Improved Technology Mapping Using A New Approach to Boolean Matching

Bhanu Kapoor
Integrated Systems Laboratory, Texas Instruments, Inc.
P. O. Box 655474, MS 446, Dallas, TX 75265
email: kapoor@hc.ti.com

Abstract

We present an improved method for technology mapping using a new approach to the Boolean matching problem. Signatures computed over OBDDs using a set of specific probability values detemine mateches between library cells and portions of the netlist. Unlike some previous methods, which may require creation of up to $O(n!)$ OBDDs for all possible permutations of module's inputs, our method requires exactly one OBDD to be created for the portion of the netlist being matched. Some results obtained on ISCAS85 benchmark circuits suggest the viability and validity of our approach.

I. Introduction

Logic synthesis has been shown to be an effective means of designing logic circuits. The computer-aided synthesis of a logic circuit involves two major steps: the optimization of a technology-independent logic [2], and technology mapping.

Technology mapping is the process of implementing a set of Boolean equationsby selecting logic gates from a library of available gates. Technology mapping is an important step in the synthesis process because the quality of the design, in terms of area, performance, and power dissipation of the circuit, depends heavily on this step.

The two operations intrinsic to technology mapping, matching and covering, are computationally difficult. For this reason, several approaches to technology mapping have been pursued and implemented in research and commercial mapping tools. Rule-based technology mappers [6, 9] and heuristic-based algorithms [7, 10, 12, 15] have been proposed.

Keutzer [10] proposed to represent library functions by trees. Fast tree matching algorithms were used to determine structural matchings. However, two functions may be structurally different but functionally equivalent. As a result, Boolean matching can provide better optimization.

The issue of Boolean matching has been addressed by several researchers [4, 5, 8, 11, 13, 14, 16]. Technology mapping tool Ceres [11] uses ordered binary decision diagrams (OBDDs) [3] for the matching purpose. It has been pointed out that in the worst case up to $O(n! \cdot 2^n)$ different ordered BDDs may be required for each match. Some heuristics based on symmetry of logic functions are developed to reduce the number of OBDDs required for the matching process.

A generalization of BDDs, for matching purpose during EPGA technology mapping, has also been used [8]. Another approach to Boolean matching [14] efficiently prunes the search space by combining shared OBDDs with unique ordering of variables.

In this paper, we describe a matching algorithm which requires only one OBDD to be created and allows deterministic matching. A set of probability values [1] are used to compute unique signatures for Boolean functions. These signatures are then used to find matchings between portions of a netlist and the elements of a given library. Once the signature of a portion of the subject graph being mapped is computed, a possible match is determined in $O(1)$ time. We also suggest some techniques to reduce the number of subgraphs examined at any node of the subject graph.

II. Preliminaries

Boolean equivalence of functions represented as *free graphs* can be determined exactly by using a specific set of probabilities. In [1], it has been shown that a specific set of probabilities can be associated with the n variables of a graph G (namely $|v_1|= \frac{1}{2^{2^0}+1}$, $|v_2|= \frac{1}{2^{2^1}+1}$, . . ., $|v_n|= \frac{1}{2^{2^{n-1}}+1}$) so that $|G(v_1, \ldots, v_n)| = \frac{x}{(2^{2^n}-1)}$, where x is an integer in the interval $0 \le x \le (2^{2^n}-1)$. There is a unique correspondence between integers x in the given interval and the 2^{2^n} Boolean functions of n variables. OBDDs [3] satisfy the properties of a free graph and can be used to compute the signature, $S_G(V) =|G(v_1, ..., v_n)|$, of a Boolean function, where V denotes the set of values assigned to variables v_1 through v_n.

