Generalized Matching: a new approach to concurrent logic optimization and library binding

L. Benini, M. Favalli* and G. De Micheli
C.I.S. Stanford University
*D.E.I.S. University of Bologna

Abstract

In this work we describe a novel approach to the classical problem of library binding. We define the concept of generalized matching, a Boolean constraint whose satisfaction allows us to find all possible input assignments for which a library element matches a given Boolean relation. Completely and incompletely specified Boolean functions are treated as particular cases. Generalized matching supports the simultaneous matching of more than one single-output cell (or the optimal use of multi-output cells) and allows the optimization of pin-assignment dependent cost functions.

We describe a fully symbolic procedure for generalized matching, and we discuss the related efficiency issues. The advantages of generalized matching over traditional Boolean matching are then discussed by means of examples.

1 Introduction

With the diffusion of reliable and effective tools for logic optimization and library binding, designers have adopted methodologies that heavily rely on automatic tools for the back-end parts of the design flow, namely the synthesis and optimization of logic circuits.

Several formalisms have been explored for the representations and exploitation of the degrees of freedom allowed in the implementation of multilevel logic circuits. We can outline three broad classes: algebraic approaches [13], *don't care* based techniques [17, 12] and Boolean relation based optimizations [14, 15]. These approaches allow the exploration of increasingly larger portions of the optimization space, at the expenses of increasing computational complexity. Such techniques are *technology independent*, because they do not rely on any particular assumption on the technology that will be used for the final implementation.

The technology-dependent part of the synthesis process has been called *library binding* (or *technology mapping*) and performs a transformation of an optimized technology-independent logic network into an equivalent network consisting only of logic blocks belonging to a predefined *library*. Library binding algorithms targeting standard cell or gate array implementations can be algorithmic or rule-based. We concentrate here on algorithmic techniques that start from an unmapped network and decompose it in simple base functions (generally two-input gates). This step is called *decomposition*. The decomposed network is then processed starting from the inputs and moving toward the outputs. During this process, subnetworks of the unmapped network (called target networks and represented by *target functions*) are selected and replaced by cell-library instances. This phase is called *covering*.

Covering algorithms can be divided into two broad classes: i) structural approaches [10, 18], based on efficient graph matching algorithms and ii) Boolean approaches [7, 19, 9], that represent library cells and portions of the unmapped network as Boolean functions, and employ mixed graph-based and Boolean techniques. Although slightly less efficient, Boolean techniques can improve upon the results obtained by the structural approach.

This work proposes a general solution to the problem of determining when a target function can be substituted by one or more library cells. This problem is usually called *Boolean matching*, and it is the core of all tools performing library binding with Boolean techniques.

Our formulation is general, because it allows to solve the matching problem even if the target block under consideration is represented as a Boolean relation. Moreover, our algorithm produces a symbolic representation of *all* possible input assignments for which a library cell matches a given target block. This information gives us the possibility to employ and optimize accurate cost functions whose value depends on the input pin assignment. Our approach is fully symbolic and takes advantage of highly optimized BDD-based tools [16]. Generalized matching supports the simultaneous matching of more than one single-output cell or the optimal use of multi-output cells. In our approach, library binding is an iterative improvement process that combines technology dependent and independent optimizations and offers the possibility to exploit degrees of freedom not generally available with traditional techniques.

2 Terminology and problem formulation

We assume that the reader is familiar with the basic concepts of Boolean algebra (see [11, 8] for a review). We will concentrate here on some more advanced concepts needed for the understanding of the following material.

The *input controllability don't care set* (CDC) for a Boolean function $f(\mathbf{x})$ (with support X) includes all input conditions that are never produced by the environment. We can define a CDC function $f_{CDC} : X \rightarrow \{0,1\}$ whose ON-set is the CDC-set of f.

The *output observability don't care set* (ODC) for f denote all input patterns that represents situations when f is not observed by the environment. We define an ODC function $f_{ODC} : X \rightarrow \{0,1\}$ whose ON-set is the ODC-set of f. The DC function $f_{DC} = f_{ODC} + f_{CDC}$ can be used to express all degrees of freedom available for the implementation $\tilde{f}$ of a single output function, namely:

$$f \cdot f'_{DC} \leq \tilde{f} \leq f + f_{DC} \tag{1}$$

or, equivalently:

$$(\tilde{f} + f'_{MIN}) \cdot (f_{MAX} + \tilde{f}') = 1 \tag{2}$$

where $f_{MIN} = f \cdot f'_{DC}$ and $f_{MAX} = f + f_{DC}$.

If we target the minimization of multi-output Boolean functions, the degrees of freedom provided by the environment can be expressed with a *Boolean relation* [14]. Intuitively, Boolean relations are generalizations of Boolean functions, where each input pattern may correspond to more than one output pattern. If we call X the input space and Y the output space, a Boolean relation $\Re$ can be represented with its *characteristic function*: $\mathcal{X} : X \times Y \rightarrow \{1,0\}$ such that $\mathcal{X}(\mathbf{x},\mathbf{y}) = 1$ if and only if $\mathbf{y}$ is one of the possible outputs of $\Re$ for the input $\mathbf{x}$. We clarify these definitions with an example.

Example 1 *Consider the Boolean relation represented by the following table:*

$in_1\, in_2$	$out_1\, out_2$
00	{00, 11}
01	{00, 11}
10	{10}
11	{10}

Its characteristic function is: $\mathcal{X} = in'_1\, out'_1\, out'_2 + in'_1\, out_1\, out_2 + in_1\, out_1\, out'_2$.

