Boolean Matching for Incompletely Specified Functions

Kuo-Hua Wang
Department of Computer Science and Information Engineering,
National Chiao Tung University, HsinChu, Taiwan 30050

TingTing Hwang
Department of Computer Science,
National Tsing Hua University, HsinChu, Taiwan 30043

Abstract

Boolean matching is to check the equivalence of two functions under input permutation and input/output phase assignment. In this paper, we will address Boolean matching problem for incompletely specified functions. We will formulate the searching of input variable mapping between two target functions as a logic equation by using multiple-valued function. Based on this equation, a Boolean matching algorithm will be proposed. Delay and power dissipation can also be taken into consideration when this method is used for technology mapping. Experimental results on a set of benchmarks show that our algorithm is indeed very effective in solving Boolean matching problem for incompletely specified functions.

1 Introduction

Boolean matching is to check the equivalence of two functions under input permutation and input/output phase assignment (so called *NPN-class*). It has been widely used in technology mapping recently [1]-[7]. Applying Boolean matching in technology mapping can improve the quality of mapped circuits and increase the mapping flexibility since it exploits implicit don't cares which was not considered in traditional tree covering algorithm. Moreover, it is able to shorten the mapping time when using a library containing complex gates with large input size. Boolean matching is also applied in logic verification, e.g., checking the equivalence of two circuits, and verifying the implementation of a specification.

Various methods for Boolean matching were proposed [1]-[11]. Mailhot et al. [1] are among the first ones to apply Boolean matching to technology mapping. They proposed an algorithm using tautology checking based on shannon decompositions. Don't cares were tackled by a lattice-based method.

Savoj et al. [2] used smoothing and consensus operators to solve Boolean matching problem. It also considered the use of don't cares in Boolean matching. Boolean unification and branch-and-bound techniques were adopted in [3]. The matching between two functions was checked by finding the *most general unifier*. Don't cares was also considered by this method. However, these methods have the disadvantage of inefficiency in terms of computation time.

32nd ACM/IEEE Design Automation Conference ®

The techniques presented in [5] were based on computing *canonical forms* of functions. In [11], Boolean matching was done by using canonical Generalized Reed-Muller forms of completely specified functions. Yet, another group of researchers take "signature" approach to solve Boolean matching. Various signatures [4, 7, 9, 10] were defined to characterize the input variables of Boolean functions, where variables with different signatures can be distinguished from each other and many impossible permutations can be pruned. The structure of Ordered Binary Decision Diagrams (OBDD's) was also utilized for Boolean matching [6, 8].These algorithms, although very efficient, failed to handle incompletely specified functions in Boolean matching.

In this paper, we propose a Boolean matching method which is not only efficient but also able to handle incompletely specified functions. We will use multiple-valued functions to represent input-variable mapping and formulate the matching process by a Boolean equation. Our method can quickly reduce the searching space and find all candidate mappings. Moreover, area, delay, and power dissipation can be easily coped with in our algorithm.

2 Boolean Matching with Don't Cares

An incompletely specified function f is a Boolean function with don't cares. It involves three sets : the on-set (f^{on}), the off-set (f^{off}) and the don't-care set (f^{dc}). In this paper, we will denote an incompletely specified function as $f = (f^{on}, f^{off})$.

With respect to the same set of inputs, two incompletely specified Boolean functions are equivalent if the following consistency condition holds.

Definition 2.1 (consistency) Two functions $f(X)$ and $g(X)$ are *consistent* if and only if $f^{on} \cap g^{off} = \emptyset$ and $f^{off} \cap g^{on} = \emptyset$. It is denoted as $f \cong g$.

Boolean matching is to check the equivalence of two functions under input permutation and input/output phase assignment. Hence, given two incompletely specified functions $f(X)$ and $g(Y)$, where $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$, Boolean matching problem for incompletely specified functions is to find a mapping (also assignment) ψ which maps x_i to a unique $y_j(\bar{y_j})$ such that $f' \cong g$, where $f' = f(\psi(X))$ (or $\bar{f}(\psi(X))$).

We will use the following example to illustrate Boolean matching for incompletely specified functions.

Example 2.1 Given two functions $f(x_1, x_2, x_3)$, and $g(y_1, y_2, y_3)$, where $f^{on} = \bar{x_1}\bar{x_2} + \bar{x_1}x_3$, $f^{off} = x_1x_3$, $g^{on} = y_1\bar{y_3}$, and $g^{off} = \bar{y_1}y_2$. Let the assignment ψ be a mapping of x_1, x_2, and x_3 to y_2, y_3, and $\bar{y_1}$, respectively. Then, $f' = f(\psi(X)) = (f'^{on} = \bar{y_2}\bar{y_3} + \bar{y_2}\bar{y_1}, f'^{off} = y_2\bar{y_1})$. The Karnaugh maps of $f, f' = f(\psi(X))$ and g are

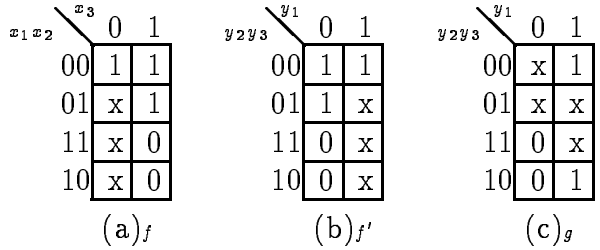

Figure 1: The Karnaugh Maps of f, f' and g.

shown in Figure 1(a), (b) and (c), respectively, where x indicates a don't care minterm. From Figure 1, we find that $f' \cong g$. Hence, f and g are matched under the assignment ψ.

