OBDD-Based Function Decomposition: Algorithms and Implementation [1]

Yung-Te Lai, Kuo-Rueih Ricky Pan, Massoud Pedram
University of Southern California
Dept. of EE-Systems
Los Angeles, CA 90089

[1]This research was supported in part by NSF under contract No. MIP-9457392.

List of Figures

List of Tables

Abstract

This paper presents algorithms for disjunctive and nondisjunctive decomposition of Boolean functions and Boolean methods for identifying common subfunctions from multiple Boolean functions. Ordered Binary Decision Diagrams are used to represent and manipulate Boolean functions so that the proposed methods can be implemented concisely. These techniques are applied to the synthesis of look-up table based field programmable gate arrays and results are presented.

1 Introduction

Most multilevel synthesis systems contain two steps: a technology-independent step that manipulates and optimizes Boolean functions and a technology-mapping step that maps Boolean functions into a set of gates in a specific target technology. The technology-independent phase is further divided into two substeps: logic restructuring that identifies common sublogic to produce a near-optimal structure and logic minimization that optimizes the logic with respect to the structure obtained in the previous step. In this paper, we consider the problem of identifying common sublogic.

There are two methods for identifying common sublogic: algebraic and Boolean. The algebraic method is fast because the logic function is represented and manipulated as an algebraic expression. Some optimality may however be lost because Boolean identities are not exploited by the algebraic methods. In comparison, the Boolean method is slow, but tends to produce better results.

The algebraic approach is based on the *division* operation, namely, rewriting a function f as $qd + r$ where q, d, and r are the quotient, divisor and remainder, respectively. The theory of division was studied by Brayton and McMullen [3] and well developed in the MIS package [4]. The identification of common sublogic is to extract common subexpressions as divisors. Because the number of divisors is huge, usually only a subset of the divisors are used. For example, *kernels* (cube-free primary divisors) are used in [4] while double- and single-cube divisors are used in [25]. Division can be also carried out by coalgebraic [13] and Boolean [4] methods.

The Boolean approach is based on the *decomposition* operation, namely, rewriting a function $f(X, Y)$ as $f'(g(X), Y)$ where the number of inputs of f' is smaller than that of f. The theory of decomposition was pioneered by Ashenhurst [2], Curtis [8] and Roth and Karp [19]. For representing functions, Karnaugh maps are used in [2, 8, 12], cubes are used in [14, 16, 19] and ordered binary-decision diagrams (OBDDs) [5] are used in [6, 9, 21]. Most of these methods, except [16] and [12], only address single output functions.

A Boolean method for extracting common subfunctions was proposed by Karp [16]. He presented an algorithm for identifying a common subfunction between two functions based on the partitioning of compatible classes [19]. This approach has two shortcomings: first, it does not apply to more than two functions and second, it does not identify more than one shared subfunction. A new Boolean extraction algorithm based on Karnaugh maps was recently proposed in [12]. Because of the size complexity of the Karnaugh map representation, this approach is only applicable to functions with small number of inputs.

In this paper, we describe OBDD-based algorithms for disjunctive and nondisjunctive decompo-

sition of Boolean functions. We then propose two methods for identifying common subfunctions of multiple-output functions. The first method is based on encoding of distinct columns in the stacked decomposition charts of the individual outputs; the second method is based on encoding all possible subfunctions that can be generated with respect to a given subset of input variables. We use OBDDs to represent functions so that our methods can be effectively carried out. Compared to [16], our proposed methods can identify multiple (≥ 2) shared functions from among multiple (≥ 2) functions. Complexity of our methods depends on the size of the bound set while that of the approach in [16] depends on the number of compatible classes. In practical applications, size of the bound sets considered are much smaller than the number of compatible classes. Finally, these methods are applied to Look-Up Table (LUT) based Field Programmable Gate Array (FPGA) synthesis.

The remainder of the paper is organized as follows. Section 2.2 presents OBDD-based algorithms for disjunctive and nondisjunctive decomposition of Boolean functions. Section 2 describes the two methods for identifying common sublogic among multiple Boolean functions and presents the corresponding OBDD-based algorithms. Section 4 shows the application of these ideas and algorithms to the synthesis of LUT-based FPGA devices. Experimental results and concluding remarks are given in section 5 and section 6.

2 OBDD-based Function Decomposition Algorithms

In this section, we first give the background for function decomposition theory. We then present decomposition algorithms for disjunctive, nondisjunctive, and multiple-output decompositions based on the OBDD representation of Boolean functions [5]. These algorithms are based on the concept of cut_set or cut_vector in the OBDD representations.

2.1 Background

Definition 2.1 A function $f(x_0, \ldots, x_{n-1})$ is said to be *decomposable* under *bound set* $\{x_0, \ldots, x_{i-1}\}$ and *free set* $\{x_{i-s}, \ldots, x_{n-1}\}, 0 < i < n, 0 \leq s$ if f can be transformed to $f'(g_0(x_0, \ldots, x_{i-1}), \ldots, g_{j-1}(x_0, \ldots, x_{i-1}), x_{i-s}, \ldots, x_{n-1})$, where $0 < j < i - s$. If s equals 0 then f is *disjunctively decomposable*; otherwise, f is *nondisjunctively decomposable*. If j equals 1 then it is *simply decomposable*. Function f' is referred as the *f-function* and each g_i is referred as a *g-function*. The reduction in variable support is equal to $i - (j + s)$. The above transformation is referred as *decomposition*. If only some of the g-functions are formed, then f is partially

decomposited.

The function decomposition theory has been studied by many researchers [2, 8, 19]. The Ashenhurst-Curtis method [2, 8] is based on an arrangement of the Karnaugh map where the rows correspond to the variables in the free set and the columns correspond to the variables in the bound set. The arrangement is referred as a *decomposition chart*. The number of distinct column vectors is referred as the *column multiplicity*.

The Roth-Karp algorithm [19] is based on the computation of *compatible classes*. Let f be a Boolean function with a *bound set* A_0 and a *free set* A_1, with $\mid A_0 \mid = i$ and $\mid A_1 \mid = n - i$. Let $B_b = B^i$ and $B_f = B^{n-i}$ where $B = \{0, 1\}$. Two variables x_1 and x_2, $x_1 \in B_b$ and $x_2 \in B_b$ are said to be *compatible* if, for all $y \in B_f$, $f(x_1, y) = f(x_2, y)$; otherwise, they are said to be incompatible. Roth and Karp show that a function has a simple disjunctive decomposition with respect to the given bound and free sets if and only if B_b can be partitioned into $k \leq 2$ classes consisting of mutually compatible elements. When a function is completely specified, compatibility is an equivalence relation and k is simply the number of equivalence classes.

The relation between a distinct column of a decomposition chart and a compatible class is bijective. The basic difference between these two methods is in the use of different function representations. One uses the Karnaugh map to represent a function while the other uses covers of the onset and offset for the function.

Theorem 2.1 [8] A function $f(x_0, \ldots, x_{n-1})$ can be transformed to $f'(g_0(x_0, \ldots, x_{i-1}), \ \ldots, \ g_{j-1}(x_0, \ldots, x_{i-1}), x_i, \ldots, x_{n-1})$ if and only if its decomposition chart has column multiplicity $\leq 2^j$.

