Boolean Matching for LUT-Based Logic Blocks With Applications to Architecture Evaluation and Technology Mapping

Jason (Jingsheng) Cong, *Fellow, IEEE,* and Yean-Yow Hwang

***Abstract*—In this paper, we present new Boolean matching methods for lookup table (LUT)-based programmable logic blocks (PLBs) and their applications to PLB architecture evaluations and field programmable gate array (FPGA) technology mapping. Our Boolean matching methods, which are based on functional decomposition operations, can characterize functions for complex PLBs consisting of multiple LUTs (possibly of different sizes) such as Xilinx XC4K CLBs. With these techniques, we conducted quantitative evaluation of four PLB architectures on their functional capabilities. Architecture evaluation results show that the XC4K CLB can implement 98% of six-input and 88% of seven-input functions extracted from MCNC benchmarks, while a simplified PLB architecture is more cost effective in terms of function implementation per LUT bit. Finally, we proposed new technology mapping algorithms that integrate Boolean matching and functional decomposition operations for depth minimization. Technology mapping results show that our PLB mapping approach achieves 12% smaller depth or 15% smaller area in XC5200 FPGAs and 18% smaller depth in XC4K FPGAs, compared to conventional LUT mapping approaches.**

***Index Terms*—FPGA architecture, logic synthesis.**

I. Introduction

THE FIELD programmable gate array (FPGA) was introduced in the mid-1980s as an alternative for the implementation of application-specific integrated circuits (ASICs). In contrast to the cell library technology and the mask-programmable gate array technology for ASICs, an FPGA does not need to go through the fabrication process for circuit implementation and is field programmable and often field reprogrammable. Although the FPGA in general has a lower gate density and slower circuit speed, its advantages of programmability, shorter design turnaround time, and lower initial nonrecurring engineering cost (good for low to medium volume production) often offset its disadvantages. A wide range of applications has been developed using FPGAs, including fast ASIC implementation, rapid system prototyping, logic emulation, and reconfigurable computing.

Manuscript received June 8, 2000; revised October 18, 2001. This work was supported in part by the National Science Foundation Young Investigator under Award MIP-9357582, by the Advanced Research Projects Agency/Computing Systems Technology Office under Contract DABT63-93-C-0055, and by Lucent Technologies, by Quickturn, and by Xilinx under the California MICRO program. This paper was recommended by Associate Editor L. Stok.

J. Cong is with the Computer Science Department, University of California, Los Angeles, CA 90024 USA.

Y.-Y. Hwang is with the Altera Corporation, San Jose, CA 95134 USA.

Publisher Item Identifier S 0278-0070(01)06885-3.

FPGAs consist of three kinds of programmable elements: programmable logic blocks (PLBs), routining resources, and input–output (I/O) blocks. Each logic block contains combinational components such as multiplexers (MUXs), simple gates (e.g., AND and OR), programmable lookup tables (LUTs), and sequential components such as flip-flops. Routing resources include segmented interconnects and switching blocks. The segmented interconnects connect to the inputs and outputs of logic blocks while the switching blocks link the segments to form long routing tracks to implement routing topology. The I/O blocks can be programmed to become the primary inputs (PIs) or primary outputs (POs) of the circuits on FPGAs.

LUTs are the basic logic blocks in many FPGAs today. A k-input LUT (k-LUT) consists of 2^k static random access memory (SRAM) cells that can store the truth table of an arbitrary k-input function. In many FPGAs, small LUTs are connected by fast local connections to form a PLB for implementation flexibility, better performance, and better utilization of silicon area. In contrast, some FPGAs use MUX-based logic blocks or product-term-based logic blocks. These blocks, although having more than k input ports, cannot guarantee an implementation for an arbitrary k-input function. New universal logic module (ULM)-based FPGA logic blocks [34] had been proposed for better covering of K-input functions, but the coverage was still incomplete (99% of four-input functions using an eight-input ULM). Since LUT is widely used in today's major FPGAs and is a true ULM for functions of its input size, we focus on implementing functions using LUT-based PLBs in this paper.

A PLB can often implement one arbitrary K-input function, where K is determined by the PLB architecture or some *wide* function of more than K inputs. Unfortunately, it is generally a difficult problem to determine if an arbitrary given wide function can be implemented by a PLB. This is called the *Boolean matching for PLB* problem. Most existing technology mapping algorithms first produce a K-LUT mapping solution, then pack LUTs into PLBs [9], [10], [18], [19], [21], [27], [30], [31], [39]. A comprehensive survey of recent FPGA technology mapping approaches can be found in [11].

In this paper, we study Boolean matching for PLB problems. We focus on two classes of (LUT-based) PLBs: PLB1 and PLB2, as shown in Fig. 1(a) and (b), respectively. Both PLBs contain two LUTs, F and G, at the first stage and another element H at the second stage where H is a 3-LUT in PLB1 and a two-input MUX in PLB2. The sets of input signals to F, G, and H are denoted as X_F, X_G, and X_H, respectively, and

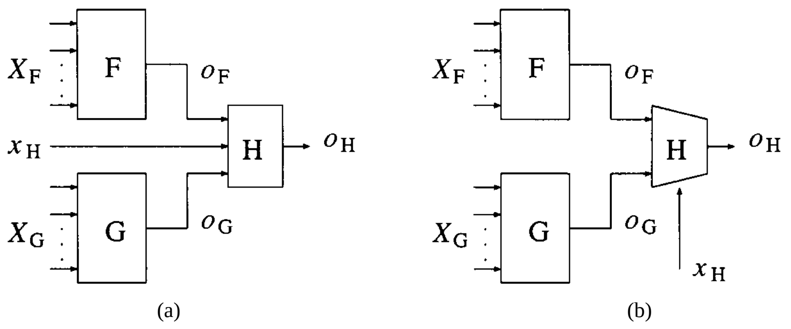

Fig. 1. Two generic PLB architectures. (a) PLB1. (b) PLB2.

the output signals of F, G, and H are denoted as o_F, o_G, and o_H, respectively. For PLB1, the 3-LUT H has inputs o_F, o_G, and external signal x_H. For PLB2, the MUX H selects either o_F or o_G to output depending on the value of external selection signal x_H.

The LUT sizes $|X_F|$ and $|X_G|$ and the appearance of x_H are architecture parameters that may vary from one PLB to another. Therefore, we shall represent PLB1 as PLB1 $(|X_F|, |X_G|, h)$, where the first two parameters are the sizes of LUTs F and G, and $h = 1$ if the x_H line exists, otherwise $h = 0$. Similarly, we shall represent PLB2 as PLB2 $(|X_F|, |X_G|)$. Under this notation, $\text{PLB1}(4, 4, 1)$ is identical to the logic block of Xilinx XC4K series FPGAs [38], which is called the configurable logic block (CLB), while $\text{PLB2}(5, 5)$ is the logic block of Lucent's ORCA series FPGAs [33], which is called the programmable function unit, and $\text{PLB2}(4, 4)$ is the Xilinx XC5200 CLB. When there is no confusion in the context, we refer to the two PLBs as PLB1 and PLB2.

Boolean matching for PLBs may lead to significant reduction on mapping area and circuit delay. For example, consider a six-variable function (in the Xilinx test suite used in Section VI-A), which is represented by the following sum-of-product form:

$$\begin{aligned} f(X) = {} & \bar{x}_1 x_3 x_5 x_6 \overline{(x_4 + x_2)} + \bar{x}_1 x_3 x_5 \bar{x}_6 (x_4 + x_2) \\ & + x_1 \bar{x}_2 x_5 x_6 (x_4 + x_2) + x_1 x_2 \bar{x}_3 \bar{x}_6 (x_4 + \bar{x}_2) \\ & + \bar{x}_1 \bar{x}_3 \bar{x}_5 x_6 (x_4 + \bar{x}_2) + x_1 x_3 x_6 (x_4 + x_2) \\ & + \bar{x}_1 \bar{x}_3 \bar{x}_6 \overline{(x_4 + \bar{x}_2)} + x_1 x_5 x_6 \overline{(x_4 + \bar{x}_2)} \\ & + x_1 \bar{x}_6 \overline{(x_4 + x_2)}. \end{aligned}$$

We compare three implementations of $f(X)$ produced by Chortle-crf [19], FlowMap [9], and Boolean matching algorithms, using XC4K CLBs. For the first two implementations, we applied the optimization script *rugged* in the sequential circuit synthesis system SIS [32], mapped the resulting gate-level network using the area-oriented mapper Chortle-crf and the depth-optimal mapper FlowMap respectively, and packed the resulting LUTs into CLBs using the efficient CLB packing procedures in [15]. The CLB networks that result from Chortle-crf and FlowMap have three levels with seven and six CLBs, respectively. However, a Boolean matching approach can obtain a single CLB implementation of $f(X)$ (Fig. 2) with each LUT implementing the following function:

$$o_H = x_6 \bar{o}_F o_G + \bar{x}_6 o_F o_G$$

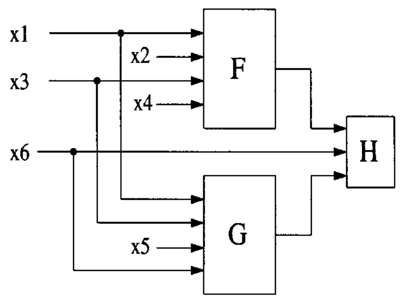

Fig. 2. Single CLB implementation of function $f(X)$.

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

In this implementation, x_1 and x_3 are bridged inputs of LUTs F and G (i.e., x_1 and x_3 are shared by F and G) and x_6 is a bridged input of LUTs H and G. Prior to this paper, it was an open problem of how to perform Boolean matching for PLBs, especially when bridged inputs are taken into consideration.

Most existing Boolean matching approaches are for ASIC design synthesis using cell libraries. A good survey can be found in [3]. Very few are targeted for LUT-based logic blocks. Boolean matching approaches were proposed in [4] and [40] for Actel's MUX-based FPGAs [1]. Their approaches cannot be applied to LUT-based PLBs directly. Mapping algorithms targeted for PLBs were proposed in [8], where linear programming was employed to compute LUT covers and PLB packings simultaneously. Functional decomposition-based mapping approaches [20], [24], [26], [37] were proposed for LUT network synthesis. None of them were targeted to implementing wide functions using PLBs. A recent work [29] studied bidecomposition of Boolean functions and applied the results to Boolean matching for some LUT-based PLBs. The results were limited. For example, the researchers were unable to solve the Boolean matching problem for the XC4K CLB completely. In this paper, we present new Boolean matching methods for PLBs. Our methods are based on classical and new functional decomposition techniques and provide a more general solution to the Boolean matching problem for LUT-based PLBs. For example, our results give exact solutions for matching functions to the XC4K CLBs. We apply our techniques to quantitative evaluation of PLB architectures (in terms of logic implementation capability) as well as to technology mapping for FPGAs that are widely used today. Our Boolean matching approaches for PLB1 and PLB2 may be extended to other LUT-based PLBs.

