Finding an Optimal Functional Decomposition for LUT-based FPGA Synthesis

Jian Qiao

Department of Electronic engineering
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
Tel: +81-3-5841-6716
Fax: +81-3-5841-8192
e-mail: qiao@silicon.t.u-tokyo.ac.jp

Makoto Ikeda Kunihiro Asada

VLSI Design and Education Center
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
Tel: +81-3-5841-6661(6671)
Fax: +81-3-5802-3903
e-mail: {ikeda, asada}@silicon.t.u-tokyo.ac.jp

Abstract— In this paper, we propose a novel approach to the optimal functional decomposition for LUT-based FPGA synthesis. We focus on exploring all design space and finding a set of $\overrightarrow{\alpha}$ components which can be merged, to the maximal extent, into multiple-output CLBs or an LUT such that the decomposition constructed from the components is also minimal. In particular, to exploit more degrees of freedom, pliable encoding has been introduced to take over the classical rigid encoding process when the latter fails to get a satisfactory solution. Experimental results on a set of MCNC91 benchmarks show that our method is promising.

I. Introduction

FPGAs(Field-Programmable Gate Arrays) have become a very popular technology in VLSI/ASIC design and system prototyping due to their short turnaround time and low manufacturing cost. One important class is the look-up table(LUT) based FPGAs. For example, Xilinx XC3090 LUT-based FPGAs [13], whose CLB has two outputs and can implement either a Boolean function up to 5 inputs or two Boolean functions up to 4 with a total at most 5 inputs. In this paper, we develop an algorithm for mapping a given Boolean function to the FPGA targets with two-output CLB architecture.

Recent years, functional decomposition has been applied to FPGA synthesis with good results. In general, two problems should be considered in single-output functional decomposition: (1) how to select the bound set variables, and (2) how to encode the compatibility classes. Research presented in [4] and [5] addressed the problem of the bound set selection, while the approaches proposed in [6] [7] [8] [9] [10] [11] dealt with the encoding problem. The existing algorithms for compatibility class encoding can be classified into two categories according to their objective criteria. The first category [6] [7] is concerned with the simplification of the resulting image function g so that g can be easily re-decomposed; while the second category focuses on producing more *PDF*s(Partially-Dependent Functions) so that these *PDF*s may be merged into multiple-output CLBs [8] [9] [10] [11]. Methods proposed in [7] [9] [11] also took into account of sharing subfunctions among the multiple outputs of a given network. However, all algorithms developed so far have employed only rigid encoding strategy to our best knowledge. That may eliminate some feasible *PDF*s and hence result in a sub-optimal decomposition solution. In [12], authors showed that the optimal decomposition may be found only by pliable encoding in the cases that the number of inputs to the image function g is less than $k = 5$. Generally, to find the optimal decomposition, all design space(more feasible decomposition forms) should be explored. For example, we know that Fig. 1 (a) gives an optimal decomposition. Anyway, the decomposition forms shown in Fig. 1 (b), two mutually compatible *PDF*s with their total inputs less than or equal to 3 can share an LUT with the image function $g$and we called it absorption, are also the optimal solutions with the same cost(there are 6 feasible decomposition forms included).

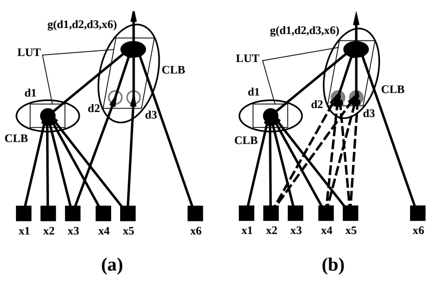

Fig. 1. Illustration of decomposition by pliable encoding. (a) Decomposition of form SVF-SVF-Alpha. (b) Decomposition of form <u>PDF3-PDF3</u>-Alpha.

Here and after, we briefly denote a decomposition form with all of its $\overrightarrow{\alpha}$ components, and let the underlined *PDF*s denote the $\overrightarrow{\alpha}$ components which can be absorbed into function g. (Notice that Fig. 1 shows pliable encoding solutions because the number of the compatibility classes is less than or equal to 4 in the case.) In this paper, with multiple-output CLB architecture in mind, we propose a novel approach to explore all design space and find a set of $\overrightarrow{\alpha}$ components such that the decomposition constructed from them is minimal in terms of the number of multiple-output CLBs or LUTs.

