A New Enhanced Constructive Decomposition and Mapping Algorithm

Alan Mishchenko
Department of EECS
University of California, Berkeley
Berkeley, CA 94720
alanmi@eecs.berkeley.edu

Xinning Wang
Intel Corporation
Strategic CAD Labs
Hillsboro, OR 97124
xinning.wang@intel.com

Timothy Kam
Intel Corporation
Strategic CAD Labs
Hillsboro, OR 97124
timothy.kam@intel.com

ABSTRACT
Structuring and mapping of a Boolean function is an important problem in the design of complex integrated circuits. Library-aware constructive decomposition offers a solution to this problem. This paper proposes novel techniques to improve the quality and runtime of constructive decomposition. The improvements are effective both in the stand-alone mapping procedure and in the context of re-synthesis applied to a mapped multi-level network. Experiments with public and proprietary benchmarks show promising results.

Categories and Subject Descriptors
B.6.3 [**Logic Design**]: Design Aids – Automatic synthesis.

General Terms
Algorithms, Performance, Experimentation, Theory.

Keywords
Functional decomposition, technology mapping, re-synthesis.

1. INTRODUCTION
Today's commercial logic synthesis tools have matured enough in terms of handling capacity, design quality, and runtime efficiency to justify their use for ASIC designs. However, widespread use of these tools in high performance designs, such as microprocessors, is still limited, mainly due to the quality of synthesis results. Synthesis tools based only on algebraic decomposition [3] often cannot compete with an experienced human designer capable of reasoning in Boolean domain. Another key factor affecting the quality of automated synthesis is the phase decoupling between technology-independent logic transformations and technology mapping in traditional logic synthesis flows.

The recent work [21] with further development [8] attempts to bridge the gap between technology-independent logic decomposition and technology mapping. A limitation of this work is that the transformations are limited to those having an algebraic structure. Constructive synthesis [10][12][13] is another novel technique combining Boolean decomposition and technology mapping in a creative way. However, the practical usefulness of this approach has limitation due to the following factors:

DAC 2003, June 2-6, 2003, Anaheim, California, USA.

- At each decomposition step, it considers only a subset of support-reducing decompositions.
- It builds a large intermediate BDDs for solving the basic 4-to-3 decomposition steps, which increases the runtime.
- The mapped netlist often contains non-decomposable blocks.

This paper addresses these limitations by proposing computationally efficient solutions for

- Considering all support-reducing decompositions.
- Performing fast decomposition in the presence of don't-cares extracted from the context of the node in the netlist.
- Pre-computing the gate library representation in the form of *supergates,* leading to fewer non-decomposable blocks.

These improvements result in reducing delay and shortening runtime. Another contribution is a re-synthesis framework, which enables selective application of computationally expensive logic synthesis techniques to critical areas in large industrial designs.

The paper is organized as follows. Section 2 introduces the basic concepts of constructive synthesis. Section 3 shows a new fast constructive decomposition algorithm based on incremental encoding, which works for both completely- and incompletely-specified functions. Section 4 shows a new technique to compute all support-reducing bound sets of a completely specified function. The following two sections describe the adaptations of the constructive decomposition for technology mapping and re-synthesis. Section 5 discusses a new representation of a gate library in the form of supergates. Section 6 presents an efficient way of computing local don't-cares from the network structure. Section 7 outlines the implementation of the decomposition engine and the re-synthesis framework. Section 8 presents experimental results. Section 9 summarizes the paper.

2. PRELIMINARIES
Let $f(X)$: $B^n \rightarrow B$, $B = \{0,1\}$, be a completely specified Boolean function (CSF). Let $f(X)$: $B^n \rightarrow \{0,1,-\}$ be an incompletely specified function (ISF). An ISF is represented by two CSFs, the *on-set* (f^1) and the *off-set* (f^0). The *support* of f, $supp(f)$, is the set of variables X, which influence the output value of f. The support size is denoted by $|X|$. For a CSF $f(X)$ and a subset of its support, X_1, the set of distinct *cofactors*, $q_1(X)$, $q_2(X)$, … , $q_\mu(X)$, of f with respect to (w.r.t.) X_1 is derived by substituting all assignments of X_1 into $f(X)$ and eliminating duplicated functions. The number of distinct cofactors, μ, is the *column multiplicity* of f w.r.t. X_1.

Given a partition of X into two disjoint subsets, X_1 and X_2, Ashenhurst-Curtis decomposition of $f(X)$ is:

$$f(X) = h(g_1(X_1), g_2(X_1), \ldots, g_k(X_1), X_2).$$

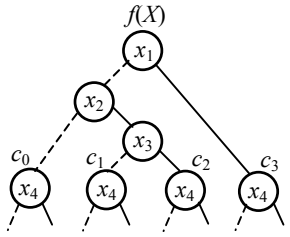
Subsets X_1 and X_2 are called the *bound set* and the *free set*, respectively. Functions $g_i(X_1)$, $1 \le i \le k$, are the *decomposition functions.* Function $h(G, X_2)$ is the *composition function*.