Obviously, a exhaustive search is not feasible because it takes $2^n \times n! \times 2$ permutations, where n is the number of inputs. Instead, we will propose a method which formulates the candidate assignments as a logic function by multiple-valued logic.

For ease of explanation, only input permutation will be discussed in the following. The result extented to input/output phase assignment will be presented in Section 3.3. Before we derive rules for matching two incompletely specified functions, we have the following definition and lemma.

Definition 2.2 Let u be a binary-valued cube. Then, *1-lit*(u) (*0-lit*(u)) is a subset of support u, where only literals of u in uncomplemented (complemented) form are in the subset.

Example 2.2 Given the input set $X = \{x_1, x_2, x_3, x_4\}$ and a cube $u = x_1\bar{x_3}x_4$. Then, *1-lit*$(u) = \{x_1, x_4\}$ and *0-lit*$(u) = \{x_3\}$.

Lemma 2.1 Let u_1 and u_2 be two binary-value cubes with respect to the same input set. Then $u_1 \cdot u_2 = \emptyset$ if and only if *1-lit*$(u_1) \cap$ *0-lit*$(u_2) \neq \emptyset$ or *0-lit*$(u_1) \cap$ *1-lit*$(u_2) \neq \emptyset$.

< *proof* :> It follows directly from the definition of AND $(\cdot)$ operation.

Boolean matching for two incompletely specified functions, $f(X)$ and $g(Y)$, is to find a mapping from X to Y so that the consistency condition holds. From Definition 2.1, the consistency of two functions f and g must satisfy $f^{on} \cap g^{off} = \emptyset$ and $f^{off} \cap g^{on} = \emptyset$. Consider condition $f^{on} \cap g^{off} = \emptyset$ first. It requires that for every pair of cubes $u_i \in f^{on}$ and $v_j \in g^{off}$, $u_i \cdot v_j = \emptyset$ should hold. That is, for any two cubes $u_i \in f^{on}$ and $v_j \in g^{off}$, in order to satisfy the consistency condition, it is only possible by mapping at least one variable of u_i in complemented form to one variable of v_j in uncomplemented form, or one variable of u_i in uncomplemented form to one variable of v_j in complemented form. For each pair of cubes, there are usually many possible partial mappings which allow the intersection of the cubes to be empty. To ensure the consistency condition, for each pair of cubes, the final mapping result must satisfy at least one of the partial mapping. The same argument is also true for the cubes in f^{off} and g^{on} to ensure condition $f^{off} \cap g^{on} = \emptyset$.

We will use the following example to illustrate that in order to satisfy the consistency condition, partial assignments are derived from pairs of cubes.

Example 2.3 Given two functions $f(x_1, x_2, x_3)$, and $g(y_1, y_2, y_3)$, where $f^{on} = \bar{x_1}\bar{x_2} + \bar{x_1}x_3$, $f^{off} = x_1x_3$, $g^{on} = y_1\bar{y_3}$, and $g^{off} = \bar{y_1}y_2$. For the pair of cubes $\bar{x_1}\bar{x_2} \in f^{on}$ and $\bar{y_1}y_2 \in g^{off}$, we have the following two partial assignments :

1. x_1 mapped to y_2, or
2. x_2 mapped to y_2.

For the pair of cubes $\bar{x_1}x_3 \in f^{on}$ and $\bar{y_1}y_2 \in g^{off}$, we have the following two partial assignments :

3. x_1 mapped to y_2, or
4. x_3 mapped to y_1.

For the pair of cubes $x_1x_3 \in f^{off}$ and $y_1\bar{y_3} \in g^{on}$, we have the following two partial assignments :

5. x_1 mapped to y_3, or
6. x_3 mapped to y_3.

To satisfy the consistency condition, at least one partial mapping derived from each pair of cubes must be satisfied in the final mapping results. That is, the final matching must satisfy the partial mappings (1 or 2) and (3 or 4) and (5 or 6).

From the above-mentioned description, we now develop rules for partial assignments for each pair of cubes $p = (u_i, v_j)$, where $u_i \in f^{on}$ (f^{off}) and $v_j \in g^{off}$ (g^{on}) as follows:

Rule 1: For each $x_i \in$ *1-lit*(u_i), derive partial mappings x_i to every $y_j \in$ *0-lit*(v_j). For each $x_i \in$ *0-lit*(u_i), derive partial mappings x_i to every $y_j \in$ *1-lit*(v_j).