We review some basic concepts related to the problem of Boolean matching. A frequently used notion is NPN-*equivalence* [7, 8]. Two Boolean functions f and g are NPN-equivalent when they are equivalent modulo the negation of the output (N), the permutation (P) of the inputs and the negation (N) of the inputs. The NPN matching problem consists of finding (if it exists) a NPN transformation that makes a cell of a predefined library equivalent to a target Boolean function.

Many approaches have been proposed for the solution of this problem [7, 2, 5, 6, 9]. We will concentrate here on the most common technique used. In a preprocessing once-for-all procedure a set of *filters* is associated with the library. Filters allow to early rule out a large number of library elements that cannot match a given target function. The library elements that pass the filters are then matched against the function.

The individuation of the correct (if any) NPN transformation is done iteratively constructing the BDDs representing the various versions of the library element modulo the NPN transformations and checking for Boolean equivalence with the target function. The number of attempts can be highly reduced if symmetries and polarity invariances are detected [7]. Efficient detection of symmetries and polarity invariances can be done employing *spectral techniques* [6].

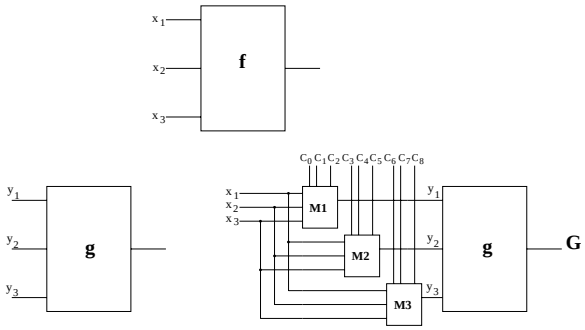

Figure 1: *Transformation of the library cell $g \rightarrow G$ for matching with function f. The first two control variables of each multiplexer are for permutation control, the last one is for polarity control.*

The approaches outlined above have been implemented with successful results [3, 2, 9]. A recent study has shown that the combined use of filters and symmetry detection is extremely effective in almost all practical cases [4].

Nevertheless, the Boolean matching problem is more complex and still partially unsolved for incompletely specified functions. Some attempts have been made to generate filters that are valid for matching incompletely specified functions [1], but they are less effective than the filters used for exact matching.

3 Exact generalized matching

Our matching algorithm is a two-step procedure. In this section we will describe it for the simplest case of matching a completely specified Boolean function (this case will be called *exact matching*).

We consider a target function $f(\mathbf{x})$ and a library cell function $g(\mathbf{y})$ with n inputs. The first step of our procedure is a transformation of the library cell function g. For the ease of understanding we will describe this transformation as it would be done using logic blocks.

On each input of g we connect the output of a multiplexer whose inputs are the same inputs of function f (Figure 1). The control inputs of each multiplexer have the following function: the first $\lceil log_2\, n \rceil$ control inputs control which of the external n inputs is multiplexed on the input of g. The last control variable controls the polarity of the selected external input.

Example 2 *In the case of Figure 1, consider multiplexer M1. If the control variables C_0 and C_1 are 00, the input x_1 is connected with y_1. If the polarity control variable C_2 is 1, the connection with y_1 will be inverting, therefore x'_1 will be seen on y_1.*

From our construction it is clear that the number of control variables needed is $N_c = n(\lceil log_2\, n \rceil + 1)$. The key

observation is that the controls variables $\mathbf{c}$ can be selected in such a way that all PN-equivalent functions of g can be generated (the inversion of the output can be obtained with one more control variable for the output polarity. We restrict to PN for the sake of simplicity).

The number of inputs of the target function and the cell library are not constrained to be the same. In general we allow m inputs for g and n inputs for f. If $n > m$ some of the inputs of f will not be used by g. In this case g can match only if some of the variables in the support of f are redundant (an unlikely condition if we do not consider *don't cares*). When $m > n$, some of the inputs of f will be connected with two or more inputs of g.

In general, the class of functions generated by assignments of $\mathbf{c}$ is actually larger than all input permutations and polarity changes. It includes the cases where two or more of the inputs of g are connected to the same external input with arbitrary polarity or some of the external inputs is left unconnected. We call this equivalence relation *extended-PN* (EPN). The extension to $ENPN$ is straightforward.

The library cell function $g(\mathbf{y})$ has been transformed to a new Boolean function $G(\mathbf{c}, \mathbf{x})$. The EPN equivalence relation is defined over the set S of all the Boolean functions with n inputs. EPN-equivalence partitions S in equivalence classes. The set of equivalence classes defined by an equivalence relation is called *quotient set*. We call G *quotient function* because it implicitly represents a quotient class, in fact, all possible assignments of the $\mathbf{c}$ variables individuate all possible functions of $\mathbf{x}$ that belong to the same class as the original library cell function g.

The construction of G is performed directly starting from the BDD of g. Thanks to the binary encoding on the control variables of the multiplexers, the size of $\mathbf{c}$ is $O(m\, log_2\, n)$. This is an important property, because we want to keep the number of variables in the BDD representation of G as small as possible for efficiency reasons.