This paper is organized as follows. Section II formulates the Boolean matching problem. After the introduction of classical and recent functional decomposition results in Section III, Boolean matching methods for PLB1 and PLB2 are presented in Sections IV and V, respectively. Section VI reports Boolean matching and architecture evaluation experimental results. In Section VII, we present technology mapping algorithms that employ our Boolean matching methods and report the experimental results in Section VIII. Section IX concludes the paper. Preliminary results of this study were presented in [13] and [14].

II. Problem Formulation

Given a multilevel network of logic gates, combinational logic synthesis transforms the given network into a network of PLBs. This transformation usually includes two major steps: 1) *logic optimization* and 2) *technology mapping*. Logic optimization transforms the given network into an equivalent network that is suitable for mapping into PLBs, e.g., into a network of fewer gates and/or smaller gates. Technology mapping then transforms the resulting network into a PLB network of minimal cost, where the cost could be network area of delay. Conventional technology mapping usually include covering the gate-level network with LUTs and packing LUTs into PLBs. The FlowMap and the Chortle-crf algorithms are two widely used technology mapping algorithms. However, as shown in the previous section, they could generate suboptimal solutions compared to Boolean matching approaches. In this paper, we focus on the Boolean matching for PLBs and then apply our results to technology mapping.

For each LUT-based PLB architecture, we define the *characteristic number* K of the PLB to be the largest number such that any function of K or fewer variables is realizable by the PLB. Functions of more than K variables are called *wide functions* with respect to the PLB. Clearly, if the PLB has more than K inputs, it can implement some wide functions. For example, if $|X_F| = |X_G| = 4$ in PLB1 and PLB2, then either PLB can implement any function of up to five variables ($K = 5$) or some wide function of up to nine variables. The Boolean matching problem for PLB is to determine, for a given wide function with respect to the PLB, whether the function can be implemented by the PLB.

Boolean Matching for PLB: Given a wide function $f(X)$ with respect to a PLB A, determine if $f(X)$ can be realized by a single PLB A.

In general, Boolean matching considers input negation and/or permutation, output inversion, bridging of inputs, and constant assignments to some inputs. For LUT-based PLBs, however, input negation, input permutation and bridging in one LUT, output inversion, and constant assignments to unused inputs do not affect the matching feasibility. Therefore, only input partitioning and input sharing among LUTs are relevant factors to our Boolean matching for PLB problem.

III. Functional Decomposition

Our solution to the Boolean matching for PLB problem relies on functionally decomposing wide functions. In this section, we shall introduce decomposition forms that are closely related to PLB matching and review existing results and present new results on functional decomposition.

A. Preliminaries

Let $f(X)$ denote Boolean function $f(x_1, x_2, \ldots, x_n)$, where $X = \{x_1, x_2, \ldots, x_n\}$. If PLB1 implements $f(X)$, we can represent $f(X)$ as $g(y_1(X_F), y_2(X_G), x_H)$, where $X_F \cup X_G \cup \{x_H\} = X$. This implies the existence of a functional decomposition of $f(X)$. If PLB2 implements $f(X)$, we can represent $f(X)$ as $\bar{x}_H \cdot y_1(X_F) + x_H \cdot y_2(X_G)$. This is the Shannon expansion of $f(X)$ with respect to the variable x_H. Therefore, decomposition of functions plays an important role in our Boolean matching approaches. Before presenting our approaches, we will give a brief review of classical and recent functional decomposition results.

Given a Boolean function $f(X)$, let $f_{\bar{x}_i}$ and f_{x_i} denote the cofactors of $f(X)$ with respect to variable x_i. Cofactors of $f(X)$ with respect to multiple variables are defined in a similar way. For example, $f_{x_1\bar{x}_2} = f(X)|_{x_1=1,x_2=0}$. The Shannon expansion of $f(X)$ with respect to x_i is $f(X) = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$. The Shannon expansion of $f(X)$ with respect to multiple variables is defined in a similar way. The cofactor set of $f(X)$ with respect to a set $B \subseteq X$, denoted $\mathrm{cs}_B(f)$, is the set of all *distinct* cofactors of $f(X)$ with respect to the variables in B. We denote the support of function $f(X)$ as $\sup(f) = X$.

Given a function $f(X)$ and a set $B = \{x_1, x_2, \ldots, x_b\} \subset X$, the *disjoint functional decomposition* of $f(X)$ under B represents $f(X)$ in the following form:

$$f(X) = g(y_1(B), y_2(B), \ldots, y_t(B), x_{b+1}, \ldots, x_n).$$

We call the set B the *bound set* and the set $X - B$ of remaining variables the *free set* of the decomposition. The decomposition is *nontrivial* if $t < b$. If $t = 1$, it is called a *simple disjoint decomposition*. Each $y_i(B)$ is called an *encoding function*. Let $Y = \{y_1, y_2, \ldots, y_t\}$ and $Y(B) = \{y_1(B), y_2(B), \ldots, y_t(B)\}$. We can write the decomposition as $f(X) = g(Y(B), X - B)$.

Two other functional decomposition forms are closely related to the disjoint decomposition: the nondisjoint decomposition and the partially dependent decomposition. A *nondisjoint* decomposition of $f(X)$ is the case when some bound set variables appear in the support of g. Let $S = \{x_j, \ldots, x_b\} \subset B$ for some $j > 1$. Then, a nondisjoint functional decomposition of $f(X)$ is to represent $f(X)$ as $g(Y(B), S, X - B) = f(X)$. The decomposition is nontrivial when $|Y| + |S| < |B|$. Variables in S are called *nondisjoint variables*. When $S = \emptyset$, the decomposition becomes disjoint.

A *partially dependent* decomposition is the case when the support of some encoding function is a *strict subset* of B, e.g., $\sup(y_1) = B_m$, where $B_m = \{x_1, \ldots, x_m\} \subset B$. Such an encoding function is called a *partially dependent* encoding function. If the support of some partially dependent encoding function contains only one variable, this encoding function can be replaced by the variable; hence, the decomposition becomes a nondisjoint decomposition.

If $f(X)$ can be represented in the form $g(y_1(X_F), y_2(X_G))$, where $X_1 \cup X_2 = X$, it is called a *bidecomposition* of $f(X)$. A bidecomposition is *disjoint* if $X_1 \cap X_2 = \emptyset$, otherwise, it is *nondisjoint*. Bidecomposition plays an important role in the Boolean matching for PLB1 when the x_H line is not used or when there are bridged inputs.

For any implementation of $f(X)$ on PLB1, existence of bridged inputs implies nondisjoint functional decomposition or bidecomposition of $f(X)$ or a combination of them. For PLB2, existence of bridged inputs implies the two cofactors share common variables.

B. Existence Conditions

We now briefly review the existence conditions for various forms of functional decomposition. Ashenhurst [2] gave the existence condition for simple disjoint decomposition.

Theorem 1: There exists a simple disjoint decomposition $g(y_1(B), X - B)$ of $f(X)$ under the bound set B if and only if $|\mathrm{cs}_B(f)| \leq 2$ [2].

The condition was extended to general disjoint decomposition ($t \geq 1$) by Curtis [17].

Theorem 2: There exists a disjoint decomposition $g(Y(B), X - B)$ of $f(X)$ under the bound set B if and only if $|\mathrm{cs}_B(f)| \leq 2^{|Y|}$ [17].

The existence of disjoint bidecomposition $g(y_1(X_1), y_2(X_2))$ of $f(X)$ can be determined by verifying the existence of simple disjoint decomposition of $f(X)$ under the bound sets X_1 and X_2, respectively. In other words, a disjoint bidecomposition can be obtained by combining two simple disjoint decompositions. This result is implied by the following theorem.

Theorem 3: Let $X_1, X_2, \ldots, X_t$ be a disjoint partition of X. A functional decomposition $g(y_1(X_1), y_2(X_2), \ldots, y_t(X_t))$ of $f(X)$ exists if and only if there exists simple disjoint decomposition of $f(X)$ under each bound set $X_1, X_2, \ldots, X_t$, respectively [2].

For nondisjoint bidecompositions of $f(X)$, it was shown in [29] that they can be obtained by applying disjoint bidecomposition to the cofactors of $f(X)$. In fact, a few forms of nondisjoint decomposition that correspond to different PLB input bridging patterns can be obtained in a similar way. Because of this, we shall defer to the next section to present the existence condition when we introduce the Boolean matching approach for each PLB input bridging pattern.

A few approaches for partially dependent decomposition and nondisjoint decomposition were proposed in the past few years [13], [20], [25], [26], [28], with different perspectives and algorithmic characteristics. In particular, an existence condition for both partially dependent decomposition and nondisjoint decomposition was given in [13]. The condition can be used to compute one partially dependent encoding function or nondisjoint variable efficiently.

Theorem 4: Let $B_m \subset B$. There exists a partially dependent decomposition $g(Y(B), X - B)$ of $f(X)$ under the bound set B with $\sup(y_1) = B_m$ if and only if $\mathrm{cs}_{B_m}(f)$ can be partitioned into two sets cs_0 and cs_1 such that [13]

$$\left| \bigcup_{h \text{ in } \mathrm{cs}_0} \mathrm{cs}_{B-B_m}(h) \right| \leq 2^{|Y|-1} \quad \text{and}$$
$$\left| \bigcup_{h \text{ in } \mathrm{cs}_1} \mathrm{cs}_{B-B_m}(h) \right| \leq 2^{|Y|-1}.$$

Proof—Only If: Assume $f(X) = g(Y(B), X - B)$ and $\sup(y_1) = B_m$. Let p be a minterm in the Boolean space defined on the variable set B_m. Define the set $\mathrm{cs}_0 = \{f(p, X - B_m) \,|\, y_1(p) = 0\}$. Let $\mathrm{cs}_{0,B}(g)$ denote the set of cofactors $g(0, y_2(B), \ldots, y_{|Y|}(B), X - B)$ with respect to variables in B. First, the set $\cup_{h \in \mathrm{cs}_0} \mathrm{cs}_{B-B_m}(h)$ of cofactors is a subset of $\mathrm{cs}_{0,B}(g)$ because every function $h \in \mathrm{cs}_0$ has the form

$$h = f(p, X - B_m)|_{y_1(p)=0} = g(0, y_2(p, B - B_m), \ldots, y_{|Y|}(p, B - B_m), X - B)|_{y_1(p)=0}.$$

Second, $|\mathrm{cs}_{0,B}(g)| \leq 2^{|Y|-1}$ because $\mathrm{cs}_{0,B}(g)$ must be a subset of $\mathrm{cs}_Y(g_{\bar{y}_1})$, the set of cofactors $g(0, y_2, \ldots, y_{|Y|}, X - B)$ with respect to variables $y_2, \ldots, y_m$. Therefore, $| \cup_{h \in \mathrm{cs}_0} \mathrm{cs}_{B-B_m}(h)| \leq 2^{|Y|-1}$. Similarly, define $\mathrm{cs}_1 = \{f(p, X - B_m) \,|\, y_1(p) = 1\}$ and we have $| \cup_{h \in \mathrm{cs}_1} \mathrm{cs}_{B-B_m}(h)| \leq 2^{|Y|-1}$. If cs_0 and cs_1 form a partition of $\mathrm{cs}_{B_m}(f)$, we have proved the condition. Otherwise, consider $\mathrm{cs}_0' = \mathrm{cs}_0 - \mathrm{cs}_1$. Then, cs_0' and cs_1 form a partition that satisfies the necessary condition.