The remainder of this paper is organized as follows. Section II describes preliminaries. The problem of compatibility class encoding is formulated in section III. Section IV explains our encoding algorithm. The preliminary experimental results are reported in section V.

II. Preliminaries

Let $B = \{0,1\}$, and $f(\mathbf{X}) : B^n \rightarrow B$, functional decomposition is a procedure that decomposes a complex function f into a set of simpler functions $\overrightarrow{\alpha}$ and g, and it can be expressed by

$$f(\mathbf{X}) = g(\overrightarrow{\alpha}(\mathbf{x}_b), \mathbf{x}_f) \quad (1)$$

where $\mathbf{X} = \{x_1, x_2, \cdots, x_n\} \in B^n$, $x_i \in B$ for $1 \leq i \leq n$, is the set of input variables. $\mathbf{x}_b = \{x_1, x_2, \cdots, x_s\} \in B^s$, $1 \leq s \leq n$, is called the *bound set*(or BS), and $\mathbf{x}_f = \{x_{s-i+1}, \cdots, x_n\} \in B^{n-s+i}$, $i \geq 0$, is called the *free set*(or FS). $\overrightarrow{\alpha} : B^s \rightarrow B^l$, $\overrightarrow{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_l)$ and $\alpha_j : B^s \rightarrow B$ for $1 \leq j \leq l$ and $1 \leq l \leq t$, is called the *encoding function*, and each α_j is called its component or sub-functions. The procedure of determining all the $\overrightarrow{\alpha}$ components is called *compatibility class encoding*. g is called the *image function*, and the size $\| g \|$ is defined as the inputs to g. The decomposition defined in Eq. (1) is *disjunctive* if $\mathbf{x}_b \bigcap \mathbf{x}_f = \phi$, and it is *non-disjunctive* if $\mathbf{x}_b \bigcap \mathbf{x}_f \neq \phi$; besides, it is a *simple decomposition* if $t = 1$ and a *complex decomposition* otherwise.

The basic theory of the classical decomposition is described as follows, and the details can be found in [1-3, 6, 10-11].

Definition II.1 *Any $\mathbf{v}_i \in B^s$ and $\mathbf{v}_j \in B^s (i \neq j)$ are said to be* compatible, *denoted by $\mathbf{v}_i \sim \mathbf{v}_j$, if and only if $f(\mathbf{v}_i, \mathbf{u}_f) = f(\mathbf{v}_j, \mathbf{u}_f)$ holds for all $\mathbf{u}_f \in B^{n-s+i}$. All mutually compatible BS vertices form a* compatibility class.

Theorem II.1 *For any $\mathbf{v}_i \in B^s$ and $\mathbf{v}_j \in B^s (i \neq j)$, if $\mathbf{v}_i \not\sim \mathbf{v}_j \Longrightarrow \overrightarrow{\alpha}(\mathbf{v}_i) \neq \overrightarrow{\alpha}(\mathbf{v}_j)$, then the decomposition defined by Eq. (1) must exist.*

Theorem II.1 gives the necessary and sufficient condition for the existence of the decomposition defined in Eq. (1). Given $\mathbf{x}_b$ is selected, let t be the size of $\overrightarrow{\alpha}$, and M be the number of compatibility classes, Theorem II.1 can be transformed to

$$t \geq \lceil log_2 M \rceil \quad (2)$$

Encoding is called *rigid* if $t = \lceil log_2 M \rceil$; otherwise, it is called *pliable*. Besides, it is *strict* if each compatibility class is assigned to only one $\overrightarrow{\alpha}$ code, and it is *non-strict* otherwise.

Definition II.2 *If a function defined on the BS space is independent of some BS variables, it is called a partially dependent function, denoted by* PDF. *The set of variables on which a function really depends is called its* support. *Especially, a function depending on only one variable is called a single-variable function, denoted by* SVF.

The decomposition constructed from at least one *PDF* is called a partially dependent decomposition, and denoted *PDD* [10]. It is worth to notice that a disjunctive decomposition can be treated as a non-disjunctive decomposition, denoted *NDD*, when at least one $\overrightarrow{\alpha}$ component is a *PDF* [11]. Here and after, we mean for an *NDD* to be a *PDD* whose at least one $\overrightarrow{\alpha}$ component can be absorbed into g without increasing the cost of g, in order to emphasize the beneficial *PDD*. Thus, our objective is to find the minimal *NDD* or *PDD*.

