4

Combined approach of ROBDDs and structural analysis in the mapping and matching of logic functions

Trullemans A.-M., Jacobi R. and Zhang Q.
UCL-Laboratoire de Microélectronique
Place du Levant, n° 3, B-1348 Louvain-la-Neuve, Belgium,
Phone: (+32) 10 47 25 65 - Fax: (+32) 10 47 86 67
e-mail: atrullemans@dice.ucl.ac.be

Abstract

The technology mapping - final step of the logic synthesis - maps the decomposed Boolean function on physical cells. We address here the decomposition and the matching steps. We present two different ROBDD-based techniques to handle the decomposition problem, and compare them. For handling the matching step, we analyse a heuristics based on symmetry and develop a new structural approach, based on controlling value analysis and observation function deduction. This last appears to be efficient regarding the CPU time for checking the match with basic cells, mostly when don't cares are present, and should be particularly interesting to handle the complex cells of FPGAs. Benchmarks are presented which validate the various heuristics.

Keywords

ROBDD, technology mapping, decomposition, matching

1 INTRODUCTION

Technology mapping is the last and decisive step in logic synthesis. It is usually performed after a technology independent minimization, which generates an optimized multi-level logic network, and a decomposition step, which splits that network into simpler subfunctions, easier to manipulate. The mapping itself consists in covering the network by implementing each node (or set of nodes) with one cell (or set of cells) selected from the target library, while trying to meet the design constraints.

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*

Gate matching is the process of selecting the library cells to implement the Boolean functions associated to the network's nodes. Network covering is the transformation of the whole Boolean network into an acceptable design by selecting which subsets of the network nodes shall be collapsed and mapped to a single cell. During the covering step, the gate matching is called several times to evaluate alternative solutions, and the mapping cost is thus heavily dependent on the matching cost.

We will concentrate on the two basic steps of decomposition and matching, and analyse innovative techniques based on ROBDD manipulation and on structural analysis of logic circuits.

Starting from a minimized description, the decomposition step will select a set of subfunctions to produce a Boolean network. We have analysed two decomposition methods based on ROBDD representations (Bryant, 1986): the direct decomposition and the path oriented decomposition. The direct decomposition is a very fast approach that extracts shared subgraphs and replaces them by new variables in the original ROBDD. The path oriented decomposition is a particular algebraic decomposition performed on ROBDDs. These two methods were tested inside the LOGOS synthesis system (Jacobi, 1993-2), on a set of benchmarks. These experiments show that, in most cases, the direct decomposition performs better than the path oriented decomposition. The results were compared to SIS (Brayton, 1987) and we find that the direct decomposition gives most of the time better results in terms of complexity of the decomposed networks.

The matching step consists in selecting in the given library the particular cell that implements a node of the Boolean network (the node function). The phase of the function, and the ordering and phase of the input variables are unknown, and this correspondence has to be established before the node function and the cell can be compared. The number of checks is directly related to the number of inputs, which greatly complicates the problem when handling technologies with complex logic cells, like field programmable gate arrays (FPGAs). The first approach proposed follows the traditional ones, which solve the problem by tree and graph matching techniques (Darringer, 1984), (Keutzer, 1987), (Aho, 1983). We reduce the complexity of the check by classifying the library cells and the functions to be mapped considering their symmetry properties. This classification method is based on the flexibility of the ROBDD structures.

However, some library cells cannot be represented as tree circuits, and this limits the applicability of the tree-based approaches. Recently, research works have focused on deriving a characteristic signature for each input variable and/or phase (Schlichtmann, 1992), (Cheng, 1993), (Mohnke, 1993). The signature method is efficient but suffers from the aliasing problem: two different input variables or input phases may produce the same signature, which leaves many variants to be checked by ROBDD comparison.

We propose here an alternative method, based on fault propagation analysis, and which is applicable to functions with don't cares: the library cell is treated as an incorrect implementation of the node function, and its structure is analysed based on the observation functions that are derived from the node function (Zhang, 1994). Using our method, incorrect input variable permutations can be easily detected, and for a correct input variable permutation, the input phase assignment can be directly determined. This reduces considerably the different ROBDDs to be checked.

2 DEFINITIONS

We define hereafter some terms used in the rest of this paper:

A *variable* is a symbol representing a single coordinate in the Boolean space.

A *literal* is a variable or its complement.

A *Boolean network* (BN) is a directed acyclic graph where each node n_i is associated to a Boolean function f_i and a literal y_i. An *edge* connecting node n_i to node n_j indicates that the variable y_i is used explicitely by the node function f_i.

The *support* of f_i is the set of variables used. A *path* in the BN is a set of successive edges that connects two nodes. The number of edges in a path p is the *length* of p. The set of nodes connected to node n_i by paths with length = 1 is the *fanin* set of n_i. The *fanout* set of n_i is the set of nodes reached from n_i by paths with length = 1.

The ROBDDs are defined as proposed by (Bryant, 1986). Each variable has an index that identifies its level in the ROBDD. A *level* l_i of a ROBDD is the set of nodes associated with the input variable x_i.

A *1-path* in a ROBDD is the set of arcs that connect the root to the 1 terminal. The set of 1-paths in a ROBDD forms a disjoint cover of the Boolean function it represents.