If: Assume $\cup_{h \in \mathrm{cs}_0} \mathrm{cs}_{B-B_m}(h)$ and $\cup_{h \in \mathrm{cs}_1} \mathrm{cs}_{B-B_m}(h)$ each contains at most 2^{t-1} members. Let p be a minterm in the Boolean space defined on the variable set B_m. Then, $f(p, X - B_m)$ represents a cofactor in $\mathrm{cs}_{B_m}(f)$. Define Boolean function $y_1(B_m)$ to be $y_1(p) = 0$ if $f(p, X - B_m) \in \mathrm{cs}_0$ and $y_1(p) = 1$ if $f(p, X - B_m) \in \mathrm{cs}_1$. Let $f_-(X - B_m)$ be an arbitrary cofactor in cs_0. We define $f_0(X)$ to be $f_0(p, X - B_m) = f(p, X - B_m)$ if $y_1(p) = 0$ and $f_0(p, X - B_m) = f_-(X - B_m)$ if $y_1(p) = 1$ for every minterm p. Then, the cofactor set of $f_0(X)$ under the bound set B, $\mathrm{cs}_B(f_0)$ is identical to $\cup_{h \in \mathrm{cs}_0} \mathrm{cs}_{B-B_m}(h)$, which contains at most 2^{t-1} members.

According to Theorem 2, there exists a disjoint decomposition of $f_0(X)$ under the bound set B with $t - 1$ encoding functions: $f_0(X) = g_0(w_2(B), \ldots, w_t(B), X - B)$. Similarly, let $f_+(X - B_m)$ be an arbitrary cofactor in cs_1. Define function $f_1(X)$ to be $f_1(p, X - B_m) = f_+(X - B_m)$ if $y_1(p) = 0$ and $f_1(p, X - B_m) = f(p, X - B_m)$ if $y_1(p) = 1$ for every minterm p. Then, there exists a disjoint decomposition of $f_1(X)$ with $t - 1$ encoding functions: $f_1(X) = g_1(z_2(B), \ldots, z_t(B), X - B)$. Because $f = \bar{y}_1 f_0 + y_1 f_1$, we have $f = \bar{y}_1 g_0 + y_1 g_1$. Define $y_i = \bar{y}_1 w_i + y_1 z_i$ $(2 \leq i \leq t)$ and $g = \bar{y}_1 g_0 + y_1 g_1$. Then, we have

$$\begin{aligned} f(X) &= \bar{y}_1(B_m) \cdot g_0(y_2(B), \ldots, y_t(B), X - B) \\ &\quad + y_1(B_m) \cdot g_1(y_2(B), \ldots, y_t(B), X - B) \\ &= g(y_1(B_m), y_2(B), \ldots, y_m(B), X - B). \end{aligned}$$

This proves the sufficient condition. □

For finding multiple partially dependent encoding functions, the authors in [13] took an approach similar to that in [36] and [37]. The existence of nondisjoint decomposition of $f(X)$ with the nondisjoint variable x_i can be checked by setting $B_m = \{x_i\}$ in Theorem 4. For the general case of multiple nondisjoint variables, we have the following theorem.

Theorem 5 : Given a bound set B and a set of nondisjoint variables $S \subset B$, there exists a nondisjoint decomposition $g(Y(B), S, X - B)$ of $f(X)$ if and only if $|\mathrm{cs}_{B-S}(h)| \leq 2^{|Y|}$ for every cofactor $h \in \mathrm{cs}_S(f)$ [13].

Proof: We prove it by mathematical induction on $|S|$.

Basis: This theorem holds for $|S| = 1$ according to Theorem 4 with $B_m = S$. Without loss of generality, assume $S = \{x_1\}$. Then, $\mathrm{cs}_{B_m}(f) = \{f_{\bar{x}_1}, f_{x_1}\}$. The only possible partition is $\mathrm{cs}_0 = \{f_{\bar{x}_1}\}$ and $\mathrm{cs}_1 = \{f_{x_1}\}$. Theorem 4 then implies the basis.

Hypothesis: Assume this theorem holds for $|S| = k - 1$. Now, we prove the case for $|S| = k$.

Only If: Without loss of generality, assume $x_1 \in S$. Define $S' = S - \{x_1\}$. Then, $f_{\bar{x}_1} = g_{\bar{x}_1}(Y(B), S', X - B)$, a decomposition with $k - 1$ nondisjoint variables. According to induction hypothesis, every cofactor $h \in \mathrm{cs}_{S'}(f_{\bar{x}_1})$ satisfies $|\mathrm{cs}_{B-S}(h)| \leq 2^{|Y|}$. Similarly, every cofactor $h \in \mathrm{cs}_{S'}(f_{x_1})$ satisfies $|\mathrm{cs}_{B-S}(h)| \leq 2^{|Y|}$. However, $\mathrm{cs}_S(f) = \mathrm{cs}_{S'}(f_{\bar{x}_1}) \cup \mathrm{cs}_{S'}(f_{x_1})$. Therefore, $|\mathrm{cs}_{B-S}(h)| \leq 2^{|Y|}$ for every $h \in \mathrm{cs}_S(f)$.

If: Assume $|\mathrm{cs}_{B-S}(h)| \leq 2^t$ for every $h \in \mathrm{cs}_S(f)$. Define $B' = B - \{x_1\}$. Since $\mathrm{cs}_S(f) = \mathrm{cs}_{S'}(f_{\bar{x}_1}) \cup \mathrm{cs}_{S'}(f_{x_1})$, every cofactor $h \in \mathrm{cs}_{S'}(f_{\bar{x}_1})$ satisfies $|\mathrm{cs}_{B-S}(h)| \leq 2^t$. According to induction hypothesis, $f_{\bar{x}_1} = g_0(W(B'), S', X - B)$, where $|W| = t$. Similarly, $f_{x_1} = g_1(Z(B'), S', X - B)$ and $|Z| = t$. Define $g = \bar{x}_1 g_0 + x_1 g_1$ and $y_i = \bar{x}_1 w_i + x_1 z_i$, where $w_i(B') \in W(B')$ and $z_i(B') \in Z(B')$. Then, we have

$$\begin{aligned} f(X) &= \bar{x}_1 \cdot g_0(Y(B), S', X - B) \\ &\quad + x_1 \cdot g_1(Y(B), S', X - B) \\ &= g(Y(B), S, X - B). \end{aligned}$$

This proves the theorem. □

IV. Boolean Matching Approaches for PLB1

In this section, we consider matching wide functions to PLB1 in Configurations A, B, C, and D shown in Fig. 3 obtained from different bridging status of input x_H. In Configuration A, x_H does not feed to LUT H; in Configuration B, x_H feeds to LUT H only; in Configuration C, x_H feeds to all three LUTs, while in Configuration D, x_H feeds to LUTs H and G. For each configuration, there might be bridged inputs to LUTs F and G, shown as dash lines in Fig. 3. There can be multiple bridged inputs to F and G as long as the total number of distinct inputs matches that of the wide function. For example, up to two bridged inputs to F and G are allowed when we consider matching a six-variable function to a XC4K CLB in Configuration D. It should be clear that the four configurations exhaust all possible ways of using PLB1. Hence, the Boolean matching problem for PLB1 can be solved by matching functions to the four configurations individually.

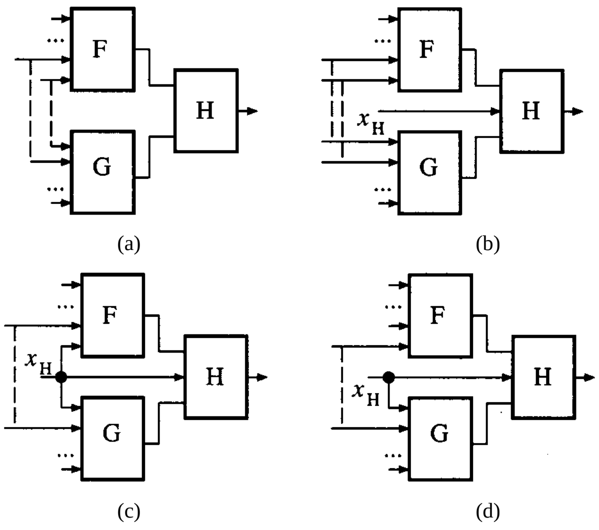

Fig. 3. Four PLB1 configurations. (a) Configuration A. (b) Configuration B. (c) Configuration C. (d) Configuration D.

A. PLB1 in Configuration A

If PLB1 in Configuration A implements $f(X)$, a bidecomposition $g(y_1(X_F), y_2(X_G))$ must exist for $f(X)$. If bridged inputs exist in the implementation, the decomposition is nondisjoint. Otherwise, it is disjoint. For the case without bridged inputs, an implementation, if it exists, can be obtained by combining simple disjoint decompositions of $f(X)$ under the bound sets X_F and X_G, respectively (according to Theorem 3).

We now consider the case when PLB1 implements $f(X)$ with bridged inputs. Let $X_F = X_1 \cup X_3$ and $X_G = X_2 \cup X_3$, where X_3 is the set of bridged inputs. In general, $|X_3| > 1$. For the case where input x_i is bridged, we have the following theorem.

Theorem 6 : Let $X_F = X_1 \cup \{x_i\}$, $X_G = X_2 \cup \{x_i\}$. A nondisjoint bidecomposition $g(y_1(X_F), y_2(X_G))$ of $f(X)$ exists if and only if there exist bidecompositions $f_{\bar{x}_i} = g_0(y_{01}(X_1), y_{02}(X_2))$ and $f_{x_i} = g_1(y_{11}(X_1), y_{12}(X_2))$ such that $g_0(y_1, y_2) = g_1(y_1, y_2)$ [29].

To determine if a function $f(X)$ can be implemented with PLB1 in Configuration A, we first enumerate bipartitions of X and test $f(X)$ for disjoint bidecompositions based on Theorem 3. If none can be found, we choose variables in X as bridged inputs enumeratively and test $f(X)$ for nondisjoint bidecompositions based on Theorem 6. When multiple bridged inputs are explored, we decompose cofactors recursively. For example, to test for a matching with bridged inputs x_i and x_j, we compute all nondisjoint bidecompositions for cofactors $f_{\bar{x}_i}$ and f_{x_i} with x_j being the nondisjoint variable and then determine if the resulting bidecompositions can satisfy the condition in Theorem 6 with x_i being the nondisjoint variable. If the condition can be satisfied, then bidecompositions of $f(X)$ with two bridged inputs x_i and x_j exist.

