Lemma 2 Let B be a prefix subset of X and S a suffix subset of B. If there exists a non-disjunctive decomposition $h(Z(B), S, X-B)$ of $f(X)$ under the bound set B, then there exists a disjunctive decomposition $g(Y(B), X-B)$ of $f(X)$ under the bound set B such that $|Z| + |S| = |Y|$.

Proof Since $f(X) = h(Z(B), S, X-B)$, $z_i = z_i(B)$ and $S \subseteq B$, we have $cs_{Z \cup S}(h) = cs_B(f)$. As a result, $|cs_B(f)| = |cs_{Z \cup S}(h)| \le 2^{|Z|+|S|}$. According to Theorem 1, there exists a disjunctive decomposition $g(Y(B), X-B)$ of $f(X)$ under the bound set B such that $|Y| = |Z| + |S|$. □

Notice that $|arg(h)| = |arg(g)|$. In other words, $h(Z(B), S, X-B)$ and $g(Y(B), X-B)$ are equally effective for decomposing $f(X)$ in terms of reducing the input size of the base function. However, there is no need to implement non-disjunctive variables in S by LUTs. As a result, the non-disjunctive decomposition of a function will reduce the implementation cost. Without loss of generality, let S be a *prefix* subset of B and rewrite h as $h(S,Z,X-B)$ (i.e. moving S in front of Z in $arg(h)$). Then the non-disjunctive decomposition $h(S,Z(B),X-B)$ of $f(X)$ is simply a partially-dependent decomposition $g(Y(B),X-B)$ of $f(X)$ such that $y_1(B)=x_1, \cdots, y_{|S|}(B)=x_{|S|}$. A sufficient and necessary condition for a decomposition such that $y_1(B)=x_1$ can be derived from Theorem 2 by assigning $m=1$ as shown in the following lemma.

Lemma 3 There exists a decomposition $g(Y(B),X-B)$ of $f(X)$ under the bound set B such that $y_1(B)=x_1$ if and only if $|cs_B(f_{x_1})| \le 2^{|Y|-1}$ and $|cs_B(f_{\bar{x}_1})| \le 2^{|Y|-1}$.

Proof Since $cs_{\{x_1\}}(f)$ contains only two cofactors $f_{\bar{x}_1}$ and f_{x_1}, according to Theorem 2, it is obviously true. □

Theorem 3 Let S and B be prefix subsets of X and $|S| \le |B|$. There exists a decomposition $g(Y(B),X-B)$ of $f(X)$ under the bound set B such that $y_i(B)=x_i$ for $1 \le i \le |S|$ if and only if $|cs_B(h)| \le 2^{|Y|-|S|}$ for every $h \in cs_S(f)$.

Proof We prove it by induction on $|S|$. Lemma 3 proves the case of $|S|=1$. Assume it holds for $|S|=m-1$. We now prove it holds for $|S|=m$.

(**only if**) Suppose $g(Y(B),X-B)$ is a decomposition of $f(X)$ and $y_i(B)=x_i$ for $1 \le i \le |S|=m$. Let $\hat{X}=X-\{x_1\}$, $\hat{Y}=Y-\{y_1\}$, $\hat{S}=S-\{x_1\}$ and $\hat{B}=B-\{x_1\}$. Consider the function $f_{x_1}(\hat{X})$. We have

$$f_{x_1}(\hat{X})=f(1,x_2, \cdots, x_n)$$

$$=g(1,y_2(1,x_2, \cdots, x_{|B|}), \cdots, y_{|Y|}(1,x_2, \cdots, x_{|B|}),x_{|B|+1}, \cdots, x_n)$$

$$=g_{y_1}(\hat{Y}_{x_1}(\hat{B}),\hat{X}-\hat{B})$$

Hence $g_{y_1}(\hat{Y}_{x_1}(\hat{B}),\hat{X}-\hat{B})$ is a decomposition of $f_{x_1}(\hat{X})$ under the bound set $\hat{B}$. Since $y_i(B)=x_i$ for $2 \le i \le m$, from the induction hypothesis, we have $|cs_{\hat{B}}(h)| \le 2^{|\hat{Y}|-|\hat{S}|}=2^{|Y|-|S|}$ for every $h \in cs_{\hat{S}}(f_{x_1})$. Similarly, $|cs_{\hat{B}}(h)| \le 2^{|Y|-|S|}$ for every $h \in cs_{\hat{S}}(f_{\bar{x}_1})$. Since $cs_S(f)=cs_{\hat{S}}(f_{\bar{x}_1}) \cup cs_{\hat{S}}(f_{x_1})$, we have $|cs_B(h)|=|cs_{\hat{B}}(h)| \le 2^{|Y|-|S|}$ for every $h \in cs_S(f)$. The (only if) condition is proved.