Rule 2: At least one partial mapping derived from each pair of cubes must be satisfied in the final result.

3 Boolean Matching Using Multiple-valued Function

In this section, we first review multiple-valued Boolean function and then show how to model assignment as multiple-valued function. Logic operations (AND and OR) will be used to search matching solution. To speed up the searching, some implementation issues will also be addressed in this section.

3.1 Representing Assignments by Multiple-Valued Functions

A **multiple-valued input, binary-valued output function** f (hereafter known as the **multiple-valued function**) is a mapping

$$f : P_1 \times P_2 \times \cdots \times P_n \rightarrow B$$

where $P_i = \{1, 2, \cdots, p_i\}$ represents p_i values that variable i may assume and $B = \{0, 1\}$ the output value of the function.

Let x_i be a variable taking a value from P_i, and S_i be a subset of P_i. ${x_i}^{S_i}$ represents the Boolean function

$${x_i}^{S_i} = \begin{cases} 0 & \text{if } x_i \notin S_i \\ 1 & \text{if } x_i \in S_i. \end{cases}$$

${x_i}^{S_i}$ is called a literal of variable x_i. For the example of $P_1 = \{1, 2, 3\}$, and $x_1 = 2$, we will have ${x_1}^{\{1,2\}} = 1$ and ${x_1}^{\{1,3\}} = 0$.

Given two Boolean functions $f(X)$ and $g(Y)$, where input set $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$. Now we show how to use multiple-valued function to represent assignment. Define a multiple-valued function F as

$$F : P_1 \times P_2 \times \cdots \times P_n \rightarrow B, \qquad (1)$$

where $P_1 = P_2 = \cdots = P_n = \{1, 2, \cdots, n\}$ and $B = \{0, 1\}$. Variable i of F corresponds to the input x_i of $f(X)$ and P_i corresponds to the set of y_i's which x_i may map to. A variable i assuming value j means that x_i is mapped to y_j. A minterm of F is evaluated true if it corresponds to an assignment. For example,

Example 3.1 Let $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$. Then a multiple-valued function F,

$$F : N \times N \times N \rightarrow B,$$

is defined, where $N = \{1, 2, 3\}$ and $B = \{0, 1\}$. If x_1 mapped to y_1, x_2 mapped to y_3 and x_3 mapped to y_2 is an assignment, then the minterm (1, 3, 2) of F is evaluated true.

A minterm is *feasible* if it may represent an assignment; otherwise, it is *infeasible*. For the same example shown in Example 3.2, the minterm (1, 1, 2) is infeasible since both x_1 and x_2 are mapped to y_1.

We define a multiple-valued function a **totality** if all feasible minterms are evaluated true.

3.2 Computing Assignments Using Multiple-valued Function

A solution to Boolean matching is searched mainly based on consistency checking. We have developed two rules in Section 2 to define the search space. In this section, we will show how to represent **Rule 1** using multiple-valued function. Then, the logic operations (AND and OR) on multiple-valued function is used to ensure **Rule 2**.

First, based on **Rule 1**, for a pair of binary-valued cubes u and v, we define $MvCube(u, v)$.

Definition 3.1 For two Boolean functions $f(X)$ and $g(Y)$, given two cubes u of f in on-set (off-set) and v of g in off-set (on-set) with respect to input sets $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$, respectively,

$$MvCube(u, v) = \sum_{x_i \in \textit{1-lit}(u)} x_i^{S_a} + \sum_{x_i \in \textit{0-lit}(u)} x_i^{S_b}, \quad (2)$$

where $S_a = \{j \mid y_j \in \textit{0-lit}(v)\}$ and $S_b = \{j \mid y_j \in \textit{1-lit}(v)\}$.

$MvCube(u, v)$ gives all partial assignments ψ which satisfy $u(\psi(X)) \cdot v = \emptyset$.

We use the following example to show how $MvCube$'s are computed using **Rule 1**.

Example 3.2 Given two incompletely specified functions $f(x_1, x_2, x_3)$ and $g(y_1, y_2, y_3)$. Let $f^{on} = \bar{x_1}\bar{x_2} + x_1x_2x_3$, $f^{off} = x_2\bar{x_3} + x_1\bar{x_3}$, $g^{on} = \bar{y_1}y_2$, and $g^{off} = \bar{y_3}y_1 + y_3\bar{y_1}\bar{y_2}$. Then $MvCube(u, v)$'s for all pairs of cubes from f^{on} and g^{off} are as follows:

$1 : MvCube(\bar{x_1}\bar{x_2}, \bar{y_3}y_1) = x_1^{\{1\}} + x_2^{\{1\}}$
$2 : MvCube(\bar{x_1}\bar{x_2}, y_3\bar{y_1}\bar{y_2}) = x_1^{\{3\}} + x_2^{\{3\}}$
$3 : MvCube(x_1x_2x_3, \bar{y_3}y_1) = x_1^{\{3\}} + x_2^{\{3\}} + x_3^{\{3\}}$ and
$4 : MvCube(x_1x_2x_3, y_3\bar{y_1}\bar{y_2}) = x_1^{\{1,2\}} + x_2^{\{1,2\}} + x_3^{\{1,2\}}$.