A *minterm* is a vertex in the n-dimensional Boolean space.

The *cofactor* of f with respect to the positive (negative) phase of an input variable x_i, denoted as $f_{xi=1}(x)$ ($f_{xi=0}(x)$), is the function obtained by setting x_i to 1 (0) in the function f.

The *Boolean difference* of a function f with respect to an input variable x_i, is the function defined as: $\partial f / \partial x_i\,(x) = f_{xi=1}(x) \oplus f_{xi=0}(x)$, where $\oplus$ is the exclusive-or operator. This function is also called the observation function of the input variable x_i, because it indicates that changes at the input x_i can be observed at the output of the function f.

An input of a logic gate is said to have a *controlling value* if its value prevents other inputs from affecting the output of the gate. It is well known that the controlling value for an AND gate is 0, the controlling value for an OR gate is 1, and no controlling value exists for inverters and EXOR gates.

3 DECOMPOSITION PROBLEM

3.1 Direct decomposition

Normally, ROBDDs are not related to the structure of a logic circuit: they are only a graph representation of a Boolean function. However, we could extract structural information from the ROBDD: each node in the diagram is the root of a subdiagram that stands for a subfunction. The *direct decomposition* consists in extracting these subfunctions from the ROBDD to find a *convenient* Boolean network representation: a decomposition adequate to the target technology (simple functions for the usual standard cells or gate arrays, functions with limited number of inputs - 4 or 5 - for lookup table FPGAs).

The *extraction* of a ROBDD subfunction starts with the selection of an interesting subdiagram, according to some objectives. This subdiagram corresponds to a node r in the ROBDD, the root of the subdiagram, and the corresponding subfunction will create a new intermediate variable y_i in the Boolean network. A new ROBDD node r_i depending on this variable is introduced in the graph, replacing the root node r. All the fathers of node r now

point to r_i. The node r_i represents the whole subfunction: its low and high sons are connected to the terminals {0,1}. In consequence, all the nodes and edges that were used only in the extracted subdiagram may be removed from the ROBDD. An example of this decomposition is shown in Figure 1.

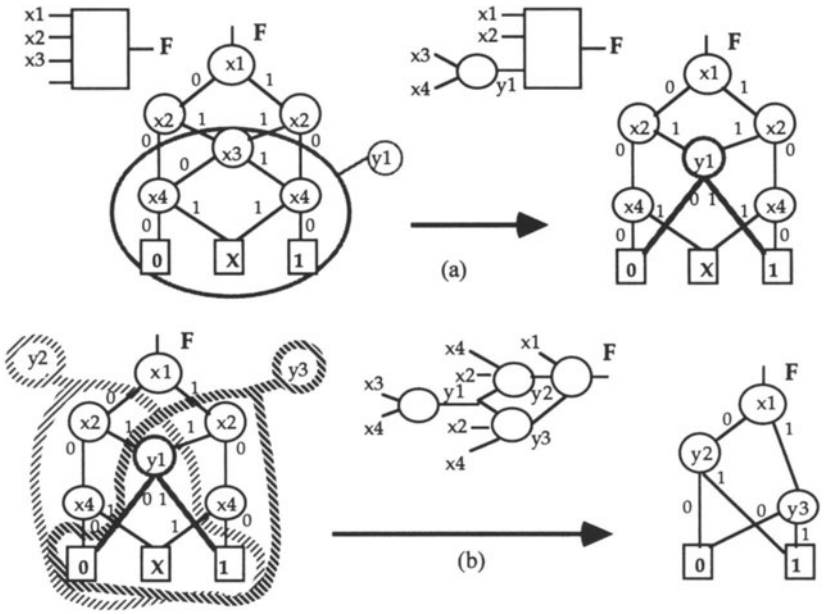

Figure 1 Example of direct decomposition of an ROBDD.

Three criteria were used to extract a subdiagram from the ROBDD:

1. Select a node with at least *NF* fathers.
2. Select a node with a subfunction depending on at least *NI* inputs.
3. Select a node whose subdiagram has a size equal or greater than *NN*.

Those values should be specified by the designer. The first criterion is related to the amount of sharing of a subfunction in the ROBDD: the number of fathers of a node n is the number of subfunctions using or depending on the subfunction associated to that node. It is a direct measure of the fanout of n. Each father of n in the ROBDD corresponds to a node that will have n in its fanin. Thus the fanin/fanout constraints may be analysed directly in the graph. The second criterion is an approximation of the function complexity measured by the number of variables it depends on. It is directly related to the complexity of the cells in the library. The third criterion provides an alternative where the size of the ROBDD of the subfunctions is used as a cost function.

3.2 Path oriented decomposition

The *path oriented decomposition* method is based on the fact that each path in a ROBDD corresponds to a cube of a disjoint cover of the function. Thus covering all the 1-paths of the ROBDD is a way of covering the function.