(**if**) Suppose $|cs_B(h)| \le 2^{|Y|-|S|}=2^{|Y|-m}$ for every $h \in cs_S(f)$. Let $\hat{X}=X-\{x_1\}$, $\hat{S}=S-\{x_1\}$ and $\hat{B}=B-\{x_1\}$. Consider the function $f_{x_1}(\hat{X})$. Since every cofactor $h \in cs_{\hat{S}}(f_{x_1})$ is in $cs_S(f)$, we have $|cs_{\hat{B}}(h)|=|cs_B(h)| \le 2^{|Y|-m}=2^{|\hat{Y}|-(m-1)}$. From the induction hypothesis, there exists a decomposition $g_1(W(\hat{B}),\hat{X}-\hat{B})$ of $f_{x_1}(\hat{X})$ under the bound set $\hat{B}$, where $W=\{w_2, \ldots, w_{|Y|}\}$, such that $w_i(\hat{B})=x_i$ for $2 \le i \le m$. Similarly, there exists a decomposition $g_0(Z(\hat{B}),\hat{X}-\hat{B})$ of $f_{\bar{x}_1}(\hat{X})$ under the bound set $\hat{B}$, where $Z=\{z_2, \ldots, z_{|Y|}\}$, such that $z_i(\hat{B})=x_i$ for

$2 \le i \le m$. Let $\hat{Y} = Y - \{y_1\}$. Define $y_1 = x_1$, $y_i(x_1, w_i, z_i) = x_1 w_i + \bar{x}_1 z_i$ for $2 \le i \le |Y|$ and $g(Y, X-B) = y_1 g_1(\hat{Y}, X-B) + \bar{y}_1 g_0(\hat{Y}, X-B)$. We claim $g(Y(B), X-B)$ is a decomposition of $f(X)$ under the bound set B. To see this, note that $y_i = w_i$ when $x_1 = 1$ and $y_i = z_i$ when $x_1 = 0$ for $2 \le i \le m$, and

$$
\begin{aligned}
f(X) &= x_1 f_{x_1}(\hat{X}) + \bar{x}_1 f_{\bar{x}_1}(\hat{X}) \\
&= x_1 g_1(W(\hat{B}), \hat{X} - \hat{B}) + \bar{x}_1 g_0(Z(\hat{B}), \hat{X} - \hat{B}) \\
&= x_1 g_1(\hat{Y}_{x_1}(\hat{B}), X - B) + \bar{x}_1 g_0(\hat{Y}_{\bar{x}_1}(\hat{B}), X - B) \\
&= g(y_1, y_2(B), \cdots, y_m(B), X - B)
\end{aligned}
$$

Since for $2 \le i \le m$, $y_i(B) = x_1 w_i(\hat{B}) + \bar{x}_1 z_i(\hat{B}) = x_1 x_i + \bar{x}_1 x_i = x_i$, we have $y_i(B) = x_i$ for $2 \le i \le m$. This completes the proof. □

Let us take the function $f(X)$ in Figure 1(a) as an example. The bound set $B = \{x_1, x_2, x_3\}$. Note that $cs_B(f_{\bar{x}_1}) = \{f_{u_1}, f_{u_2}\}$ and $cs_B(f_{x_1}) = \{f_{u_2}, f_{u_3}\}$. According to Lemma 3, there is a decomposition $g_3(Y(B), X-B)$ of $f(X)$ such that $y_1(B) = x_1$. We construct $g_3(Y(B), X-B)$ by defining $(g_3)_{\bar{y}_1\bar{y}_2} = f_{u_1}$, $(g_3)_{\bar{y}_1 y_2} = (g_3)_{y_1\bar{y}_2} = f_{u_2}$ and $(g_3)_{y_1 y_2} = f_{u_3}$. The BDD of $g_3(Y, X-B)$ is shown in Figure 1(d). The encoding functions $y_1(B)$ and $y_2(B)$ in $g_3(Y(B), X-B)$ are specified in columns 7 and 8 of Table 1, respectively. Note that columns 1 and 7 are identical, i.e. $y_1(B) = x_1$. Also note that $g_3(Y(B), X-B)$ is a non-strict decomposition of $f(X)$ since $g_{\bar{y}_1 y_2} = g_{y_1\bar{y}_2} = f_{u_2}$. If we consider only strict decomposition for $f(X)$, there will be no non-disjunctive decomposition found.

A theorem similar to Theorem 3 had been presented in [LaPV93]. The difference is the variable ordering in $arg(f)$. In our case the non-disjunctive set S is a *prefix* subset of the bound set B while in [LaPV93] S is a *suffix* subset of B. Our variable ordering makes the testing condition for the existence of a non-disjunctive decomposition much simpler. Of course, we adopt this ordering because we analyze the non-disjunctive decomposition from the perspective of partially-dependent decomposition.

2.5. Multiple-Output Decomposition

The number of encoding functions might decrease when multiple functions are decomposed together. Let $g_1(Y(B), X-B)$ and $g_2(Z(B), X-B)$ be decompositions of $f_1(X)$ and $f_2(X)$ under the bound set B, respectively. Assume $|Y| = |Z|$. If not, we introduce dummy variables to make them equal. When $f_1(X)$ and $f_2(X)$ are decomposed independently, it requires $|Y| + |Z|$ encoding functions. But if $f_1(X)$ and $f_2(X)$ are decomposed together and some common encoding functions can be found for them, then fewer encoding functions are needed. For example, if $y_1(B) = z_1(B)$, then only $|Y| + |Z| - 1$ encoding functions are generated. In order to reduce the implementation cost, we are interested in computing a maximal set of common encoding functions for a given set of functions. We define the multiple-output decomposition problem as follows.