2.1 Computing Signatures Over OBDDs

The computation of signature of a Boolean function is similar to that of function density [3] using the probability values for the variables in the Boolean function. This computation is recursive in nature and is defined as follows:

$$S_1(V) = 1 \quad (1)$$

$$S_0(V) = 0 \quad (2)$$

$$S_G(V) = S_x(V) S_{G_{x\leftarrow 1}}(V) + (1 - S_x(V)) S_{G_{x\leftarrow 0}}(V) \quad (3)$$

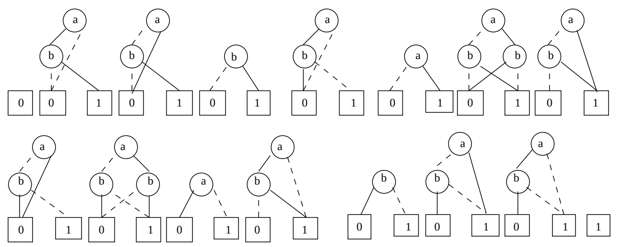

Figure 1: OBDDs for all two-input functions.

Thus, given an OBDD representation of G, we can compute the signature by traversing the graph depth first, labeling each vertex by the signature of the subgraph rooted at that vertex. Fig. 1 shows all the functions of two variables with their OBDDs sorted by increasing values of their signatures. The signatures range from $\frac{0}{15}$ to $\frac{15}{15}$, for the assignment $a = \frac{1}{3}$ and $b = \frac{1}{5}$. Fig. 2 shows an OBDD of the function $f = a+bc$, and its signature using the probability assignment $a = \frac{1}{3}$, $b = \frac{1}{5}$, and $c = \frac{1}{17}$. It should be noted that the signature is unique only for a given permutation of these assignments.

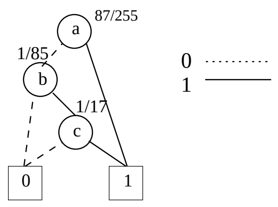

Figure 2: Computing signature for $f = a + bc$

2.2 Outline

Before getting into the details of various steps involved in the technology mapping process described here, a brief description of the methodology is outlined here to help clear up some of the key points regarding this approach.

As the first step in the process, all possible signatures for the cells in a given library will be determined. A hash table will be created using signature as the *key* and a library cell with the particular variable assignment as the *hash value.*

During the mapping process, a subgraph of a given subject graph will be selected to see if it can be implemented using one of the library cells. If the subgraph has n variables, then the signature of the subgraph will be computed by randomly assigning the values to the variables from the set $(\frac{1}{2^{2^0}+1}, \frac{1}{2^{2^1}+1}, \ldots, \frac{1}{2^{2^{n-1}}+1})$.

An important result regarding these signatures guarantees states that two Boolean function with identical signatures can always be matched. This result forms a cornerstone of this methodology. Using this result, a matching, if any, is established between the subgraph and the library cells. As a result, once the signature of the subgraph is computed, the matching itself can be established in $O(1)$ time.

III. Computing Library Signatures

For optimal results, it is required that all possible signatures be precomputed for each library cell. For a library cell with n inputs, different permutations of assignment from the set $(\frac{1}{2^{2^0}+1}, \frac{1}{2^{2^1}+1}, \ldots, \frac{1}{2^{2^{n-1}}+1})$ may result in multiple signatures. Because of the logic symmetry found in most of the library cells, each cell requires only a few signatures. For example, the MCNC `lib2` benchmark library has only 140 distinct signatures.

Lemma 1 *An n-input AND/OR/NAND/NOR gate has exactly one signature.* □

An interesting comparison can be made with the number of tree patterns required to represent these gates in the case of tree-matching based systems such as [2, 10]. The required number of tree-patterns has been studied in detail in [12]. It is shown that the number grows quickly with increasing value of n. For example, 8-input AND gates require 23 tree patterns and 16-input AND gates require 10905 tree patterns. In comparison, each n-input has exactly one signature.