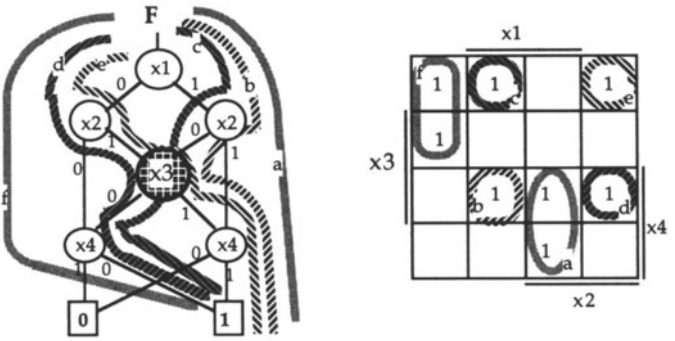

Figure 2 Path cover of an ROBDD.

To decompose the ROBDD into a set of functions, one has to select subsets of the ROBDD paths and to cover them with the extracted subfunctions. This is done by exploiting the graphical nature of the ROBDD, in order to get a faster algorithm:

1. Select a node in M, the global ROBDD, according to some cost function: n is called the *splitting node*. Let S be the set of paths through n.
2. Build QD, a new ROBDD with S. All arcs of QD that are not defined point to the 0 terminal.
3. Let $S1$ be the set of the remaining paths of M: build R, a new ROBDD with $S1$. All the undefined arcs point to the 0 terminal.
4. The sum of the ROBDDs QD and R is a cover of M:
   $M = QD + R$.

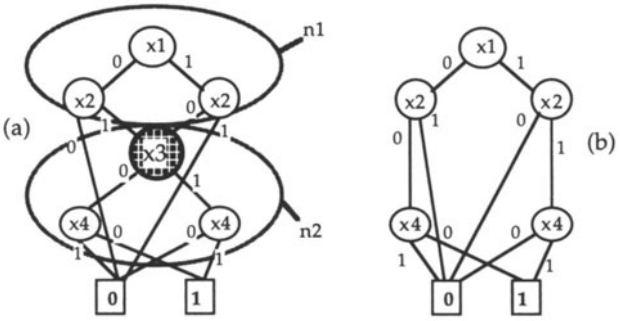

Figure 3 Path oriented decomposition: ROBDD corresponding to QD (a) and R (b).

A good criterion to select the splitting node is hard to define. The heuristic used here is to select, among the nodes that split the ROBDD in subfunctions with support of similar size, the node that covers more 1-paths. The support of the functions are thus reduced at each decomposition step, leading to simpler functions. A more detailed analysis of the splitting nodes can be found in (Jacobi, 1993-2).

3.3 Experimental results

We checked both algorithms on some experiments: the results are presented in Table 1. For each circuit, the ROBDD size is first reduced using a swapping technique (Jacobi, 1991), (Calazans, 1992), then we apply a decomposition algorithm. Direct decomposition was tested with three parameter settings : (*NF*=2, *NI*=2, 3, 4). The quality of the result is measured via a set of three numbers: (*Nodes-ROBDD-Arcs*). *Nodes* is the number of generated subfunctions (nodes of the Boolean network - corresponds to the size of the network). *ROBDD* is the total number of nodes of the ROBDD of each subfunction (complexity of each subfunction), and *Arcs* is the number of interconnections in the Boolean network (interconnection complexity). These parameters give an estimate of the complexity of the multi-level representation.

Table 1 Decomposition benchmarks - (Nodes-ROBDD-Arcs)

	Direct Decomposition			*Path Decomposition*	*Sis*
	NF=2 NI=2	*NF=2 NI=3*	*NF=2 NI=4*		
5xp1	65-276-162	52-237-143	**38-187-119**	87-427-236	42-244-120
alu3	55-231-135	35-168-106	**28-142-95**	49-244-138	**28**-192-145
bw	102-420-279	90-382-266	**63-276**-227	228-1040-568	65-400-**216**
dekoder	17-61-39	15-55-37	12-**42**-34	**7-47-25**	10-55-30
dk27	22-74-49	15-51-40	13-**43**-35	16-73-41	**11**-55-**30**
f51m	62-275-163	57-260-153	**37-197-117**	116-518-278	42-243-119
fo2	15-60-36	11-48-31	8-35-24	**4-30-16**	12-56-32
misex1	34-148-92	32-139-90	**18-91**-68	48-229-130	**18**-118-**67**
misex2	86-322-184	54-223-153	42-**181**-136	44-276-185	**30**-190-**113**
r53	21-95-55	19-89-53	**14-69-41**	19-101-53	**14**-85-**41**
r73	41-193-113	39-187-111	31-**159**-96	46-286-153	**28**-193-**93**
r84	57-265-155	54-256-152	44-223-137	**26-219-102**	37-337-140
z4	24-128-60	12-73-36	**8-49-26**	12-80-40	12-64-30

We give in the last column the results obtained with SIS (Brayton, 1987). The best results are indicated in boldface character. Direct Decomposition with *NI*=4 gives the best result in most of the cases. Sometimes Path Oriented gives the best one, but it may also produce very bad solutions (bw and f51m). These algorithms appear to be fast (time range from a few seconds to at most a few minutes, implemented in LISP on a personal computer), so we propose to apply both and select the best solution. The results are slightly better than SIS in

terms of the complexity of a Boolean network where each node function is represented by a ROBDD. We have supposed that all subfunction's ROBDDs have the same variable ordering.