Multiple-Output Decomposition Problem Given a multiple-output function with outputs $f_1(X), \cdots, f_m(X)$ such that each of them is decomposable under a bound set B, compute a maximal set of encoding functions which are common to *all* the outputs.

The multiple-output decomposition problem can be reduced to the problem of computing partially-dependent encoding functions. Consider the decomposition of a two-output function with outputs $f_1(X)$ and $f_2(X)$. Assume each output is decomposable under a bound set B. Define the auxiliary function $F(s,X) = \bar{s} f_1(X) + s f_2(X)$ where s is a selection variable. We have the following theorem.

Theorem 4 Let B be a prefix subset of X and $\hat{B} = \{ s \} \cup B$. A function $w(B)$ is a common encoding function in the decomposition of $f_1(X)$ and $f_2(X)$ under the bound set B if and only if $w(B)$ is a partially-dependent encoding function in a non-disjunctive decomposition of $F(s,X)$ under the bound set $\hat{B}$ with s as a non-disjunctive variable.

Proof (**only if**) Let $g_1(Y(B), X-B)$ and $g_2(Z(B), X-B)$ be decompositions of $f_1(X)$ and $f_2(X)$ under the bound set B, respectively, and $y_1(B) = z_1(B) = w(B)$. Let $W = \{ w_1, \cdots, w_{|Y|} \}$. Define $w_i(s, y_i, z_i) = \bar{s} y_i + s z_i$ for $1 \le i \le |Y|$ and $G(s, W, X-B) = \bar{s} g_1(W, X-B) + s g_2(W, X-B)$. We claim $G(s, W, X-B)$ together with $w_i(\hat{B}) = w_i(s, y_i(B), z_i(B)) = \bar{s} y_i(B) + s z_i(B)$ for $1 \le i \le |Y|$ constitute a non-disjunctive decomposition of $F(s,X)$ under the bound set $\hat{B}$ with s as a non-disjunctive variable. To see this, note that $w_i(\hat{B})|_{s=0} = y_i(B)$ and $w_i(\hat{B})|_{s=1} = z_i(B)$, and

$$G(s, W(\hat{B}), X-B) = \bar{s} g_1(W(\hat{B})|_{s=0}, X-B) + s g_2(W(\hat{B})|_{s=1}, X-B)$$

$$= \bar{s} g_1(Y(B), X-B) + s g_2(Z(B), X-B) = \bar{s} f_1(X) + s f_2(X) = F(s,X).$$

Since $s \in arg(F)$ and $s \in arg(G)$, $G(s, W(\hat{B}), X-B)$ is a non-disjunctive decomposition of $F(s,X)$ with s as a non-disjunctive variable. Since $y_1(B) = z_1(B) = w(B)$, we have $w_1(\hat{B}) = w(B)$. Hence $w(B)$ is a partially-dependent encoding function in the non-disjunctive decomposition of $F(s,X)$ under the bound set $\hat{B}$ with s as a non-disjunctive variable. The (only if) condition is proved.

(**if**) Suppose $G(s, W(\hat{B}), X-B)$ is a non-disjunctive decomposition of $F(s,X)$ under the bound set $\hat{B}$ with s as a non-disjunctive variable and $w_1(\hat{B}) = w(B)$ is a partially-dependent encoding function of the decomposition. Let $Y = \{ y_1, \ldots, y_{|W|} \}$. Define $y_i(B) = w_i(\hat{B})|_{s=0}$ for $1 \le i \le |W|$. Then $G_{\bar{s}} = G(0, W(\hat{B})|_{s=0}, X-B) = G(0, Y(B), X-B)$. Since $F(s,X) = G(s, W(\hat{B}), X-B)$, we have $f_1(X) = F_{\bar{s}} = G_{\bar{s}} = G(0, Y(B), X-B)$. Hence $G(0, Y(B), X-B)$ is a decomposition of $f_1(X)$ under the bound set B. Similarly, let $Z = \{ z_1, \ldots, z_{|W|} \}$ and define $z_i(B) = w_i(\hat{B})|_{s=1}$ for $1 \le i \le |W|$. We have $f_2(X) = F_s = G_s = G(1, Y(B), X-B)$. So $G(1, Y(B), X-B)$ is a decomposition of $f_2(X)$ under the bound set B. Since $w_1(\hat{B}) = w(B)$, according to the definition of $y_i(B)$ and $z_i(B)$, we have $y_1(B) = z_1(B) = w(B)$. Hence $w(B)$ is a common encoding function in the decomposition of $f_1(X)$ and $f_2(X)$ under the bound set B. This completes the proof. □

Theorem 4 allows us to reduce the two-output decomposition problem to the problem of computing partially-dependent encoding functions. For a given m-output function, we can create an auxiliary function with $\lceil log(m) \rceil$ selection variables and apply the reduction in a similar way. Note that this approach is able to compute non-strict common encoding functions since the partially-dependent decomposition in our formulation can be non-strict.