Lemma 1. [1][7] The decomposition of $f(X)$ with k functions $g_1(X_1)$, $g_2(X_1)$, …, $g_k(X_1)$ exists if and only if (iff) $\lceil log_2\mu \rceil \le k \le n$, where μ is the column multiplicity of f w.r.t. X_1, and $n = |X_1|$.

Following [10], we say that the bound set X_1 leads to an n-to-k *support-reducing* decomposition if k satisfies $\lceil log_2\mu \rceil \le k < n$. All support-reducing bound sets of a fixed size can be computed for a given CSF $f(X)$ using the algorithm described in Section 4.

This paper assumes that the reader is familiar with the basic concepts of Binary Decision Diagrams (BDDs) [4]. The BDD representation of $f(X)$ is convenient for decomposition because the cofactors of f w.r.t. X_1 can be found by identifying all the nodes in the BDD of f that have incoming edges from the nodes located above the cut separating X_1 from X_2 in the variable order [14][23].

Example 1. Fig. 1 shows the upper part of the BDD for some function $f(X)$. The bound set $X_1 = \{x_1, x_2\}$ leads to the decomposition with three distinct cofactors. This bound set is not support-reducing because $k = \lceil log_2 3 \rceil = 2$ and $n = |X_1| = 2$. The bound set $X_1 = \{x_1, x_2, x_3\}$ leads to the decomposition with four distinct cofactors, denoted c_0, c_1, c_2, and c_3. This bound set is support-reducing because $k = \lceil log_2 4 \rceil = 2$ and $n = |X_1| = 3$.



$x_1 \setminus x_3$ \ x_2	0 0	0 1	1 1	1 0
0	0	0	2	1
1	3	3	3	3

Figure 1. The function and the decomposition pattern.

3. DECOMPOSITION ALGORITHM

3.1 Algorithm Outline

Decomposition of a logic function represents the function as a network of elementary sub-functions. The pseudo-code of constructive decomposition is shown in Fig. 2. Given a function f and a set of elementary sub-functions $\{f_i\}$, decomposition is performed iteratively as long as f is not one of the elementary sub-functions. In each iteration, a bound set X is determined. Next, decomposition functions $\{g_i\}$, $g_i \in \{f_i\}$, and the composition function h are derived. The functions $\{g_i\}$ are added to the network and the decomposition proceeds to decompose function h, which depends on fewer variables than f.

```
network ConstructiveDecomposition( function f, functions {fi})
{
   N = empty network;
   while ( f ∉ {fi} ) {
      X  = DetermineBoundSet( f )
      {gi} = DeriveDecompositionFunctions( f, X, {fi});
      h  = DeriveCompositionFunction( f, X, {gi} );
      AddToNetwork( N, {gi} );
      f = h;
   }
   AddToNetwork( N, f );
   return N;
}
```

Figure 2. The pseudo-code of constructive decomposition.

3.2 Completely Specified Functions

The decomposition of the CSF $f(X)$ with the bound set X_1 is characterized by a *decomposition pattern* defined in [10] as a mapping of the bound set minterms into the corresponding cofactors. For CSFs, each minterm maps into a unique cofactor.

Example 2. The four distinct cofactors created by the bound set $X_1 = \{x_1, x_2, x_3\}$ in Fig. 1 are c_0, c_1, c_2, and c_3. The decomposition map is shown in Fig. 1 (right), where each minterms is labeled by the number of the corresponding cofactor.

To derive the decomposition functions, we introduce k new variables $g_0, g_1, ..., g_{k-1}$ and encode μ cofactors of f w.r.t. X_1 using unique minterms composed of the new variables. The i-th decomposition function, $g_i(X)$, is derived by ORing of the bound set minterms associated with the cofactors, whose codes have variable g_i in the positive polarity. Different encoding of cofactors can lead to different sets of decomposition functions.

Example 3. ORing of the bound set minterms associated with the cofactors of $f(X)$ shown in Fig. 1 yields the following functions:

$$m(c_0) = \bar{x}_1\bar{x}_2,\ m(c_1) = \bar{x}_1 x_2\bar{x}_3,\ m(c_2) = \bar{x}_1 x_2 x_3,\ m(c_3) = x_1.$$

For the natural encoding of cofactors, $code(c_0)$=00, $code(c_1)$=01, $code(c_2)$=10, $code(c_3)$=11. The decomposition functions are:

$$g_0(X) = m(c_1) \vee m(c_3) = \bar{x}_1 x_2\bar{x}_3 \vee x_1;$$

$$g_1(X) = m(c_2) \vee m(c_3) = \bar{x}_1 x_2 x_3 \vee x_1.$$

Simpler decomposition functions can be derived by assigning adjacent codes to cofactors c_1 and c_2, $code(c_0)$=00, $code(c_1)$=01, $code(c_2)$=11, $code(c_3)$=10:

$$g_0(X) = m(c_1) \vee m(c_2) = \bar{x}_1 x_2;$$

$$g_1(X) = m(c_2) \vee m(c_3) = x_2 x_3 \vee x_1.$$

3.3 Incompletely Specified Functions

The cofactors of an ISF are themselves ISFs. When decomposing ISFs, each bound set minterm is associated with an ISF of the corresponding cofactor. In this case, it is possible to define a compatibility relation of the bound set minterms. This relation is true for two minterms iff there exists an assignment of don't-cares in the corresponding cofactors, which makes the cofactors equal.