Definition 2.2 Given a Boolean function f, a bound set B and the decomposition chart C of f and B, the *column_vector* $\mathcal{V}$ of f and B is defined as $\mathcal{V}^f = \langle v_{2^{|B|}-1}, \ldots, v_0 \rangle$ where $v_0 = 0$ and $v_i = j$ if v_i is the j^{th} (from the right) distinct column of $\mathcal{V}^f$. The *column_set* $\mathcal{S}^f$ of f and B is $\{0, \ldots, k-1\}$ if there are k distinct columns in the decomposition chart. The *bit_size* of $\mathcal{V}^f$ or $\mathcal{S}^f$ is $\lceil \log_2 \mid \mathcal{S} \mid \rceil$.

We use $\mathcal{V}^f_{i,j}$ to denote a column_vector with bound set size i and bit_size j and $\mathcal{S}^f_{i,j}$ to denote the column_set of $\mathcal{V}^f_{i,j}$.

Example 2.1 Let $f = x_0x_1x_2x_3 + x_0x_1x_2\bar{x_3}\bar{x_4} + x_0x_1\bar{x_2}\bar{x_4} + x_0\bar{x_1}x_2\bar{x_4} + x_0\bar{x_1}\bar{x_2}\bar{x_3} + \bar{x_0}x_1x_2\bar{x_4} + \bar{x_0}x_1\bar{x_2}\bar{x_3} + \bar{x_0}\bar{x_1}x_2x_3 + \bar{x_0}\bar{x_1}x_2\bar{x_3}x_4 + \bar{x_0}\bar{x_1}\bar{x_2}\bar{x_3}$. If the decomposition chart of $f(x_0, x_1, x_2, x_3, x_4)$ with respect to the bound set $\{x_0, x_1, x_2\}$ and the free set $\{x_3, x_4\}$ is constructed, then there will be three distinct columns, namely $[1100]^t$, $[1011]^t$ and $[1010]^t$. Thus, at least two g-functions are needed, that is, $\mathcal{V}^f_{3,2} = \langle 1, 2, 2, 0, 2, 0, 1, 0 \rangle$, $\mathcal{S}^f_{3,2} = \{0, 1, 2\}$ and the bit_size is 2. □

2.2 Ordered Binary Decision Diagrams

Definition 2.3 [5] An OBDD is a directed acyclic graph consisting of two types of nodes. A *nonterminal* node **v** is represented by a 3-tuple $\langle variable(\mathbf{v}), child_l(\mathbf{v}), child_r(\mathbf{v})\rangle$ where $variable(\mathbf{v}) \in \{x_0, \ldots, x_{n-1}\}$. A terminal node **v** is either **0** or **1**. There exist an index function $index(x) \in \{0, \ldots, n-1\}$ such that for every nonterminal node **v**, either $child_l(\mathbf{v})$ is a terminal node or $index(variable(\mathbf{v})) < index(variable(child_l(\mathbf{v})))$, and either $child_r(\mathbf{v})$ is a terminal node or $index(variable(\mathbf{v})) < index(variable(child_r(\mathbf{v})))$. There is no nonterminal node **v** such that $child_l(\mathbf{v}) = child_r(\mathbf{v})$, and there are no two nonterminal nodes **u** and **v** such that $\mathbf{u} = \mathbf{v}$. The function denoted by $\langle x, \mathbf{v}_l, \mathbf{v}_r\rangle$ is $xf_l + \overline{x}f_r$ where f_l and f_r are the functions denoted by $\mathbf{v}_l$ and $\mathbf{v}_r$, respectively. The functions denoted by **0** and **1** are the constant function 0 and 1, respectively.

We use the following notation.

1. The left edge of a node represent 1 or the true edge and the right edge represents 0 or the false edge.

2. **v** represents both a BDD node and the BDD rooted by node **v**.

3. $index(\mathbf{v})$: the index of the variable associated with node **v**. If **v** is a terminal node, then $index(\mathbf{v}) = n$.

4. $$l\_child(\mathbf{v}, i) = \begin{cases} child_l(\mathbf{v}) & \text{if } index(\mathbf{v}) = i, \\ \mathbf{v} & \text{otherwise.} \end{cases}$$

   $$r\_child(\mathbf{v}, i) = \begin{cases} child_r(\mathbf{v}) & \text{if } index(\mathbf{v}) = i, \\ \mathbf{v} & \text{otherwise.} \end{cases}$$

5. When $B = \{x_0, \ldots, x_{i-1}\}$ represents a bound set, $index(x_0) < \ldots < index(x_{i-1})$, $head(B) = x_0$, $tail(B) = \{x_1, \ldots, x_{i-1}\}$, and $last(B) = x_{i-1}$.

Definition 2.4 Given an OBDD node **v** representing $f(x_0, \ldots, x_{n-1})$ and a bit vector $\langle b_0, \ldots, b_{i-1}\rangle$, the function $eval$ is defined as

$$eval(\mathbf{v}, \langle\rangle) = \mathbf{v},$$
$$eval(\mathbf{v}, \langle b_0, \ldots, b_{i-1}\rangle) = \mathbf{v}',$$

where $\mathbf{v}'$ is the OBDD representing function $f(b_0, \ldots, b_{i-1}, x_i, \ldots, x_{n-1})$. When i is known, we also use $eval(\mathbf{v}, p)$ for $eval(\mathbf{v}, \langle b_0, \ldots, b_{i-1}\rangle)$ where $p = 2^{i-1}b_0 + \ldots + 2^0 b_{i-1}$.

2.3 Disjunctive Decomposition

Definition 2.5 Given an OBDD **v** representing $f(x_0, \ldots, x_{n-1})$ with variable ordering $x_0, \ldots, x_{n-1}$ and bound set $B = \{x_0, \ldots, x_{i-1}\}$, we define

$$cut\_set(\mathbf{v}, B) \;=\; \{\mathbf{u} \mid \mathbf{u} = eval(\mathbf{v}, p), 0 \leq p < 2^i\}.$$

In the above definition, each element in $cut\_set(\mathbf{v}, B)$ corresponds to a distinct column in Ashenhurst-Curtis decomposition charts [2, 8]. Furthermore, $\lceil \log_2 \mid cut\_set(\mathbf{v}, B) \mid \rceil$ determines the minimum number of G-functions required for a decomposition of f under B.

Example 2.2 The OBDD representation and decomposition chart of the function in Ex. 2.1 are shown in Fig. 1 (a) and (b), respectively. Here, $cut\_set(\mathbf{f}, \{x_0, x_1, x_2\}) = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$. Nodes **a**, **b**, and **c** correspond to distinct columns 1100, 1010, and 1011, respectively. Since there are three distinct columns f is not simple decomposable under bound set $\{x_0, x_1, x_2\}$ and free set $\{x_3, x_4\}$. □

Figure 1 goes here.

When the bound variables are on the top of the OBDDs, the computation of the cut_set is straightforward as shown next. The time complexity of computing cut_sets depends on the size of the OBDD representation.