The time complexity of the direct decomposition method (Jacobi, 1993-2) is linear with respect to the ROBDD size. The path oriented decomposition method is more complex: every node of the subfunctions is checked for redundancy, which is done by checking for implication between its son's functions. This gives a global time complexity bound by $O(n^2)$, where n is the ROBDD size. This is to be compared to the Boolean methods involving logic minimization, which is a NP-complete problem.

The ROBDD extensions proposed in the literature, namely the strong canonical form and the negative edges, would be usefully used in these direct and path oriented decomposition methods. The negative edges allows a fast verification of the existence and use of the complement of the extracted subfunctions in both methods (this is implemented here by an equivalence checking algorithm). The strong canonical form will present interest in the path oriented decomposition, for it keeps the global ROBDD always updated.

4 MATCHING PROBLEM

4.1 Defining the matching problem

Let $f(x_1, \ldots, x_n)$ be the function of a node in the Boolean network to be mapped, and $g(y_1, \ldots, y_n)$ the function of a cell in the library. For simplicity, we assume that f is a completely specified function with one output.

The matching problem between the two functions f and g can be stated as follows: Does there exist an input variable permutation π, an input phase assignment ϕ and an output phase assignment θ, such that $f(\mathbf{x}) = \theta\{ g[\pi\phi(\mathbf{y})]\}$. The number of variants to be considered are: $n!$ * 2^n * 2 .

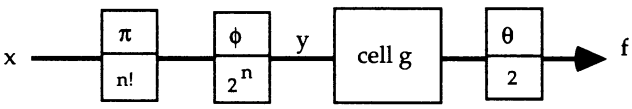

Figure 4 The matching problem: input and output transformations.

4.2 Exploiting symmetry

4.2.1 Symmetry classification

The BDD structure, i.e., the BDD graph without node values and with unlabelled arcs, represents a family of functions. This is exemplified in Figure 5.

Considering the ROBDD representation, the matching of a node function with a cell can be realized by first finding two isomorphic ROBDD structures and then extracting from them the variables and output phases. The isomorphism between the graphs can be discovered by classifying the ROBDDs according to their symmetry classes. If *BDDf* is the ROBDD of the node function to be matched, the ROBDD matching steps are:

1. Identify the symmetry classes (SymCs) of the *BDDf*.

2. Sort the classes in the descending order of cardinality.
3. Match the node function ROBDD with the cells that have the same number of SymCs and the same SymCs cardinality.

This algorithm is based on the fact that two functions f and g can match only if they have the same number of symmetry classes and if for each symmetry class of f there is a corresponding symmetry class in g with the same cardinality.

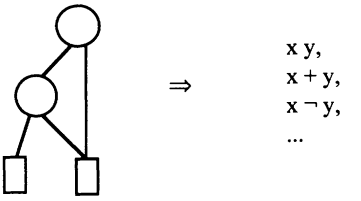

Figure 5 BDD structure and its family of functions.

To find the symmetry classes of a function, we just have to check the ROBDD of the function: two variables are symmetric in a ROBDD if they can be exchanged without modifying the topology of the graph. If we already have one ROBDD of the function, we could simply apply incremental techniques in order to avoid the cost of building a new ROBDD. If we exchange two adjacent variables we can easily check if the ROBDD will be modified or not. The next theorem allows to check this without exchanging the variables (Jacobi, 1993-2):

Theorem 1 Let S be the set of subgraphs associated to two adjacent variables in the ROBDD ordering (x_i, x_{i+1}) which have at least 2 nodes. Each subgraph has one root and can lead to 2, 3 or 4 different nodes in the ROBDD. x_i and x_{i+1} are *phase free* symmetric either if all successor in S lead to only 2 nodes in the ROBDD or if they lead to three nodes: two nodes are associated to paths $\neg x_i \neg x_j$ and $x_i x_j$ in the graph and the third one is associated to paths $\neg x_i x_j$ and $\neg x_j x_i$.

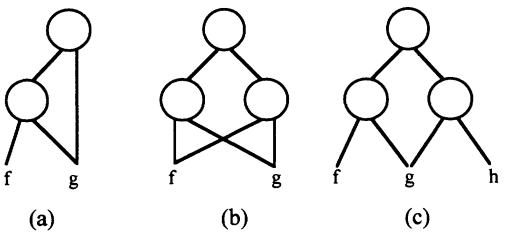

Figure 6 Examples of adjacent variables subgraphs for check of phase free symmetry.

This idea is illustrated in Figure 6. Subgraphs (a) and (b) are phase free symmetric with only two different nodes, while (c) is phase free symmetric with three different ones.

The algorithm to generate the phase free symmetry classes uses the incremental techniques proposed in (Jacobi, 1991) to swap adjacent variables in the ROBDD ordering. The number of steps is upper bounded by $n(n\text{-}1)/2$. Usually there are less comparisons due to the transitive property of symmetric variables: if one variable belongs to a SymCs it does not need to be further compared. After all variables are pairwise tested, the ROBDD is classified by putting the SymCs with larger cardinality closer to the root.