And, $MvCube(u, v)$'s for all pairs of cubes from f^{off} and g^{on} are

$5 : MvCube(x_2\bar{x_3}, \bar{y_1}y_2) = x_2^{\{1\}} + x_3^{\{2\}}$ and
$6 : MvCube(x_1\bar{x_3}, \bar{y_1}y_2) = x_1^{\{1\}} + x_3^{\{2\}}$.

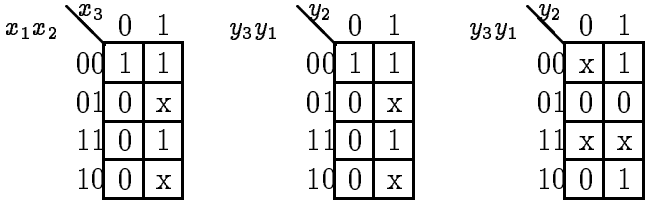

Figure 2: The Karnaugh Maps of f, f' and g.

For two incompletely specified functions $f = (f^{on}, f^{off})$ and $g = (g^{on}, g^{off})$, to ensure consistency condition, **Rule 2** says that the final mapping must comply with at least one partial assignment developed for each pair of cubes from f^{on} (f^{off}) and g^{off} (g^{on}). Based on the definition of $MvCube$ and **Rule 2**, we have the following theorem.

Theorem 3.1 Given two incompletely specified functions $f = (f^{on}, f^{off})$ and $g = (g^{on}, g^{off})$, the candidate assignments ψ matching f to g is formulated as

$$\psi = \textbf{totality} \cdot \bigcap_{\substack{i=1 \text{ to } |f^{on}| \\ j=1 \text{ to } |g^{off}|}} MvCube(u_i^{on}, v_j^{off}) \cdot \bigcap_{\substack{i=1 \text{ to } |f^{off}| \\ j=1 \text{ to } |g^{on}|}} MvCube(u_i^{off}, v_j^{on}), \quad (3)$$

where $|f^{on}|$, $|f^{off}|$, $|g^{on}|$, and $|g^{off}|$ are the numbers of cubes for f^{on}, f^{off}, g^{on}, and g^{off}, respectively, $u_i^{on} \in f^{on}$, $u_i^{off} \in f^{off}$, $v_j^{on} \in g^{on}$, and $v_j^{off} \in g^{off}$.

< *proof* :> Directly from **Rule 1** and **Rule 2**. The anding with **totality** is to ensure infeasable solutions are removed.

The same example of Example 3.2 is used to show that based on Theorem 3.1 all possible assignments can be derived using logic operations (AND and OR) on a multi-valued function as follows.

Example 3.3 Given two incompletely specified functions $f(x_1, x_2, x_3)$ and $g(y_1, y_2, y_3)$. Let $f^{on} = \bar{x_1}\bar{x_2} + x_1x_2x_3$, $f^{off} = x_2\bar{x_3} + x_1\bar{x_3}$, $g^{on} = \bar{y_1}y_2$, and $g^{off} = \bar{y_3}y_1 + y_3\bar{y_1}\bar{y_2}$. The candidate assignments ψ matching f to g is formulated as :

$$\begin{aligned}
\psi = \; & \textbf{totality} \cdot MvCube(\bar{x_1}\bar{x_2}, \bar{y_3}y_1) \cdot MvCube(\bar{x_1}\bar{x_2}, y_3\bar{y_1}\bar{y_2}) \cdot \\
& MvCube(x_1x_2x_3, \bar{y_3}y_1) \cdot MvCube(x_1x_2x_3, y_3\bar{y_1}\bar{y_2}) \cdot \\
& MvCube(x_2\bar{x_3}, \bar{y_1}y_2) \cdot MvCube(x_1\bar{x_3}, \bar{y_1}y_2) \\
= \; & \textbf{totality} \cdot (x_1^{\{1\}} + x_2^{\{1\}}) \cdot (x_1^{\{3\}} + x_2^{\{3\}}) \cdot (x_1^{\{3\}} + x_2^{\{3\}} + x_3^{\{3\}}) \cdot \\
& (x_1^{\{1,2\}} + x_2^{\{1,2\}} + x_3^{\{1,2\}}) \cdot (x_2^{\{1\}} + x_3^{\{2\}}) \cdot (x_1^{\{1\}} + x_3^{\{2\}}) \\
= \; & x_1^{\{3\}}x_2^{\{1\}}x_3^{\{2\}} + x_1^{\{1\}}x_2^{\{3\}}x_3^{\{2\}}.
\end{aligned}$$

There are two cubes $x_1^{\{3\}}x_2^{\{1\}}x_3^{\{2\}} + x_1^{\{1\}}x_2^{\{3\}}x_3^{\{2\}}$ in the result. The first cube represents an assignment ψ_1 which maps x_1, x_2, x_3 to y_3, y_1, y_2, respectively, and the second cube represents an assignment ψ_2 which maps x_1, x_2, x_3 to y_1, y_3, y_2, respectively. By the first assignment, Figure 2(a), (b) and (c) shows the Karnaugh maps of f, $f' = \psi_1(X)$ and g, respectively. It is clear that f' and g are consistent and thus f and g are matched by ψ_1.