$cut\_set(\mathbf{v}, B)$ /* B is on the top of the OBDD */
{
  if $(index(\mathbf{v}) > index(last(B)))$ return({ **v** });
  else return($cut\_set(child_l(\mathbf{v}), B) \cup cut\_set(child_r(\mathbf{v}), B)$);
}

To move a bound variable x to the top of an OBDD, we create a new BDD node **v** such that $variable(\mathbf{v}) = x$, $child_l(\mathbf{v}) = \mathbf{f_x}$ and $child_r(\mathbf{v}) = \mathbf{f_{\overline{x}}}$ where $\mathbf{f_x}$ and $\mathbf{f_{\overline{x}}}$ are the cofactors of f with respect to x and $\overline{x}$, respectively.

We show how to perform the disjunctive decomposition of a function directly on its OBDD representation.

Algorithm *decomp*:
Given a function f represented in an OBDD $\mathbf{v_f}$ and a bound set B, a disjunctive decomposition with respect to B is carried out by the following steps:

1. Compute the cut_set with respect to B. Let $cut\_set(\mathbf{v}, B) = \{\mathbf{u_0}, \ldots, \mathbf{u_{k-1}}\}$.
2. Encode each node in the cut_set by $\lceil \log_2 k \rceil = j$ bits. Let the encoding of $\mathbf{u_q}$ be q.
3. Construct $\mathbf{v_{f'}}$ to represent function f' by replacing the top part of $\mathbf{v_f}$ by a new set of variables $g_0, \ldots, g_{j-1}$ such that $eval(\mathbf{v_{f'}}, q) = \mathbf{u_q}$ for $0 \leq q < k-1$, $eval(\mathbf{v_{f'}}, q) = \mathbf{u_{k-1}}$ for $k-1 \leq q < 2^j$.
4. Construct $\mathbf{v_{g_p}}$'s to represent g_p's, $0 \leq p < j$ by replacing each node $\mathbf{u}$ with encoding $b_0, \ldots, b_{j-1}$ in the cut_set by terminal node $\mathbf{b_p}$.

Example 2.3 As an example of how $decomp$ works, consider the OBDDs shown in Fig. 2. Since x_4-node in **f** has encoding 01, it has been replaced by terminal nodes **0** and **1** in $\mathbf{g_0}$ and $\mathbf{g_1}$ respectively. The evaluation of $x_0 = 1$, $x_1 = 0$, and $x_2 = 1$ in **f** ends at x_4-node in the cut_set. The evaluation of the same pattern 101 in $\mathbf{g_0}$ and $\mathbf{g_1}$ produce function values 0 and 1 for new variables g_0 and g_1. Then, the evaluation of 01 in $\mathbf{f'}$ also ends at the same x_4-node.

Because there is no encoding 11 in the cut_set, variables g_0 and g_1 can never be 11. We can assign arbitrary value for this pattern. In this example, we assign the left x_3-node so that the left g_1-node can be reduced in $\mathbf{f'}$. □

Figure 2 goes here.

Theorem 2.2 Given an OBDD $\mathbf{v_f}$ with variable ordering $x_0 < \ldots < x_{n-1}$ representing $f(x_0, \ldots, x_{n-1})$, a bound set $B = \{x_0, \ldots, x_{i-1}\}$, the $cut\_set(\mathbf{v_f}, B) = \{\mathbf{u_0}, \ldots, \mathbf{u_{k-1}}\}$ and the $decomp$ algorithm returning OBDDs $\mathbf{v_{f'}}, \mathbf{v_{g_0}}, \ldots, \mathbf{v_{g_{j-1}}}$, then

$$f(x_0, \ldots, x_{n-1}) = f'(g_0(x_0, \ldots, x_{i-1}), \ldots, g_{j-1}(x_0, \ldots, x_{i-1}), x_i, \ldots, x_{n-1})$$

where $f', g_0, \ldots, g_{j-1}$ are the functions denoted by $\mathbf{v_{f'}}, \mathbf{v_{g_0}}, \ldots, \mathbf{v_{g_{j-1}}}$, respectively.

2.4 Nondisjunctive Decomposition

Before describing how to perform nondisjunctive decomposition based on OBDD representation, we extend the concept of cut_set in the following definition.

Definition 2.6 Let $R = \{x_0, \ldots, x_{s-1}\}$, $S = \{x_s, \ldots, x_{i-1}\}$, and $T = \{x_i, \ldots, x_{n-1}\}$, $0 < s < i < n$. Given an OBDD **v** representing $f(x_0, \ldots, x_{n-1})$, a bound set $R \cup S$, and a free set $S \cup T$, we define

$$cut\_set\_nd(\mathbf{v}, R, S, p) = \{eval(\mathbf{w}, p) \mid \mathbf{w} \in cut\_set(\mathbf{v}, R)\},$$

where $0 \leq p < 2^{|S|}$.

With the above definition, $cut\_set(\mathbf{v}, B)$ can be represented by $cut\_set\_nd(\mathbf{v}, B, \phi, 0)$. In the following, we present a pseudo code for computing cut_set_nd and an example of it.

$cut\_set\_nd(\mathbf{v}, R, S, p)$ $\quad /* \, 0 \leq p < 2^{|S|} \, */$
{
  if $(index(\mathbf{v}) < index(head(S)))$
    return($cut\_set\_nd(child_l(\mathbf{v}), R, S, p) \cup cut\_set\_nd(child_r(\mathbf{v}), R, S, p)$);
  else if $(index(head(S)) \leq index(\mathbf{v}) \leq index(last(S)))$ {
    $q = 2^{index(last(S)) - index(\mathbf{v})}$;
    if ($q \leq p$) $\quad /*$ then traverse down through left edge $*/$
      return($cut\_set\_nd(child_l(\mathbf{v}), R, S, p - q)$);
    else return($cut\_set\_nd(child_r(\mathbf{v}), R, S, p)$);
  }
  else return($\{\mathbf{v}\}$);
}

Example 2.4 The OBDD in Figure 1 (a) has

$$cut\_set\_nd(\mathbf{f}, \{x_0, x_1\}, \{x_2\}, 0) = \{\mathbf{a}, \mathbf{b}\},$$
$$cut\_set\_nd(\mathbf{f}, \{x_0, x_1\}, \{x_2\}, 1) = \{\mathbf{b}, \mathbf{c}\},$$
$$cut\_set\_nd(\mathbf{f}, \{x_0\}, \{x_1, x_2\}, 0) = \{\mathbf{a}\},$$
$$cut\_set\_nd(\mathbf{f}, \{x_0\}, \{x_1, x_2\}, 1) = \{\mathbf{b}, \mathbf{c}\},$$
$$cut\_set\_nd(\mathbf{f}, \{x_0\}, \{x_1, x_2\}, 2) = \{\mathbf{a}, \mathbf{b}\}, \text{ and}$$
$$cut\_set\_nd(\mathbf{f}, \{x_0\}, \{x_1, x_2\}, 3) = \{\mathbf{b}, \mathbf{c}\}.$$

□

Algorithm $decomp\_nd$:
Given a function f represented in an OBDD $\mathbf{v_f}$, a bound set $\{x_0, \ldots, x_s, \ldots, x_{i-1}\}$, and a free set $\{x_s, \ldots, x_{i-1}, \ldots, x_{n-1}\}$, a nondisjunctive decomposition with respect to the given bound set and free set is carried out in the following steps:

1. Compute $cut\_set\_nd(\mathbf{v_f}, R, S, r)$ for $0 \leq r < 2^{|S|}$ where $R = \{x_0, \ldots, x_{s-1}\}$ and $S = \{x_s, \ldots, x_{i-1}\}$. Let $cut\_set\_nd(\mathbf{v_f}, R, S, r) = \{\mathbf{u_{r,0}}, \ldots, \mathbf{u_{r,l}}\}$, $max\{\mid cut\_set\_nd(\mathbf{v_f}, R, S, r) \mid \} = k$, and $j = \lceil \log_2 k \rceil$.