4.2.2 Matching algorithm

```
function MatchBDD (mbd, order): matchlist;
begin
        (mbd,order) = ClassBDD (mbd, order);
        vector = GetSCsVector(mbd);  /* Symmetry class vector */
        G = GetCells(vector);
        foreach cell in G
          if (match = MatchCell(mbd, G.mbd, order))
          then put match in matchlist; /* match = cell and inverters needed */
        return (matchlist);
end;
function MatchCcell (bddf, bddg, order): match;
begin
        initialize PhaseVector;
        if (RecMatchCell (bddf, bddg, PhaseVector))
          then return (BuildMatch ( order, PhaseVector))
          else return (NULL);
end;
function RecMatchCcell (bddf, bddg, phv)
begin
        case (bddf,bddg) of
          (they have different indices): return(false);
          (both are terminals):
              if terminals are distinct then invert the output;
              return(true);
          (only one is a terminal): return (false);
          ((RecMatchCell (lowv (bddf), lowv (bddg), phv) and
              RecMatchCcell (highv (bddf), highv (bddg) phv)):
              return(true);
          ((RecMatchCell (lowv (bddf), highv (bddg), phv) and
              RecMatchCcell (highv (bddf), lowv (bddg) phv)):
              phv[index(bddf)] = NEGATED;
              return(true);
end;
```

Figure 7 Matching algorithm based on symmetry classes.

Figure 7 presents the matching algorithm. The function *MatchBDD* takes a ROBDD (mbd) with its variable order, and returns the set of matches that can implement the function. Each

match is formed by the cell and the set of inputs/output inverters generated by the matching algorithm. The function is first classified according to its symmetry classes. Then the SymCs cardinality vector is used to address the set of cells in the library that are candidates to match the function. The function *MatchCell* takes the ROBDD of the function (bddf) and of the cell (bddg), and returns the input assignment with the respective input and output phases, or NULL if they are not matchable. Function *RecMatchCell* checks the matching by traversing both ROBDDs with direct or complemented match at each node.

4.3 Structural analysis

The controlling value analysis and observation function deduction method was first proposed for the location of multiple design errors in an incorrect gate-level circuit (Zhang, 1994). It can also be used for checking the match between a node function f and a library cell g. In this section, we first give a brief review of this method, and then show its application to the matching problem.

4.3.1 Controlling value analysis and observation function deduction

Consider the required function $f(\mathbf{x})$, and its proposed - and maybe incorrect - implementation by the circuit C, which realizes the function $g(\mathbf{x})$. For simplicity, we assume that the circuit C is exclusively composed of inverters and AND gates. To locate the design errors within the circuit C, its primary inputs are first initialized with the corresponding observation functions derived from the function $f(\mathbf{x})$. This is based on the following theorem (Zhang, 1994).

Theorem 2 The function $g(\mathbf{x})$ is said to be equivalent to the function $f(\mathbf{x})$, or equivalent to its complement if, and only if $\partial g / \partial x_i (\mathbf{x}) = \partial f / \partial x_i (\mathbf{x})$ for any input x_i.

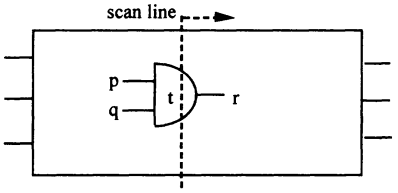

Figure 8 A logic gate t in the incorrect circuit C.

The detection of the possible design errors associated with the logic gate t in the incorrect circuit C, can be derived from the controlling value analysis. We assume that the observation functions of the gate inputs p and q with respect to the required function $f(\mathbf{x})$ are known. We present here two theorems adapted from (Zhang, 1994).

Theorem 3 The gate inputs p and q are the correct inputs of the logic gate t if, and only if
$\partial f / \partial p (\mathbf{x}) \cdot q(\mathbf{x}) = 0$ or $\partial f / \partial p (\mathbf{x}) \cdot \neg q (\mathbf{x}) = 0$,
and $\partial f / \partial q (\mathbf{x}) \cdot p(\mathbf{x}) = 0$ or $\partial f / \partial q (\mathbf{x}) \cdot \neg p(\mathbf{x}) = 0$.

Theorem 4 If $\partial f / \partial p\,(\mathbf{x}) \cdot q(\mathbf{x}) = 0$, the input q has the controlling value 1 for the gate t, otherwise it has the controlling value 0.

If the observation functions of the inputs of a logic gate t cannot satisfy Theorem 3, an interconnection design error is detected. Furthermore, if a gate input is proved to have the controlling value 1 for an AND gate, a missing inverted error at the signal line is detected.

If, for a given gate t with inputs p, q and output r, there is no design error or all design errors have been corrected (for example by adding inverters), the observation function of the gate output r can be deduced from that of the gate inputs p, q according to the following theorem.

Theorem 5 $\partial f / \partial r\,(\mathbf{x}) = \partial f / \partial p\,(\mathbf{x}) + \partial f / \partial q\,(\mathbf{x})$.