III. Encoding Formulation

We formulate the problem of compatibility class encoding as follows [12]:

Definition III.1 *For function f, given $\mathbf{x}_b$ is fixed, the objective of compatibility class encoding is to determine t according to Exp. (2), and find a set of t $\overrightarrow{\alpha}$ components such that the decomposition constructed from them is minimal in terms of the number of CLBs or LUTs.*

The encoding problem can be settled by assigning an $\overrightarrow{\alpha}$ code to each BS vertex according to Theorem II.1 . Generally, a *PDD* may be a better solution when some $\overrightarrow{\alpha}$ components can be merged to 2-output CLBs in pairs; while an *NDD* must be the best because some its components can be reduced and absorbed into g. So it is helpful to find all *SVF*s and *PDF*s which can serve as the $\overrightarrow{\alpha}$ components.

Definition III.2 *The group of 2^r BS vertices, resulted from changing only certain r variables $x_{j1}, x_{j2}, \cdots, x_{jr}$, where $x_{ji} \in \mathbf{x}_b$ for all $1 \leq i \leq r$ and $1 \leq r \leq s$, is called an r-adjacent group with respect to the r variables $x_{j1}, x_{j2}, \cdots, x_{jr}$. For example, 4 BS vertices* (00100), (00101), (00110), *and* (00111) *form a 2-adjacent group. If we assign the group of BS vertices to the onset, and the remainders to the offset, the resulting function is independent of the 2 variables at the* 0*th and* 1*th positions.*

Given $\| \mathbf{x}_b \| = 5$, there are totally 5 sets of 1-adjacent groups with respect to one variable x_i for $1 \leq i \leq 5$. Similarly, there are totally 10 sets of 2-adjacent groups with respect to certain 2 variables x_{j1}, x_{j2}; 10 sets of 2-adjacent groups with respect to certain 3 variables x_{j1},

x_{j2}, and x_{j3}; and 5 sets of 4-adjacent groups with respect to certain 4 variables x_{j1}, x_{j2}, x_{j3}, and x_{j4}; where $1 \le j1, j2, j3, j4 \le 5$, and $j1 \neq j2 \neq j3 \neq j4$. In Fig. 2 , we illustrate some sets of r-adjacent groups for all $1 \le r \le 4$.

00000	00010	00100	00110	01000	01010	01100	01110	10000	10010	10100	10110	11000	11010	11100	11110
00001	00011	00101	00111	01001	01011	01101	01111	10001	10011	10101	10111	11001	11011	11101	11111

(a)

00000	00100	01000	01100	10000	10100	11000	11100
00001	00100	01001	01100	10001	10101	11001	11101
00010	00110	01010	01110	10010	10110	11010	11110
00011	00111	01011	01111	10011	10111	11011	11111

(b)

00000	01000	10000	11000
00001	01001	10001	11001
00010	01010	10010	11010
00011	01011	10011	11011
00100	01100	10100	11100
00101	01101	10101	11101
00110	01110	10110	11110
00111	01111	10111	11111

(c)

00000 00001 00010 00011 00100 00101 00110 00111 01000 01001 01010 01011 01100 01101 01110 01111
10000 10001 10010 10011 10100 10101 10110 10111 11000 11001 11010 11011 11100 11101 11110 11111

(d)

Fig. 2. Illustration of some sets of r-adjacent groups for all $1 \le r \le 4$. (a) A set of 16 1-adjacent groups with respect to one variable x_1. (b) A set of 8 2-adjacent groups with respect to two variables x_1 and x_2. (c) A set of 4 3-adjacent groups with respect to three variables x_1, x_2, and x_3. (d) A set of 2 4-adjacent groups with respect to four variables x_1, x_2, x_3, and x_4.

Definition III.3 *Given a set of BS vertices, a Boolean function defined on the BS space can be constructed from the set of BS vertices by putting them to the onset and the remainders to the offset. The procedure is called the assignment of the set of BS vertices to the function.*

By Definition III.3 , any bipartition of all the BS vertices corresponds to a Boolean function on the BS space by the assignment of either part of the bipartition to the onset. So we can get all *PDF*s which are independent of certain r variables if we bipartition and assign the BS vertices according to the r-adjacent groups for $1 \le r \le 4$.