2. Construct $\mathbf{v_{f'}}$ to represent function f' in two steps:

   (a) Construct $\mathbf{v_q}$, $0 \leq q < k$, such that $eval(\mathbf{v_q}, r) = \mathbf{u_{q,r}}$ where $\mathbf{u_{q,r}}$ is the q^{th} element in $cut\_set\_nd(\mathbf{v_f}, R, S, r)$ or the last element if $q > \mid cut\_set\_nd(\mathbf{v_f}, R, S, r) \mid$.

   (b) Construct $\mathbf{v_{f'}}$ such that $eval(\mathbf{v_{f'}}, q) = \mathbf{v_q}$ for $0 \leq q < k-1$ and $eval(\mathbf{v_{f'}}, q) = \mathbf{v_{k-1}}$ for $k-1 \leq q < 2^j$.

3. Construct $\mathbf{v_{g_p}}$'s to represent g_p's for $0 \leq p < j$:
   Replace each node $\mathbf{u_{q,r}}$ (q^{th} node of $cut\_set\_nd(\mathbf{v_f}, R, S, r)$) from $\mathbf{v_f}$ by the terminal node whose value is b_{q_p} where b_{q_p} is the p^{th} bit from the least significant bit of integer q.

Note that, a node **u** may be the i^{th} element of $cut\_set\_nd(\mathbf{v_f}, R, S, r_1)$ and the j^{th} element of $cut\_set\_nd(\mathbf{v_f}, R, S, r_2)$ which requires different encodings for **u**. This does not cause a problem because we can first duplicate the node **u** and then assign each copy a different encoding.

Example 2.5 One possible nondisjunctive decomposition of the OBDD in Fig. 1 (a) with respect to the bound set $\{x_0, x_1, x_2\}$ and the free set $\{x_2, x_3, x_4\}$ is shown in Fig. 3. In this decomposition, we use the following coding: $\{\mathbf{a} = \mathbf{u_{0,0}}, \mathbf{b} = \mathbf{u_{0,1}}\} = cut\_set\_nd(\mathbf{v_f}, \{x_0, x_1\}, \{x_2\}, 0)$ and $\{\mathbf{c} = \mathbf{u_{1,1}}, \mathbf{b} = \mathbf{u_{1,0}}\} = cut\_set\_nd(\mathbf{v_f}, \{x_0, x_1\}, \{x_2\}, 1)$. □

Figure 3 goes here.

Theorem 2.3 Given an OBDD $\mathbf{v_f}$ with variable ordering $x_0, \ldots, x_{n-1}$ representing $f(x_0, \ldots, x_{n-1})$, and $k = max\{|\ cut\_set\_nd(\mathbf{v_f}, \{x_0, \ldots, x_{s-1}\}, \{x_s, \ldots, x_{i-1}\}, r)\ |\ 0 \leq r < 2^{i-s}\}$, $2^{j-1} < k \leq 2^j$, the algorithm $decomp\_nd$ returns $j+1$ OBDDs $\mathbf{v_{f'}}, \mathbf{v_{g_0}}, \ldots, \mathbf{v_{g_{j-1}}}$ such that
$f(x_0, \ldots, x_{n-1}) = f'(g_0(x_0, \ldots, x_{i-1}), \ldots, g_{j-1}(x_0, \ldots, x_{i-1}), x_{s-1}, \ldots, x_{i-1}, \ldots, x_{n-1})$
where $f', g_0, \ldots, g_{j-1}$ are the functions denoted by $\mathbf{v_{f'}}, \mathbf{v_{g_0}}, \ldots, \mathbf{v_{g_{j-1}}}$, respectively.

3 Common Subfunction Extraction

In this section, we present two methods to extract common subfunctions from multiple Boolean functions. The first method is based on the stacking of the decomposition charts of the individual outputs; the second method is based on the examination of all possible g-functions that can be generated. We first describe an algorithm for generating column_vectors using the OBDD representation. We then present a multiple-output decomposition algorithm that is useful when column encoding and shared subfunction encoding is used.

3.1 Column Encoding

Our first method is called *column encoding* which is carried out as follows: we first stack up the decomposition charts for individual functions and then encode the distinct column patterns. This is equivalent to finding a common encoding for all functions.

Example 3.1 Consider a multiple-output function F: $f_0 = x_0x_4 + x_1x_2x_4$, $f_1 = x_0x_3 + x_1x_3 + \bar{x}_0\bar{x}_1\bar{x}_4$ and $f_2 = x_0x_3x_4 + x_2x_3x_4$ with bound set $B = \{x_0, x_1, x_2\}$.

If we stack the decomposition charts of F, then there will be four distinct column patterns. We can thus use two bits to encode each column pattern (defining two g-functions g_0 and g_1). The f-functions are determined from combining identical columns of the stacked decomposition chart. In particular,

$$
\begin{aligned}
&g_0(x_0, x_1, x_2) = x_0 + x_1, \\
&g_1(x_0, x_1, x_2) = x_0 + x_2, \\
&f'_0(g_0, g_1, x_3, x_4) = g_0g_1x_4, \\
&f'_1(g_0, g_1, x_3, x_4) = g_0x_3 + \bar{g}_0\bar{x}_4, \text{ and} \\
&f'_2(g_0, g_1, x_3, x_4) = g_1x_3x_4.
\end{aligned}
$$

□

Note that the above method can identify common subexpressions that algebraic division based methods cannot. After the above decomposition, the literal count of the resulting circuit is 14. On the other hand, the best we could achieve by the algebraic method, is 16 as shown below:

$$
\begin{aligned}
y_0 &= x_0 + x_1, \\
y_1 &= x_0 + x_2, \\
f'_0 &= x_0x_4 + x_1x_2x_4, \\
f'_1 &= y_0x_3 + \overline{y}_0\overline{x}_4, \text{ and} \\
f'_2 &= y_1x_3x_4.
\end{aligned}
$$

Lemma 3.1 Given a multiple-output Boolean function $F = \langle f_0, \ldots, f_{m-1} \rangle$ on variable set X and bound set $B \subset X$, if the column multiplicity of the stacking of individual decomposition charts is k such that $2^{j-1} < k \leq 2^j$, then F can be transformed to the following:

$$\langle f'_0(g_0(B), \ldots, g_{j-1}(B), X - B), \ldots, f'_{m-1}(g_0(B), \ldots, g_{j-1}(B), X - B) \rangle.$$

To perform column encoding, we use the following operator.