The second step of our procedure consists of defining a Boolean formula [11] that has at least one satisfying assignment if and only if there exists a function EPN-equivalent to g that is equivalent to f. Intuitively, this problem can be solved observing that there is an EPN matching if and only if there exists an assignment $\mathbf{c_0}$ to the control variables $\mathbf{c}$ of $G(\mathbf{c}, \mathbf{x})$ such that $G(\mathbf{c_0}, \mathbf{x})$ is equal to $f(\mathbf{x})$ for all possible values of $\mathbf{x}$. The key point is to realize that this condition can be immediately translated into the following Boolean formula:

$$M(\mathbf{c}) = \forall_{\mathbf{x}} \left[G(\mathbf{c}, \mathbf{x}) \overline{\oplus} f(\mathbf{x}) \right] \tag{3}$$

where $\forall_{\mathbf{x}}$ represents the universal quantifier (consensus) applied to all variables in $\mathbf{x}$ and $\overline{\oplus}$ represents the XNOR operator. The formula above can be seen as a function of the control variables $\mathbf{c}$. We will call it *matching function*, $M(\mathbf{c})$, for brevity. Notice that the ON-set of M contains all possible input assignments and polarity inversions that match with f. There are two important consequences. First, the formula above can be efficiently computed in a fully symbolic way, using BDDs. Second, our procedure finds *all* possible matchings given f and g, not only a particular one.

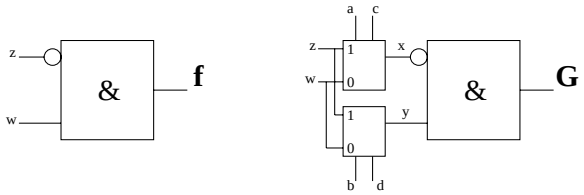

Figure 2: *Target function f and quotient function G of example 3.*

Example 3 *The library cell function is $g = x'y$. We want to find the matching with $f = wz'$. Figure 2 shows $G(a,b,c,d,w,z) = (c \oplus (za + wa'))'(d \oplus (zb + wb'))$, where a,c and b,d are the control variables. We take the XNOR of f and G:*

$$f \overline{\oplus} G = (wz') \overline{\oplus} ((c \oplus (za + wa'))'(d \oplus (zb + wb')))$$

Computing the consensus of the above Boolean formula with respect to w and z (the order does not matter), we obtain $M(a,b,c,d) = ab'c'd' + a'bcd$. There are two minterms in $M(a,b)$ that correspond to the two possible solutions. Minterm $ab'c'd'$ corresponds to assigning z to x and w to y without any polarity change. Minterm $a'bcd$ corresponds to assigning z to y and w to x changing both polarities. The correctness and completeness of the solution set represented by M can be verified by inspection.

From a practical point of view, our algorithm operates as follows. Given the BDD of f and the BDD of G (constructed starting from g), the BDD of $G(\mathbf{c}, \mathbf{x}) \overline{\oplus} f(\mathbf{x})$ is constructed. The last step is the computation of the consensus over all variables in $\mathbf{x}$ that produces $M(\mathbf{c})$.

Notice that our matching procedure requires more Boolean variables than the traditional approach, therefore the BDD computations involved are very likely to be more expensive. The efficiency of filters and symmetry detection is such that for a very high percentage of cases the traditional approach can find a matching doing only a single BDD comparison (performed in constant time), therefore the traditional matching approach would appear to be more convenient. This may be true, especially if we are not interested in obtaining all possible matchings. We will show in the next section that our approach is applicable to Boolean matching problems more general that exact matching, where traditional techniques cannot be applied.

4 Generalized matching

We will first consider the case of incompletely specified single-output target function. Next, we generalize to multi-output target function.

4.1 Matching incompletely specified functions

In a general multilevel circuit, single-output target functions have ODCs and CDCs. These degrees of freedom should be exploited when the library binding step is performed. The matching problem with DCs (DC-matching, for brevity) can be formulated as follow. Given a Boolean

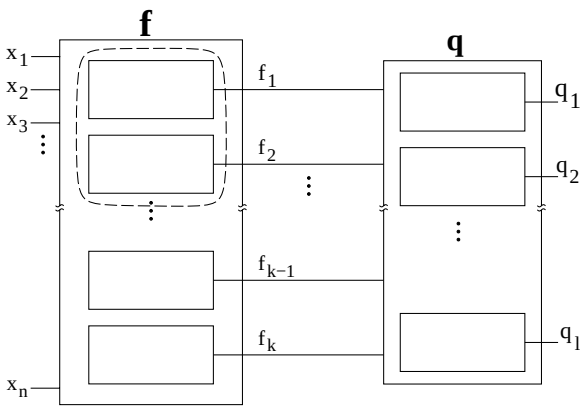

Figure 3: *The general multi-output matching problem. The unbound target functions of* **f** *are enclosed by the dashed line.*

target function f and its DC function f_{DC}, a library cell function g matches f if and only if there is an input permutation and polarity assignment $g \overset{NPN}{\rightarrow} \tilde{g}$ such that $(\tilde{g} + f'_{MIN})(\tilde{g}' + f_{MAX})$ is true for all input combinations.