Theorem III.1 *For function f, given $\mathbf{x}_b$ and hence t is determined, any Boolean function on the BS space corresponds to a bipartition of the BS vertices. If the number of compatibility classes in each part of the bipartition is not greater than 2^{t-1}, the function is said to be a* feasible $\overrightarrow{\alpha}$ *component; namely, there must exist at least one decomposition which can be constructed from the feasible component. Proof. Trivial.* □

Definition III.4 *Given any m functions($2 \le m \le t$) on the BS space, all the BS vertices may be partitioned up to 2^m parts; if the number of the compatibility classes in each part is not greater than 2^{t-m}, we say the m functions are* mutually compatible *with respect to the given encoding problem.*

Theorem III.2 *There must exist at least one decomposition solution in which the m functions($2 \le m \le t$) serve as the m components of $\overrightarrow{\alpha}$, if and only if the m functions are mutually compatible with respect to the given encoding problem. Proof. Trivial.* □

According to Theorem III.2 , to find an optimal decomposition, we just need to extract a set of mutually compatible *PDF*s(including *SVF*s) form all feasible *PDF*s such that the decomposition constructed from them is minimal in terms of CLB or LUT count. The compatibility checking of the *PDF*s can can be done easily according to Theorem III.2 .

IV. Our new approach

A. Detection of all feasible PDFs

We utilize a *backtracking* algorithm [12] to find all feasible *PDF*s. As illustrated in Fig. 3 , for each sort of r-adjacent groups with $1 \le r \le 4$, a tree of maximal depth $2^r - 1$ is constructed to emulate the assignment of the BS vertices according to the r-adjacent groups; namely, the groups corresponding to the temporary path of the tree are assigned to the onset, and the remainders to the offset, of a *PDF*. Without loss of generality, the number of compatibility classes included in the r-adjacent groups which correspond to the path from the root is treated as the *constraint* for backtracking, and branch pruning is made as the number is greater than 2^{t-1} according to Theorem III.1 . A potential *PDF*, on the other hand, is evaluated for feasibility by checking the numbers of the compatibility classes in both the onset and the offset. All feasible *PDF*s can be found effectively by traversing the trees corresponding to all sets of the r-adjacent groups for $1 \le r \le 4$.

B. Finding an optimal decomposition

Notice that only a set of feasible *PDF*s, which are mutually compatible, are helpful for finding the minimal decomposition. So our chief concern is a set of *PDF*s which can be merged into a minimal number of 2-output CLBs or 5-LUTs. Then the optimal decomposition can be constructed from them.

B.1 Encoding strategies

For function f, given $\mathbf{x}_b$ is fixed, the number M of compatibility classes will be uniquely determined. According to Theorem II.1 , the number t of $\overrightarrow{\alpha}$ components required to encode the compatibility classes is given by Exp. (2). When we limit the BS size to $\parallel \mathbf{x}_b \parallel = k = 5$), each $\overrightarrow{\alpha}$ component can be implemented by a k-LUT or a CLB; so setting t to its minimum $t_0 = \lceil log_2 M \rceil$ is a common encoding strategy; and that is rigid encoding. However, if we let t be greater than $t_0 = \lceil log_2 M \rceil$, all

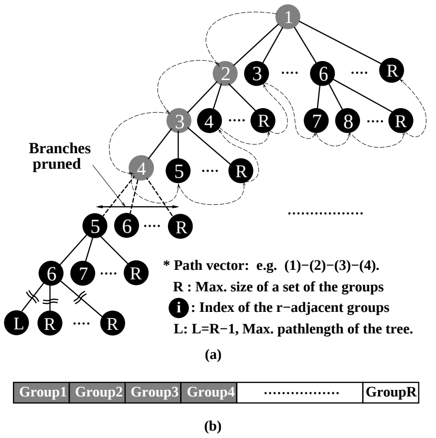

Fig. 3. Detection of all feasible PDFs which are independent of certain r variables for all $1 \leq r \leq 4$. (a) The search tree constructed for a set of r-adjacent groups. (b) The given set of r-adjacent groups.

possible *PDF*s, especially *SVF*s, will be feasible according to Theorem III.1 and we have much encoding choice; and that is pliable encoding. Our encoding process is SVF-dominated, and pliable encoding allows us to find more beneficial *NDD*s [12]. In this paper, we check more decomposition forms for the optimal solution.

B.2 Encoding considerations when $\| g \| > k$

When $\| g \| > k = 5$, we just take consideration of rigid encoding because increasing t may increase the cost of implementing g. we consider the following 3 cases.