Example 4. Suppose cofactors c_0, c_1, c_2, and c_3 in Fig. 1 depend on variables x_4 and x_5 and have truth tables shown as columns in Fig. 3 (left). The truth tables of c_1 and c_2 can be made equal by assigning the don't-care minterm 10 of cofactor c_1 to 1 and the don't-care minterms 01, 11 of cofactor c_2 to 0 and 1, respectively.

x_4x_5	c_0	c_1	c_2	c_3
00	1	1	1	0
01	1	0	-	0
10	1	-	1	1
11	0	1	-	0

$x_1x_2x_3$ / $x_1'x_2'x_3'$	0 0 0	0 0 1	0 1 1	0 0 1	1 1 1	1 1 0	1 1 1	1 0 0
000	1	1	0	0	0	0	0	0
001	1	1	0	0	0	0	0	0
011	0	0	1	**1**	0	0	0	0
010	0	0	**1**	1	0	0	0	0
110	0	0	0	0	1	1	1	1
111	0	0	0	0	1	1	1	1
101	0	0	0	0	1	1	1	1
100	0	0	0	0	1	1	1	1

Figure 3. Cofactor truth tables and the compability relation.

The compatibility relation R can be computed using the on/off-set representation of the ISF $f = (f^0, f^1)$ as follows:

$$R(X_1,X_1') = \neg\exists X_2[f^1(X_1, X_2) f^0(X_1', X_2) \vee f^0(X_1, X_2) f^1(X_1', X_2)]$$

$$= \forall X_2[\bar{f}^1(X_1, X_2)\bar{f}^1(X_1', X_2) \vee \bar{f}^0(X_1, X_2)\bar{f}^0(X_1', X_2)].$$

This formula states that two bound set minterms, X_1 and X_1', are compatible iff there is no assignment A of the free set variables X_2

such that the cofactors of $f(X)$ w.r.t. X_1 and X_1' are not compatible, i.e., one of them is in the on-set while the other in the off-set:

$$f^1(X_1, A)f^0(X_1', A) \vee f^0(X_1, A)f^1(X_1', A).$$

In the presence of don't-cares, the minimum column multiplicity for the given bound set can be computed by graph coloring of the graph representing the compatibility relation. In this graph, the vertices correspond to the bound set minterms. The two vertices have an edge between them (can be colored by the same color) iff the corresponding mintems belongs to the compatibility relation.

Lemma 2. [22] A support-reducing decomposition of $f(X)$ with bound set X_1 exists iff graph coloring of the graph of $R(X_1, X_1')$ yields the number of minimum colors, μ, satisfying $\lceil log_2\mu \rceil < n$.

Example 5. Consider the decomposition pattern in Fig. 1 (right) and the cofactor truth tables in Fig. 3 (left). The compatibility relation is in Fig. 3 (right), where the 1's in bold are produced by the compatibility of c_1 and c_2 due to don't-cares. Graph coloring of the graph of this relation yields $\mu = 3$. According to Lemma 2, decomposition with $k = \lceil log_2 3 \rceil = 2 < n = 3$ exists.

Deriving the decomposition functions in the case of ISFs is performed by encoding the classes of bound set minterms colored by the same color, similarly to encoding the cofactors of CSFs.

3.4 Selecting Decomposition Functions

Several methods for selecting the decomposition functions have been proposed. Some of them exploit the cofactor encoding but do not use don't-cares [15][18][20]. Other methods use don't-cares but do not take advantage of the cofactor encoding [22]. The previous work on constructive decomposition theoretically allows for don't-cares [10][13], but in practice don't-cares are not extracted from the netlist in the context of re-synthesis [11].

The decomposition algorithm introduced in this section exploits the freedom provided by the cofactor encoding in the case of CSFs, and both graph coloring and cofactor encoding in the case of ISFs, to facilitate selection of decomposition functions with desirable properties. For example, we can, as in [18], prefer single-variable decomposition functions replaceable by wires in the netlist, or we can choose functions that map into gates with short delay, which is relevant to the goal of this paper.