Note that the bidecompositions of cofactors $f_{\bar{x}_i}$ and f_{x_i} are not unique in general. When we compute nondisjoint bidecompositions, in order to check if g_0 can be equal to g_1 for some feasible bidecompositions, we invert $y_{01}(X_1)$, $y_{02}(X_2)$, or both and compute the corresponding g_0 function under each case. In total, we obtain four equivalent bidecompositions of $f_{\bar{x}_i}$. Then, we compare each resulting function g_0 with g_1 for a match. Since $|\sup(g_0)| = 2$, we have exhausted all bidecompositions of $f_{\bar{x}_i}$.

A special case that requires particular attention is when one of the cofactors is a constant (zero or one). For example, assume that $f_{\bar{x}_i}$ is a constant. Then, we can always obtain a nondisjoint bidecomposition of $f(X)$ by duplicating g_1 for g_0 followed by producing constant encoding functions $y_{01}(X_1)$ and $y_{02}(X_2)$ such that g_0 will return the same value as $f_{\bar{x}_i}$. The condition in Theorem 6 is, thus, satisfied for this special case. When both cofactors are constant, we choose an arbitrary two-input function, e.g., the XOR function, for both g_0 and g_1 function and choose (constant) encoding functions accordingly.

We illustrate the special case by an example. Let $X = \{a, b, c, d, e, j\}$ and $f(X) = \bar{a} + a(bcd + ej)$. We want to check if a nondisjoint bidecomposition $f(X) = g(y_1(a, b, c, d), y_2(a, e, j))$ exists. According to Theorem 6, we first compute two cofactors with respect to a and obtain $f_{\bar{a}} = 1$ and $f_a = bcd + ej$. Note that f_a can be easily decomposed into the form $g_1(y_{11}(b, c, d), y_{12}(e, j))$, where $g_1(y_{11}, y_{12}) = y_{11} + y_{12}$, $y_{11}(b, c, d) = bcd$, and $y_{12}(e, j) = ej$. In order to obtain a nondisjoint bidecomposition of $f(X)$, we would like to represent $f_{\bar{a}} = 1$ as $g_0(y_{01}, y_{02})$ such that $g_0 = g_1$. We choose $g_0(y_{01}, y_{02}) = y_{01} + y_{02}$ with $y_{01} = 1$ and $y_{02} = 1$. Then, combining the two decompositions, we have $f(X) = g(y_1(a, b, c, d), y_2(a, e, j))$, where $y_1(a, b, c, d) = \bar{a} \cdot g_{01} + a \cdot g_{11}(b, c, d) = \bar{a} + abcd$, $y_2(a, e, j) = \bar{a} \cdot g_{02} + a \cdot g_{12}(e, j) = \bar{a} + aej$, and $g(y_1, y_2) = y_1 + y_2$. To achieve efficient computation, functional decomposition operations are performed using ordered binary decision diagrams [5] in our implementation, as in approaches such as [6], [25], and [26].

B. PLB1 in Configuration B

If PLB1 in Configuration B implements $f(X)$, a decomposition $g(y_1(X_F), y_2(X_G), x_H)$ must exist for $f(X)$. There could be bridged inputs between LUTs F and G, but x_H does not bridge to either of them.

Let $X_{FG} = X_F \cup X_G$. We consider two cases: 1) $|\mathrm{cs}_{X_{FG}}(f)| = 3, 4$ or 2) $|\mathrm{cs}_{X_{FG}}(f)| = 2$. (It is impossible that $|\mathrm{cs}_{X_{FG}}(f)| > 4$ because any function $h(x_H)$ can only be one of zero, one, x_H, or $\bar{x}_H$.) In Case 1, there exists a disjoint decomposition of $f(X)$ under the bound set X_{FG} with two encoding functions (according to Theorem 2). Let $f(X) = g(y_1(X_{FG}), y_2(X_{FG}), x_H)$ be the decomposition. Then, PLB1 in Configuration B can implement $f(X)$ if and only if the encoding functions are feasible for LUTs F and G. This implies a partially dependent decomposition such that $|\sup(y_1)| \le |X_F|$ and $|\sup(y_2)| \le |X_G|$. By checking every possible input variable for x_H, we can determine the existence of a match for Case 1 based on Theorem 4.

In Case 2, a simple disjoint decomposition $g(y_1(X_{FG}), x_H)$ of $f(X)$ exists under the bound set X_{FG}, which does not match directly to configuration B. However, it is possible to replace a large encoding function $(y_1(X_{FG}))$ with two small encoding functions. In particular, we proved that a function $f(X) = g(y_1(X_{FG}), x_H)$ can be represented as $g'(z_1(X_F), z_2(X_G), x_H)$ when the following condition holds.

Theorem 7: Let $f(X) = g(y_1(X_{FG}), x_H)$ and $X_F \cup X_G = X_{FG}$. A partially dependent decomposition $g'(z_1(X_F), z_2(X_G), x_H)$ of $f(X)$ exists if and only if a bidecomposition $h(z_1(X_F), z_2(X_G))$ of $y_1(X_{FG})$ exists.

Proof—Only If: Consider $f_{x_H} = g(y_1(X_{FG}), 1)$. Because $g(y_1, 1) = y_1$ or $\bar{y}_1$, we have $f_{x_H} = y_1(X_{FG})$ or $\bar{y}_1(X_{FG})$. On the other hand, $f_{x_H} = g'_{x_H}(z_1(X_F), z_2(X_G), 1) = h(z_1(X_F), z_2(X_G))$. Therefore, $y_1(X_{FG}) = h(z_1(X_F), z_2(X_G))$ or $y_1(X_{FG}) = \bar{h}(z_1(X_F), z_2(X_G))$. In either case, a bidecomposition of $y_1(X_{FG})$ is found.

If: Since $f(X) = g(y_1(X_{FG}), x_H) = g(h(z_1(X_F), z_2(X_G)), x_H) = g'(z_1(X_F), z_2(X_G), x_H)$, the necessary condition is proved. □

Note that Cases 1 and 2 cover all possible implementations of $f(X)$ on PLB1 in Configuration B. Therefore, using the partially dependent decomposition algorithm in [13] and Theorem 7, we can determine if a wide function can be implemented on PLB1 in Configuration B. Note that $X_F \cap X_G \neq \emptyset$ is not excluded in both cases. Therefore, bridged inputs to LUTs F and G have been considered implicitly.

C. PLB1 in Configuration C

If PLB1 in Configuration C implements $f(X)$, a decomposition $g(y_1(X_F), y_2(X_G), x_H)$ of $f(X)$ must exist where x_H belongs to X_F and X_G. Besides x_H, other bridged inputs to LUTs F and G may exist in the implementation. We proved the following theorem for matching $f(X)$ to Configuration C.

Theorem 8: Let $x_H \in X_F \cap X_G$, $X_1 = X_F - \{x_H\}$, and $X_2 = X_G - \{x_H\}$. A decomposition of the form $g(y_1(X_F), y_2(X_G), x_H)$ for $f(X)$ exists if and only if there exist bidecompositions $f_{\bar{x}_H} = g_0(y_{01}(X_1), y_{02}(X_2))$ and $f_{x_H} = g_1(y_{11}(X_1), y_{12}(X_2))$.

Proof—Only If: Define $y_{01}(X_1) = y_1(X_1)|_{x_H=0}$, $y_{02}(X_2) = y_2(X_2)|_{x_H=0}$, and $g_0(X_1 \cup X_2) = g_{\bar{x}_H}(X_1 \cup X_2)$. Then, $f_{\bar{x}_H} = g_0(y_{01}(X_1), y_{02}(X_2))$. Similarly, define $y_{11}(X_1) = y_1(X_1)|_{x_H=1}$, $y_{12}(X_2) = y_2(X_2)|_{x_H=1}$, and $g_1(X_1 \cup X_2) = g_{x_H}(X_1 \cup X_2)$. Then, $f_{x_H} = g_1(y_{11}(X_1), y_{12}(X_2))$. The necessary condition is proved.

If: Define $y_1(X_F) = \bar{x}_H y_{01}(X_1) + x_H y_{02}(X_2)$, $y_2(X_G) = \bar{x}_H y_{11}(X_1) + x_H y_{12}(X_2)$, and $g = \bar{x}_H g_0 + x_H g_1$. Then, $f(X) = \bar{x}_H f_{\bar{x}_H} + x_H f_{x_H}$ implies

$$\begin{aligned} f(X) = {} & \bar{x}_H g_0(y_1(X_1, x_H), y_2(X_2, x_H)) \\ & + x_H g_1(y_1(X_1, x_H), y_2(X_2, x_H)). \end{aligned}$$

A decomposition $f(X) = g(y_1(X_F), y_2(X_G), x_H)$ is then obtained. □

To test the matching of a function $f(X)$ to Configurations C of PLB1, we enumerate x_H from X and compute bidecompositions for the cofactors $f_{\bar{x}_H}$ and f_{x_H} such that the condition in Theorem 8 is satisfied. Because x_H feeds to LUTs F, G, and H, the existence condition for Configuration C is less constrained compared to Configurations A and D (Section IV-D).

D. PLB1 in Configuration D

If PLB1 in Configuration D implements $f(X)$, a decomposition $g(y_1(X_F), y_2(X_G), x_H)$ of $f(X)$ must exist with $x_H \in X_G$. Other bridged inputs to LUTs F and G may exist. We prove the following result.

Theorem 9: Let $x_H \in X_G$ and $X_2 = X_G - \{x_H\}$. A decomposition of the form $g(y_1(X_F), y_2(X_G), x_H)$ for $f(X)$ exists if and only if $f_{\bar{x}_H} = g_0(y_{01}(X_F), y_{02}(X_2))$ and $f_{x_H} = g_1(y_{11}(X_F), y_{12}(X_2))$ such that $y_{01}(X_F) = y_{11}(X_F)$.

Proof: The proof is similar to that of Theorem 8 except that we require $y_{01}(X_F) = y_{11}(X_F) = y_1(X_F)$. □

To implement a function $f(X)$ on a PLB1 in Configurations D, we select each x_H in turn from X followed by computing bidecompositions of $f_{\bar{x}_H}$ and f_{x_H} to satisfy the condition in Theorem 9. Since bidecompositions of functions are not unique, for every bidecomposition $g_0(y_{01}(X_F), y_{02}(X_2))$ of $f_{\bar{x}_H}$, we derive another bidecomposition with inverted encoding function $\bar{y}_{01}(X_F)$. Both bidecompositions are then tested for the condition in Theorem 9. Note that we do not derive four bidecompositions, as in the matching to Configuration A, because it is $y_{01}(X_F)$ and $y_{11}(X_F)$ under comparison rather than g_0 and g_1.