2.6. Algorithms for Maximal Partially-Dependent Decompositions

We now give an algorithm to compute, for a given function $f(X)$ and a bound set B, a partially-dependent decomposition $g(Y(B), X-B)$ of $f(X)$ such that $Y(B)$ contains a maximal number of partially-dependent encoding functions. Our approach first computes a set of *all* feasible partially-dependent encoding functions according to the condition given in Theorem 2 and then chooses a maximal number of *compatible* encoding functions from the set to compose $g(Y(B), X-B)$.

Given a function $f(X)$ and a bound set B, let $Y_1(f)$ be the set of all feasible partially-dependent encoding functions. For an encoding function $w(z_1, z_2, \cdots, z_m) \in Y_1(f)$ where $z_i \in B$ for $1 \le i \le m < |B|$, let $cs_1(w)$ and $cs_2(w)$ be a partition of $cs_{\{z_1, \ldots, z_m\}}(f)$ according to $w(B) = 1$ or 0, respectively, and let CS_w and $CS_{\overline{w}}$ represent $\bigcup_{h \in cs_1(w)} cs_B(h)$ and $\bigcup_{h \in cs_2(w)} cs_B(h)$, respectively.

Lemma 4 Let $w_1(B), \cdots, w_k(B)$ be k partially-dependent encoding functions in $Y_1(f)$ and $\hat{w}_i \in \{w_i, \overline{w}_i\}$. There exists a decomposition $g(Y(B), X-B)$ of $f(X)$ such that $y_i(B) = w_i(B)$ for $1 \le i \le k \le |Y|$ if and only if for all possible assignments of $\hat{w}_1, \cdots, \hat{w}_k$,

$$|CS_{\hat{w}_1} \cap CS_{\hat{w}_2} \cap \cdots \cap CS_{\hat{w}_k}| \le 2^{|Y|-k}$$

Proof Omitted.

We say $w_1(B), \cdots, w_k(B)$ are *compatible* encoding functions if they satisfy the condition in Lemma 4. Given a function $f(X)$ and a bound set B, we compute a maximal set of compatible encoding functions as follows. We first compute the set $Y_1(f)$ (to be discussed). Next, for every two functions $w_i(B)$ and $w_j(B)$ in $Y_1(f)$, we examine whether they are compatible encoding functions according to Lemma 4. If it is the case, we form a pair (w_i, w_j) and add it to a set $Y_2(f)$. As a result, $Y_2(f)$ will be the set of all compatible pairs of partially-dependent encoding functions. In the third step, we search for all compatible triples of encoding functions (w_i, w_j, w_l) subject to (w_i, w_j), (w_j, w_l) and (w_l, w_i) are all in $Y_2(f)$. We continue this process until the first $Y_k(f) = \emptyset$ is found. A maximal set of compatible encoding functions for $f(X)$ under the bound set B can be any member in $Y_{k-1}(f)$. The algorithm is outlined in Figure 3.

1 **function** Maximal_Partially_Dependent(f, X, B)
2 /* f = input function, X = input variable set , B = bound set*/

3 compute $Y_1(f)$
4 **for** $k=2$ **to** ∞ **do**
5     $Y_k(f)=\varnothing$
6   **for** every k members $s_1, \ldots, s_k \in Y_{k-1}(f)$ satisfying $|s_1 \cup \cdots \cup s_k| = k$ **do**
7         $S=s_1 \cup \cdots \cup s_k$
8   **if** S satisfies Lemma 4 **then** add S to $Y_k(f)$
9   **if** $Y_k(f)=\varnothing$ **return** a member in $Y_{k-1}(f)$

Figure 3 An algorithm for computing a maximal set of partially-dependent encoding functions.

We now compute $Y_1(f)$. Given a function $f(X)$ and a bound set B, we compute all the possible partially-dependent encoding functions $w(z_1, \ldots, z_m)$ for every subset $\{z_1, \ldots, z_m\} \subset B$ as follows. First, we compute $cs_{\{z_1, \ldots, z_m\}}(f)$. Next, we find a partition cs_1 and cs_2 of $cs_{\{z_1, \ldots, z_m\}}(f)$ and examine whether cs_1 and cs_2 satisfy Theorem 2. If it is the case, we have found a partially-dependent encoding function $w((z_1, \ldots, z_m)$ and add it to $Y_1(f)$. We exhaust every possible partition of $cs_{\{z_1, \ldots, z_m\}}(f)$ in this step. It should not be hard to see, after testing all $m < |B|$, we have computed $Y_1(f)$ at the end of this procedure. The algorithm is outlined in Figure 4.

For the case of computing a maximal set of non-disjunctive variables, we simply let $m=1$ in Figure 4. Alternatively, we can replace the condition on line 7 of Figure 4 by Lemma 3 and the condition on line 8 of Figure 3 by Theorem 3, respectively, to reduce the computation time.