The proposed selection algorithm is based on the incremental encoding of cofactors [18]. The input to the algorithm is a set of candidate functions and a decomposition pattern represented by the mapping of bound set minterms into cofactors in the case of CSF, or by the compatibility relation in the case of ISFs. Without specifying what desirable properties the functions should have, we assume that the functions have been pre-computed and are available as an array sorted in the decreasing order of desirability. The algorithm returns a set of functions that satisfy the decomposition pattern and occur as early as possible in the sorted array. This solution is optimal according to the selected criteria. Below we consider only the case for ISFs, as a more difficult one.

Let X be the bound set. Let $R(X, X')$ be a compatibility relation, and let $g_1(X)$, $g_2(X)$, …, $g_p(X)$ be some functions. The *cofactor relations*, $R_j(X, X')$, $0 \le j < 2^p$, of relation $R(X, X')$ w.r.t. the given functions are defined as follows. For each j, $0 \le j < 2^p$, compute $S_j(X)$, the product of functions $g_1(X)$, $g_2(X)$, …, $g_p(X)$, with polarities specified by the binary representation of j. $S_j(X)$ is a function over the bound set variables. Then,

$$R_j(X, X') = R(X, X') \wedge S_j(X) \wedge S_j(X').$$

The motivation for considering $R_j(X, X')$ is given by the following observation. Suppose in the process of decomposition, we have selected a set of decomposition functions, $g_1(X)$, $g_2(X)$, …, $g_p(X)$, which split the bound set space into 2^p subspaces containing minterms that are not distinguished by the selected functions. Restricting $R(X, X')$ to these subspaces is needed to check whether the decomposition with the given functions exists.

Lemma 3. Let $R(X, X')$ be the compatibility relation and $g_1(X), g_2(X)$, …, $g_p(X)$, $0 \le p \le k$, be a set of decomposition functions. The decomposition with these p functions and the additional k-p functions exists iff the graphs of each cofactor relation $R_j(X, X')$, $0 \le j < 2^p$, can be colored using at most μ colors, satisfying $\lceil log_2\mu \rceil \le k$-$p$.

Proof: The decomposition exists iff all incompatible bound set minterms are distinguished. The graphs of cofactor relations can be colored as stated in Lemma 3 iff the additional k-p decomposition functions can be selected to distinguish the remaining minterms in the subspaces of the compatibility relation induced by the already selected decomposition function. *Q. E. D.*

Example 6. Consider the compatibility relation in Fig. 3 and the ordered set of functions (g_0, g_1), $g_0(X) = x_1 \vee x_2$, $g_1(X) = x_1$. The gray areas in Fig. 4 show the on-sets of the cofactor relations w.r.t. g_0 (left) and both g_0 and g_1 (right). Decomposition with g_0 exists because each cofactor relations w.r.t. g_0 can be colored with at most two colors. The decomposition with both g_0 and g_1 exists because each cofactor relations w.r.t. both g_0 and g_1 can be colored with one color. On the other hand, decomposition with $g_2(X) = x_1\bar{x}_2$ does not exist because one of the cofactor relations produced by this function (not shown in Fig. 4) requires three colors. Note that using don't-cares leads to the decomposition with simpler functions, $g_0(X) = x_1 \vee x_2$ and $g_1(X) = x_1$, compared to the solutions obtained without don't-cares in Example 3.

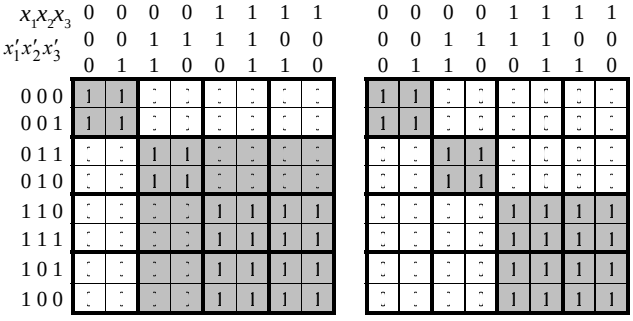


Figure 4. The cofactor relations.

The above decomposition theory can be used to create a robust decomposition algorithm. Given a set of candidate functions, we try selecting subsets, each containing k functions. The functions are selected in the order of their appearance in the sorted array, which is sorted using the arrival times of the inputs. The first subset that satisfies Lemma 3 is returned as the solution. If, for the given k, we tried all sets of k functions and none of them leads to the decomposition, we increment k as long as $k < n$, and apply the same procedure under the relaxed conditions.

The proposed algorithm is robust in the sense that it can solve decomposition problems with thousands of candidate functions. The runtime of the algorithm for a typical 5-variable decomposition pattern strongly depends on the probability of the existence of decomposition but is always reasonable and rarely exceeds 0.1 sec. The trade-off between runtime and quality can be controlled by a set of parameters.

4. BOUND SET COMPUTATION

A straightforward way of computing all support-reducing bound sets of size n consists in counting the number of different cofactors for each bound set of the given size. The bounds sets

with the column multiplicity μ satisfying $\lceil log_2\mu \rceil < n$ are, by definition, support-reducing.