Case 1: $t_0 = 2$. The best solution of the case is an *NDD* of form *SVF-ALPHA* if there is a feasible *SVF*(by *ALPHA*, we mean any feasible function of 5 inputs, and the functions connected by hyphens are compatible mutually). If, on the other hand, there are no feasible *SVF*s, a pair of *PDF*s, denoted *PDF-PDF*, is a better solution because they can be merged into a 2-output CLB. Anyway, there exists a decomposition of form *ALPHA-ALPHA* at worst according to Theorem III.1 . In short, encoding is performed in order of *SVF-ALPHA*, *PDF-PDF*, and *ALPHA-ALPHA*.

Case 2: $t_0 = 3$. Encoding is performed in order of *SVF-SVF-ALPHA*, *SVF-PDF-PDF*, *SVF-ALPHA-ALPHA*, *PDF-PDF-ALPHA*, and *ALPHA-ALPHA-ALPHA* at worst. Feasible *SVF*s can be used to filter out the *PDF*s which are not mutually compatible with them when checking the form *SVF-ALPHA-ALPHA*.

Case 3: $t_0 = 4$. Encoding is performed in order of *SVF-SVF-SVF-ALPHA*, *SVF-SVF-PDF-PDF*, *SVF-SVF-ALPHA-ALPHA*, *SVF-PDF-PDF-ALPHA*, *SVF-ALPHA-ALPHA-ALPHA*, *PDF-PDF-PDF-PDF*, *PDF-PDF-ALPHA-ALPHA*, and *ALPHA-ALPHA-ALPHA-ALPHA* at worst.

B.3 Encoding considerations when $\| g \| = k$

Only rigid encoding is considered in this case. As shown in Fig. 4 (a), the decomposition of form *PDF2-PDF2-Alpha* is also an beneficial *NDD* of the minimum cost. So encoding is performed in order of *SVF-SVF-ALPHA*, *SVF-PDF2-Alpha*, *PFF2-PDF2-Alpha*, *SVF-PDF-PDF*, *SVF-ALPHA-ALPHA*, *PDF-PDF-ALPHA*, and *ALPHA-ALPHA-ALPHA* at worst.

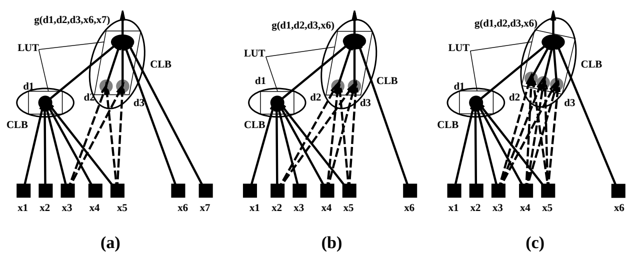

Fig. 4. Illustration of beneficial NDD solutions: (a) NDDs of forms PDF2-PDF2-Alpha when $\| g \| = k$(rigid encoding). (b) NDDs of forms PDF3-PDF3-Alpha when $\| g \| < k$(pliable encoding). (c) NDDs of forms PDF3-PDF3-PDF3-Alpha when $\| g \| < k$(pliable encoding).

B.4 Encoding considerations when $\| g \| < k$

When $\| g \| < k = 5$, pliable encoding should be considered to explore much encoding choice if there exist no satisfactory solutions by rigid encoding, while keeping the cost of implementing g invariant. The situation occurs mainly when f is a function of 6 or 7 variables. Generally, the numbers of feasible *PDF*s may be very large(over 167000 in pliable encoding), and the manipulations of the *PDF*s become impractical, so we have defined bitwise-operations for their manipulations.

Case 1: $t_0 = 2$ and $\| g \| = 3$; f is a 6-input function as shown in Fig. 4 (b). Rigid encoding is performed at first to search the optimal solution in order of forms *SVF-ALPHA*, *PDF2-ALPHA*, *PDF3-ALPHA*, *PDF-PDF*(*PDF2* and *PDF3* are *PDF*s of 2 and 3 inputs, respectively). When it fails to find a satisfactory solution by rigid encoding, pliable encoding is introduced to search for a better one in order of *SVF-SVF-ALPHA*, *SVF-PDF2-ALPHA*, *PDF3-PDF3-ALPHA*(including 4 decomposition forms), *SVF-PDF-PDF* and *PDF2-PDF-PDF*. (Notice that they are all beneficial *NDD*s.) If it still fails to find a better solution once by pliable encoding, a recursive call is introduced and described in Case 3.