4.3.2 Application to the matching problem

In the problem of matching a node function f with a library cell g, most of the input variables, at least for many practical functions (and if the function has no don't cares), can be identified by characteristic signatures (Schlichtmann, 1992), (Cheng, 1993), (Mohnke, 1993) and symmetric variables (Keutzer, 1987), (Mailhot, 1993), (Jacobi, 1993-1). Thus, the total number of possible input variable permutations and the total number of possible input phase assignments have already been reduced, but there may remain ambiguity for some variables (aliases). We will explain the strategy of the structural analysis on the example of Figure 9, where we have to match the node function f (represented here for simplicity by its Karnaugh map) with some cell g, which will be analysed as an equivalent netlist of elementary gates.

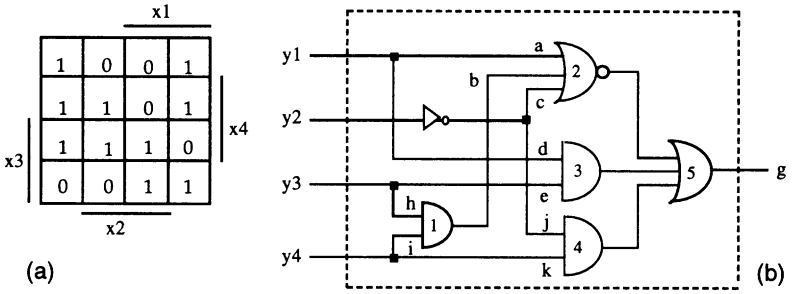

Figure 9 Example of match: (a) Karnaugh map of function f. (b) netlist equivalent of library cell g.

Table 2 The possible input variable permutations and phases

	Input variable permutation π				*Input phase assignment* ϕ			
No.	*y1*	*y2*	*y3*	*y4*	*y1*	*y2*	*y3*	*y4*
1	x1	x3	x2	x4	?	?	1	0
2	x1	x3	x4	x2	?	?	0	1
3	x3	x1	x2	x4	?	?	1	0
4	x3	x1	x4	x2	?	?	0	1

In this particular example, after checking signatures and symmetry, it remains only 4 cases to be checked (Table 2).

For each of these input variable permutations, the inputs of the library cell g are initialized with the observation functions derived from the node function f. The design errors considered here for the library cell g are only wire exchanges at the inputs and missing inverters at the inputs and/or at the output. Based on these observation functions, the current input variable permutation π is first verified according to Theorem 3, then the input phase assignment ϕ is verified according to Theorem 4.

With respect to the input variables of the node function f, the observation functions can be computed as:

$$\partial f / \partial x1\ (\mathbf{x}) = x3\ \neg x4 + x2\ \neg x3\ x4 + \neg x2\ x3. \tag{1}$$

$$\partial f / \partial x2\ (\mathbf{x}) = \neg x3\ \neg x4 + x1\ x4. \tag{2}$$

$$\partial f / \partial x3\ (\mathbf{x}) = x1\ x4 + x1\ x2 + \neg x1\ \neg x2\ \neg x4. \tag{3}$$

$$\partial f / \partial x4\ (\mathbf{x}) = \neg x1\ x2 + \neg x2\ x3. \tag{4}$$

Note that the observation function $\partial f / \partial x_i(\mathbf{x})$ contains the input combinations on which the input x_i can be observed, or in other words, the stuck-at-faults at the input x_i that can be propagated. These observation functions will be used to check the input variable permutation, and also to determine the input phase assignment.

For each of the possible input variable permutations, the corresponding inputs of the library cell g are initialized with these observation functions.

Let us consider the variable permutation $\pi(y1, y2, y3, y4) = (x1, x3, x4, x2)$ as shown in Figure 10. For this variable permutation, the input phase assignment is:

$$\phi(y1, y2, y3, y4) = (?, ?, 0, 1). \tag{5}$$

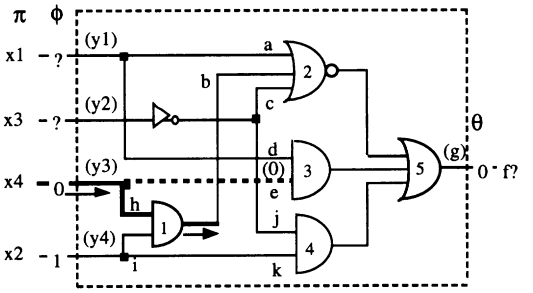

Figure 10 Checking the input variable permutation $\pi(y1, y2, y3, y4) = (x1, x3, x4, x2)$.

According to the structure of the library cell g, the function $[\partial f / \partial x4\ (\mathbf{x}) \cdot \neg d\ (\mathbf{x})]$ can be used as partial observation function of the signal line h. If $\phi(y1) = 0$, the observation function of the signal line h is: $\partial f / \partial h\ (\mathbf{x}) = \neg x1\ x2 + \neg x1\ \neg x2\ x3$. It indicates that the signal line h

can be observed in spite of the logic values at the input $x2$, which is contradictory to the fact that the gate 1 is an AND gate. If $\phi(y1) = 1$, we have $\partial f / \partial h\,(\mathbf{x}) = x1 \neg x2\, x3$. This function does satisfy Theorem 3. Thus, the input phase of the variable $y1$, $\phi(y1) = 1$, is determined according to Theorem 4. Similarly, for the signal line i, we have $\partial f / \partial i\,(\mathbf{x}) = x1x3x4$ if $\phi(y2) = 0$. This observation function also satisfies Theorem 3, and the input phase of the variable $y2$, $\phi(y2) = 0$, is determined.