In this section, we describe an efficient bound set computation using BDDs. Although our method also creates all bound sets, it does so implicitly using cache to avoid repeated computations. For this reason, this method has better performance compared to the explicit enumeration using the straightforward method. The computation is divided into several steps:

Step 1. In the BDD package, introduce three sets of variables, *X*, *Y*, and *Z*, each having size $m = supp(f)$. Interleave the variables to get the variable order:

$$x_1 < y_1 < z_1 < x_2 < y_2 < z_2 < ... < x_m < y_m < z_m$$

Step 2. Construct function *f* using variables *X* and perform simultaneous composition of $f(X)$ with functions $x_i(X,Y,Z) = \text{ITE}(x_i, y_i, z_i)$, for all $1 \le i \le m$. Given the interleaved variable order, the resulting function $f(X,Y,Z)$ has three times more internal nodes than $f(X)$. A schematic representation of the BDDs for *f*, before and after composition, is shown in Fig. 5 (left and center).

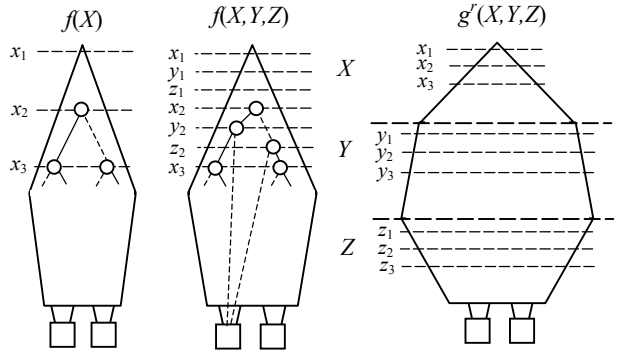


Figure 5. Transformation for the BDD for *f*(*X*) during computation of all support-reducing bound sets.

Step 3. Compute $g(X,Y,Z) = f(X,Y,Z) \wedge Tuples_{n,m}(X)$, where $Tuples_{n,m}(X)$ is the function, whose on-set minterms depend on *X* and contain exactly *n* positive literals. The BDD representation of $Tuples_{n,m}(X)$ can be efficiently derived, as shown in [9].

Step 4. Permute variables in $g(X,Y,Z)$ in such a way that variables *X* are above variables *Y* and variables *Y* are above variables *Z*. One of such variable orders, shown in Fig. 5 (right), is:

$$x_1 < x_2 < ... < x_m < y_1 < y_2 < ... < y_m < z_1 < z_2 < ... < z_m.$$

In the resulting BDD $g^r(X,Y,Z)$, variables *X* encode all bound sets of size *n*. In each assignment $(x_0, x_1, \ldots, x_m)$ of variables *X*, 1's stand for the bound set variables, while 0's stand for the free set variables. In the cofactor $g^r_{(x0,\ x1,\ \ldots,\ xm)}(Y, Z)$, the bound set is represented by *Y* and the free set is represented by Z.

Step 5. It is now possible to count the number of different nodes pointed to below the cut separating variables of *Y* and *Z* in each cofactor $g^r_{(x0,\ x1,\ \ldots,\ xm)}(Y, Z)$. This number is equal to the column multiplicity μ in the decomposition with the corresponding bound set. If $\lceil log_2\mu \rceil < n$, the bound set is support-reducing. The computation for all bound sets is performed in one traversal of the upper part of the BDD representing $g^r(X, Y, Z)$.

Note that Step 4 in the above computation typically leads to a substantial increase in the size of the BDD for $g^r(X, Y, Z)$ compared to $g(X, Y, Z)$. However, this increase can be controlled in Step 3, by further restricting the set of considered bound sets.

The overall performance of this computation is reasonable for typical functions encountered in a netlist. The computation of all 3-, 4-, and 5-variable support-reducing bound sets for a typical 10-variable function takes about 0.05 sec on a 1GHz computer with 256Mb RAM. Similar computation for a 20-variable function takes about 0.5 sec. Further increase in the support size leads to an exponential increase in runtime, unless the number of computed bound set is further constrained, for example, by only considering a subset of the support variables. It should be noted that, in many cases, bound set computation can be made much faster by first detecting disjoint-support decomposition [2][16].

The above algorithm works for CSFs. When dealing with ISFs, we derive support-reducing bound sets by first applying the BDD minimization algorithm *restrict* [6] to produce a CSF.

5. GATE LIBRARY REPRESENTATION

Constructive decomposition aims at constructing a decomposed network functionally equivalent to a CSF or ISF, with the following requirements: (1) the decomposition functions can be efficiently implemented using the gate library; (2) the delay/area trade-off of the decomposed network satisfies the constraints.

To facilitate selection of the decomposition functions that satisfies the given constraints, we introduce the concept of a *supergate*. Essentially, a supergate is a single-output network composed of several logic levels of library gates. The upper bound on the delay of this network is imposed to limit the total number of supergates generated, which can be very large even for small gate libraries.