The increased complexity of this kind of Boolean matching stems from the fact that not all filters and symmetries are invariant when f_{DC} is not null. In this case our approach completely avoids the problems, because it is possible to formulate Boolean matching with DC as a simple Boolean formula. Since the quotient function $G(\mathbf{c}, \mathbf{x})$ implicitly represents the complete EPN class of functions generated by g, we can simply insert it in the formula of DC-matching. We then universally quantify out $\mathbf{x}$, enforcing the validity of the condition for all input configurations. In symbols:

$$M(\mathbf{c}) = \forall_{\mathbf{x}} \left[(G(\mathbf{c}, \mathbf{x}) + f_{MIN}(\mathbf{x})') \cdot (f_{MAX}(\mathbf{x}) + G(\mathbf{c}, \mathbf{x})') \right] \quad (4)$$

The result of the consensus is again the matching function $M(\mathbf{c})$ whose ON-set is the set of all possible assignments of the control variables that satisfy the matching condition. Observing the formula, two points are of interest. First, when $f_{DC} = 0$ we have $f_{MIN} = f_{MAX} = f$ and equation (4) degenerates to equation (3). Second, solving the DC-matching problem in our approach is not more difficult than solving the exact matching problem, the only difference being that a different formula must be universally quantified (the number of Boolean variables involved do no change).

Another interesting point is that our procedure can be applied to target functions and library cell functions with different number of inputs. Therefore it is able to find a match even when the minimum cost library element g compatible with f has smaller or larger support than f.

4.2 Matching Boolean relations

We consider the scenario shown in Figure 3. We have a logic network where we identify a set of logic blocks $q_1, q_2, ..., q_l$ (represented by multi-output Boolean function **q**). Consider the set of logic blocks that are predecessors of **q**, which we call $f_1, f_2, ..., f_n$ (represented by a multi-output Boolean function **f**). We focus on the library binding problem for the components **f**. In the traditional single-output approach, we would bind the components of **f** one by one (considering *don't cares* conditions). Using generalized matching it is possible to perform concurrent binding of two or more target functions.

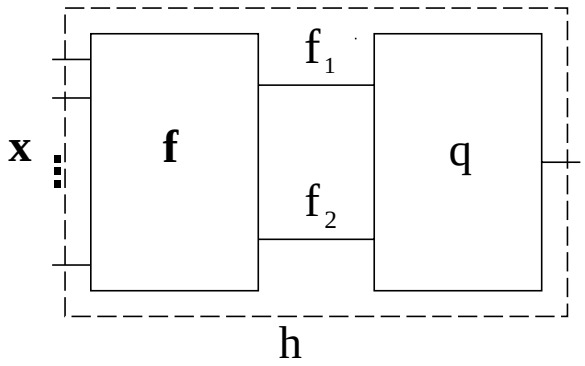

Figure 4: *A typical example of situation where BR-matching is applicable.*

We will show that concurrent binding requires to find a group (*cluster*) of single-output cells (or a single multi-output cell) that satisfy a Boolean constraint expressed as a Boolean relation. This flavor of generalized matching will therefore be called BR-matching. Roughly speaking, BR-matching is more powerful than DC-matching, as in the case of the corresponding technology independent optimizations. The components of **f** that will be concurrently mapped are called *unbound*. The remaining components of **f** are considered as *bound* and we will not attempt to bind the corresponding target functions to library cells.

The two limiting cases of this situation are when only one component is considered unbound and when all components are considered unbound. The first limiting case has already been addressed in the previous section. We will discuss here the second limiting case (**f** is *fully-unbound*), from which all the intermediate situations can be easily derived. In order to keep the formalism as simple as possible, we will analyze the case of a fully-unbound, two-output target Boolean function **f**, as shown in Figure 4 (the extension to the general multi-output case is straightforward). Moreover, we will assume that the composite function $h(\mathbf{x})$ is completely specified and single-output (this hypothesis will be relaxed later).

Whenever $h = 1$, we know that the function q must be $q = 1$ as well (their outputs coincide). The opposite holds when $h = 0$. We can translate this simple observation in a Boolean constraint:

$$qh + q'h' \quad (5)$$

that must hold for each value of $\mathbf{x}$. Notice that the support of q is not $\mathbf{x}$, but the vector **f** (in our case consisting of f_1 and f_2). We want to test if two library cell functions g_1 and g_2 (or a two-output library element) can implement block **f**, without changing the external behavior of h. We will use the quotient functions $G_1(\mathbf{c_1}, \mathbf{x})$ and $G_2(\mathbf{c_2}, \mathbf{x})$ that

implicitly represent the EPN classes. The constraint (5) enforced on all input vectors becomes:

$$M(\mathbf{c_1},\mathbf{c_2}) = \forall_{\mathbf{x}}[q(G_1(\mathbf{c_1},\mathbf{x}),G_2(\mathbf{c_2},\mathbf{x}))h(\mathbf{x}) + q(G_1(\mathbf{c_1},\mathbf{x}),G_2(\mathbf{c_2},\mathbf{x}))'h(\mathbf{x})'] \quad (6)$$

We clarify the meaning of this Boolean formula through an example.

Example 4 *Consider a block* $\mathbf{f}$ *with three inputs* (x_1, x_2, x_3) *and two outputs* f_1 *and* f_2 *that are connected to the inputs of an AND gate. We have* $q = f_1 f_2$. *Assume that the global function is* $h = x_1 x_2' + x_3$. *The Boolean constraint enforced by this structure is:*

$$(f_1 f_2)(x_1 x_2' + x_3) + (f_1' + f_2')(x_1' x_3' + x_2 x_3')$$

which is the characteristic function of the following BR

$x_1x_2x_3$	f_1f_2
000	{10,01,00}
001	{11}
010	{10,01,00}
011	{11}
100	{11}
101	{11}
110	{10,01,00}
111	{11}

The Boolean relation is constructed by AND-ing the only admissible output {11} ($f_1 f_2$) *with all input configurations corresponding to the ON-set of* h: $x_1 x_2' + x_3$, *and by AND-ing the admissible outputs* {10,01,00} ($f_1' + f_2'$) *with the remaining input configurations (the OFF-set of* h: $x_1' x_3' + x_2 x_3'$).