3.3 Extension to Input/Output Phase Assignment

The method presented in the previous subsections is good only for input permutation. In this section, we will

extend this method to dealing with input/ouput phase assignments.

Input Phase Assignment

Input phase assignment allows that each input x_i maps to an unique y_j or $\bar{y_j}$. Since each x_i may map to y_j or $\bar{y_j}$, the number of values of each multiple-valued variable is doubled. Therefore, given two Boolean functions $f(X)$ and $g(Y)$, where input set $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$, the multiple-valued functions for representing assignments extended to input phase assignment is defined as:

$$F : P_1 \times P_2 \times \cdots \times P_n \rightarrow B, \quad (4)$$

where $P_1 = P_2 = \cdots = P_n = \{1, 2, \cdots, n, n+1, \cdots, 2n\}$ and $B = \{0, 1\}$. For each P_i, the first n represents y_j in uncomplemented form and the second n (i.e., $n + j$) represents y_j in complemented form.

As a result, to deal with input phase assignment, the definition of $MvCube(u, v)$ is modified as follows.

$$MvCube(u, v) = \sum_{x_i \in \textit{1-lit}(u)} x_i^{S_a} + \sum_{x_i \in \textit{0-lit}(u)} x_i^{S_b}, \quad (5)$$

where $S_a = \{j \mid y_j \in \textit{0-lit}(v)\} \cup \{j + n \mid y_j \in \textit{1-lit}(v)\}$ and $S_b = \{j \mid y_j \in \textit{1-lit}(v)\} \cup \{j + n \mid y_j \in \textit{0-lit}(v)\}$.

Output Phase Assignment

To deal with output phase assignment, we simply consider matching f and $\bar{g}$ (the complement of g). Given two functions $f = (f^{on}, f^{off})$ and $g = (g^{on}, g^{off})$. Since $\bar{g} = (g^{off}, g^{on})$, To match f and the complement of g, Equation (3) is modified as:

$$\psi = \textbf{totality} \cdot \bigcap_{\substack{i=1 \text{ to } |f^{on}| \\ j=1 \text{ to } |g^{on}|}} MvCube(u_i^{on}, v_j^{on}) \cdot \bigcap_{\substack{i=1 \text{ to } |f^{off}| \\ j=1 \text{ to } |g^{off}|}} MvCube(u_i^{off}, v_j^{off}). \quad (6)$$

With the above modification, our Boolean matching method is able to handle not only input permutation but also input/output assignment.

3.4 Implementation Issues

It is necessary to compute Equation (3) to obtain all candidate assignments. However, computing Equation (3) directly is impractical since the number of cubes may be too large. Instead, the intersection operation can be performed using two nested loops. Let f and g be two functions to be matched. At the beginning of the Boolean matching algorithm, the assignment ψ is set to be true. During each iteration, for each $u_i \in f^{on}$ (f^{off}) and each $v_j \in g^{off}$ (g^{on}), $MvCube(u_i, v_j)$ is computed, and ψ is set to be $\psi \cdot MvCube(u_i, v_j)$.

In the computation, totality function is not actually calculated. Operation of anding *totality* can be completed by checking the feasibility of each cube in the resultant cube list of $MvCube$ operation in each iteration. Moreover, redundant cubes is likely generated in each iteration. A cube is *redundant* if it is equal to or **completely** contained in another cube. To prevent the number of cubes from growing too large, redundant and infeasible cubes must be deleted during the computation. For that purpose, a procedure *Reduction* is designed to remove redundant and infeasible cubes. A user defined parameter, *threshold*, will be set up to control the time point when to call *Reduction*. When the number of cubes exceeds the threshold, *Reduction* is called.

```
Algorithm Partial-Assignment(ψ, f, g)
Input: f, g = binary-valued covers;
Output: return ψ or ∅;
Begin
    if (ψ is ∅) then return ∅;
    Sort the cubes in f and g by the cube size;
    for each cube u_i ∈ f do
        for each cube v_j ∈ g do
            N = MvCube(u_i, v_j);
            /* generate partial assignments for this
                pair of cubes */
            ψ = ψ ⋂ N;
            if (| ψ |> threshold) then
            /* | ψ |: the number of cubes in ψ */
                ψ = Reduction(ψ);
            /* remove redundant and infeasible
                cubes */
            endif
            if (ψ is ∅) then return ∅;
        endfor
        ψ = Reduction(ψ);
    endfor
    return ψ;
End
```