Lemma 2 *An n-input OAI/AOI$k_1k_2...k_m$ gate, $n = k_1 + k_2 + ... + k_m$, has*

$$\frac{n!}{(k_1!k_2!...k_n!)(\prod_{i=1}^{max(k_1,k_2,...,k_m)}(S_i!))}$$

distinct signatures, where S_i's are the cardinalities of the symmetry classes. □

Typically, number of tree patterns required are smaller than the number of signatures in the case of OAI/AOI gates. However, fairly small number of signatures are needed to represent typical OAI/AOI gates found in the libraries. Table 2 shows the number of signatures for some of the common OAI/AOIk_1k_2 gates.

Table 2: AOI Signatures

(k_1, k_2)	Signatures
(1, 2)	3
(1, 3)	4
(3, 3)	10
(3, 4)	10

Boolean matching addresses the problem of determining the equivalence of two Boolean functions regardless of the structure of their representation and under an arbitrary permutation of their inputs. Given two Boolean functions $f(x_1, x_2, ..., x_n)$ and $g(y_1, y_2, ..., y_n)$, if there exists a permutation $P(X) : \{x_1, x_2, ..., x_n\} \rightarrow \{y_1, y_2, ..., y_n\}$ such that $f(x_1, x_2, ..., x_n) = g(y_1, y_2, ..., y_n)$ then f and g are said to be matching functions.

Earlier we made a claim that Boolean functions with equal signatures can be matched. This is a key result for this approach. In the following text, we prove this result.

Theorem 1 *There exists a specific set V of probabilities (namely $|v_1| = \frac{1}{2^{2^0}+1}$, $|v_2| = \frac{1}{2^{2^1}+1}$, . . ., $|v_n| = \frac{1}{2^{2^n}+1}$) associated with n variables of a Boolean function F, and $\hat{V}$, a permutation of V, associated with n variables of a Boolean function G such that $S_F(V) = S_G(\hat{V}) \Rightarrow$ F and G are matching functions.*

Proof: Let the n variables that F and G depend on are $(x_1, x_2, ..., x_n)$. The set of probabilities V is given by:

$$x_1 = |v_1|, x_2 = |v_2|, ..., x_i = |v_i|, ..., x_j = |v_j|, ..., x_n = |v_n|$$

Consider a $\hat{V}$ in which values of x_i and x_j are swapped. The same argument holds for any number of swaps. The set of probabilities $\hat{V}$ is given by:

$$x_1 = |v_1|, x_2 = |v_2|, ..., x_i = |v_j|, ..., x_j = |v_i|, ..., x_n = |v_n|$$

Consider a function F_θ, obtained from F by swapping variables x_i and x_j. The signature of F_θ under the assignment $\hat{V}$ will be exactly the same as the signature of F under the assignment V. We get, $S_F(V) = S_{F_\theta}(\hat{V})$. We are given that $S_F(V) = S_G(\hat{V})$. Hence, $S_{F_\theta}(\hat{V}) = S_G(\hat{V})$. This implies [1], F_θ and G are equivalent functions. F_θ has been obtained from F using a permutation of variables. Therefore, F can be obtained from G using a permutation of variables. Hence, F and G are matching functions. $\Box$

The pin matchings can be determined by the variables with identical values in F and G. Thus given all possible signatures of the cells in the library, a matching can be determined by creating just one OBDD and computing the signature using a random assignment of these probabilities to the variables in the OBDD.

In order to consider the different phases for the inputs of the subgraph under consideration, the subject graph is modified with the *inverter-pair* heuristic. Each node of the subgraph can be implemented in either positive or negative phase. The subgraph is modified using *inverter-pair heuristic* [12].

Redundant inverter pairs are added to all those edges in the subject graph which connect two two-input gates. A pair is also added to those primary inputs and outputs which are available in only one phase. As a result, each node in the subject graph is available in both phases. The library is augmented by a cell which implements a pair of inverters. This cell is assigned a zero area and zero delay cost. Fig. 3 shows a subject tree modified using inverter pair heuristic. Three inverter pairs have been added.