Definition 3.1 Given an OBDD **v** representing $f(x_0, \ldots, x_{n-1})$ with variable ordering $x_0 < \ldots < x_{n-1}$ and bound set $B = \{x_0, \ldots, x_{i-1}\}$, we define

$$cut\_vector(\mathbf{v}, B) \quad = \quad \langle eval(\mathbf{v}, 2^i - 1), \ldots, eval(\mathbf{v}, 0) \rangle.$$

Example 3.2 The OBDD representation of the multiple-output function in Example 3.1 is shown in Fig. 4. With the bound set $B = \{x_0, x_1, x_2\}$, we have the following cut_vectors:

$$
\begin{aligned}
cut\_vector(\mathbf{f_0}, B) &= \langle \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{0}, \mathbf{0}, \mathbf{0} \rangle, \\
cut\_vector(\mathbf{f_1}, B) &= \langle \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{c} \rangle, \text{ and} \\
cut\_vector(\mathbf{f_2}, B) &= \langle \mathbf{d}, \mathbf{d}, \mathbf{d}, \mathbf{d}, \mathbf{d}, \mathbf{0}, \mathbf{d}, \mathbf{0} \rangle.
\end{aligned}
$$

Note that each node corresponds to a column in the decomposition chart. For example, node **a** corresponds to column $[1010]^t$ while node **b** corresponds to column $[1100]^t$. □

Figure 4 goes here.

In the following procedure for $cut\_vector(\mathbf{v}, B)$, we assume that the bound variables B are on top of the OBDD. In addition, if $B = \{x_0, \ldots, x_{i-1}\}$, then $index(x_0) < \ldots < index(x_{i-1})$, $head(B) = x_0$, and $rest(B) = \{x_1, \ldots, x_{i-1}\}$.

$cut\_vector(\mathbf{v}, B)$
{
  if ($B == \phi$) return($\langle \mathbf{v} \rangle$);
  if ($index(\mathbf{v}) == index(head(B))$)
    return($concatenate(cut\_vector(child_l(\mathbf{v}), rest(B)), cut\_vector(child_r(\mathbf{v}), rest(B))$);
  else $/* index(\mathbf{v}) > index(head(B)) */$
    return($concatenate(cut\_vector(\mathbf{v}, rest(B)), cut\_vector(\mathbf{v}, rest(B))$);
}

Definition 3.2 Operator $column\_encode$ is defined as follows:

$$column\_encode([v_{0,2^i-1}, \ldots, v_{0,0}], \ldots [v_{m-1,2^i-1}, \ldots, v_{m-1,0}]) = [u_{2^i-1}, \ldots, u_0]$$

where $u_0 = 0$ and $u_p = q$ if $[v_{0,p}, \ldots, v_{m-1,p}]$ is the q^{th} distinct m-tuple of $[v_{0,0}, \ldots, v_{m-1,0}]$, $\ldots$, $[v_{0,2^i-1}, v_{m-1,2^i-1}]$.

Example 3.3 $column\_encode(\langle 1,1,1,1,1,0,0,0\rangle, \langle 1,1,1,1,1,1,0,0\rangle, \langle 2,2,2,2,2,1,0,0\rangle) = \langle 2,2,2,2,2,1,0,0\rangle$.

Definition 3.3 Operator $select$ is defined as

$$select(j, \langle v_{0,2^i-1}, \ldots, v_{0,0}\rangle, \ldots, \langle v_{m-1,2^i-1}, \ldots, v_{m-1,0}\rangle) = \langle v_{0,k}, \ldots, _{m-1,k} \rangle$$

where $\langle v_{0,k}, \ldots, v_{m-1,k}\rangle$ is the j^{th} distinct *m*-tuple of $\langle v_{0,0}, \ldots, v_{m-1,0}\rangle$, $\ldots$, $\langle v_{0,2^i-1}, \ldots, v_{m-1,2^i-1}\rangle$. If j is greater than the number of distinct m-tuples, then $\langle v_{0,k}, \ldots, v_{m-1,k}\rangle$ is the last distinct m-tuple.

Example 3.4 Let $\mathcal{V}^{f_0} = \langle 2,2,1,1,1,0,0,0\rangle$ and $\mathcal{V}^{f_1} = \langle 2,2,2,2,2,1,1,0\rangle$, then we have the following:

$column\_encode(\mathcal{V}^{f_0}, \mathcal{V}^{f_1}) = \langle 3,3,2,2,2,1,1,0\rangle$,
$select(0, \mathcal{V}^{f_0}, \mathcal{V}^{f_1}) = \langle 0,0\rangle$,
$select(1, \mathcal{V}^{f_0}, \mathcal{V}^{f_1}) = \langle 0,1\rangle$,
$select(2, \mathcal{V}^{f_0}, \mathcal{V}^{f_1}) = \langle 1,2\rangle$, and
$select(3, \mathcal{V}^{f_0}, \mathcal{V}^{f_1}) = \langle 2,2\rangle$. □

Given a multiple output function $\langle f_0, \ldots, f_{m-1} \rangle$ represented by a vector of OBDDs and a bound set $\{x_0, \ldots, x_{i-1}\}$, after the computation of cut_vectors and column encoding, the g- and f- functions are constructed as follows: For any input pattern $b = b_0, \ldots, b_{i-1}$, if the evaluation of b on f_k, $0 \leq k < m$, ends at node **v** with encoding $e_0, \ldots, e_{j-1}$, then we let $g_0(b), \ldots, g_{j-1}(b)$ produce function values $e_0, \ldots, e_{j-1}$ and $f'_k(g_0(b), \ldots, g_{j-1}(b), x_i, \ldots, x_{n-1})$ result in node **v**. Consequently, $f_k(b_0, \ldots, b_{i-1}, x_i, \ldots, x_{n-1}) = f'_k(g_0(b_0, \ldots, b_{i-1}), \ldots,$
$g_{j-1}(b_0, \ldots, b_{i-1}), x_i, \ldots, x_{n-1})$ for every input pattern $b_0, \ldots, b_{i-1}$ and $0 \leq k < m$. The following procedure gives the details of our algorithm.

Algorithm $decomp\_mo\_ce$:
Given a vector of OBDDs $\langle \mathbf{v}_0, \ldots, \mathbf{v}_{m-1} \rangle$ representing $\langle f_0(x_0, \ldots, x_{n-1}), \ldots, f_{m-1}(x_0, \ldots, x_{n-1}) \rangle$ with variable ordering $x_0, \ldots, x_{n-1}$ and a bound set $B = \{x_0, \ldots, x_{i-1}\}$.

1. Compute $\mathcal{V}^k = cut\_vector(\mathbf{v}_k, B) = \langle \mathbf{u}_{k,2^i-1}, \ldots, \mathbf{u}_{k,0} \rangle$, $0 \leq k < m$.

2. Compute $\mathcal{V}_{i,j} = column\_encode(\mathcal{V}^0, \ldots, \mathcal{V}^{m-1})$. Encode each element v_p, $(0 \leq p < 2^i)$ of $\mathcal{V}_{i,j}$ by j bits $d_{p,0} \ldots d_{p,j-1}$ such that $v_p = 2^{j-1}d_{p,0} + \ldots + 2^0 d_{p,j-1}$.

3. Construct each g-function $g_q(x_0, \ldots, x_{i-1})$, $0 \leq q < j$, as
$$g_q(x_0, \ldots, x_{i-1}) = [d_{2^i-1,q} \ldots d_{0,q}] \quad \text{(truth table of } g_q\text{)}$$
where $g_q(b_0, \ldots, b_{i-1}) = d_{p,q}$ if $2^{i-1}b_0 + \ldots + 2^0 b_{i-1} = p$.