A special case that requires particular attention is when one of the cofactors is a constant (zero or one). For example, assume that $f_{\bar{x}_H}$ is a constant. Then, we can always obtain a decomposition of $f(X)$ by duplicating y_{11} for y_{01} after obtaining the bidecomposition of f_{x_H} followed by producing $g_0(y_{01}, y_{02})$ such that its value does not depend on the y_{01} input (e.g., $g_0(y_{01}, y_{02}) = y_{02}$, where y_{02} is the constant $f_{\bar{x}_H}$). Therefore, the condition in Theorem 9 is always satisfied for the special case. When both cofactors are constant, we set both y_{01} and y_{11} to constant and construct g_0 and g_1 accordingly to obtain a decomposition.

If there is no bridged input between LUTs F and G (i.e., $X_F \cap X_G = \emptyset$), we may also use the following theorem for efficient matching to Configuration D.

Theorem 10: Let $x_H \in X_G$ and $X_F \cap X_G = \emptyset$. A decomposition of the form $g(y_1(X_F), y_2(X_G), x_H)$ for $f(X)$ exists if and only if a simple disjoint decomposition $h_F(y_1(X_F), X_G)$ and a nondisjoint decomposition $h_G(X_F, y_2(X_G), x_H)$ of $f(X)$ both exist.

Proof—Only If: The two decompositions h_F and h_G of $f(X)$ can be obtained by collapsing y_1 and y_2 into g, respectively. The necessary condition is proved. □

If: Since $X_F \cap X_G = \emptyset$, the existence of simple disjoint decomposition h_F of $f(X)$ implies the existence of a simple disjoint decomposition of h_G under the bound set X_F, which in turn implies the existence of decomposition form $g(y_1(X_F), y_2(X_G), x_H)$ for $f(X)$. This proves the sufficient condition.

Using Theorem 10 is more efficient than using Theorem 9 for matching to Configuration D when there is no bridged inputs between LUTs F and G. We first compute a simple disjoint decomposition under X_F and then identify a nondisjoint variable x_H in X_G. If both are successful, we compute the whole decomposition. Since computing a simple disjoint decomposition and identifying a nondisjoint variable can be performed efficiently, we save runtime for the case when $X_F \cap X_G = \emptyset$.

V. Boolean Matching for PLB2

Boolean matching for PLB2 is simpler than that for PLB1 (but PLB2 is not as powerful as PLB1 for implementing wide functions as shown in Section VI). It is easy to see that bridging x_H to LUTs F or G does not help in matching wide functions to PLB2. Such a matching can be obtained by performing Shannon expansions.

Theorem 11: PLB2 can implement $f(X)$ if and only if a Shannon expansion $f(X) = \bar{x}_H y_1(X_F) + x_H y_2(X_G)$ can be obtained for some $x_H \in X$.

Therefore, given a wide function, we enumerate every input as the MUX selection signal and check if the supports of two cofactors contain no more than $|X_F|$ and $|X_G|$ variables, respectively. Once the constraints are met, we obtain a matching for PLB2.

VI. Boolean Matching-Based Architecture Evaluation

We applied our Boolean matching approaches in two experiments. First, we employed them to map 1868 benchmark circuits provided by Xilinx, Inc. All circuits are known to be implementable in one XC4K CLB, but only up to 76% of them were mapped successfully with Xilinx internal tools or any other commercial FPGA tools [23]. Using Boolean matching techniques developed in this paper, we were able to achieve 100% single CLB implementation for this set of circuits. Second, we employed the matching approaches in the evaluation of four different PLB architectures (shown in Fig. 5) based on the percentage of wide functions (extracted from MCNC benchmarks) that can implemented with each PLB. Our quantitative approach can help design better FPGA logic blocks.

A. Boolean Matching for XC4K CLB

Since $|X_F| = |X_G| = 4$, wide functions are functions with six to nine variables for the XC4K CLB. We refine the XC4K CLB architecture into the configurations a to h shown in Fig. 4, with respect to the input sizes of six, seven, eight, and nine. For example, the configurations 6.b and 6.c are instances of Configurations A and C, respectively. Bridged inputs are explicitly shown in Fig. 4. There are as many as eight configurations for circuits with six inputs while there is only one configuration for circuits with nine inputs. Note that the configurations are not exclusive. For example, configurations 6.a, 6.b, 6.c, and 6.d are increasingly more capable in implementing six-input functions (but also require longer and longer runtime).

Refined configurations that involve multiple bridged inputs (e.g., 6.g) may combine some Configurations A, B, C, and D. Therefore, matching functions to configurations of multiple bridged inputs may require a sequence of functional decomposition operations described in Section IV. For example, configuration 6.d is a combination of Configurations A and C. In order to implement wide functions in configuration 6.d, we apply matching procedures for both Configuration C (based on Theorem 8) and Configuration A (based on Theorem 6). Given a function $f(X)$, denote the bridged inputs in configuration 6.d as x_H (to all LUTs) and x_i (to LUTs F and G only). According to Theorem 8, it requires that both cofactors $f_{\bar{x}_H}$ and f_{x_H} have (nondisjoint) bidecompositions with x_i being the nondisjoint variable. Consequently, we verify that both cofactors $f_{\bar{x}_H \bar{x}_i}$ and $f_{\bar{x}_H x_i}$ have (disjoint) bidecompositions that satisfy Theorem 6 and similarly for $f_{x_H \bar{x}_i}$ and $f_{x_H x_i}$. If the verification results are positive, then $f(X)$ can be

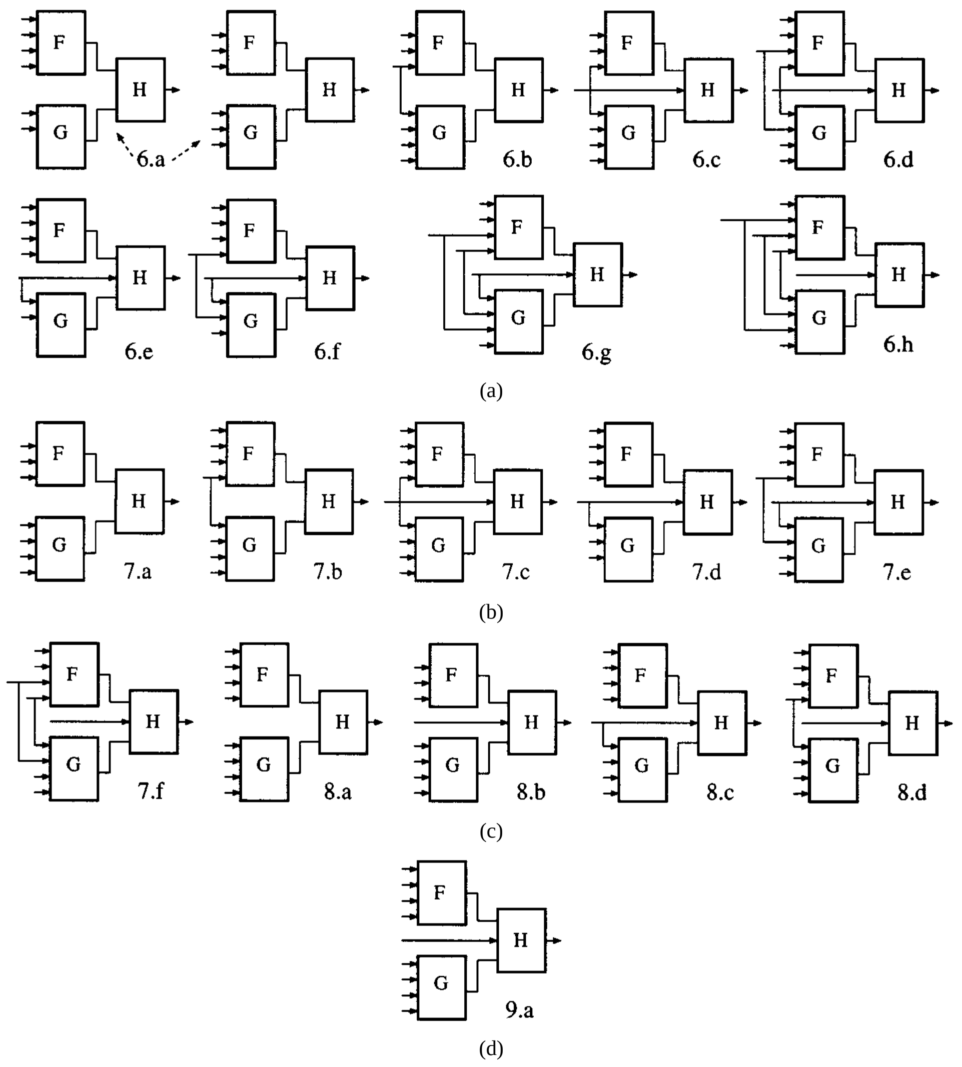

Fig. 4. Elaborated configurations of XC4K CLB. (a) Six-input configurations. (b) Seven-input configurations. (c) Eight-input configurations. (d) Nine-input configuration.

decomposed in a way that matches configuration 6.d. Clearly, matching to configuration 6.g (which involves three bridged inputs) is the most time-consuming procedure, while matching to configurations without bridged inputs (6.a, 7.a, 8.a, 8.b, and 9.a) employs only simple disjoint decompositions. It is worth noting that configurations 6.h and 8.d are instances of Configuration B to which partially dependent decomposition is employed for a match. Since bridged inputs are taken into account implicitly in partially dependent decomposition, there is no need for computing cofactors using Shannon expansions with bridged inputs.

Initially, 20 out of 1868 Xilinx benchmark circuits had five or less inputs. After applying the SIS *rugged* script, we obtained an additional 292 circuits of five or less inputs. The remaining 1576 circuits consist of 393, 371, 423, and 389 circuits of six, seven, eight, and nine inputs, respectively. Each circuit was then matched to the refined configurations in alphabetical order (i.e., 6.a before 6.b before 6.c, etc.) so that implementation of the least bridged inputs could be obtained. The results are presented in Table I. Note that 30 circuits of six inputs are matched to configuration 6.g which involves three bridged inputs.

TABLE I
NUMBER OF CIRCUITS MATCHED TO THE CONFIGURATIONS

Number of ciruits matched to each configuration

input size \ configurations	.a	.b	.c	.d	.e	.f	.g	.h
6	35	104	20	170	14	14	30	6
7	63	86	28	72	122	0	▨	▨
8	105	106	212	0	▨	▨	▨	▨
9	389	▨	▨	▨	▨	▨	▨	▨

B. PLB Architecture Evaluation

In this section, we present our evaluation on four PLB architectures which are variations in PLB1 and PLB2 families. Their diagrams are shown in Fig. 5, where (a) XC4K CLB is PLB1$(4,4,1)$; (b) PLB2$(*,4)$ could have different sizes of

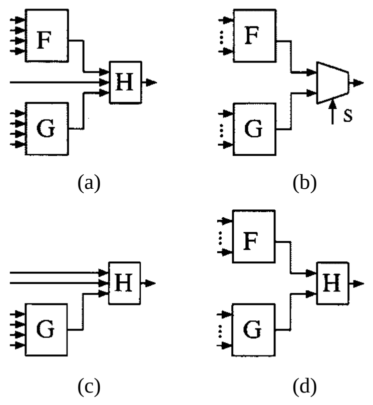

Fig. 5. Four PLB architectures. (a) XC4K CLB. (b) $\mathrm{PLB2}(*,4)$. (c) $\mathrm{PLB1}(1,4,1)$. (d) $\mathrm{PLB1}(*,4,0)$.