According to Theorem 5, the observation function of the signal line b can be deduced as: $\partial f / \partial b\,(\mathbf{x}) = \partial f / \partial i\,(\mathbf{x}) + \partial f / \partial h\,(\mathbf{x})$. The same analysis process continues for the remaining logic gates. In fact, not all logic gates in the library cell need to be analysed because the possible design errors occur only at the primary inputs. In our method, only the logic gates whose inputs contain the primary inputs are considered. If no conflicts have been found, the ROBDDs for the subcircuit f and for the library cell g are then constructed for equivalence checking.

In our example, for the variable permutation $\pi(y1, y2, y3, y4) = (x1, x3, x4, x2)$, the input phase assignment $\phi(y1, y2, y3, y4) = (1, 0, 0, 1)$ and the output phase assignment $\theta = 0$, the ROBDDs of the two functions f and g are isomorphic. Thus, the library cell g is a match of the node function f.

4.3.3 Experimental results

Table 3 Check of matchable cells

Library cell		*complete ROBDD match*		*structural analysis*	
Cell name	*nb inputs*	*BCT*	*CPU (sec)*	*BCT*	*CPU (sec)*
AO04D1	3	36	0.23	1	0.00
FN02D1	3	1	0.02	1	0.00
OA04D1	3	36	0.25	1	0.02
AO01D1	4	12	0.03	1	0.03
AO5D1	4	236	1.60	1	0.00
AO08D1	4	347	2.38	1	0.00
FN03D1	4	49	0.37	1	0.02
FN04D1	4	49	0.35	1	0.00
OA01D1	4	58	0.42	1	0.02
OA05D1	4	236	1.65	1	0.00
OA08D1	4	340	2.35	1	0.03
AO03D1	5	3084	23.28	1	0.00
AO07D1	5	2908	21.75	1	0.05
OA03D1	5	3457	25.82	1	0.03
OA07D1	5	2641	19.23	1	0.03
AO02D1	6	3892	30.77	1	0.00
AO06D1	6	4441	3.52	1	0.03
OA02D1	6	4234	22.92	1	0.03
OA06D1	6	3892	31.25	1	0.03
ex-A07	7	556852	6019.85	1	0.02

We implemented a simple ROBDD check and the structural check for basic cells in a prototype written in C++. We checked the match with standard cells of an industrial library (library VSC450L offered in the COMPASS CAD tools), composed mostly of simple cells.

The node function to be mapped is defined as the library function, with the addition of 5% random don't cares. In these tests, of course, signature can not be used, for it does not apply if don't cares are present. In Table 3, we compared the number of ROBDD checks (BCT) to be handled to confirm the correspondence between the function and the cell, and the CPU time (in seconds) on a SUN Sparc20 workstation. The last one (ex-A07) is a test on some more complex cell artificially constructed, to verify how the algorithm handles the complexity.

Table 4 Check of non matchable cells

Library cell		*complete ROBDD match*		*structural analysis*	
Cell name	*nb inputs*	*BCT*	*CPU (sec)*	*BCT*	*CPU (sec)*
AO04D1	3	48	0.28	0	0.00
FN02D1	3	48	0.35	6	0.03
OA04D1	3	48	0.32	0	0.02
AO01D1	4	384	2.58	0	0.02
AO5D1	4	384	2.55	0	0.02
AO08D1	4	384	2.67	0	0.02
FN03D1	4	384	2.57	0	0.02
FN04D1	4	384	2.68	0	0.00
OA01D1	4	384	2.57	0	0.02
OA05D1	4	384	2.67	0	0.02
OA08D1	4	384	2.63	0	0.02
AO03D1	5	3840	28.93	0	0.03
AO07D1	5	3840	28.02	0	0.08
OA03D1	5	3840	28.60	0	0.03
OA07D1	5	3840	27.95	0	0.05
AO02D1	6	46080	370.60	0	0.15
AO06D1	6	46080	369.23	0	0.03
OA02D1	6	46080	365.30	0	0.25
OA06D1	6	46080	369.88	0	0.10
ex-A07	7	645120	5763.65	0	0.37

In Table 4, we introduced a modification in the function, so that the cell could not be a match, and of course the ROBDD checks are to be considered for all the possible permutations and phases.

In every case, the structural analysis appears to be very rapid. It handles very well the complexity.

5 CONCLUSION

The two decomposition methods presented here produce multi-level networks from a ROBDD description. The direct decomposition is very fast and produces good results. Of course, to be effective it needs a previous reordering to reduce the ROBDD size. It is easy to tune the complexity of the node functions (support size of the function). Fine granularity decomposition may be obtained by restricting the support size to two variables, for example. This produces a larger network, but with simpler node functions. Reducing the amount of sharing of a subfunction will also reduce the network size, by identifying subfunctions that

are used several times in the multi-level representation. It should be used concurrently with the support size, as an alternative cost function. The path oriented decomposition has not yet produced interesting results : it has to be improved by a deeper analysis of the splitting node selection. In conclusion, the direct decomposition seems particularly interesting for the synthesis on multiplexor-based cells, for instance. Current research includes the application of this algorithm in the logic synthesis of MUX-based FPGAs, and the development of new Boolean decomposition techniques based on partial symmetry detection in ROBDDs.