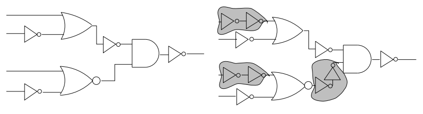

Figure 3: Modification using inverter-pair heuristic

3.1 An Example of Technology Mapping Process

Fig. 4 shows a subject graph on which the inverter-pair heuristic has already been applied. It should be noted that any two-input gate can be used in the subject tree. Also, it is not necessary to use two-input gates, however, smaller gates allow more choices for the mapping process. This results in better optimized circuits.

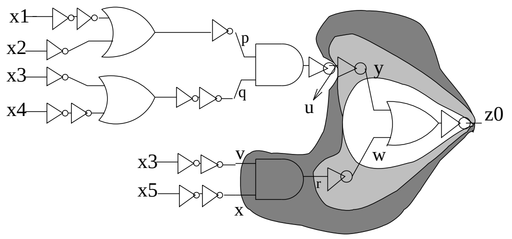

Figure 4: An example circuit for technology mapping

Table 3 contains the signatures for various cells in our example library. In this example library, only the AOI21 cell has more than one signature.

The mapping process involves selecting a subgraph of the subject graph and computing its signature. For example, for the subtree ($z0$, y, w) in Fig. 4, if y is assigned the value $\frac{1}{3}$ and w is assigned the value $\frac{1}{5}$, then the computed signature is $\frac{14}{15}$, which has an entry in library table pointing to NOR2. Pin a is matched with wire y and pin b is matched with w. Similarly, subgraphs ($z0$, u, r) and ($z0$, u, v, x) are matched with AND2 and AND3 cells, respectively.

Table 3: Hash Table for Example Library

$S_f(V)$	Library Cell, Var Assignment
$\frac{2}{3}$	INV1X, ($a = \frac{1}{3}$)
$\frac{1}{15}$	AND2, ($a = \frac{1}{3}$, $b = \frac{1}{5}$)
$\frac{8}{15}$	NAND2, ($a = \frac{1}{3}$, $b = \frac{1}{5}$)
$\frac{14}{15}$	NOR2, ($a = \frac{1}{3}$, $b = \frac{1}{5}$)
$\frac{1}{245}$	AND3, ($a = \frac{1}{3}$, $b = \frac{1}{5}$, $c = \frac{1}{17}$)
$\frac{254}{255}$	NAND3, ($a = \frac{1}{3}$, $b = \frac{1}{5}$, $c = \frac{1}{17}$)
$\frac{128}{255}$	NOR3, ($a = \frac{1}{3}$, $b = \frac{1}{5}$, $c = \frac{1}{17}$)
$\frac{87}{255}$	AOI21, ($a = \frac{1}{5}$, $b = \frac{1}{17}$, $c = \frac{1}{3}$)
$\frac{55}{255}$	AOI21, ($a = \frac{1}{3}$, $b = \frac{1}{17}$, $c = \frac{1}{5}$)
$\frac{31}{255}$	AOI21, ($a = \frac{1}{3}$, $b = \frac{1}{5}$, $c = \frac{1}{17}$)

AOI21 presents a case where a library element has more than one signature. If subtree ($z0$, p, q, w) is selected as a candidate and p, q, and w are randomly assigned the values $\frac{1}{3}$, $\frac{1}{5}$, and $\frac{1}{17}$, then the signature computed for this subtree is $\frac{31}{255}$. This can matched with AOI21 gate with pin matchings (p a), (q b), and (w, c).

3.2 Pruning Candidate Subgraphs

The number of possible candidates for a matching at node v_f grows with the size of the subject trees. However, this can significantly reduced using some knowledge about the library cells. Following heuristics have been used reduce the number of candidate subtrees for matching:
(1) Library Cell Size:

Each selected leaf-DAG has a number of leaf variables associated with it. For a given library, we know the maximum number of inputs to any of its cells. Only those candidate trees at v_f, which have less number of leaves than this maximum number for a library, are considered for the matching. Typically, most libraries contain cells up to 8 inputs.
(2) Phase-Based Pruning:

A characteristic of the library cells, in most of the libraries available today, is the *unateness-of-phase* of the input nodes with respect to the output node. The term unateness-of-phase has been used to mean the following: If two leaf nodes contain equal number of inversions, along every path from these nodes to the root node, then these nodes are said to be of unate phase with respect to the root node. Fig. 5 shows some examples of subtrees having unate and non-unate leaf phases.