1 **function** Compute_Y1(f, X, B)
2 /* f = input function, X = input variable set , B = bound set*/

3 $Y_1(f)=\varnothing$
4 **for** every set $\{z_1, \ldots, z_m\} \subset B$ **do**
5   compute $cs_{\{z_1, \ldots, z_m\}}(f)$
6   **for** every partition cs_1 and cs_2 of $cs_{\{z_1, \ldots, z_m\}}(f)$ **do**
7   **if** cs_1 and cs_2 satisfies Theorem 2 **then**
8   **for** every $f(b_1, \cdots, b_m, x_{m+1}, \cdots, x_n) \in cs_1$ **do**
9                 $w(b_1, \cdots, w_m)=1$
10   **for** every $f(b_1, \cdots, b_m, x_{m+1}, \cdots, x_n) \in cs_0$ **do**
11                 $w(b_1, \cdots, w_m)=0$
12             $Y_1(f)=Y_1(f) \cup \{w(z_1, \ldots, z_{m)}\}$
13 **return** $Y_1(f)$

Figure 4 An algorithm for computing $Y_1(f)$.

2.7. Summary

We present the theory of partially-dependent decomposition in this section. The sufficient and necessary condition for the existence of partially-dependent decomposition is given in Theorem 2. We derive theorems for non-disjunctive decomposition (Theorem 3) and multiple-output decomposition (Theorem 4) from the perspective of partially-dependent decomposition. We also give algorithms for computing a maximal set of partially-dependent encoding functions as well as non-disjunctive variables for a given function $f(X)$ and a bound set B in this section.

3. Applications to LUT-based FPGA

The lookup-table (LUT) based FPGA is one of the most popular FPGA architecture in recent years. A K-input LUT (K-LUT) is a function cell which can implement an arbitrary function of up to K inputs. The methods which transform a given network into an equivalent LUT network can be classified into three categories. The first category includes technology mapping approaches which treat networks as graphs and use LUTs to cover the graphs [FrRV91a, CoDi94a]. Large fanin gates have to be decomposed into gates of K or fewer fanins in order to produce a mapping solution. The second category includes logic synthesis approaches in which the functionality of the input network is represented by certain Boolean structure (e.g. truth table) and then decomposed into subfunctions of at most K variables [ChMa92, LaPV93] such that these subfunctions altogether compute the original function and individually can be implemented by a K-LUT. Methods in the third category combine technology mapping and logic synthesis [MuNS90, SaTh93, CoDi93b]. In this approach, technology mapping and logic synthesis are applied in serial or interleavingly to improve the performance and cost. Our approach falls into the third category. We apply technology mapping followed by iterative resynthesis of LUT subnetworks to obtain the final LUT network.

Given an input network, our primary goal is to generate an LUT network of minimal level (i.e. depth), and the secondary goal is area minimization. Several technology mapping algorithms have been proposed to achieve the first objective [FrRV91c, MuSB91a, SaTh93, ChCD92, CoDi94a] and reported good results. In particular, the FlowMap [CoDi94a] algorithm produces depth-optimal mapping solutions for K-bounded networks. To further decrease the number of levels in the final LUT network, logic synthesis has been integrated into the technology mapping process [CoDi93b] to overcome the combinatorial limit of technology mapping.

In this paper, we propose an approach which applies technology mapping followed by logic synthesis to achieve the mapping goal. We first employ a depth-oriented technology mapper such as FlowMap (or its enchancement CutMap [CoHw95]) to produce an initial LUT network of small depth. We then improve the depth as well as area of the mapping solution by resynthesis of LUT subnetworks. Two advantages are in our approach. First, it allows us to apply resynthesis iteratively instead of just once in the mapping phase. We experience continuing reduction in area through a subsequent of resynthesis steps in our experiments. Second, it is possible to calculate a precise gain of a resynthesis in our approach while this gain is difficult to estimate in the technology mapping process since an LUT network is not yet completely constructed. The precise gain enables us to make better decisions for area minimization. We now describe our approach in detail.

3.1. Notations

In order to describe our algorithm, we introduce some notations. A combinational network can be represented by a directed acyclic graph (DAG) where a node represents a function cell and a directed edge (u,v) represents a connection from the output of node u to the input of node v. A primary input (PI) is a node of zero fanin and a primary output (PO) is a node without outgoing edge. The height of a node v, denoted $height(v)$, is the number of edges on the longest path from any PI to v. A PI node has a height of zero. Let $input(v)$ be the set of fanin nodes of node v and $output(v)$ be the set of fanout nodes of node v. Given a subgraph H of a network, we use $input(H)$ to denote the set of distinct nodes outside H which supply inputs to nodes in H. A fanout-free cone (FFC) at v, denoted FFC_v, is a cone rooted at v such that for every $u \neq v \in FFC_v$, $output(u) \subseteq FFC_v$ [CoDi94b]. The MFFC at v, denoted $MFFC_v$, is the FFC at v which contains a maximal number of nodes. An important property of MFFCs is that MFFCs are either disjoint or one contains another [CoLB94]. As a result, any network can be uniquely partitioned into MFFCs.