4. Compute $select(r, \mathcal{V}^0, \ldots, \mathcal{V}^{m-1}) = \langle \mathbf{u}_{0,s_r}, \ldots, \mathbf{u}_{m-1,s_r} \rangle$, $0 \leq r < 2^j$, $0 \leq s_r < 2^i$, s_r is any l such that $v_l = r$.

5. Construct each f-function $f'_k(g_0, \ldots, g_{j-1}, x_i, \ldots, x_{n-1})$, $0 \leq k < m$, as
$$f'_k(b_0, \ldots, b_{j-1}, x_i, \ldots, x_{n-1}) = [\mathbf{u}_{k,s_{2^j-1}} \ldots \mathbf{u}_{k,s_0}],$$
where $f'_k(b_0, \ldots, b_{j-1}, x_i, \ldots, x_{n-1}) = \mathbf{u}_{s_r,k}$ if $2^{j-1}b_0 + \ldots + 2^0 b_{j-1} = r$.

Example 3.5 The application of $decomp\_mo\_ce$ on the multiple-output function in Example 3.1 is summarized as follows:

1. $cut\_vector(f_0, B) = \langle \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{0}, \mathbf{0}, \mathbf{0} \rangle = \mathcal{V}^0_{3,1}$,
$cut\_vector(f_1, B) = \langle \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{c} \rangle = \mathcal{V}^1_{3,1}$,
$cut\_vector(f_2, B) = \langle \mathbf{d}, \mathbf{d}, \mathbf{d}, \mathbf{d}, \mathbf{d}, \mathbf{0}, \mathbf{d}, \mathbf{0} \rangle = \mathcal{V}^2_{3,1}$ (see Fig. 4),

2. $column\_encode(\mathcal{V}^0_{3,1}, \mathcal{V}^1_{3,1}, \mathcal{V}^2_{3,1}) = \langle 3, 3, 3, 3, 3, 2, 1, 0 \rangle \Rightarrow \langle 11, 11, 11, 11, 11, 10, 01, 00 \rangle$,

3. $g_0(x_0, x_1, x_2) = [11111100]$,
   $g_1(x_0, x_1, x_2) = [11111010]$,

4. $select(0, \mathcal{V}^0_{3,1}, \mathcal{V}^1_{3,1}, \mathcal{V}^2_{3,1}) = \langle \mathbf{0}, \mathbf{c}, \mathbf{0} \rangle$,
   $select(1, \mathcal{V}^0_{3,1}, \mathcal{V}^1_{3,1}, \mathcal{V}^2_{3,1}) = \langle \mathbf{0}, \mathbf{c}, \mathbf{d} \rangle$,
   $select(2, \mathcal{V}^0_{3,1}, \mathcal{V}^1_{3,1}, \mathcal{V}^2_{3,1}) = \langle \mathbf{0}, \mathbf{b}, \mathbf{0} \rangle$,
   $select(3, \mathcal{V}^0_{3,1}, \mathcal{V}^1_{3,1}, \mathcal{V}^2_{3,1}) = \langle \mathbf{a}, \mathbf{b}, \mathbf{d} \rangle$,

5. $f'_0(g_0, g_1, x_3, x_4) = [\mathbf{a000}]$,
   $f'_1(g_0, g_1, x_3, x_4) = [\mathbf{bbcc}]$, and
   $f'_2(g_0, g_1, x_3, x_4) = [\mathbf{d0d0}]$.

The resulting g- and f-functions are shown in Fig. 5 (a) and (b), respectively. To see $f_k(x_0, x_1, x_2, x_3, x_4) = f'_k(g_0(x_0, x_1, x_2), g_1(x_0, x_1, x_2), x_3, x_4)$, consider the evaluation of $x_0 = 0$, $x_1 = 1$, and $x_2 = 0$ on f_1, g_0, g_1, and f'_1 as an example:

$$f_1(0, 1, 0, x_3, x_4) = \mathbf{b} = x_3,$$
$$g_0(0, 1, 0) = 1,$$
$$g_1(0, 1, 0) = 0, \text{ and}$$
$$f'_1(1, 0, x_3, x_4) = \mathbf{b} = x_3.$$

□

Figure 5 goes here.

The following theorem proves the correctness of $decomp\_mo\_ce$.

Theorem 3.1 The $decomp\_mo\_ce$ algorithm performs the following transformation

$$f_k(x_0, \ldots, x_{n-1}) = f'_k(g_0(x_0, \ldots, x_{i-1}), \ldots, g_{j-1}(x_0, \ldots, x_{i-1}), x_i, \ldots, x_{n-1}),$$

where $0 \leq k < m, 0 < i < n$.

In practice, it is unlikely that a multiple-output function is decomposable. For example, if we directly apply column encoding to every output, then the resulting column_vector for the stacked decomposition chart will often be $\mathcal{V}_{i,i}$. Output partitioning is thus useful to improve decomposability. We partition the outputs into groups such that the column_vector of the stacked

decomposition chart for each group corresponds to a decomposable function (i.e., the number of required g-functions required is less than size of the bound set) and the total number of g-functions required to implement all groups is minimum. This problem is formulated as follows.

Definition 3.4 Given a set of column_vectors $\mathcal{V}_{i,j_k}^{f_k}$'s with respect to m Boolean functions $F = \langle f_0, \ldots, f_{m-1} \rangle$ and bound set B, partition this set into $P_0, \ldots, P_{\ell-1}$ such that the resulting column_vector $\mathcal{V}_{i,j_q}^{q}$ of each P_q satisfies $i > j_q$ and $\sum_{q=0}^{l-1} j_q$ is minimum.

We use the following greedy algorithm to solve this problem.

Algorithm $output\_grouping$:
Assume every column_vector $\mathcal{V}_{i,j_k}^{f_k}$ satisfies $i > j_k$.

1. Order $\mathcal{V}_{i,j_k}^{f_k}$ in nonincreasing order of $\mid \mathcal{S}_{i,j_k}^{f_k} \mid$. Initialize $\mathcal{V}_{i,j_0}^{0}$ to the null set.

2. Starting from the first element of the above list, merge as many column_vectors $\mathcal{V}_{i,j_k}^{f_k}$'s as possible into $\mathcal{V}_{i,j_0}^{0}$ as long as column_encode$(\mathcal{V}_{i,j_0}^{0}, \mathcal{V}_{i,j_k}^{f_k}) = \mathcal{V}_{i,j_r}$ satisfies $i > j_r$. As soon as $i \leq j_r$, initiate a new group of outputs. Repeat until all column_vectors are processed.

The above algorithm is based on the following observation: A $\mathcal{V}_{i,j_k}^{f_k}$ with larger $\mathcal{S}_{i,j_k}^{f_k}$ has better chance to *contain* another $\mathcal{V}_{i,k_l}^{f_l}$ with smaller $\mathcal{S}_{i,k_l}^{f_l}$. For example, if we have column_encode($\langle 2,2,2,1,1,0,0,0 \rangle$, $\langle 3,3,3,2,2,1,1,0 \rangle$) = $\langle 3,3,3,2,2,1,1,0 \rangle$, then all the g-functions required for the first $\mathcal{V}_{3,2}$ are contained in those for the second $\mathcal{V}_{3,2}$. Thus, by putting these two column_vectors into the same set, we only need two g-functions for both functions.