TABLE II
IMPLEMENTATION OF CUTS ON XC4K IN VARIOUS CONFIGURATIONS

	XC4K Configurations					all
	A-br	A	B	C	D	
6-cuts	51%	90%	98%	91%	49%	99%
7-cuts	34%	62%	88%	63%	33%	92%

LUT F but LUT G has four inputs; (c) $\mathrm{PLB1}(1,4,1)$ has a degenerated LUT F of wire connection; and (d) $\mathrm{PLB1}(*,4,0)$ does not have the x_H line and LUT G has four inputs.

We evaluate PLBs based on the number of wide functions (extracted from MCNC benchmarks) that each PLB can implement and the number of SRAM bits in LUTs. The bits in LUTs (for storing truth tables) are called *LUT bits* in the sequel. Our approach is as follows. First, we compute for each node the complete set of seven-feasible cuts [16] (where each *cut* corresponds to the inputs to a supernode at the node). The number of cuts largely depends on the size of the circuit. For example, there are 285, 422, and 734 instances of five cuts, six cuts, and seven cuts in *5xp1*, while there are 28 875, 65 245, and 157 028 instances of five cuts, six cuts, and seven cuts in *des*. Second, for each cut (i.e., supernode), we compute its function and match the function to PLBs. We say a cut can be implemented by a PLB if the corresponding function can be matched to the PLB. We report the percentage of successful implementation of cuts in Tables II and III. Finally, we divide the number of implemented cuts by the number of LUT bits for each PLB to represent PLB functional capability. In other words, we measure the efficiency of LUT bits in wide function implementation. Our measurement, of course, is only one aspect of PLB architectures. Other important factors such as required routing resources are not taken into account in the evaluation. Nevertheless, we think that it is important to see the capability of various PLBs in implementing wide functions.

We match six cuts to XC4K CLB in Configurations A, B, C, and D, and report the average percentages of matched cuts in Table II. Additionally, we report the results on Configuration A while disallowing input bridging (A-br) to see its impact on wide function implementation. Comparing the results of Configurations A versus A-br as well as the results of Configurations C versus D, we see the percentages of matches increase substantially when the inputs to LUTs F, G, and H are allowed to bridge with each other. Also, we notice that each of Configurations A, B, and C alone can implement over 90% of six cuts in the MCNC benchmarks. Overall, 99% of six cuts (all) can be implemented using the XC4K CLB.

TABLE III
IMPLEMENTATION OF CUTS ON $\mathrm{PLB2}(*,4)$, $\mathrm{PLB1}(1,4,1)$, AND $\mathrm{PLB1}(*,4,0)$

	PLB2($\|X_F\|$,4)			PLB1(1,4,1)		PLB1($\|X_F\|$,4,0)	
	(3,4)	(4,4)	(5,4)	SD	ND	(3,4,0)	(4,4,0)
5-cuts	98%	100%	100%	96%	98%	89%	97%
6-cuts	5%	8%	98%	85%	n/a	33%	89%

For the implementation of seven cuts on XC4K CLBs, it is interesting to see that Configuration B (based on partially dependent decomposition) is more capable than other configurations (based on bidecompositions). Overall, 92% of seven cuts can be implemented using the XC4K CLB.

In the evaluation of $\mathrm{PLB2}(*,4)$ architectures, we consider three configurations: $\mathrm{PLB2}(3,4)$, $\mathrm{PLB2}(4,4)$, and $\mathrm{PLB2}(5,4)$. In other words, we fixed LUT G as a 4-LUT and considered LUT F as a 3-LUT, 4-LUT, and 5-LUT, respectively. Note that it is not guaranteed that a single $\mathrm{PLB2}(3,4)$ can realize an arbitrary five-input function. However, experimental results in Table III show that $\mathrm{PLB2}(3,4)$ implements 98% of five cuts. Experiments also show that $\mathrm{PLB2}(3,4)$ and $\mathrm{PLB2}(4,4)$ implement only 5% and 8% of six cuts, respectively, while $\mathrm{PLB2}(5,4)$ implements 98% of 6-LUTs. It is worth noting that shrinking LUT F from 4-LUT to 3-LUT loses marginally in terms of functional capability but saves substantially on LUT bits (25%). Also, by expanding LUT F from 4-LUT to 5-LUT, PLB2 gains substantial capability on six-input function implementation with an additional 25% more LUT bits.

$\mathrm{PLB1}(1,4,1)$ has the least LUT bits among the four PLBs under evaluation. Although implementation of five cuts is not guaranteed, experiments show that 96% of five-cuts can be implemented by applying simple disjoint decomposition (SD) on the five-input functions. If nondisjoint decomposition (ND) is also applied, an additional 2% of five cuts can be implemented. $\mathrm{PLB1}(1,4,1)$ also implements 85% of six cuts (using SD) as well.

For $\mathrm{PLB1}(*,4,0)$, we considered two architectures $\mathrm{PLB1}(3,4,0)$ and $\mathrm{PLB1}(4,4,0)$. Although implementation of five cuts is not guaranteed, we found that $\mathrm{PLB1}(3,4,0)$ can implement most five cuts and $1/3$ of six cuts, and $\mathrm{PLB1}(4,4,0)$ can implement 97% of five cuts and 89% of six cuts.

We compared these PLBs in terms of the unit-bit implementation capability (UBIC), which is defined as the number of cuts that a PLB can implement divided by the number of LUT bits in that PLB. The comparison is presented in Table IV. Among the four evaluated PLB architectures, $\mathrm{PLB1}(1,4,1)$ has the highest UBIC for five cuts and six cuts. In addition, we notice that $\mathrm{PLB2}(3,4)$ has high UBIC for five cuts and $\mathrm{PLB1}(4,4,0)$ has

TABLE IV
FUNCTIONAL CAPABILITY PER EACH LUT BIT FOR PLBs

		#cuts per LUT bit (UBIC)		
PLBs	#bits	5-cut	6-cut	7-cut
XC4K	40	1327	2923	6389
PLB2(3,4)	24	2167	249	-
PLB2(4,4)	32	1659	298	-
PLB2(4,5)	48	1106	2436	-
PLB1(1,4,1)	24	2167	4226	-
PLB1(3,4,0)	28	1687	1364	-
PLB1(4,4,0)	36	1430	2950	-

high UBIC for six cuts. The XC4K CLB has very high UBIC for seven cuts.

VII. Application to Technology Mapping for FPGAs

In this section, we incorporate our Boolean matching techniques into technology mapping algorithms for depth minimization. Our mapping algorithms are targeted to the XC4K CLB $(\mathrm{PLB1}(4,4,1))$ and the XC5200 CLB $(\mathrm{PLB2}(4,4))$ architectures. However, they are applicable to general PLB1 and PLB2 types of FPGAs. We formulate the following problem.

Technology Mapping for XC4K/XC5200 Series FPGAs: Given a network N, compute a functionally equivalent PLB network of XC4K CLBs or XC5200 CLBs such that the depth of the PLB network is minimum.

Our approaches inherit the polynomial-time FlowMap algorithm [9] that can guarantee the minimum depth in LUT mapping solutions. Given a Boolean network N of logic gates that have no more than K fan-ins, FlowMap first computes the minimum level for each node v in all LUT mapping solutions. This level is called the *LUT label* of v, denoted as $\mathrm{lut\_label}(v)$ in this paper. After computing the labels, FlowMap generates a mapping solution based on them.

Computing node labels is the key operation in FlowMap and is briefly reviewed as follows. Every PI has a minimum level of zero. Remaining node labels are computed from PIs to POs in a topological order. Let N_v denote the subnetwork rooted at node v. A *cut* in N_v is a set of nodes that separates v from PIs. A cut is K feasible if the node cutset contains at most K nodes. The *height* of a cut is defined as the largest label for nodes in the cut. Let p be the largest label among the fan-ins of v. It was shown in [9] that $\mathrm{lut\_label}(v) = p$ if there is a K-feasible cut of height $p-1$ in N_v, which can be verified using the max-flow min-cut algorithm. If such a cut cannot be found, then $\mathrm{lut\_label}(v) = p+1$. Our mapping algorithms take similar steps to compute the minimum depth in CLB networks.

A. Technology Mapping for XC4K FPGAs

In parallel to the definition of LUT label, the minimum level of node v in any CLB network is called the *CLB label* of v, denoted $\mathrm{clb\_label}(v)$. In general, $\mathrm{clb\_label}(v) \leq \mathrm{lut\_label}(v)$. The largest CLB label in a network N is called the CLB depth $\mathrm{CLB\_depth}(N)$ of the network.

Our mapping algorithm for the XC4K CLB is called BM-Map. Before mapping, the input network is decomposed into a two-bounded network. BM-Map has three phases: initialization, labeling, and mapping. In initialization, BM-Map computes the set $C_5(v)$ of all five-feasible cuts for every node $v \in N$. (Wide cuts are generated using five-feasible cuts for runtime consideration). In the labeling phase, BM-Map computes both $\mathrm{lut\_label}(v)$ and $\mathrm{clb\_label}(v)$ using the set $C_5(v)$. Nodes are proceeded in a topological order in both procedures. Finally, BM-Map produces a XC4K CLB network in the mapping phase.

The cutset $C_5(v)$ is obtained using cut enumeration techniques in [16]. It computes K-feasible cuts in N_v by merging the cuts of the fan-ins of v and rejecting those cuts that are not K feasible. In theory, the number of K-feasible cuts grows exponentially with respect to K. However, for $K \leq 5$, this computation is efficient in practice. For most benchmarks, we see about 30–70 five-feasible cuts per node.

Let p be the largest LUT label among the fan-ins of v. Instead of using max-flow min-cut procedures, BM-Map determines if $\mathrm{lut\_label}(v) = p$ by looking for a cut of height $p-1$ in the cutset $C_5(v)$. To compute the $\mathrm{clb\_label}(v)$, let q be the largest CLB label among the fan-ins of v. BM-Map performs the following two checks.

(C1) If there exists a K-feasible cut of height $q-1$ in $C_5(v)$, then $\mathrm{clb\_label}(v) = q$.

(C2) Otherwise, if there exists a nine-feasible cut of height $q-1$ of which the corresponding wide function can be successfully matched to the XC4K CLB, then $\mathrm{clb\_label}(v) = q$.