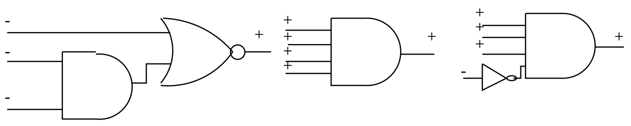

Figure 5: Unate and non-unate phases

For example, in the MCNC library (`lib2`), all gates, except the XOR and XNOR gates, satisfy the unateness-of-phase property. Thus, with the knowledge that all functions with more than n inputs satisfy this property, all the candidate trees with more than n leaves having non-unate leaf with respect to the node v_f can be discarded for the matching process. The phase calculation needs to be done only once for a tree. This can be done using a simple breadth-first search procedure starting at the root of the tree. The root node is assigned a phase and its inputs are assigned an opposite phase if the gate at this node is inverting, otherwise the inputs are assigned the same phase.

IV. Experimental Results

The technology mapping system, called MARS, described in this paper has been implemented in Common LISP, running on a SPARCStation 10. The library used for experimentation is the library distributed with the `LGSynth91` benchmark suite under the name of `lib2`. It is composed of 29 gates.

The signatures for the library cells are computed at the beginning, only once. For the MCNC library, there are **140 distinct** signatures. It takes approximately **2 seconds** to compute these signatures and setup the signature hash table.

For technology mapping, a simple gate with large number of inputs provides a good test example. A set of AND gates, with number of inputs varying from 8 to 128, were used for this purpose. Table 4 shows an approximately linear growth in the CPU time with respect to the the size of the gates.

Table 4: Results on AND gates

gate	area	CPU
AND8	6032	0.6
AND16	13456	1.8
AND32	27376	4.4
AND64	54288	9.2
AND128	109968	20.8

Table 5 contains the results obtained over the **unoptimized** ISCAS85 benchmark circuits. The circuits were technology mapped using the our system and the SIS [2] system. For SIS, the results were optimized using the `map` command with `-m 0` options. This produces the minimum area circuit with no consideration for load limits.

Columns 3 and 4 contain the area of the mapped circuit and the CPU times in seconds, obtained using the MARS system. Columns 5 and 6 contain the area of the mapped circuits and the CPU times in seconds, for a DECStation 5000/125, obtained using the SIS system. As a results of using Boolean matching as opposed to structural matching over trees, MARS produces circuits with smaller areas. The gains in area range from 0.4% to 17.2% for various examples, with comparable run-times.

In the method presented here, the equivalence of two graphs is decided deterministically by assigning by assigning the values $\frac{1}{3}, \frac{1}{5}, \frac{1}{17}, \ldots, \frac{1}{2^{2^{n-1}}+1}$ to the variables. The rapid growth in the denominator is necessary on account of the large number of Boolean functions.

For a typical library available today, the maximum value of n is 8 and the maximum cost of computing signature involves additions and multiplications of integers represented using 128 bits. This is approximately four times the cost of computing a signature with 32-bit integer values. Hence, the cost of computing a signature can be said to be of the order of the cost of creating the OBDD. Also, this computation is carried out only once for a candidate being matched.