We will allow EP matching (input polarity inversions are not allowed) for the sake of simplicity. Our candidate library cell functions for BR-matching of block $\mathbf{f}$ *are* $g_1(y_1,y_2) = (y_1' y_2')'$ *(for* f_1*) and* $g_2(y_1,y_2) = (y_1 y_2')'$ *(for* f_2*). We need* $N_c = 2 * 4$ *control variables for P matching (two control variables for each input of the library cells). The quotient functions are:* $G_1 = ((c_1' c_2' x_1 + c_1' c_2 x_2 + c_1 c_2' x_3)'(c_3' c_4' x_1 + c_3' c_4 x_2 + c_3 c_4' x_3)')'$ *and* $G_2 = ((c_5' c_6' x_1 + c_5' c_6 x_2 + c_5 c_6' x_3)(c_7' c_8' x_1 + c_7' c_8 x_2 + c_7 c_8' x_3)')'$

Replacing in the expression of the Boolean constraint all the occurrences of f_1, f_2 *with* G_1 *and* G_2 *we obtain the Boolean formula for BR-matching. We then universally quantify the formula with respect to* x_1, x_2 *and* x_3. *The resulting matching function* M *is* $M(c_1,c_2,...,c_8) = c_1' c_2' c_3 c_4' c_5' c_6 c_7 c_8' + c_1 c_2' c_3' c_4' c_5' c_6 c_7 c_8'$ *(representing all allowed input assignments). There are two minterms because of the symmetry of the library element* g_1.

Notice that formula (6) reduces to formula (3) if the block $\mathbf{f}$ has one output. The number of control variables needed is $N_c = m_1(\lceil log_2(n)\rceil+1)+m_2(\lceil log_2(n)\rceil+1)$ where m_1 and m_2 are respectively the number of inputs of the library cells to be inserted in place of f_1 and f_2 and n is the number of external inputs.

The general fully-unbound multi-output case (when the block $\mathbf{f}$ has k outputs) is conceptually the same as the two-output case above described. From the practical point of view, however, the complexity of BR-matching increases very rapidly with the number of outputs of block $\mathbf{f}$. First, the number of control variables is $O(k\, m\, log_2\, n)$ (where m is the number of inputs of the library cell and n is the number of inputs of the block $\mathbf{f}$). Second, the number of possible clusters of library cells to be tried is $O(N^k)$ (where N is the number of cells in the library).

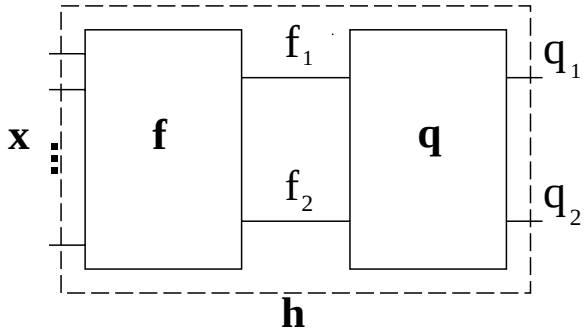

Figure 5: *A situation where* BR^2*-matching is applicable.*

Let us now consider the general case in which only some of the target functions of block $\mathbf{f}$ are unbound. For each unbound component, the quotient function G of a candidate library cell will be inserted in (6), while the bound target functions will be left untouched. The advantage of this approach is that it can be applied to situations where the number of control variables needed for fully-unbound BR-matching is too high, or the number of possible clusters of library cells is excessive.

Moreover, many libraries include multi-output cells (like full adders and decoders). We can restrict the use of BR-matching to the multi-output cells predefined in our library. In this case, the dependence from k of the number of control variables disappears, because if a matching input assignment exists, it must be the same for all the output functions (the quotient function for a multi-output cell does not have more control variables than the quotient function of a single-output cell with the same number of inputs).

Our last step is to introduce the most powerful form of generalized matching: we will consider the case of BR-matching when the block h itself is described by a Boolean relation. We call this case BR^2*-matching*. Consider the characteristic function $\mathcal{X}(\mathbf{x})$ of the Boolean relation representing the block h (obviously, the block h is a multi-output function $\mathbf{h}$, otherwise the specification as a Boolean relation would be meaningless). We assume for simplicity that h has only two outputs: $h_1(\mathbf{x})$ and $h_2(\mathbf{x})$ (corresponding to the output of the $\mathbf{q}$ block: q_1 and q_2). This case is depicted in Figure 5. We call $\mathbf{G}$ the array of quotient functions of cells that we want to place into block $\mathbf{f}$. The Boolean formula for BR^2-matching is:

$$\begin{aligned} M(\mathbf{c}) = \forall_{\mathbf{x}}[\ & \mathcal{X}_{h_1 h_2}(\mathbf{x}) q_1(\mathbf{G}(\mathbf{c},\mathbf{x})) q_2(\mathbf{G}(\mathbf{c},\mathbf{x})) + \\ & \mathcal{X}_{h_1 h_2'}(\mathbf{x}) q_1(\mathbf{G}(\mathbf{c},\mathbf{x})) q_2(\mathbf{G}(\mathbf{c},\mathbf{x}))' + \\ & \mathcal{X}_{h_1' h_2}(\mathbf{x}) q_1(\mathbf{G}(\mathbf{c},\mathbf{x}))' q_2(\mathbf{G}(\mathbf{c},\mathbf{x})) + \\ & \mathcal{X}_{h_1' h_2'}(\mathbf{x}) q_1(\mathbf{G}(\mathbf{c},\mathbf{x}))' q_2(\mathbf{G}(\mathbf{c},\mathbf{x}))'\ ] \end{aligned} \quad (7)$$