Figure 3: The *Partial-Assignment* Algorithm

The computation time of Equation (3) heavily depends on the sequence of pairs of u_i and v_j chosen for computing $MvCube(u_i, v_j)$. A good sequence may quickly reduce the searching space, i.e., only a small number of cubes remains in the cube list. Our heuristic is to compute the cubes with less number of literals first. The reason behind this heuristic is that if u_i and v_j have less number of literals, then the number of resulting cubes for operation $MvCube(u_i, v_j)$ is smaller. Moreover, since anding operation is performed during the process, if the initial search space is small, the subsequent operations is restricted in a smaller search space. The above Boolean matching algorithm is implemented as procedure *Partial-Assignment* shown in Figure 3.

4 Minimum Weighted Matching for the Best Assignment

In Section 3, We have shown an algorithm to find all candidate mappings which satisfy consistency conditions. To find a specific one among all candidates mappings, we model the problem as a matching problem on a bipartite graph. Also, We show that using this method, delay and power dissipation can be easily taken into account during technology mapping.

4.1 Bipartite Graph Matching

If two functions are matched, the result of *Partial-Assignment* will contain a sum of cubes and each cube corresponds to a set of candidate mappings. For example,

Example 4.1 Given $f(X)$, g(Y), and two input sets $X = \{x_1, x_2, x_3, x_4\}$ and $Y = \{y_1, y_2, y_3, y_4\}$. Let cube $c = x_1^{\{1,4\}} x_2^{\{3\}} x_3^{\{1,2,4\}} x_4^{\{2\}}$ be obtained by procedure *Partial-Assignment*. Then, two assignments can be deduced from this cube; one is mapping of x_1 to y_1, x_2 to y_3, x_3 to y_4 and x_4 to y_2 and the other is mapping of x_1 to y_4, x_2 to y_3, x_3 to y_1 and x_4 to y_2.

To find a specific mapping, we must ensure that assignment from x_i's to y_i's is an onto mapping. The selection of a specific solution can be modeled as a matching problem on a bipartite graph. Given functions $f(X)$ and $G(Y)$, where $X = \{x_1, x_2, \cdots, x_n\}$ and

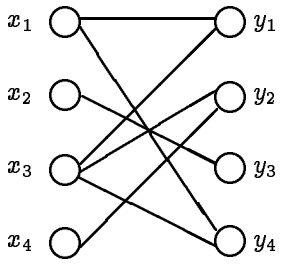

Figure 4: Bipartite Graph of $c = x_1^{\{1,4\}} x_2^{\{3\}} x_3^{\{1,2,4\}} x_4^{\{2\}}$.

$Y = \{y_1, y_2, \cdots, y_n\}$, and a cube $c = x_1^{S_1} x_2^{S_2} \cdots x_n^{S_n} \in \psi$ obtained by procedure *Partial-Assignment*. A bipartite graph $B = (U, V, E)$ will be constructed for c, where $U = X$, $V = Y$, and $e_{i,j} \in E$ is an edge connecting vertices x_i and y_j if $j \in S_i$. An assignment corresponds to a perfect match on the bipartite graph. Figure 4 shows the corresponding bipartite graph of Example 4.1. There are two perfect matching in the graph. Each corresponds to an assignment.

4.2 Considerations of Delay and Power Dissipation

Boolean matching technique is commonly used for technology mapping. When the result of procedure *Partial-Assignment* contains many assignments, choice must be made to select a specific one. Different choice may result in different cost. For example, during technology mapping, a subcircuit (f) in the Boolean network is mapped to a logic cell g in cell library. In most cases, f is incompletely specified since implicitly don't cares is used. If f and g are matched and the result contains many candidate assignments, the delay and power dissipation of the mapped circuits may vary for different assignments because different assignment means different input correspondence.

For delay and power dissipation optimization in technology mapping, the basic idea is to assign weights on edges in B and then find a minimum weighted matching. In the following, delay and power dissipation optimization are discussed.

Consider delay-driven technology mapping first. The goal is to map a late arrival signal to an input line whose longest path to the output node is shortest. Consider a subcircuit $f(X)$, a logic gate $g(Y)$ and input set $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$. Let a_i be the arrival time of signal x_i and d_j be the delay time of input y_j to the output of g. Then, the weight of $e_{i,j}$, $weight(e_{i,j})$, is defined as

$$weight(e_{i,j}) = \quad a_i + d_j \quad \text{for } j \in S_i. \tag{7}$$

Our objective is to find a perfect match whose largest cost on an edge is minimized. With some modification on the algorithm of perfect match, this problem can be solved in polynomial time. With these cost function and objective function, the arrival time of output g can be minimized.