Table 5: RESULTS ON ISCAS85 BENCHMARK CIRCUITS

STATISTICS		MARS		SIS	
ckt	gates	area	cpu	area	cpu
c432	160	230144	20.4	234784	8.2
c499	202	379552	16.6	458432	11.0
c880	383	351248	21.7	369344	10.2
c1355	546	609696	38.9	713632	14.8
c1908	880	607376	46.9	610160	19.0
c2670	1193	854224	70.2	882528	29.2
c3540	1669	1247696	88.5	1276000	38.7
c5315	2307	1944644	108.6	1971072	62.4
c6288	2406	2939904	28.8	3291616	58.1
c7552	3512	2665216	110.0	2800240	88.0

For libraries which may contain 8-to-1 muxes and 16-input AOIs, the size of encoding will be quite large. However, in this scenario, an approach of mixed structural and Boolean matching will be quite useful. All the subject graph portions with small number of leaves, for example up to 8, can be subjected to the Boolean process while matching for larger subject-tree can take the course of structural matching. The gains, in terms of area, performance, and power dissipation of the circuit, to be made by using efficient Boolean matching during the technology mapping process can be significant. The method for determing probabilistic equivalence [1] of Boolean function will significantly reduce the compuational complexity for matchings involving large number of inputs.

V. Summary

We have described a new Boolean matching algorithm based on signature computation using an unique set of probability values [1]. For the typical libraries available today, the cost of computing a signature is of the order of the cost of creating the OBDD. Once the signature of a portion of subject graph is computed, a possible match is determined in $O(1)$ time. We have also suggested some techniques to reduce the number of subgraphs examined for the purpose of matching. We have demonstrated the effectiveness of this approach through an application to optimize area of the ISCAS85 benchmark circuits.

Acknowledgment

I will like to thank Vigyan Singhal of U. C. Berkeley for helping me with SIS benchmark results. I am grateful to Dr. J. Brock Barton and Dr. P. Bosshart of Texas Instruments for their valuable suggestions.

References

[1] M. Blum, A. K. Chandra, and M. N. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, pages 80–82, March 1980.

[2] R. Brayton, G. Hactel, and A. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, pages 264–300, February 1990.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, pages 677–691, August 1986.

[4] J. R. Burch and D. E. Long. Efficient boolean function matching. In *Proc. of International Conference on Computer-Aided Design*, pages 408–411, 1992.

[5] K-C Chen. Boolean matching based on boolean unification. In *European Design Automation Conference*, pages 346–351, 1993.

[6] J. Darringer, D. Brand, W. Joyner, and L. Trevillyan. Lss: A system for production logic synthesis. *IBM Journal of Research and Developement*, pages 537–545, September 1984.

[7] E. Detjens, G. Ganot, R. Rudell, and A. Sangiovanni-Vincentelli. Technology mapping in mis. In *Proc. of International Conference on Computer-Aided Design*, pages 1062–1081, 1987.

[8] S. Ercolani and G. De Micheli. Technology mapping for electrically programmable gate arrays. In *Proc. of Design Automation Conference*, pages 234–239, 1991.

[9] D. Gregory, K. Bartlett, A. DeGeus, and G. Hachtel. Socrates: A system for automatically synthesizing and optimizing combinational logic. In *Proc. of Design Automation Conference*, pages 79–85, 1986.

[10] K. Keutzer. Dagon: Technology mapping and local optimization. In *Proc. of Design Automation Conference*, pages 341–347, 1987.

[11] F. Mailhot and G. De Micheli. Algorithms for technology mapping based on binary decision diagrams and on boolean operations. *IEEE Trans. on Computer-Aided Design*, pages 599–620, May 1993.

[12] R. Rudell. *Logic Synthesis for VLSI Design.* PhD thesis, University of California, Berkeley, CA, April 1989.

[13] H. Savoj, M. J. Silva, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Boolean matching in logic synthesis. In *European Design Automation Conference*, pages 168–174, 1992.

[14] U. Schlichtmann and F. Brglez. Efficient boolean matching in technology mapping with very large cell libraries. In *CICC*, pages 3.6.1–3.6.6, 1993.

[15] H. Touati. *Performance-Orieneted Technology Mapping.* PhD thesis, University of California, Berkeley, CA, 1991.

[16] K. Zhu and D. F. Wong. Fast boolean matching for fpgas. In *European Design Automation Conference*, pages 352–357, 1993.