The supergates are computed independently for each support size of the decomposition functions. The library for a larger support size includes the supergates with smaller supports because the decomposition functions can depend on fewer variables than the bound set size. In our experiments, we pre-compute the library of supergates by setting a limit on the supergate delay to be several delays of the inverter. For a typical gate library, the delay of three or four inverters leads to several thousand four-input supergates, which almost completely eliminates the possibility of generating non-decomposable blocks for most benchmark functions.

Some pre-computed supergates may have equivalent functionality but different delay/area parameters. Such supergates represent alternative implementation choices. Depending on the situation, we may use a complex gate, or a functionally equivalent combination of simpler gates. We remove from the library only *dominated* supergates, having area and pin-to-pin delays larger than or equal to another supergate of the same functionality.

We sort the supergates in each decomposition step, for each decomposition pattern. For example, if the minimum delay is sought, the supergates are sorted by increasing delay. Given the sorted array of supergates, the decomposition algorithm selects the decomposition functions that optimize the given criteria.

In summary, introducing supergates, as described in this section, facilitates decomposition by reducing the number of non-decomposable blocks and enhancing the scope of decomposition from single gates (one level) to gate clusters (multiple levels).

6. DON'T-CARE COMPUTATION

The traditional technology mapping methods [25] and their improvements [21][8] work on a NAND or AND/OR graph derived by the algebraic decomposition of the network. Using the uniform network representation allows the mapping algorithms to perform optimization across the boundaries of logic blocks.

A potential weakness of the constructive decomposition, on the other hand, is that each node is decomposed in isolation from other nodes. Even if the decomposed functions are cached and reused, logic sharing is difficult to create because the nodes considered earlier may produce decomposition functions that are useless for the decomposition of the nodes encountered later.

We compensate for this weakness of constructive decomposition by computing don't-cares for each node using its context in the network. These don't-cares contain important information about

the reduced controllability and observability of the node, which allows the decomposition algorithm, in a roundabout way, to optimize across the boundaries of logic blocks.

The previous work on the use of don't-cares in the optimization of Boolean networks [24] considers only the compatible and satisfiability don't-cares. Our don't-care computation method is similar to [17] in that is computes the *complete* don't-cares. The difference is, we compute don't-cares in a dynamically adjusted context of the node rather than in the scope of the total network.

The context of the node is determined "on-the-fly", when the don't-care computation is called. This allows us to integrate this algorithm into applications performing frequent updated to the network structure, such as the re-synthesis framework. The re-convergent paths in the vicinity of the node produce most of the don't-cares for a node. Therefore, our algorithm includes as many re-convergent paths as possible into a sub-network surrounding the node. This sub-network is used for the computation of the complete don't-cares, as shown in [17].

The performance of the don't-care computation is controlled by several parameters specifying the size of the sub-network (the number of logic levels on the fanin/fanout side of the node), the limit on the BDD size, and the timeout. Moreover, because the computation is performed in a limited scope around the node, its runtime is independent of the size of the Boolean network.

7. IMPLEMENTATION ISSUES

The following subsections briefly discuss the implementation of two main components of our decomposition system. The *decomposition engine* (CDM) performs decomposition of a CSF or an ISF. It returns the implementation of the function in the form of a network of supergates. The *re-synthesis framework* (RESYN) takes a mapped netlist, determines the gate clusters to be re-synthesized, collapses these gates into a single node, calls one or more decomposition engines to decompose this node, selects the best decomposition, and replaces the collapsed node by the decomposed network if the decomposition is acceptable. The re-synthesis framework performs this operation until there is no improvement, or until a fixed number of iterations is reached.

7.1 Decomposition Engine

We developed two versions of the decomposition engine using alternative data structures, BDDs and bit-strings, to represent the candidate functions and search for the decomposition. Extensive testing has shown that bit-strings lead to significantly faster processing compared to the BDDs used in [10][12][13].

In our approach, BDDs are used only to derive the compatibility relation $R(X_1, X_1')$. This computation involves $2*|X_1| + |X_2|$ variables, where X_1 is the bound set and X_2 is the free set. This number is much less than $k*2^{|X_1|} + |X_2|$, which is the number of variables in the representation of the constraint function to encode all decomposition choices in [10] (page 63, formula 4.16).

Instead of implicitly encoding decomposition choices in a BDD, we use the explicit search through the space of decompositions as described in Section 3.3. This search can be efficiently implemented using bit-strings, because for a bound set of five variables or less, the truth table of each decomposition function can be stored in one machine word. The compatibility relation can also be represented using bit-strings, along with its cofactors. The solutions of graph coloring for small relations are pre-computed and stored in a hash table, so that checking the existence of decomposition with a particular set of decomposition functions is reduced to several table lookups. This is the reason why our implementation is fast enough to test many support-reducing bound sets, each of them with a large set of candidate functions.