Where $\mathcal{X}_{h_1,h_2}$, $\mathcal{X}_{h_1,h_2'}$, $\mathcal{X}_{h_1',h_2}$ and $\mathcal{X}_{h_1',h_2'}$ are the cofactors of the characteristic function $\mathcal{X}$ with respect to h_1 and h_2. The meaning of this Boolean formula is similar the one of the simpler BR-matching case. The detailed derivation is not presented here for space reasons.

In summary, we have shown that all flavors of generalized matching, from exact matching to BR^2-matching,

can be expressed as a satisfiability problem on a Boolean function. This is an interesting result from a theoretic point of view. The matching problem has traditionally been solved as a combination of a combinatorial problem (finding the correct permutation and polarity assignment for inputs and outputs) and a Boolean satisfiability problem (testing the equality of two Boolean functions). Our approach merges the two in a pure satisfiability problem on an extended Boolean space (input variables and control variables).

A mapper based on generalized matching has a structure that is very similar to BDD-based technology-independent optimization tools. First, BR-based matching can find matchings across fan-out points. Second, its efficacy is dominated by the size of the BDDs representing the matching functions $M(\mathbf{c})$. Third, generalized matching can be applied iteratively, because the exploitation of indifference conditions on a portion of a network can reveal indifference conditions on other portions that where hidden before.

5 Exploiting the matching function

The solution of the generalized matching problem involves the identification of the matching function $M(\mathbf{c})$ (where $\mathbf{c}$ are control variables) whose satisfying assignments (ON-set) are all possible input assignments and polarity changes that enable the substitution of a target function f with a library cell function g. This is an useful information that can be exploited for the individuation of the best input assignment according to a pre-defined cost function.

Simple delay and power models that do not depend on input assignment do not accurately describe the behavior of real circuits. We will consider here a timing-related and a power-related cost function for which input assignment is significant.

A timing-related cost function is the *path-dependent delay*. In order to accurately describe the timing behavior of CMOS circuits, the path-dependent delay model assigns a different delay to each path between inputs and output (we will consider single output blocks for simplicity). If the environment of a block is completely known (as it is the case for the optimization of an already mapped circuit), the path delays can be accurately calculated based on look-up tables, fan-in and fan-out information. Otherwise, if this model is used during the first mapping pass, only an estimate of the possible fan-out is known.

The cost of a library element could be defined to be the latest arrival time of its output for a given input assignment:

$$Cost = Max\{t_{in_1}+\delta_{in_1}, t_{in_2}+\delta_{in_2}, ..., t_{in_n}+\delta_{in_n}\} \quad (8)$$

where t_{in_i} and δ_{in_i} are respectively the arrival times and the path delays of the n inputs. Obviously, the cost of a library element depends on the input assignment.

Example 5 *Consider a 2 input NAND gate. Assume that it can safely replace (because of don't care conditions) a 2 input XOR gate. The cost (latest arrival time of the output) of the XOR gate is* $Cost = 12.3$. *The arrival time of the two inputs is* $t_{in_1} = 11.2$ *and* $t_{in_2} = 11.7$. *The path delays of the NAND are* $\delta_{in_1} = 0.5$, $\delta_{in_2} = 0.7$. *The replacement of the XOR gate with the NAND gate is convenient only if the external input* in_2 *is connected with the gate input 1 (*$Cost = Max\{11.7 + 0.5, 11.2 + 0.7\} = 12.2$*). The other input assignment gives a cost* $Cost = 12.4$ *that is higher than the one of the XOR gate.*

The formulation of suitable cost functions for minimum power dissipation mapping is still an open research topic. As an example we will refer to the power dissipation model proposed in [20] where the charge and discharge of internal nodes is taken into account. In this case the cost function (power dissipated on a gate) is defined as:

$$Cost = \sum_{j \in internal\_nodes} P^{int}_{avg}(j) + \sum_{i \in inputs} P^{inp}_{avg}(i) \quad (9)$$

Where the two contributions are respectively the average power dissipated on internal nodes and the average power dissipated on the inputs of the gate plus the accumulated power cost of input i assuming a match with the gate under consideration. The first summation is strongly dependent on input assignment, in fact $P^{int}_{avg}(j)$ is proportional to the expected number of transitions on internal node j. As for the second summation, it may become dependent on input assignment if the input loads are not equal. Clearly, the minimization of such a cost function is guaranteed only if all admissible input assignments are known.

A simple procedure that finds the minimum of a cost function dependent on input assignment enumerates all the minterms of $M(\mathbf{c})$ and computes the cost function for each of them. The assignment corresponding to the minterm that produces the minimum cost function value is chosen. The essential advantage of this procedure is that it computes the cost function (a possibly expensive task) only for the minterms of M, that represent all admissible EPN transformations.