Given a node v in network N, let N_v denote the subnetwork consisting of node v and all the predecessors of v. A cut in N_v is a partition $(X_v, \overline{X}_v)$ of nodes in N_v such that PI nodes are in X_v and $v \in \overline{X}_v$. The cutset of a cut, denoted $n(X_v, \overline{X}_v)$, is defined as $input(\overline{X}_v)$. The height of a cut, denoted $height(X_v, \overline{X}_v)$, is the maximal height of nodes in $n(X_v, \overline{X}_v)$. A min-cut of height p in N_v can be computed by the max-flow min-cut algorithm [CoDi94a].

Let N be a network and $v \in N$. Our algorithm resynthesizes the node v with respect to a cut $(X_v, \overline{X}_v)$ in N_v as follows. Let $Q(X_v, \overline{X}_v)$ denote the subnetwork which consists of $\overline{X}_v$ and $n(X_v, \overline{X}_v)$. Nodes in $n(X_v, \overline{X}_v)$ are regarded as the PIs and node v is the PO of $Q(X_v, \overline{X}_v)$. The functionality of $Q(X_v, \overline{X}_v)$, denoted $f_Q(X_v, \overline{X}_v)$, is first computed and represented in some logic form (BDD in our case). Then, a resynthesis is carried out to implement $f_Q(X_v, \overline{X}_v)$ by an LUT subnetwork $R(X_v, \overline{X}_v)$ with the output node v'. In this case, we can replace the node $v \in N$ and nodes in $MFFC_v$ by $R(X_v, \overline{X}_v)$ such that $output(v') = output(v)$ and $input(R(X_v, \overline{X}_v)) = n(X_v, \overline{X}_v)$. Such a replacement will not change the functionality of N. Note that it is not all nodes in $\overline{X}_v$ can be replaced by $R(X_v, \overline{X}_v)$ because some nodes in $\overline{X}_v$ may have outputs to nodes outside $\overline{X}_v$. Only $MFFC_v$ can be replaced. The gain of the replacement is $|MFFC_v| - |R(X_v, \overline{X}_v)|$. Our resynthesis algorithm makes such replacement only when the gain is positive.

3.2. The CutSyn Algorithm

Given an LUT network N, the CutSyn algorithm uses two resynthesis procedures to reduce the number of LUTs in N: (1) the MFFC resynthesis and (2) the min-cut resynthesis. In the MFFC resynthesis procedure, N is partitioned into MFFCs. For each partitioned MFFC $MFFC_i$ with its root node r_i, we compute a resynthesis R_i of r_i with respect to the cut $(N_{r_i} - MFFC_i, MFFC_i)$. If R_i is superior to $MFFC_i$ in terms of depth or area, we replace the entire $MFFC_i$ by R_i without changing the functionality of N since $MFFC_i$ is fanout-free. The MFFC resynthesis process terminates when every MFFC has been tried for a resynthesis.

The min-cut resynthesis procedure in CutSyn includes two phases. In the first phase, we compute the *best* resynthesis for each node $v \in N$ as follows. First, we compute min-cuts of height p in N_v for $0 \le p < height(v)$. For each min-cut $(X_v, \bar{X}_v)$ of height p, we compute a resynthesized network, denoted $R_p(X_v, \bar{X}_v)$, of v with respect to the cut $(X_v, \bar{X}_v)$ of height p. Note that the smaller the p is, the further the cut of height p is away from v. However, $R_{p_1}(X_v, \bar{X}_v)$ might be smaller than $R_{p_2}(X_v, \bar{X}_v)$ even if $p_1 < p_2$. The gain of replacement is estimated for each $R_p(X_v, \bar{X}_v)$ and the best choice is recorded. In the second phase, we traverse nodes from POs to PIs in reverse topological order. Nodes with a positive gain and *non-zero* fanouts will be replaced by their corresponding resynthesis networks. Obviously, some nodes in the original network becomes zero-fanout nodes and will be removed from the network. The resulting network has the same functionality as the original network.

We use BDD [Br86a] to represent functions due to its compactness and efficiency in manipulation. It is well-known that the size of BDD for a function $f(X)$ depends heavily on the order of variables in X. The existence of a non-trivial functional decomposition of $f(X)$ under a bound set B also relies on the size of cofactor set $cs_B(f)$. As a result, it is crucial to compute BDD based on a good variable ordering in order to decompose functions successfully. CutSyn computes the initial BDD of node v with respect to a cut $(X_v, \bar{X}_v)$ based on a variable order determined by a height-driven depth-first-search heuristic. However, a dynamic variable reordering technique is used in the followed synthesis steps in order to maintain a small BDD and a small cofactor set. The resynthesis in CutSyn is based on the theory of partially-dependent decomposition developed in Section 2.