If both checks fail, then $\mathrm{clb\_label}(v) = q+1$.

We take two actions to save runtime in label computation. First, we do not exhaust all XC4K configurations in Boolean matching. We only test configurations (in Fig. 4) 6.a and 6.c for six cuts, 7.c and 7.f for seven cuts, 8.c for eight cuts, and 9.a for nine cuts because they have high matching percentages (Table II) with relatively low decomposition complexity. Second, to obtain wide cuts (of six to nine nodes) at each node v, we simply merge the cuts in $C_5(u_1)$ and $C_5(u_2)$, where u_1 and u_2 are fan-ins of v. By doing so, we trade the completeness of nine-feasible cuts for runtime.

After every node in the network has been labeled, BM-Map generates a CLB mapping solution that respects labels. A PO node v is a *critical* PO node if $\mathrm{LUT\_label}(v) \geq \mathrm{CLB\_depth}(N)$. For those critical POs and their fan-in networks, BM-Map covers them with CLBs to guarantee the CLB depth computed in the labeling phase. For the remaining noncritical POs and their fan-in networks, BM-Map covers them with LUTs to save area. At last, BM-Map packs LUTs into CLBs using an efficient procedure *match_4k* proposed in [15]. The BM-Map algorithm is outlined in Fig. 6.

B. Technology Mapping for XC5200 FPGAs

While the XC4K CLB can implement a large number of six cuts and seven cuts, it was shown in previous sections that the

```
function BM-Map ($N$)
/* $N$ = input network */

Compute $C_5(v)$ for every $v \in N$
Compute lut label($v$) for every $v \in N$
Compute clb label($v$) for every $v \in N$ using (C1) and (C2)
    in Section 7.1 (matched wide cuts saved)
CLB depth($N$) = max{ CLB label($v$) | $v \in$ N}

for each critical PO $v$ with LUT label($v$) $\geq$ CLB depth($N$) do
    Get the cut of height clb label($v$) $-$ 1
    Map $v$ to a CLB with inputs from the cut
    Map every $u$ in the cut recursively using CLBs

for each remaining non-critical PO $v$ do
    Select a cut from $C_5(v)$
    Map $v$ to an LUT with inputs from the cut
    Map every $u$ in the cut recursively using LUTs

Pack LUTs into CLBs using match 4k
```

Fig. 6. BM-Map algorithm.

XC5200 CLB implements only a very small percentage of them. However, we can exploit the XC5200 CLB architecture based on an interesting result observed in our experiments.

Let us first introduce two concepts. A wide function $f(X)$ is *fully implementable* on a XC5200 CLB if each cofactor f_{x_i} and $f_{\bar{x}_i}$ depends on at most four variables for some $x_i \in X$ (Theorem 11), while $f(X)$ is *partially implementable* if only one of the two cofactors satisfies the condition. In Table V, the columns titled "fit" and "1/2-fit" show the number of cuts that are fully and partially implementable on XC5200 CLBs, respectively. Although very few six cuts and seven cuts are fully implementable, most of them are partially implementable. Based on this observation, we consider using XC5200 CLBs for partially implementable functions. [Note that most wide functions are fully implementable on XC4K CLBs (see Table II). As a result, there is little benefit to consider partially implementable functions.]

Our mapping algorithm for XC5200 CLBs, called BMD-Map, enhances BM-Map by incorporating functional decomposition (D) operations into label computation. In initialization, BMD-Map computes the cutset $C_7(v)$ of all seven-feasible cuts for every node $v \in N$. (We can afford testing more wide cuts since matching to XC5200 CLB is much less time consuming.) In the labeling phase, BMD-Map determines if $\text{clb\_label}(v) = q$ using the following three checks.

(C1) If there exists a five-feasible cut of height $q-1$ in $C_7(v)$, then $\text{clb\_label}(v) = q$.

(C2) Otherwise, if there exists a wide cut of height $q-1$ in $C_7(v)$ of which the corresponding wide function is full implementable, then $\text{clb\_label}(v) = q$.

(C3) Otherwise, if there exists a wide cut of height $q-1$ in $C_7(v)$ of which the corresponding wide function is partially implementable and the infeasible cofactor can be decomposed to obtain a new cut of height $q-1$ (explained in the next paragraph), then $\text{clb\_label}(v) = q$. We refer to the decomposition of the infeasible cofactor as a *decomposition operation*.

Otherwise, $\text{clb\_label}(v) = q+1$.

TABLE V
MATCHING WIDE FUNCTIONS TO XC5200 CLBs

Circuits	6-cuts	fit	1/2-fit	7-cuts	fit	1/2-fit
5xp1	422	186	233	734	159	563
9sym	1256	235	1020	2333	143	2173
9symml	1156	177	978	2195	84	2080
C499	9716	370	8826	27599	636	22601
C880	3969	210	3727	9079	75	8858
alu2	6666	573	6026	18231	482	17494
alu4	14841	1140	13376	40332	604	38845
apex6	2691	253	2437	4370	152	4216
apex7	743	76	657	1342	39	1283
count	94	15	79	87	0	87
des	65245	5177	59192	157028	2903	147891
duke2	4606	180	4426	10106	151	9955
misex1	266	57	209	400	19	381
rd84	2351	209	2142	5307	50	5198
rot	4857	335	4410	10362	177	9980
vg2	410	32	378	850	29	806
z4ml	36	0	35	36	0	34
total	119325	9225	108151	290391	5703	272445
ratio	1	8%	91%	1	2%	94%

We illustrate the decomposition operation (C3) in Fig. 7. Let $X = \{x_1, \ldots, x_6\}$ be a six cut of v and all nodes in X have a label $q-1$. Let $f(X)$ denote the function of the subnetwork rooted at v with inputs from X [see Fig. 7(a)]. Let $f(X) = \bar{x}_1 F(X_F) + x_1 I(X_I)$ and assume that $|X_I| = 5$. Then, $f(X)$ is partially implementable on XC5200 CLB [see Fig. 7(b)]. The dash line represents possible bridged inputs to LUTs. Decomposition operations are performed iteratively in three steps.

1) Choose a two-variable bound set $B \subset X_I$.
2) Perform a simple disjoint decomposition $I(X_I) = G(x_7(B), X_I - B)$ under the bound set B.
3) If the decomposition is successful, create a node u for $x_7(B)$ [see Fig. 7(c)] and compute a min-cut of height $q-2$ in the cone N_u rooted at u. If the min-cut is K feasible, then $l(u) = q-1$ and, therefore, $l(v) = q$ since all the inputs of F, G, and H have a label $q-1$.

Steps 1 to 3 are iterated for all possible bound sets until a success is found or bound sets are exhausted. In the latter case, we assign $l(v) = q+1$.

VIII. Technology Mapping Experimental Results

We conducted experiments on a Sun ULTRA2 workstation with 256 MB of memory. The tested circuits are MCNC benchmarks which were optimized using the SIS *rugged* script [32] and decomposed into two-input networks using the *dmig* algorithm [7], [35]. The mapping goal was to minimize the CLB depth with consideration to area minimization.

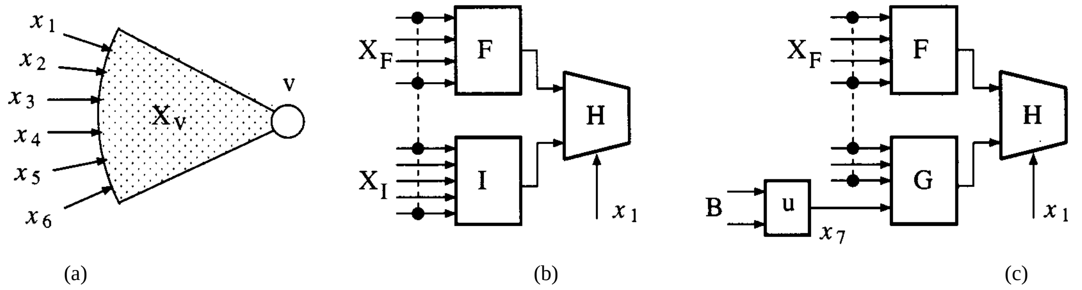

Fig. 7. Decomposition operation in step (S3) for XC5200 CLB. (a) Six-input supernode. (b) Partial implementation on XC5200 CLB. (c) Implementation after decomposition operation.

TABLE VI
BM-Map Mapping Results for XC4K CLBs

Circuits	FlowMap			BM-Map(10)			BM-Map(50)		
	D	CLB	T(s)	D	CLB	T(s)	D	CLB	T(s)
5xp1	3	17	0.5	2	19	4.1	2	19	5.1
9sym	5	35	1.1	4	39	11.9	4	39	20.0
9symml	5	35	1.1	4	38	15.2	4	39	23.6
C499	4	34	4.7	4	63	21.6	4	63	34.2
C880	8	65	3.0	7	89	15.8	7	91	89.7
alu2	9	90	3.8	7	108	53.9	6	96	97.8
alu4	9	144	7.4	8	204	95.2	8	199	181.6
apex6	5	155	4.9	4	184	31.3	4	186	34.6
apex7	4	40	1.4	3	46	18.4	3	46	15.4
count	5	19	0.7	4	24	2.7	4	24	2.6
des	5	672	32.9	5	978	746.4	4	764	1180.8
duke2	4	93	2.8	4	118	21.7	4	118	47.3
misex1	2	10	0.3	2	11	1.6	2	11	2.1
rd84	4	34	1.2	3	44	7.3	3	45	14.7
rot	7	130	5.0	6	156	29.5	6	159	44.4
vg2	3	15	0.4	3	20	14.2	2	23	20.1
z4ml	2	3	0.2	2	3	0.5	2	3	0.6
total	84	1591	71.4	72	2144	1091.3	69	1925	1814.6
arith.mean	1.00	1.00	1.00	0.86	1.35	15.28	0.82	1.21	25.41
geo.mean	1.00	1.00	1.00	0.86	1.25	8.41	0.82	1.24	13.18

In the first experiment, we applied FlowMap and BM-Map to map MCNC benchmarks into XC4K CLBs. After technology mapping, *match_4k* [15] was employed to pack LUTs into XC4K CLBs. We limited the maximal number of wide cuts to ten and 50 in BM-Map(10) and BM-Map(50) and reported corresponding results in Table VI. Compared to FlowMap, BM-Map obtained 14% and 18% smaller depth when ten and 50 wide cuts were tested, respectively. However, BM-Map uses substantially more CLBs compared to FlowMap. After careful examination, we found that a large percentage of 5-LUTs mapped by FlowMap were decomposed by *match_4k* into 2-LUTs and 4-LUTs and subsequently packed into CLBs with other 4-LUTs. This is very efficient for area minimization. This benefit does not happen to BM-Map because it uses LUTs only to cover noncritical portions of the input network and obtains much less 5-LUTs. In general, the more cuts that are tested for matching in BM-Map, the better the mapping results will be, but the longer the runtime. The ratio, however, is biased significantly by the circuit *des*.