Case 2: $t_0 = 2$ and $\| g \| = 4$; f is a 7-input function. Rigid encoding is performed at first to search the optimal solution in order of *SVF-ALPHA*, *PDF2-ALPHA*, *PDF-PDF*. Pliable encoding is introduced, when it fails to find a satisfactory solution by rigid encoding, to search a better one in order of *SVF-SVF-ALPHA*, *PDF2-PDF2-ALPHA*(including *SVF-PDF2*-

ALPHA, <u>*PDF2-PDF2*</u>-*ALPHA*), and then *SVF-PDF-PDF*. If there exists no better solution even by pliable encoding, a solution of form *ALPHA-ALPHA* is chosen at the worst.

Case 3: If a better solution cannot be found in Case 1 even once by pliable encoding, a recursive application of pliable encoding may be considered in search for a better decomposition in order of *SVF-SVF-SVF-ALPHA* and <u>*PDF3-PDF3-PDF3*</u>-*ALPHA*(including 8 decomposition forms). If it still fails to find an beneficial NDD, a solution of form *ALPHA-ALPHA* is chosen at worst.

Example 1: Let us examine the function shown in Fig. 5 . Given $\mathbf{x}_b = \{x_1, x_2, x_3, x_4, x_5\}$, then $M = 4$; and the encoding chart which lists all BS vertices associated with their compatibility class IDs is shown in Fig. 6 (a). Because there exist no feasible *SVF*s or *PDF*s in the case of rigid encoding($t_0 = 2$), there are no beneficial NDD or PDD solutions, and we have a sub-optimal solution of form *ALPHA-ALPHA*, as shown in Fig. 6 (c). We need 3 LUTs or 3 2-output CLBs to implement the function. However, by pliable encoding($t = 3 > t_0$), and partition the BS space as shown in Fig. 6 (b), we can find 2 mutually compatible *SVF*s and get the optimal solution of form *SVF-SVF-ALPHA*, as shown in Fig. 6 (d). We instead need only 2 LUTs or 2 2-output CLBs to implement it.

$$\begin{aligned}
\{8\} \quad = \quad & x_1 x_2 x_3 x_4 x_5 + x_1 x_2 x_3 \bar{x}_4 \bar{x}_5 x_6 + x_1 x_2 \bar{x}_3 x_4 \bar{x}_5 \bar{x}_6 \\
+ \quad & x_1 x_2 \bar{x}_3 \bar{x}_4 x_5 x_6 + x_1 \bar{x}_2 x_3 x_4 \bar{x}_5 x_6 + x_1 \bar{x}_2 x_3 \bar{x}_4 x_5 \\
+ \quad & x_1 \bar{x}_2 \bar{x}_3 x_4 x_5 x_6 + x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \bar{x}_5 \bar{x}_6 + \bar{x}_1 x_2 x_3 x_4 \bar{x}_5 x_6 \\
+ \quad & \bar{x}_1 x_2 x_3 \bar{x}_4 x_5 + \bar{x}_1 x_2 \bar{x}_3 x_4 x_5 x_6 + \bar{x}_1 x_2 \bar{x}_3 \bar{x}_4 \bar{x}_5 \bar{x}_6 \\
+ \quad & \bar{x}_1 \bar{x}_2 x_3 x_4 x_5 + \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4 \bar{x}_5 x_6 + \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \bar{x}_5 \bar{x}_6 \\
+ \quad & \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 x_5 x_6
\end{aligned}$$

Fig. 5. A function of 6 input variables.

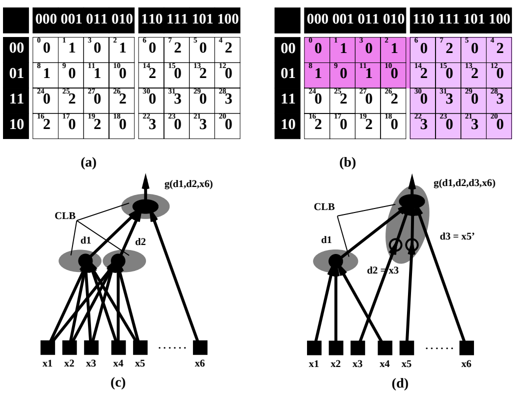

Fig. 6. (a) Encoding chart. (b) Partition of the BS space by 2 feasible SVFs in pliable encoding. (c) Decomposition by rigid encoding($t_0 = 2$). (d) Decomposition by pliable encoding($t = 3$).

In general, it is time-consuming to extract a mutually compatible *PDF-PDF* pair when the number of feasible *PDF*s is large, especially in the case of pliable encoding. So we may make an approximation by replacing a *PDF4-PDF4* pair with a *PDF3-PDF4*, *PDF2-PDF4*, or *PDF3-PDF3*.