Example 3.6 Given a set of column_vectors as following:

$\mathcal{V}^{f_0} = \langle 2,2,2,2,1,1,0,0 \rangle$,
$\mathcal{V}^{f_1} = \langle 3,3,2,2,2,2,1,0 \rangle$,
$\mathcal{V}^{f_2} = \langle 2,2,1,1,1,1,0,0 \rangle$,
$\mathcal{V}^{f_3} = \langle 2,2,2,1,1,0,0,0 \rangle$,
$\mathcal{V}^{f_4} = \langle 2,2,2,2,2,2,1,0 \rangle$,
$\mathcal{V}^{f_5} = \langle 2,2,2,1,1,1,0,0 \rangle$.

If we apply column_encode on every outputs without using output_grouping algorithm, then column_encode($\mathcal{V}^{f_0}, \ldots, \mathcal{V}^{f_5}$) = $\langle 6,6,5,4,3,2,1,0 \rangle$ which is $\mathcal{V}_{3,3}$.

If we apply output_grouping algorithm, then three groups will be produced.

group 1: column_encode($\mathcal{V}^{f_1}, \mathcal{V}^{f_4}$) = $\langle 3,3,2,2,2,2,1,0 \rangle = \mathcal{V}_{3,2}$.

group 2: column_encode($\mathcal{V}^{f_0}, \mathcal{V}^{f_2}$) = $\langle 3,3,2,2,1,1,0,0\rangle = \mathcal{V}_{3,2}$.
group 3: column_encode($\mathcal{V}^{f_3}, \mathcal{V}^{f_5}$) = $\langle 3,3,3,2,2,1,0,0\rangle = \mathcal{V}_{3,2}$. □

3.2 Shared Subfunction Encoding

After computing the column vectors of a multiple output function with respect to a bound set B, it is possible to develop a decomposition scheme that minimizes the number of required g-functions by sharing these functions among the original functions as described next.

Example 3.7 Let $\mathcal{V}_{3,2}^{f1} = \langle 2,3,2,0,1,2,1,0\rangle$ and $\mathcal{V}_{3,2}^{f2} = \langle 1,0,1,3,2,1,1,0\rangle$, then column_encode($\mathcal{V}_{3,2}^{f1}, \mathcal{V}_{3,2}^{f2}$) $= \langle 2,5,2,4,3,2,1,0\rangle = \mathcal{V}_{3,3}$. If we encode 0 as 00, 1 as 01, 2 as 10, and 3 as 11 for both $\mathcal{V}_{3,2}^{f1}$ and $\mathcal{V}_{3,2}^{f2}$, then we have

23201210	10132110
[11100100]	[00011000]
[01001010]	[10110110]

which requires four g-functions: [11100100], [01001010], [00011000], and [10110000].

If we encode 0 as 00, 1 as 11, 2 as 01, and 3 as 10 for $\mathcal{V}_{3,2}^{f1}$, and 0 as 00, 1 as 01, 2 as 11, and 3 as 10 for $\mathcal{V}_{3,2}^{f2}$, then we have

23201210	10132110
[01001010]	[00011000]
[10101110]	[10101110]

which requires only three g-functions: [01001010], [00011000], and [10101110]. □

To achieve the above type of subfunction sharing, we present a *shared subfunction encoding* scheme as follows: For each output function, we compute all g-functions which can be produced from every possible encoding. We then identify the minimum number of g-functions which produce valid encodings for every function with respect to the given bound set.

The optimum g-function sharing can only be found if we allow the assignment of multiple function values (codes) to the same pattern (multi-coding). This is, however, very expensive. We trade some optimality for computational efficiency by requiring that a unique function value (code) be assigned to each pattern (uni-coding). In this case, for $\mathcal{S}_{i,j}$ with set size k, there will be $C_k^{2^j} k!$ different encodings. In the remaining of this paper, we only consider uni-coding.

Although there are many different encodings, the number of different g-functions which can be generated from these encodings is small as described next.

Definition 3.5 Given $cut\_vector\ \mathcal{V}_{i,j}$ and $cut\_set\ \mathcal{S}_{i,j}$ for some decomposition, let $\mathcal{S}_{i,j}$ be partitioned into S_0 and S_1 such that $0 \in S_0$, $\mid S_0 \mid \leq 2^{j-1}$ and $\mid S_1 \mid \leq 2^{j-1}$. A *permissible g-function* of $\mathcal{V}_{i,j}$ with respect to S_0 and S_1, denoted by $pg_{\mathcal{V}_{i,j},S_0,S_1}$, is defined as:

$$pg_{\mathcal{V}_{i,j},S_0,S_1} = [b_{2^i-1} \ldots b_0],$$

where $b_p = 0$ if $v_p \in S_0$ and $b_p = 1$ otherwise, $0 \leq p < 2^i$.

The *pg-set* of $\mathcal{V}_{i,j}$ is the set of all pg's of $\mathcal{V}_{i,j}$ obtained by enumerating all two-block partitions of $\mathcal{S}_{i,j}$ into S_0 and S_1 satisfying the conditions stated above. The restrictions on $\mid S_0 \mid$ and $\mid S_1 \mid$ are needed to ensure that a valid j-bit encoding of the nodes in the cut_set of f with respect to B can be found.

Example 3.8 Let $\mathcal{V}_{3,2} = \langle 2, 2, 1, 1, 1, 1, 0, 0 \rangle$, then

$$\begin{array}{lcl} pg_{\mathcal{V}_{3,2},\{0\},\{1,2\}} & = & [11111100], \\ pg_{\mathcal{V}_{3,2},\{0,1\},\{2\}} & = & [11000000] \text{ and} \\ pg_{\mathcal{V}_{3,2},\{0,2\},\{1\}} & = & [00111100]. \end{array}$$

The pg-set of $\mathcal{V}_{3,2}$ is $\{[11111100], [11000000], [00111100]\}$. Note that we need not consider $pg_{\mathcal{V}_{3,2},\{1\},\{0,2\}}$ because it is equal to $\overline{pg_{\mathcal{V}_{3,2},\{0,2\},\{1\}}}$ and that is why we force $0 \in S_0$ at $\mathcal{S}_{i,j}$ partition in definition 3.5. □

The cardinality of the pg-set of $\mathcal{V}_{i,j}$ is given by the following equation:

$$\mid pg\_set \mid \ = \ C^{k-1}_{k-2^{j-1}-1} + \ldots + C^{k-1}_{2^{j-1}-1} = \sum_{l=k-2^{j-1}}^{2^{j-1}} C^{k-1}_{l-1}$$

where $2^{j-1} < \mid \mathcal{S}_{i,j} \mid = k \leq 2^j$. Because neither $\mid S_0 \mid$ nor $\mid S_1 \mid$ can exceed 2^{j-1}, the minimum and maximum sizes of S_0 are $k - 2^{j-1}$ and 2^{j-1}, respectively. $k-1$ and $l-1$ are used because $0 \in S_0$.

Example 3.9 For $\mid \mathcal{S}_{i,3} \mid = k = 3, 4, 6$, and 8, the cardinalities of their pg-sets are computed as follows:

$$\begin{array}{ll} k = 3\text{:} & C^2_0 + C^2_1 = 1 + 2 = 3, \\ k = 4\text{:} & C^3_1 = 3, \\ k = 6\text{:} & C^5_1 + C^5_2 + C^5_3 = 5 + 10 + 10 = 25, \text{ and} \\ k = 8\text{:} & C^7_3 = 35. \end{array}$$

□

Note that there are $C_k^{2^j} k! = 40,320$ different encodings for $\mid \mathcal{S}_{i,3} \mid = 8$ while only 35 different pg-functions can be generated.