6 Examples and practical issues

The purpose of this work was to describe a new approach to the matching problem and more in general to library binding. We have implemented and tested the matching algorithm, but we have not yet developed a complete library binding tool based on these ideas. Therefore we only show the application of generalized matching on a simple but meaningful example. We will then discuss the issue of finding a good ordering on the BDD variables for maximum efficiency during generalized matching.

Example 6 *In this example we will consider EP-matching (input polarity changes are not allowed) for the sake of simplicity. Assume that we have a simple library containing 4 cells: two-input XOR (*$Cost = 2$*), two-input AND (*$Cost = 2$*), inverter NOT (*$Cost = 1$*), two-input AND1 (logic function* $in_1 in_2'$, $Cost = 3$*). An implicit cell is the "WIRE" whose cost is zero. We want to optimize the mapped network of Figure 6. Notice that the mapping cannot be improved with Boolean methods using don't cares because the external DC-set is empty and the XOR on the output does not introduce any ODC on its fan-ins.*

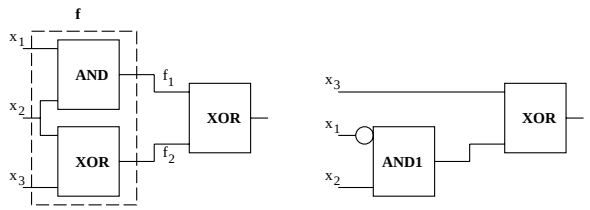

Figure 6: *Application of BR-matching on a multi-output sub-block.*

We apply BR-matching to the multi-output target function consisting of the first XOR and the AND (enclosed in the dashed box **f***). The Boolean condition for BR-matching is:*

$$h(G_1G_2' + G_1'G_2) + h'(G_1G_2 + G_1'G_2')$$

that must be universally quantified on the input variables x_1, x_2, x_3. Only the cell clusters that give a lower cost than the current one ($Cost = 4$) must be considered. The candidate cell clusters are listed in the following table.

Cost	Cluster
0	WIRE-WIRE
1	WIRE-NOT NOT-WIRE
2	NOT-NOT WIRE-AND XOR-WIRE WIRE-XOR
3	NOT-AND AND-NOT XOR-NOT NOT-XOR AND1-WIRE WIRE-AND1

The number of control variables needed is $4 * 2 = 8$ in the worst case (two two-input cells and three primary inputs), but we will need only $3 * 2 = 6$ for our restricted candidate set of clusters.

Applying BR-matching, we find that WIRE-AND1 is a correct replacement. The quotient functions are $G_1 = c_1'c_2'x_1 + c_1'c_2x_2 + c_1c_2'x_3$ (for WIRE) and $G_2 = (c_3'c_4'x_1 + c_3'c_4x_2 + c_3c_4'x_3)(c_5'c_6'x_1 + c_5'c_6x_2 + c_5c_6'x_3)'$ (for AND1). The matching function is $M = c_1c_2'c_3'c_4c_5'c_6'$. The final solution is shown in Figure 6. The optimized network has a lower cost and is fan-out free. Notice that this replacement could not have been found with traditional methods, unless resorting to technology-independent optimizations.

From the example above we can draw some general observations. First, generalized matching is well suited for re-mapping or local optimization. Heuristics must be developed that direct the re-mapping effort on regions of a large network where improvements are required. Second, the efficiency of our procedure will improve if methods that avoid the generation of useless library cell clusters are developed.

Generalized matching involves a BDD-based consensus computation. It is well known that universal quantification of Boolean formulas may be computationally expensive. In our prototype implementation we have used standard Boolean operators as implemented in a well known BDD-package [16]. The efficiency of BDD-based Boolean manipulation strongly depends on input variable ordering. The Boolean equation of generalized matching are expressed on a common support that can be divided into two subsets: control variables $\mathbf{c}$ and input variables $\mathbf{x}$. We have investigated the best ordering between these two sets. It is very unlikely that an ordering that interleaves control variables and input variables will give good results in the general case, because of the structure of the quotient functions.

	$Order\{\mathbf{c},\mathbf{x}\}$			$Order\{\mathbf{x},\mathbf{c}\}$		
$N_\mathbf{x}/N_\mathbf{c}$	BDD_e	T_e	T_c	BDD_e	T_e	T_c
2 / 4	22	$< .01$	$< .01$	14	$< .01$	$< .01$
3 / 9	188	$< .01$	$< .01$	88	$< .01$	.01
4 / 12	763	.05	.02	192	$< .01$	.02
5 / 20	4286	.32	.07	916	$< .01$	.08
6 / 24	15693	1.3	.26	2020	$< .01$	.40
7 / 28	61506	7.2	2.1	5566	$< .01$	1.8
8 / 32	211769	34	4.3	10688	$< .01$	6.8
9 / 45	–	–	–	41110	$< .01$	34

Table 1: Performance of generalized matching applied to functions with increasing support size. Time is in seconds.

Table 1 shows the result of our test. Both the target and the library cell function were N-inputs AND gates ($N = 2, 3, ..., 9$). This function has been chosen because of the compactness of its BDD and its complete symmetry. The first column of the table contains the number of input variables versus the number of control variables. The second and fifth column contain the size of the BDD representing the Boolean formula to be universally quantified (in our case the matching formula is $G\overline{\oplus}f$) for the two orderings. The third and sixth column contain the time needed for the computation of the matching formula on a SPARCstation-IPX with 32MB of memory. The fourth and seventh column contain the time needed for the computation of the consensus on the matching formula.