7.2 Re-Synthesis Framework

The re-synthesis framework takes the following parameters: arrival times of the primary inputs, required times of the primary outputs, a *window* parameter, a *depth* parameter, and an *iteration limit*. Each iteration of re-synthesis proceeds as follows:

1. Compute the slack of each node using timing analysis by propagating the arrival and required times. Timing-critical outputs and internal nodes are identified. A node n is considered timing-critical if the value $slack(n)+window$ is negative.

2. For each timing-critical node, count the number of critical paths passing through this node. The node g with the largest path count is chosen. The given number (*depth*) of levels of the transitive fanins of g are collapsed into g, resulting in a larger node g_c.

3. Don't-cares are computed for node g_c taking into account its context in the network. Next, g_c is decomposed with the don't-cares, using the arrival time information of its fanins. If the arrival time of the output of the resulting logic cone satisfies the required time, and the resulting area does not increase above a certain limit, the collapsed node g_c is replaced by the resulting logic cone. If the decomposition is not accepted, the original node g is marked to prevent future attempts to re-synthesize it.

4. The re-synthesis continues until either all the timing-critical nodes are tried, or until the *iteration limit* is reached.

Additionally, the re-synthesis framework features a high-effort re-synthesis mode. In this mode, a number of fanin-logic collapsings are tried for each timing-critical node. In Step 2, different subsets of nodes in the transitive fanin of g are chosen for collapsing and decomposition in such a way that the most timing critical path is always included in the subset. Thus, for each timing-critical node, different logic cones are tried, one at a time, until an acceptable re-synthesis is found, or until all logic collapsing's have been tried, or until the number of collapsing attempts reaches a limit.

8. EXPERIMENTAL RESULTS

The algorithm is implemented in SIS environment [26] using BDD package CUDD [27].

In the first experiment, shown in Table 1, we applied the new decomposition algorithm (column "CDM") to a set of MCNC benchmarks using *mcnc.genlib* library and compared it with the results obtained by SIS and M31 [10]. The results for SIS are obtained by running *script.rugged* and *speed_up*, followed by the delay-oriented mapping. The results for M31 are taken from [10]. The performance of our tool was geared towards reducing delay, which could lead to an increase in area. The resulting networks are verified using the verification command in SIS.

In the second experiment, shown in Table 2, we applied the re-synthesis framework to a set of mega-block designs extracted from a high performance microprocessor at the authors' affiliation and compared the results with those obtained using the existing project synthesis flow. The delay and area numbers reported are after cell sizing using a proprietary cell sizer. It can be seen that performance improvements are achieved with reasonable area increase. This preliminary result demonstrated that it is feasible to use computationally expensive Boolean techniques to improve the synthesis quality of practical-sized designs through re-synthesis.

Table 1. Synthesis results for MCNC benchmarks.

Benchmark	Area			Delay		
	SIS	M31	CDM	SIS	M31	CDM
rd53	50	56	40	17.9	14.3	7.7
rd73	98	75	59	35.6	17.4	9.6
rd84	205	107	199	26.4	22.6	11.2
9sym	310	84	36	35.8	18.9	10.2
parity	75	75	33	12.4	15.5	9.9
my_adder	285	667	495	57.3	26.0	18.2
comp	168	240	198	24.0	25.3	17.8
z4ml	50	59	41	25.3	14.5	7.9
t481	53	56	21	11.8	11.9	7.5
pm1	87	72	87	11.1	10.8	4.9
c8	175	187	130	27.5	18.5	7.8
x4	552	624	578	38.3	22.2	24.4
count	216	272	273	58.6	23.3	15.3
pcler8	142	235	164	23.5	16.9	9.9
la1	146	166	205	22.5	15.6	8.6
sct	113	129	144	57.9	16.2	7.1
apex7	332	358	556	33.6	29.2	15.3
i2	296	295	363	21.7	17.9	18.8
Total	3353	3757	3622	541.2	337	212.1
Ratio, %	100	112	108	100	62.3	39.2

Table 2. Re-synthesis results for proprietary benchmarks.

Benchmark	Area		Delay	
	existing	RESYN	existing	RESYN
Design1	199570	210960	537	495
Design2	846067	847006	249	211
Design3	804329	824450	207	188
Design4	260704	270430	262	220
Design5	715683	717824	186	148
Total	2856353	2870670	1441	1262
Ratio, %	100	101.6	100	87.6

9. CONCLUSIONS AND FUTURE WORK

We presented several new algorithms for constructive decomposition, concerning: (1) bound set selection, (2) using don't-cares in decomposition, (3) pre-computing a set of supergates for the given gate library. Using these algorithms led to the following advantages: (a) 4-to-3 and 5-to-4 decompositions take reasonable time, (b) non-decomposable blocks are practically never created, (c) a better delay/area trade-offs are often found. The context of decomposition has been enhanced by (i) the fast extraction of local don't-cares for the node from the node's immediate context in the network, and (ii) developing a flexible re-synthesis framework, which uses several decomposition engines and selects the best result. The experimental results show that the new algorithm performs well on the benchmark functions. Future work include extending re-synthesis framework to incorporate more accurate delay modeling and physical design aspects as well as integrating constructive decomposition with bi-decomposition [5][19][27][28] techniques.