Now, consider minimization of power dissipation in technology mapping. Power can be minimized by considering the input part and output part of a gate. For the input part of a gate, the goal is to have a smaller product of input-transition-density and input-load-capacitance. For the output part of a gate, the goal is to have a lower output-transition-density. Consider a subcircuit $f(X)$, a logic cell $g(Y)$ and input set $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$. Let t_{x_i} be the transition-density of input x_i, t_{y_j} be the transition-density of g with respect to input y_j (the transition probability of g when y_i transits), and l_{y_j} be the input-load-capacitance of input line y_j. Then, to minimize power dissipation, the weight of $e_{i,j}$, $weight(e_{i,j})$, is defined as

$$weight(e_{i,j}) = \quad t_{x_i} \cdot t_{y_j} \cdot l_{y_j} \quad \text{for } j \in S_i. \tag{8}$$

With this cost function, the minimum-cost matching will result in an assignment with low power dissipation.

```
Algorithm Boolean-Matching(f, g)
Input: f = (f^on, f^off);
       g = (g^on, g^off);
Output: return (Success, ψ) if f and g are matched;
otherwise, return Failure;
Begin
    ψ = 1;
    ψ = Partial-Assignment(ψ, f^on, g^off);
    ψ = Partial-Assignment(ψ, f^off, g^on);
    if (S ≠ ∅) then
        ψ = Minimum-Matching(ψ);
        /*  ψ: a minimum-cost assignment  */
        return (Success, ψ);
    else
        return Failure;
    endif
End
```

Figure 5: The *Boolean-Matching* Algorithm

A complete Boolean matching algorithm is shown in Figure 5. Procedure *Partial-Assignment* is called to compute all candidate assignments. If the result is not empty, procedure *Minimum-Matching* is called to find a best assignment with respect to the defined cost function.

5 Experimental Results

The proposed Boolean matching algorithm has been implemented in C language on SUN Sparc station. Our algorithm can be applied both to completely specified and incompletely specified functions. To demonstrate the efficiency of our algorithm, 31 benchmarks of MCNC circuits have been tested. For each circuit, a new circuit was generated by randomly permuting its input variables. Then we applied our matching algorithm to match these two circuits. Two sets of experiments were conducted to test the matchings of completely specified and incompletely specified functions, respectively.

The first experiment was performed to match completely specified functions. Table 1 shows the experimental results. The columns with labels **on** and **off** give the number of cubes of the on-set and off-set for each circuit, respectively. The column **prod** is the product of **on** and **off** which is the number of cube pairs processed for finding candidate assignment. The column **CPU** shows the running time in seconds. The columns **without** and **with** show the running time of without using and using speed-up heuristic described in Section 3.4, respectively.

Table 1 shows that 6 out of 31 cases can not be completed in 3 hours CPU time without using speed-up heuristic. For 17 out of the remaining completed 25 cases, the running time can be reduced varying from 1% to 97% with speed-up heuristic. Summing up the total running time except the 6 incomplete cases, using speed-up heuristic achieves 61% less running time than without using this heuristic.

Table 1: Benchmarking Results for Completely Specified Functions

circuits	No. of Cubes		prod	CPU	
	on	off		without	with
5xp1	65	76	4940	0.71	0.34
*alu4	575	609	350175		250.34
*apex1	206	1234	254204		1591.24
*apex3	280	756	211680		2012.25
*apex4	435	1471	639885		327.73
bw	22	66	1452	0.15	0.13
b12	42	29	1218	0.17	0.10
clip	119	151	17969	301.46	7.89
cm138a	9	8	72	0.08	0.08
cm150a	17	16	272	16.53	16.53
cm151a	17	17	289	0.51	0.51
cm152a	8	8	64	0.35	0.34
cm162a	25	35	875	0.25	0.10
cm163a	24	27	648	0.10	0.06
cm42a	7	10	70	0.01	0.01
cm82a	23	23	529	0.11	0.11
cm85a	48	57	2736	4.69	1.34
con1	9	9	81	0.02	0.01
duke2	87	314	27318	9.01	6.36
ex5	73	134	9916	5.01	4.87
f2	9	10	90	0.01	0.01
f51m	76	78	5928	1.08	0.65
inc	31	70	2170	0.70	0.64
misex1	12	30	360	0.07	0.07
misex2	28	73	2044	0.20	0.15
*misex3	678	679	472566		275.52
misex3c	197	616	121352	147.86	146.30
*pdc	251	987	247737		137.92
sao2	58	80	4640	10.21	2.04
vg2	110	194	21340	21.91	14.17
z4ml	59	59	3481	1.20	1.20
total	1175	2190	229854	522.40	204.01

The second experiment is designed for incompletely specified functions. To show the different size of don't-care set, we perform this experiment by removing 10%, 20%, and 30% of the cubes in on-set and off-set of the second circuit. Then the second circuit (incompletely specified function) is matched to the first one (completely specified function). Table 2 shows the experimental results. The columns labeled **-10%**, **-20%**, and **-30%** show the results of cases where 10%, 20%, and 30% of cubes are removed, respectively. The columns **product** and **CPU** show the number of cube pairs processed and the running time in seconds. Table 2 shows that matching functions with smaller size of on-set and off-set needs less running time. In average, the running time of the cases for removing 20% and 30% cubes are 15% and 20% quicker than the case of removing 10% cubes, respectively.