4. Experimental Results

We have implemented the CutSyn algorithm using the C language and incorporated our implementation into the SIS [SeSL92] logic synthesis package. We use MCNC combinational benchmarks in our experiments. These benchmark circuits are first optimized for area and then decomposed into simple-gate 2-input networks for speed using standard SIS routines. The same set of benchmarks with similar optimization process have been used in several previous algorithms including Chortle-d [FrRV91c], FlowMap [CoDi94a], MIS-pga-delay [MuSB91a], TechMap-D [SaTh93], CutMap [CoHw95] and FlowSyn [CoDi93b]. All these algorithms target for delay minimization in LUT synthesis and mapping. Among them, Chortle-d, FlowMap and CutMap use pure combinatorial-based technology mapping graph covering techniques while MIS-pga-delay, TechMap-D and FlowSyn combine both technology mapping and logic synthesis.

We compared CutSyn with these existing synthesis and mapping algorithms for delay minimization. The best pure technology mapping results in terms of achieving optimal depth with minimal area was reported by CutMap [CoHw95]. Among the approaches which combines technology mapping and logic synthesis, the results by TechMap-D [SaTh93] and FlowSyn [CoDi93b] are considerably better than others. Therefore, we compare CutSyn results with results by CutMap, TechMap and FlowSyn. Table 3 presents the comparison of results. The subtotal is the sum of results for benchmarks which appear in every data set. We always use CutMap to generate initial LUT networks for CutSyn. Hence the result by CutMap is the starting point for CutSyn. From Table 3, we see CutSyn is able to improve CutMap mapping solutions by 18% in both depth and area. Comparing to other two combined approaches, the results by CutSyn outperforms TechMap-D and FlowSyn by 2% to 11% in depth and 10% to 39% in area.

	CutMap		TechMap-D		FlowSyn		**CutSyn**	
Circuit	LUT	depth	LUT	depth	LUT	depth	LUT	depth
5xp1	23	3	17	2	20	2	17	2
9sym	62	5	9	3	7	3	6	3
9symml	58	5	9	3	7	3	6	3
C499	143	5	148	4	133	5	129	5
C880	205	8	213	7	232	8	197	8
alu2	144	8	197	8	113	6	72	5
alu4	245	10	-	-	249	9	201	10
apex4	732	6	-	-	-	-	714	6
apex6	233	4	252	5	257	4	226	4
apex7	80	4	86	4	89	4	78	4
count	69	3	71	4	75	3	55	3
des	986	5	1395	8	893	4	877	5
duke2	178	4	175	4	187	4	172	4
e64	159	3	-	-	-	-	157	3
misex1	15	2	18	2	15	2	14	2
rd84	45	4	16	3	13	3	13	2
rot	239	6	315	6	262	6	237	6
vg2	39	4	36	4	45	4	34	4
z4ml	12	3	9	2	6	2	6	2
subtotal	2531	73	2966	69	2354	63	2139	62
	+18%	+18%	+39%	+11%	+10%	+2%	1	1

Table 3 Results by CutMap, TechMap-D, FlowSyn and CutSyn.
(Subtotal is computed based on the subset of benchmarks on which all methods reported results)

We record the occurrence of partially-dependent encoding functions during functional decompositions in the CutSyn program in order to obtain a frequency statistics. Because we choose the K-LUT input size to be $K=5$, the bound set B contains at most 5 variables. In a non-trivial decomposition, the number of encoding functions ny will be in the range $1 \le ny \le 4$. As a result, we divide the statistics into four groups S_1, S_2, S_3 and S_4. An encoding function $y(B)$ is counted as a partially-dependent encoding function if it does not depend on every variable in the bound set B, i.e. $arg(y) \subset B$. For each instance of functional decomposition which generates ny encoding functions, we add the number of encoding functions as well as the number of partially-dependent encoding functions into the statistics group S_{ny}. The statistics data is collected during the entire post-processing optimization stage. Note that CutSyn might decide not to use some decomposition results (when there is no gain by resynthesis) while the statistics reported in Table 4 includes *every* decomposition tried during resynthesis. Numbers under columns *ny* and *npdy* are the numbers of encoding function and partially-dependent encoding functions, respectively. For the group S_1, no partially-dependent encoding function is generated. This is because CutSyn always removes redundant variables before resynthesis. As a result when there is only one encoding function, it must depend on all variables in the bound set. From the table, we see partially-dependent encoding functions occur at a high probability of 35% to 53% in functional decomposition. Therefore, the use of partially-dependent functional decomposition can introduce significant reduction in area of LUT networks.

5. Conclusion and Future Works

In this paper, we present the theory of partially-dependent functional decomposition and derive the theories of non-disjunctive decomposition and multiple-output decomposition from the perspective of partially-dependent decomposition. We observe non-disjunctive decompositions are able to reduce the number of encoding functions comparing to disjunctive decompositions and we show the computation of common encoding functions in the decomposition of multiple-output functions can be reduced to the computation of partially-dependent encoding functions. We give algorithms to compute a maximal set of partially-dependent encoding functions and non-disjunctive variables for a given function $f(X)$ and a bound set B. For LUT-based FPGA mapping and synthesis, we propose an approach which first employs depth-optimal mapping algorithms to generate an depth-optimal LUT network followed by iterative resynthesis of LUT subnetworks to reduce the depth and area of the final LUT network. A post-mapping algorithm, CutSyn, is proposed for resynthesis in this paper. Experimental results show an average of 18% reduction in both depth and the number of LUTs can be achieved comparing the initial LUT network obtained by the depth-optimal technology mapping algorithm CutMap. We compare our approach with TechMap-D and FlowSyn algorithms. Our approach outperforms both of them by 2% to 11% in depth and 10% to 39% in area. We also collect statistics of the occurrence of partially-dependent encoding functions during the resynthesis procedure. The statistics show that partially-dependent encoding functions occur at a high probability of 35% to 53% in the