10. ACKNOWLEDGEMENTS

The first author has been supported by a research grant from Intel Corporation. The authors thank Robert Brayton, Tsutomu Sasao, and Marek Perkowski for illuminating discussions. Special thanks are due to the pioneers of constructive decomposition, Victor Kravets and Karem Sakallah, for providing us with M31 software and for sharing helpful insights into their work.

11. REFERENCES

[1] R. L. Ashenhurst, "The decomposition of switching functions," *Computation Lab*, Harvard University, 1959, Vol. 29, pp. 74-116.

[2] V. Bertacco and M. Damiani, "Disjunctive decomposition of logic functions," *Proc. ICCAD '97*, pp. 78-82.

[3] R. K. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," *Proc. ISCAS '82*, pp. 29-54.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comp.*, Vol. C-35, No. 8 (August, 1986), pp. 677-691.

[5] J. Cortadella, "Bi-decomposition and tree-height reduction for timing optimization". Proc. *IWLS '*02, pp. 233-238.

[6] O. Coudert, C. Berthet, and J. C. Madre, "Verification of synchronous sequential machines based on symbolic execution," in *Automatic Verification Methods for Finite State Systems*. Springer-Verlag, 1989, pp. 365-373.

[7] A. Curtis. New Approach to the Design of Switching Circuits. Van Nostrand, Princeton, NJ, 1962.

[8] D.-J. Jongeneel, R. Otten, Y. Watanabe, R. K. Brayton, "Area and search space control for technology mapping," *Proc. DAC '00*, pp. 86-91.

[9] T. Kam, T. Villa, R. K. Brayton, A. Sangiovanni-Vincentelli. *Synthesis of Finite State Machines: Functional Optimization.* Kluwer Academic Publishers, 1997.

[10] V. N. Kravets. Constructive Multi-Level Synthesis by Way of Functional Properties. Ph. D. Thesis, University of Michigan, 2001.

[11] V. N. Kravets. Private communication.

[12] V. N. Kravets and K. A. Sakallah, "Constructive library-aware synthesis using symmetries," *Proc. DATE '00*, pp. 208-216.

[13] V. N. Kravets and K A. Sakallah, "Re-synthesis of multi-level circuits under tight constraints using symbolic optimization," *Proc. ICCAD '02*, pp. 687-693.

[14] Y. T. Lai, M. Pedram, S. B. K. Vrudhula, "BDD-based decomposition of logic functions with applications to FPGA synthesis," *Proc. DAC '93*, pp. 642-647.

[15] Ch. Legl, B. Wurth, and K. Eckl, "Computing support-minimal subfunctions during functional decomposition," *IEEE Trans. VLSI*, 6(3), pp. 354-363, Sept. 1998.

[16] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition," *Proc. SASIMI '98*, pp. 44-50.

[17] A. Mishchenko and R. K. Brayton, "Simplification of non-deterministic multi-valued networks", *Proc.ICCAD'02*, pp.557-562.

[18] A. Mishchenko and T. Sasao, "Encoding of Boolean functions and its application to LUT cascade synthesis," *Proc. IWLS '02*, pp. 115-120.

[19] A. Mishchenko, B. Steinbach, and M. Perkowski, "An algorithm for bi-decomposition of logic functions," *Proc. DAC '01*, pp. 103-108.

[20] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum functional decomposition using encoding," *Proc. DAC '94*, pp. 408-414.

[21] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. CAD*, 16(8), 1997, pp. 813-833.

[22] M. Perkowski, R. Malvi, S. Grygiel, M. Burns, and A. Mishchenko, "Graph coloring algorithms for fast evaluation of Curtis decompositions," *Proc. DAC 99*, pp. 125-130.

[23] T. Sasao, "FPGA design by generalized functional decomposition". In T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[24] H. Savoj, R. K. Brayton, and H. Touati, "Extracting local don't cares for network optimization," *Proc. ICCAD* '91, pp. 514-517.

[25] E. Sentovich, et al. "SIS: A system for sequential circuit synthesis", *Tech. Rep. UCB/ERI, M92/41, ERL*, Dept. of EECS, Univ. of California, Berkeley, 1992.

[26] F. Somenzi. BDD package "CUDD v. 2.3.0." http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html

[27] T. Stanion and C. Sechen, "Quasi-algebraic decomposition of switching functions," *Proc. AR VLSI '95*, pp. 358-367.

[28] C. Yang and M. Ciesielski, "BDS: A BDD-based logic optimization system". *IEEE Trans. CAD*, Vol. 21 (7), July 2002, pp. 866-876.