6 Conclusions

In this paper, we have addressed the Boolean matching problem for incompletely specified functions. Multiple-valued functions are used to model assignment. We formulate the searching of candidate assignments by using a Boolean equation. A Boolean matching algorithm based on the computation of this equation is proposed. Some implementation issues to speed up the searching process are also presented. Moreover, delay and power dissipation may be taken into account when this method is used for technology mapping. Experimental results show that our algorithm is indeed efficient for many benchmark circuits.

Table 2: Benchmarking Results for Incompletely Specified Functions

circuits	-10%		-20%		-30%	
	prod	CPU	prod	CPU	prod	CPU
5xp1	8828	0.93	7852	0.84	6865	0.80
alu4	629953	478.39	560165	422.35	498768	373.73
apex1	456950	2665.32	405698	2061.94	355471	2074.16
apex3	380912	3326.28	338464	2976.01	296296	2762.98
apex4	1150666	635.64	1024368	564.00	894799	500.65
bw	2552	0.43	2266	0.36	2002	0.32
b12	2165	0.27	1923	0.22	1681	0.20
clip	32222	14.82	28625	12.45	25028	11.13
cm138a	127	0.09	110	0.08	93	0.07
cm151a	510	4.03	442	3.74	374	3.90
cm152a	112	0.89	96	0.37	80	0.30
cm162a	1545	0.25	1400	0.17	1195	0.19
cm163a	1143	0.41	1017	0.10	864	0.18
cm42a	123	0.04	106	0.02	89	0.01
cm82a	920	0.34	828	0.32	736	0.29
cm85a	4899	4.18	4326	4.07	3753	2.41
con1	144	0.04	126	0.03	108	0.03
duke2	49026	11.15	43503	10.58	37893	9.01
ex5	17724	8.87	15824	7.88	13716	6.89
f2	161	0.03	142	0.03	123	0.03
f51m	10624	1.63	9392	1.39	8238	1.29
inc	3843	1.18	3416	1.06	2989	0.99
misex1	624	0.21	558	0.20	492	0.16
misex2	3645	26.30	3230	21.69	2815	7.97
misex3	850276	534.81	765226	478.25	669748	414.98
misex3c	223304	272.03	193636	238.86	169299	205.10
pdc	444963	344.30	395439	291.45	345915	262.26
sao2	8336	5.63	7392	5.04	6448	4.53
vg2	38346	25.68	34122	23.15	29788	18.40
z4ml	6254	4.52	5546	4.24	4838	3.65
total	4293831	8368.73	3855238	7130.89	3380504	6666.61
norm.	100	100	90	85	79	80

References

[1] F. Mailhot, and G. De Micheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," in *Proc. of EDAC'90*, pp.212-216, 1990.

[2] H. Savoj, M.J. Silva, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Boolean Matching in Logic Synthesis," in *Proc. of Euro-DAC'92*, pp.168-174, 1992.

[3] K-C Chen, "Boolean Matching Based on Boolean Unification," in *Proc. of Euro-DAC'93*, pp.346-351, 1993.

[4] U. Schlichtmann, F. Brglez, and P. Schneider, "Efficient Boolean Matching Based on Unique Variable Ordering," in *IWLS'93*.

[5] J.R. Bruch, and D. Long, "Efficient Boolean Function Matching," in *Proc. of ICCAD'92*, pp.408-411, 1992.

[6] K. Zhu, and D.F. Wong, "Fast Boolean Matching for Field Programmable Gate Arrays," in *Proc. of Euro-DAC'93*, pp.352-357, 1993.

[7] U. Schlichtmann, F. Brglez, and M. Hermann, "Characterization of Boolean Functions for Rapid Matching in EPGA Technology Mapping," in *Proc of DAC'92*, pp.374-379, 1992.

[8] Y.T. Li, S. Sarma, and P. Massoud, "Boolean Matching using Binary Decision Diagrams with Applications to Logic Synthesis and Verification," in *Proc. of ICCD'92*, pp.452-458, 1992.

[9] D.I. Chen and M. Marek-Sadowska, "Verifing Equivalence of Functions with Unknown Input Correspondence," in *Proc. of EDAC'93*, pp.81-85, 1993.

[10] J. Mohnke, and S. Malik, "Permutation and Phase Independent Boolean Comparison," in *Proc. of EDAC'93*, pp.86-92, 1993.

[11] C.C. Tsai, and M. Marek-Sadowska, "Boolean Matching Using Generalized Reed-Muller Forms," in *Proc. of DAC'94*, pp.339-344, 1994.