We have analysed various methods to solve the matching problem. Characteristic signatures and symmetric variables are really efficient to reduce the search space of the matching problem. But it may leave ambiguities between some input variables and for some phases. Moreover, these techniques are not applicable in the presence of don't cares. Using the new structural analysis proposed here, incorrect variable permutations can be easily checked, and the input phase assignment can be directly determined. Compared with the ROBDD based techniques (Mailhot, 1993), the time needed to compute the ROBDDs for the library cell with all the possible input variable permutations and input phase assignments, can be saved.

This structural matching technique will be implemented in a complete technology mapper, along with the proposed decomposition method, and tested on realistic examples, mapping on standard cells and on FPGAs.

6 REFERENCES

Aho, A., Hopcroft, J. and Ullman, J. (1983) *Data Structures and Algorithms*. Addison-Wesley, Reading, Massachusetts.

Brayton, R. K., Rudell, R., Sangiovanni-Vincentelli, A. and Wang, A. (1987) MIS: A multiple-Level Logic Optimization System. *IEEE Trans. on Computer-Aided Design*, **CAD-6**, no 6.

Bryant, R.E. (1986) Graph-Based Algorithm for Boolean Function Manipulation. *IEEE Trans. on Computers*, **C-35**, no 8.

Burgun, L. (1993) Multi-level Synthesis Based on Binary Decision Diagrams. *VIII SBMicro*, Campinas.

Calazans, N., Zhang, Q., Jacobi, R., Yernaux, B. and Trullemans, A-M. (1992) Advanced Ordering and Manipulation Techniques for Binary Decision Diagrams. *Proceedings EDAC*, Bruxelles.

Cheng, D. I. et al (1993) Verifying equivalence of functions with unknown input correspondence. *EDAC/EURO-ASIC*, Paris.

Darringer, J. A. et al. (1984) LSS: A System for Production Logic Synthesis. *IBM Journal of Research and Development*, **28**, no 5.

Jacobi, R., Calazans, N. and Trullemans, C. (1991) Incremental Reduction of Binary Decision Diagrams. *Proceedings of International Symposium on Circuits and Systems*, Singapore.

Jacobi, R. et al (1993-1) Boolean mapping with binary decision diagrams. *IFIP Workshop on Logic and Architecture Synthesis*, Grenoble.

Jacobi, R. (1993-2) *A Study of the Application of Binary Decision Diagrams on Multi-level Logic Synthesis*. Doctoral Thesis, Université Catholique de Louvain, Belgium.

Keutzer, K. (1987). DAGON: Technology Binding and Local Optimization by DAG Matching. *24th ACM/IEEE Design Automation Conference*.

Mailhot, F. et al (1993) Algorithms for technology mapping based on binary decision diagrams and on Boolean operations. *IEEE Trans. on Computer-Aided Design*, **CAD-12**, no 5.

Mohnke, J. et al (1993) Permutation and phase independent Boolean comparison. *EDAC/EURO-ASIC*, Paris.

Schlichtmann, U. et al (1992) Characterization of Boolean functions for rapid matching in EPGA technology mapping. *29th DAC.*

Zhang, Q. (1994) *Logic verification and design error diagnosis for combinational circuits.* Ph.D. Dissertation, Université Catholique de Louvain, Belgium.

7 BIOGRAPHY

Anne-Marie Trullemans-Anckaert was born in Brussels, Belgium, in 1944. She received the engineering degree and the Doctorat en Sciences Appliquées from the Université Catholique de Louvain (UCL), Belgium, in 1967 and 1974, respectively. From 1967 she undertook research, at the Laboratoire de Microélectronique of UCL, in circuit simulators, symbolic layout editors and digital CAD tools, and was involved in the teaching of microelectronics. She is now Associated Professor at UCL. Her interest is in the development of CAD tools for the simulation and the design of MOS VLSI. Recently, she concentrate her research on digital circuits synthesis tools from HDL.

Ricardo Pezzuol Jacobi received his B.S. degree in electrical engineering and his M.S. degree in Computer Science at the Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in Dec. 82 and Aug. 86, respectively. He received his Doctoral degree from Université Catholique de Louvain, Belgium, in Dec. 93. His research interest includes logic synthesis for FPGAs, layout oriented technology mapping and software and hardware codesign. He is an adjoin professor of the Informatics Institute at the UFRGS since Dec. 1989.

Qinhai Zhang received the B.Sc. degree in physics and the M.Sc. degree in electrical engineering, both from Fudan University, Shanghai, China. In 1995, he received the Ph.D. degree from the Université Catholique de Louvain, Belgium. Currently he is a research assistant at the Université Catholique de Louvain, Belgium. He was an assistant from 1985 to 1987, and a lecturer from 1988 to 1990, in the E.E. department at Fudan University, Shanghai, China. In 1990, he joined the Electronics Department of Université Catholique de Louvain. His research interests include logic synthesis, formal verification, design error diagnosis and circuit layout.