	S_1		S_2		S_3		S_4	
	ny	npdy	ny	npdy	ny	npdy	ny	npdy
5xp1	2	0	10	2	3	1	0	0
9sym	5	0	30	6	48	17	16	5
9symml	4	0	26	3	36	14	20	5
C499	51	0	140	92	12	7	0	0
C880	289	0	546	241	474	267	176	86
alu2	55	0	444	96	462	143	492	170
alu4	312	0	1016	335	591	216	560	179
apex4	1403	0	2694	1049	2586	777	1184	380
apex6	304	0	182	114	0	0	0	0
apex7	3	0	20	11	3	2	0	0
count	121	0	108	78	6	4	0	0
des	1369	0	5046	3435	1308	835	128	66
duke2	99	0	108	51	45	21	8	5
e64	2	0	0	0	0	0	0	0
misex1	0	0	10	5	0	0	0	0
rd84	8	0	38	10	9	3	0	0
rot	158	0	156	102	0	0	0	0
vg2	36	0	112	78	45	25	8	4
z4ml	5	0	0	0	0	0	0	0
Total	4226	0	10686	5708	5628	2332	2592	900
Ratio	1	0	1	53%	1	41%	1	35%

Table 4 Hit rate of partially-dependent encoding functions.

decomposition of functions.

In our approach, we consider the resynthesis of LUT subnetworks individually. It will be beneficial if we can resynthesize multiple LUT subnetworks together according to a common cut and share encoding functions. We are in the process of implementing multiple-output decomposition routines to further improve the results.

Acknowledgement

This work is partially supported by NSF Young Investigator (NYI) Award MIP-9357582, and grants from Xilinx, Xerox PARC, and AT&T under NSF NYI and California MICRO programs.

References

[AS59] Ashenhurst, R. L., ‘‘The Decomposition of Switching Functions,’’ *Proc. Int’l Symp. on Theory of Switching Functions*, 1959.

[Br86a] Bryant, R. E., ‘‘Graph-based Algorithms for Boolean Function Manipulation,’’ *IEEE Trans. on Computers*, Vol. **C-35**, pp. 677-691, Aug. 1986.

[ChCD92] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, ‘‘DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization,’’ *IEEE Design and Test* *of Computers*, pp. 7-20, Sep. 1992.

[ChMa92] 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.

[CoDi93b] 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.

[CoDi94a] Cong, J. and Y. Ding, ''An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs,'' *IEEE Trans. on Computer-Aided Design*, Vol. **13**, pp. 1-12, Jan. 1994.

[CoDi94b] 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.

[CoHw95] Cong, J. and Y.-Y. Hwang, ''Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping,'' *Proc. ACM 3rd Int'l Symp. on Field Programmable Gate Arrays*, pp. 68-74, Feb. 1995.

[CoLB94] Cong, J., Z. Li, and R. Bagrodia, ''Acyclic Multi-Way Partitioning of Boolean Networks,'' *Proc. ACM/IEEE 31st Design Automation Conf.*, pp. 670-675, June 1994.

[Cu61] Curtis, H. A., ''A Generalized Tree Circuit,'' *Journal of the ACM*, Vol. **8**(4) pp. 484-496, 1961.

[FrRV91a] Francis, R. J., J. Rose, and Z. Vranesic, ''Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs,'' *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1991.

[FrRV91c] Francis, R. J., J. Rose, and Z. Vranesic, ''Technology Mapping of Lookup Table-Based FPGAs for Performance,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 568-571, Nov. 1991.

[HwJS95] 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.

[Karp63] Karp, R. M., ''Functional Decomposition and Switching Circuit Design,'' *J. Soc. Indust. Appl. Math.*, Vol. **11, 2**, pp. 291--335, Jun. 1963.

[LaPP94] 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.

[LaPV93] 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.

[MuNS90] Murgai, R., Y. Nishizaki, N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, ''Logic Synthesis Algorithms for Programmable Gate Arrays,'' *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[MuSB91a]
Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ''Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 572-575, Nov. 1991.

[RoKa62] Roth, J. P. and R. M. Karp, ‘‘Minimization Over Boolean Graphs,’’ *IBM Journal of Research and Development*, pp. 227-238, April 1962.

[SaSN95] Sawada, H., T. Suyama, and A. Nagoya, ‘‘Logic Synthesis for Look-Up Table based FPGAs using Functional Decomposition and Support Minimization,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 353-358, Nov. 1995.

[SaTh93] 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.

[SeSL92] 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.

[WuEA95] 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.