V. Experimental results

The algorithm described above is implemented in language C, and incorporated into SIS-1.2 program [14]. To assess our algorithm, we conducted two experiments over a large set of mcnc91 logic synthesis benchmarks: one applies the approach described in Section IV with consideration of pliable encoding when the conventional rigid encoding fails to get a satisfactory solution, denoted *with_pliable*; the other is without consideration of pliable encoding(switching off all pliable encoding procedures in Section IV), denoted *without_pliable*. The scripts for logic optimization and FPGAs mapping are exactly the same as given in [12]. Our method has been applied to the decomposition step for *Xilinx XC3090* FPGAs target. The experiments are conducted on a *Sun SPARC Ultra-60* workstation, and the preliminary experimental results are shown in Fig. 7 Columns 3 and 4. We have also shown the results from the decomposition algorithm provided in SIS-1.2 program(denoted *SIS-1.2* in Column 2) for evaluation. The area(LUTs/CLBs), depth(Levels) and the computational expense(seconds) are given as the criteria.

On an average, our algorithms both with and without pliable encoding require considerably fewer LUTs(33% and 25%) and 2-output CLBs(27% and 21%) than that from SIS-1.2 program. That is because our approach explores all the design space during encoding process and hence can find more beneficial *NDD* or *PDD* solutions. Moreover, the results from *with_pliable* show some reduction in terms of the number of LUTs or 2-output CLBs, compared with those from *without_pliable*. It is due to that more beneficial *NDD*s may be found only by introducing pliable encoding. As can be expected, our methods need more computational resources(time and space), especially in the case of pliable encoding. In Fig. 8 , we made a comparison of our algorithm with the state-of-the-art mathods: $HEU + EXACT$ RK_dec [7] and $ISODEC\_S$ [11]. As can be observed, our method is quite effective for reducing the number of LUTs; but it haven't performed so good as expected in terms of the number of multiple-output CLBs, especially for the benchmarks with higher parallelism among their multiple outputs, such as *alu*2 and *alu*4. That is due to our single-output decomposition method without consideration of sharing sub-functions among the multiple outputs.

Benchmarks	SIS−1.2	without_pliable	with_pliable	#PI/#PO
	#LUT/#level/sec//#CLB	#LUT/#level/sec//#CLB	#LUT/#level/sec//#CLB	
9symml	7 3 0.4 7	6 3 18.7 6	6 3 18.7 6	9 1
alu2	91 21 28.9 76	89 30 26.2 75	89 30 97.6 75	10 6
alu4	296 39 198.4 243	237 25 690.4 216	220 22 1216.6 176	14 8
apex2	198 16 301.3 168	109 13 768.0 98	99 11 1698.2 88	39 3
apex6	232 20 10.5 189	172 7 281.2 144	171 7 628.6 143	135 99
apex7	59 7 3.3 47	57 7 2.9 44	57 7 3.0 44	49 37
b9	52 5 2.8 42	39 5 39.8 34	38 3 525.6 32	41 21
clip	36 5 2.6 29	22 3 40.5 21	20 3 144 19	9 5
duke2	149 8 6.3 137	133 8 780.5 125	129 7 1838.6 121	22 29
example2	129 7 13.6 106	107 5 10.4 84	102 6 296.8 68	85 66
frg1	39 14 10.4 38	32 7 29.6 31	30 8 169.6 29	28 3
i7	103 2 6.4 102	103 2 13.2 102	103 2 14.8 102	199 67
misex2	49 3 1.4 41	34 4 68.8 30	32 5 1268.6 26	25 18
misex3c	260 11 18.3 216	143 9 126.0 134	135 10 1628.0 110	14 14
rd84	13 3 1.2 13	12 3 16.6 11	12 3 466.8 10	8 4
sao2	52 5 2.0 51	23 5 56.8 22	22 3 201.8 21	10 4
too_large	184 31 568.9 166	138 11 305.5 114	133 11 1265.6 108	38 3
vda	260 11 18.9 156	206 10 286.6 156	197 9 510.8 147	17 39
vg2	27 6 1.8 25	21 6 6.2 19	20 6 46.2 17	25 8
t481	14 4 8.5 13	5 3 26.8 5	5 3 26.9 5	16 1
Total(LUT/CLB)	2250 / 1865	1688 / 1471	1517 / 1347	793 / 36