Clearly, the ordering with the support variables $\mathbf{x}$ on top has superior performance, even if the time for the consensus computation is slightly smaller for the opposite ordering. Keeping the size of the matching formula as small as possible is important for two reasons. First, the formula must be completely computed before the consensus operation can be performed. Second, the formula is not fixed (it reduces to a simple XNOR only for exact matching) and it is not possible to design customized algorithm for its computation.

In contrast, the consensus on $\mathbf{x}$ is always the last step of generalized matching, and the complete knowledge of its ON-set is not always needed (only a single satisfying assignment is necessary if a simple "yes-no" answer is sought for). A specialized algorithm for the consensus computation could exploit this property to increase the efficiency.

7 Conclusions

We have described a BDD-based fully-symbolic procedure called *generalized matching*, that allows us to find all possible input assignments for which a library cell function matches a given Boolean function or relation. Generalized matching supports the simultaneous matching of more than one single-output cell (or the optimal use of multi-output cells) and allows the optimization of pin-assignment dependent cost functions.

Much work has to be done before its practical application may become competitive with state-of-the-art tools for library binding. Cells with support between 2 and 6, that constitute the largest fraction of most libraries, are matched in a time-efficient fashion. For larger cells (and clusters of cells) the performance degrades.

Many directions of improvement are under exploration. The detection of symmetries in the library cells may help in constructing quotient functions with a smaller number of control variables. Filters can be exploited to early rule out large numbers of candidate cells. A specialized algorithm is under development for the efficient computation of the consensus.

We believe that generalized matching is practically appealing as a post-processing step after traditional library binding. The availability of the full set of compatible input assignments allows a more accurate search of the minimum cost when complex cost functions are employed. Boolean Relation based optimizations offer the possibility of exploiting additional degrees of freedom for multi-output cell matching or concurrent optimization of clusters of cells during re-mapping.

8 Acknowledgements

This research is supported by NSF and ARPA under contract number MIP-9421129.

References

[1] H. Savoj, M. J. Silva, et al., "Boolean matching in logic synthesis," in *EURO-DAC, Proceedings of the European Design Automation Conference, EURO-VHDL*, pp. 168–174, Sep. 1992.

[2] U. Schlichtmann, F. Brglez and M. Hermann, "Characterization of Boolean functions for rapid matching in EPGA technology mapping," in *DAC, Proceedings of the Design Automation Conference*, pp. 374–379, June 1992.

[3] J. Mohnke and S. Malik, "Permutation and phase independent Boolean comparison," *Integration, The VLSI Journal*, pp. 109–129, Dec. 1993.

[4] U. Schlichtmann, F. Brglez and P. Schneider, "Efficient Boolean matching based on unique variable ordering," in *Proceedings of the IWLS93*, May 1993.

[5] J. R. Burch and D. E. Long, "Efficient Boolean function matching," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 408–411, Nov. 1992.

[6] K. C. Chen and J. C. Y. Yang, "Boolean matching algorithms," in *Proceedings of the International Symposium on VLSI Technology, Systems, and Applications*, pp. 44-48, May 1993.

[7] F. Mailhot and G. De Micheli, "Technology mapping using Boolean matching and Don't care sets," in *EDAC, Proceedings of the European Design Automation Conference*, pp. 212–216, Aug. 1990.

[8] G. De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, 1994.

[9] Y. T. Lay, S. Sastry and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," in *ICCD, Proceedings of the International Conference on Computer Design*, pp. 452–458, Oct. 1992.

[10] K. Keutzer, "DAGON: technology binding and local optimization by DAG matching," in *DAC, Proceedings of the Design Automation Conference*, pp. 341–347, June 1987.

[11] F. Brown. *Boolean reasoning*. Kluwer Academic Publishers, 1990.

[12] M. Damiani and G. De Micheli, "Don't care specifications in combinational and synchronous logic circuits," *IEEE Transactions on CAD/ICAS*, vol CAD-12, pp. 365–388, March 1993.

[13] R. Brayton, G. Hachtel et al., "MIS: A multiple-level logic optimization system," *IEEE Transactions on CAD/ICAS*, vol CAD-6, pp. 1062–1081, Nov. 1987.

[14] F. Somenzi, R. K. Brayton, "Minimization of Boolean relations," in *IEEE, Proceedings of the International Symposium on Circuits and Systems*, pp. 738–473, May 1989.

[15] Y. Watanabe, L. Guerra and R. K. Brayton, "Logic optimization with multi-output gates," in *DAC, Proceedings of the Design Automation Conference*, pp. 416–420, June 1993.

[16] K. Brace, R. Rudell and R. Bryant, "Efficient implementation of a BDD package," in *DAC, Proceedings of the Design Automation Conference*, pp. 40–45, June 1993.

[17] H. Savoj, R. K. Brayton and H. Touati, "Extracting local don't cares for network optimization," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 514–517, Nov. 1991.

[18] E. Detjens, G. Gannot et al., "Technology mapping in MIS," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 116–119, Nov. 1987.

[19] H. Sato, N. Takahashi et al., "Boolean technology mapping for both ECL and CMOS circuits based on permissible functions and binary decision diagrams," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 286–289, Nov. 1990.

[20] C. Y. Tsui, M. Pedram and A. Despain, "Power efficient technology mapping under an extendend power model" *IEEE Transactions on CAD/ICAS*, vol. 13, pp. 1110–1121, Sep. 1994.