In the second experiment, we applied our mapping approach to the technology mapping for XC5200 FPGAs. After mapping, LUTs were packed into XC5200 CLBs. One CLB is allocated for one 5-LUT as well as for one pair of 4-LUTs. We compared BMD-Map with the CutMap algorithm [12]. CutMap also inherits the FlowMap algorithm, but in addition performs simultaneous area minimization. It obtains the same LUT depth as obtained by FlowMap, but uses 18% less 5-LUTs on average for industrial benchmarks [22]. The mapping results are reported in Table VII. Compared to CutMap, BM-Map obtains 7% smaller depth with 15% smaller area, and BMD-Map obtains 12% smaller depth with 6% larger area. The runtime of BM-Map and BMD-Map are both one order of magnitude

TABLE VII
BMD-MAP MAPPING RESULTS FOR XC5200 CLBs

	CutMap			BM-Map			BMD-Map		
Circuits	D	CLB	T(s)	D	CLB	T(s)	D	CLB	T(s)
5xp1	3	18	0	3	19	3	3	19	2
9sym	5	48	1	4	43	10	4	46	9
9symml	5	46	1	4	45	10	4	48	8
C499	4	53	7	4	50	323	4	50	272
C880	8	101	3	8	89	45	8	89	43
alu2	9	118	4	8	106	148	7	124	132
alu4	9	194	11	9	184	374	8	206	327
apex6	5	186	7	4	165	22	4	175	19
apex7	4	56	1	3	52	6	3	57	6
count	5	28	0	5	23	1	4	28	1
des	5	925	116	5	714	1750	5	1040	1588
duke2	4	131	4	4	121	70	4	129	66
misex1	2	12	0	2	9	2	2	9	1
rd84	4	40	2	4	37	45	4	40	37
rot	7	166	6	6	154	62	5	188	56
vg2	3	19	0	3	18	4	3	18	5
z4ml	2	4	0	2	4	0	2	4	0
total	84	2145	163	78	1833	2875	74	2270	2572
arith.mean	1.00	1.00	1.00	0.93	0.85	17.64	0.88	1.06	15.78
geo.mean	1.00	1.00		0.93	0.91		0.90	0.99	

longer than CutMap. Comparing BM-Map with BMD-Map, we can see the impact of functional decomposition in XC5200 mapping.

IX. CONCLUSION

We have presented new Boolean matching methods for LUT-based PLBs and their applications to architecture evaluation and FPGA technology mapping. Our Boolean matching methods employ functional decomposition operations to represent functions in forms corresponding to the target PLB architecture. Existence conditions for new functional decomposition forms are given and proved. We applied the methods to the evaluation of PLB architectures in terms of logic implementation capability. Experimental results show that the Xilinx XC4K CLB can implement 98% and 88% of six- and seven-variable functions extracted from MCNC benchmarks, respectively, while a simplified PLB architecture implements the largest amount of functions per LUT bit. We developed new technology mapping algorithms that employ the Boolean matching techniques for depth minimization. Compared to conventional LUT mapping approaches, experimental results show 18% and 12% depth reduction on average for the Xilinx XC4K series and XC5200 series FPGAs, respectively, with up to 15% area reduction in XC5200 FPGAs. Our Boolean matching techniques can be useful for designing future FPGA architectures and better utilization of FPGA silicon resources.

ACKNOWLEDGMENT

The authors would like to thank Dr. S. Kelem from Xilinx for providing the test suite used in Section VI-A.

REFERENCES

[1] *Actel, Data Sheet: 3200DX FPGAs*, 1995.

[2] R. L. Ashenhurst, "The decomposition of switching functions," in *Proc. Int. Symp. Theory of Switching Functions*, Apr. 1957, pp. 74–116.

[3] L. Benini and G. D. Micheli, "A survey of Boolean matching techniques for library binding," *ACM Trans. Design Automat. Electron. Syst.*, vol. 2, pp. 193–226, July 1997.

[4] T. Besson, H. Bouzouzou, M. Crastes, I. Floricica, and G. Saucier, "Synthesis on multiplexer-based F.P.G.A. using binary decision diagrams," in *Proc. Int. Conf. Computer Design*, Oct. 1992, pp. 163–167.

[5] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.

[6] S.-C. Chang and M. Marek-Sadowska, "Technology mapping via transformations of function graphs," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1992, pp. 159–162.

[7] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-map: Graph-based FPGA technology mapping for delay optimization," *IEEE Des. Test Comput.*, pp. 7–20, Sept. 1992.

[8] A. Chowdhary and J. Hayes, "General modeling and technology mapping technique for LUT-based FPGAs," in *Proc. 5th ACM/SIGDA Int. Symp. FPGAs*, Feb. 1997, pp. 43–49.

[9] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1–12, Jan. 1994.

[10] ——, "On area/depth trade-off in LUT-based FPGA technology mapping," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 137–148, June 1994.

[11] ——, "Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 1, no. 2, pp. 145–204, 1996.

[12] J. Cong and Y.-Y. Hwang, "Simultaneous depth and area minimization in LUT-based FPGA mapping," in *Proc. ACM 3rd Int. Symp. FPGA*, Feb. 1995, pp. 68–74.

[13] ——, "Partially dependent functional decomposition with applications in FPGA synthesis and mapping," in *Proc. ACM 5th Int. Symp. FPGA*, Feb. 1997, pp. 35–42.

[14] ——, "Boolean matching for complex PLBs in LUT-based FPGAs with application to architecture evaluation," in *Proc. ACM 6th Int. Symp. FPGA*, Feb. 1998, pp. 27–34.

[15] J. Cong, J. Peck, and Y. Ding, "RASP: A general logic synthesis system for SRAM-based FPGAs," in *Proc. ACM 4th Int. Symp. FPGA*, Feb. 1996, pp. 137–143.

[16] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proc. 7th ACM/SIGDA Int. Symp. FPGAs*, Feb. 1999, pp. 29–35.

[17] H. A. Curtis, "A generalized tree circuit," *J. ACM*, vol. 8, no. 4, pp. 484–496, 1961.

[18] A. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for FPGA technology mapping," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1319–1332, Nov. 1994.

[19] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," in *Proc. 28th ACM/IEEE Design Automation Conf.*, June 1991, pp. 613–619.

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

[21] ——, "An iterative area/performance trade-off algorithm for LUT-based FPGA technology mapping," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 13–17.

[22] Y.-Y. Hwang, "Logic synthesis for lookup-table-based field programmable gate arrays," Ph.D. dissertation, Univ. California, Los Angeles, CA, 1999.

[23] S. Kelem, FPGA Designs, Comp. Sci. Dept., Univ. California, Los Angeles, CA. Presentation.

[24] Y.-T. Lai, K.-R. R. Pan, and M. Pedram, "FPGA synthesis using function decomposition," in *Proc. Int. Conf. Computer Design: VLSI in Computers*, Oct. 1994, pp. 30–35.

[25] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD-based decomposition of logic functions with application to FPGA synthesis," in *Proc. 30th ACM/IEEE Design Automation Conf.*, June 1993, pp. 642–647.

[26] C. Legl, B. Wurth, and K. Eckl, "An implicit algorithm for support minimization during functional decomposition," in *Proc. Eur. Design and Test Conf.*, Mar. 1996, pp. 412–417.

[27] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 564–567.

[28] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM J. Res. Devel.*, pp. 227–238, Apr. 1962.

[29] T. Sasao and J. T. Butler, "On bidecompositions of logic functions," in *Proc. Int. Workshop Logic Synthesis*, June 1997, pp. 1–6.

[30] P. Sawkar and D. Thomas, "Area and delay mapping for table-look-up based field programmable gate arrays," in *Proc. 29th ACM/IEEE Design Automation Conf.*, June 1992, pp. 368–373.

[31] M. Schlag, J. Kong, and P. K. Chan, "Routability-driven technology mapping for lookup table-based FPGAs," in *Proc. 1992 IEEE Int. Conf. Computer Design*, Oct. 1992, pp. 86–90.

[32] 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," Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M92/41, May 1992.

[33] *ORCA OR2C-A/OR2T-A Series FPGAs Data Sheet*, Lucent Technologies, Inc., Allentown, PA, 1996.

[34] S. Thakur and D. Wong, "On designing ULM-based FPGA logic modules," in *Proc. ACM 3rd Int. Symp. FPGA*, Feb. 1995, pp. 3–9.

[35] A. Wang, "Algorithms for Multi-Level Logic Optimization," Univ. California, Berkeley, CA, Memo. UCB/ERL M89/50, Apr. 1989.

[36] B. Wurth, K. Eckl, and K. Antreich, "Functional multiple-output decomposition: Theory and an implicit algorithm," in *Proc. ACM/IEEE Design Automation Conf.*, June 1995, pp. 54–59.

[37] B. Wurth, U. Schlichtmann, K. Eckl, and K. J. Antreich, "Functional multiple-output decomposition with application to technology mapping for lookup table based FPGAs," *ACM Trans. Design Automat. Electron. Syst.*, pp. 313–350, July 1999.

[38] *The Programmable Gate Array Data Book*, Xilinx, San Jose, CA, 1997.

[39] H. Yang and D. Wong, "Edge-map: Optimal performance driven technology mapping for iterative LUT based FPGA designs," in *Proc. 1994 IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 150–155.

[40] K. Zhu and D. F. Wong, "Fast Boolean matching for field-programmable gate arrays," in *Proc. Eur. Design Automation Conf.*, Mar. 1993, pp. 352–357.

Jason (Jingsheng) Cong (S'88–M'90–SM'96–F'01) received the B.S. degree in computer science from the Peking University, Beijing, China, in 1985, and received the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana-Champaign, in 1987 and 1990, respectively.

He is currently a Professor in the Computer Science Department and Co-Director of the VLSI CAD Laboratory at University of California, Los Angeles. His research interests include computer-aided design of VLSI circuits and systems, rapid prototyping of computer and communication systems, design and analysis of algorithms, and computer architecture.

Dr. Cong received the Best Graduate Award from Peking University in 1985, the Ross Martin Award for Excellence in Research from the University of Illinois in 1989, the National Science Foundation Research Initiation Award and Young Investigator Award in 1991 and 1993, respectively, the Northrop Corporation Outstanding Junior Faculty Research Award from University of California, Los Angeles, in 1993, the ACM SIGDA Meritorious Service Award in 1998, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS Best Paper Award in 1995. He is an Associate Editor of *ACM Transactions on Design Automation of Electronic Systems*.

Yean-Yow Hwang received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in and the M.S. and Ph.D. degrees in computer science from the University of California, Riverside and Los Angeles, in 1987 and 1999, respectively.

He was a Research Engineer at Synopsys, Inc. and is currently a Technical Staff Engineer at the Altera Corporation, San Jose, CA. His research interests include algorithms, logic synthesis, post-layout optimization, VLSI design tools, and FPGA architectures.