Fig. 7. *Experimental results.*

Benchmarks	HEU+EXACT RK_dec[7]	ISODEC_S[11]	with_pliable	#PI/#PO
	#LUT / #CLB	#LUT / #CLB	#LUT / #CLB	
9symml	6 6	8 7	6 6	9 1
alu2	70 65	52 44	89 75	10 6
alu4	216 169	231 48	220 176	14 8
apex2	−− −−	165 75	99 88	39 3
apex6	209 163	207 126	171 143	135 99
apex7	63 51	71 40	57 44	49 37
b9	36 29	41 −−	38 32	41 21
clip	23 23	22 16	20 19	9 5
duke2	117 99	213 121	129 121	22 29
example2	−− −−	139 63	102 68	85 66
frg1	−− −−	40 −−	30 29	28 3
i7	−− −−	103 −−	103 102	199 67
misex2	32 28	40 21	32 26	25 18
misex3c	−− −−	152 −−	135 110	14 14
rd84	13 12	12 10	12 10	8 4
sao2	27 27	22 21	22 21	10 4
too_large	−− −−	165 −−	133 108	38 3
vda	−− −−	351 −−	197 147	17 39
vg2	23 22	59 17	20 17	25 8
t481	−− −−	−− −−	5 5	16 1

Fig. 8. *Comparison with the state-of-the-art methods.*

VI. Conclusions

In this paper, we address compatibility class encoding problem. With multiple-output CLB architecture in mind, we explore all design space and focus on extracting a set of $\overrightarrow{\alpha}$ components which can be merged into a minimal number of CLBs or LUTs. To exploit more degrees of freedom, pliable encoding has been introduced when it fails to find a satisfactory solution only by the classical rigid encoding. Our preliminary experimental results are quite encouraging, and show that pliable encoding should be taken into account for finding an optimal decomposition.

Acknowledgements

The authors thank Dr.Murgai for providing us with sis-1.2 program and for many helpful discussions about how to use it.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," in *Proc. Int. Symp. Theory of Switching Functions*, Apr. 1959.

[2] J. P. Roth, and R. M. Karp, "Minimization over boolean graphs", *IBM J. Res. Develop.*, vol.6, pp.227-238, April 1962.

[3] H. A. Curtis, "A generalized tree circuit," *J. ACM*, vol.8(4), pp.484-496, 1961.

[4] W. Z. Shen, J. D. Huang, and S. M. Chao, "Lambda set selection in Roth-Karp decomposition for LUT-based FPGA technology mapping," in *32nd ACM/IEEE Design Automation Conference*, pp.65-69, June 1995.

[5] J-H. Jiang, J-Y. Jou, J-D. Huang, and J-S. Wei, "A variable partitioning algorithm of BDD for FPGA technology mapping," *IEICE Trans. Fundamentals*, vol.E80-a, no.10,pp.1813-1819, 1997.

[6] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum functional decomposition using encoding," in *Proc. 31st ACM/IEEE Design Automation Conference*, pp.408-414, June 1994.

[7] J-H. R. Jiang, J-Y. Jou, and J-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," in *Proc. 35th ACM/IEEE Design Automation Conference*, pp.712-717, June 1998.

[8] J-D. Huang, J-Y. Jou, and W-Z. shen, "Compatible class encoding in Roth-Karp decomposition for two-output LUT architecture," in *Proc. Int. Conf. Computer Aided Design*, pp.359-363, Nov. 1995.

[9] H. Sawada, T. Suyama, and A. Nagoya, "Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization," in *Proc. Int. Conf. Computer Aided Design*, pp.353-358, Nov. 1995.

[10] J. Cong, and Y-Y. Hwang, "Partially dependent functional decomposition with applications in FPGA synthesis and mapping," in *Proc. ACM/SIGDA Int'l Symp. on FPGAs*, pp.35-42, Feb. 1997.

[11] C. Legl, B. Wurth, and K. Eckl, "Computing support-minimal subfunctions during functional decomposition," *IEEE Trans. on VLSI*, vol.6, no.3, pp.354-363, Sep. 1998.

[12] J. Qiao, M. Ikeda, and K. Asada, "Optimum functional decomposition for LUT-based FPGA synthesis," in *Proc. 10th int. Conference on Field-Programmable Logic and Applications, FPL2000*, pp.555-564, Aug. 2000.

[13] Xilinx Inc., 2069, Hamilton Ave. San Jose, CA-95125, *The Programmable Gate Array Data Book.*

[14] E. Sentovich, 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.