Because each pg function defines a partial (1-bit) encoding, a complete encoding of $\mathcal{S}_{i,j}$ is determined by selecting exactly j pg's. However, not every subset of a pg-set, forms a valid partial encoding. This leads to the following definition of *compatibility of pg's*.

Definition 3.6 A *k-bit partition* P^k of $\mathcal{S}_{i,j}$ is defined as a partitioning of $\mathcal{S}_{i,j}$ into $P^k = \{S_0, \ldots, S_{2^k-1}\}$, $k \leq j$ such that $\forall S_q \in P^k, \mid S_q \mid \leq 2^{j-k}$.

Definition 3.7 A *partial k-bit encoding* of P^k is defined as follows:

$$\text{if } s \in S_p, \text{ then } s \text{ is encoded as } b_0 \ldots b_{k-1} x_k \ldots x_{j-1}$$

where $2^{k-1}b_0 + \ldots + 2^0 b_{k-1} = p$ and $x_k, \ldots, x_{j-1}$ are unassigned bits.

Thus, a k-bit partition defines a k-bit encoding of the $\mathcal{S}_{i,j}$. Note that if there is a S_q such that $\mid S_q \mid > 2^{j-k}$, then we cannot find a $j-k$ bit encoding of S_q and, therefore, cannot generate a valid j bit encoding of $\mathcal{S}_{i,j}$.

Definition 3.8 Given k- and l-bit partitions $P^k = \{S_0, \ldots, S_{2^k-1}\}$ and $Q^l = \{T_0, \ldots, T_{2^l-1}\}$ of $\mathcal{S}_{i,j}$ and assuming $k + l \leq j$, we define a merge operator $\mathcal{M}$ as follows:

$$\mathcal{M}(P^k, Q^l) = \{R_0, \ldots, R_{2^{k+l}-1}\},$$

where $R_{2^l p+q} = S_p \cap T_q, S_p \in P^k, T_q \in Q^l$. If $\mathcal{M}(S^k, T^l)$ is a $(k+l)$-bit-partition of $\mathcal{S}_{i,j}$, then P^k and Q^l are *compatible*. That is, if every $R_z \in \mathcal{M}(P^k, Q^l)$ satisfies $\mid R_z \mid \leq 2^{k+l}$, then P^k and Q^l are compatible.

In other words, if P^k and Q^l are compatible, then they can be used together to produce a $(k+l)$-bit encoding of $\mathcal{S}_{i,j}$.

Example 3.10 Let $\mathcal{S}_{5,4} = \{0,1,2,3,4,5,6,7,8,9\}$. some of 1-bit partitions of $\mathcal{S}_{5,4}$ are:

$$\begin{aligned} P^1 &= \{\{0,1,2,3,4\},\{5,6,7,8,9\}\}, \\ Q^1 &= \{\{0,1,2,5,6,7\},\{3,4,8,9\}\}, \\ U^1 &= \{\{0,1,2,3,4,5\},\{6,7,8,9\}\}, \\ V^1 &= \{\{0,2,4,6,8\},\{1,3,5,7,9\}\}. \end{aligned}$$

We have:

$$
\begin{aligned}
\mathcal{M}(P^1, Q^1) &= \{\{0,1,2\},\{3,4\},\{5,6,7\},\{8,9\}\} = P^2, \\
\mathcal{M}(Q^1, U^1) &= \{\{0,1,2,5\},\{6,7\},\{3,4\},\{8,9\}\}, \\
\mathcal{M}(P^1, U^1) &= \{\{0,1,2,3,4\},\{\},\{5\},\{6,7,8,9\}\}, \\
\mathcal{M}(P^2, V^1) &= \{\{0,2\},\{1\},\{4\},\{3\},\{6\},\{5,7\},\{8\},\{9\}\}, \\
\mathcal{M}(P^2, U^1) &= \{\{0,1,2\},\{\},\{3,4\},\{\},\{5\},\{6,7\},\{\},\{8,9\}\}.
\end{aligned}
$$

P^1 and Q^1, Q^1 and U^1, and P^2 and V^1 are compatible, but P^1 and U^1, and P^2 and U^1 are not compatible. Note that this example shows that compatibility relation is *not* transitive.

P^1 defines the following encoding:

$$
\begin{aligned}
&0, 1, 2, 3 \text{ and } 4 \text{ are encoded by } 0g_1g_2g_3, \\
&5, 6, 7, 8 \text{ and } 9 \text{ are encoded by } 1g_1g_2g_3.
\end{aligned}
$$

P^2 defines the following encoding:

$$
\begin{aligned}
&0, 1 \text{ and } 2 \text{ are encoded by } 00g_2g_3, \\
&3 \text{ and } 4 \text{ are encoded by } 01g_2g_3, \\
&5, 6 \text{ and } 7 \text{ are encoded by } 10g_2g_3, \\
&8 \text{ and } 9 \text{ are encoded by } 11g_2g_3.
\end{aligned}
$$

□

After generating the pg-sets for individual Boolean functions, we select a minimum number of pg-functions which produce valid, complete encoding of each function. This problem is formulated as follows:

Definition 3.9 *Minimum subfunction covering problem*: Given a collection PGS of pg-sets $\{pgs_1, pgs_2, \ldots, pgs_n\}$ and $G = \cup_{i=1}^{n} pgs_i$, find a subset $G' \subseteq G$ with minimum cost such that for each pg-set pgs_z with bit_size j, there is a subset of G' with size j which induces a j-bit-partition of pgs_z.

To see the complexity of the above problem, consider a restricted version of it as follows:

Definition 3.10 *Reduced minimum subfunction covering problem*: Given a collection PGS of pg-sets, find a minimum cardinality subset $G' \subseteq G$ such that G' contains at least one element from each pg-set in PGS_n.

This is the *hitting set problem* which is NP-complete [11].

We use the following greedy algorithm for solving the minimum subfunction covering problem.

Algorithm $decomp\_mo\_sse$:
Given a vector of OBDDs $\langle \mathbf{v}_0, \ldots, \mathbf{v}_{m-1} \rangle$ representing $\langle f_0(x_0, \ldots, x_{n-1}), \ldots, f_{m-1}(x_0, \ldots, x_{n-1}) \rangle$ with variable ordering $x_0, \ldots, x_{n-1}$ and a bound set $B = \{x_0, \ldots, x_{i-1}\}$.

1. Compute $\mathcal{V}^k = cut\_vector(\mathbf{v}_k, B) = \langle \mathbf{u}_{k,2^i-1}, \ldots, \mathbf{u}_{k,0} \rangle$, $0 \leq k < m$.
2. Compute $\mathcal{V}_{i,j} = column\_encode(\mathcal{V}^0, \ldots, \mathcal{V}^{m-1})$.
3. Compute the pg-set corresponding to each $\mathcal{V}_{i,j}$. Annotate each pg-set with a value count initialized to its bit_size (this is for indicating when a function has a complete encoding) and a 0-bit-partition bp initialized to its column_set (this is for checking compatibility).
4. Find a pg g that occurs in the pg-sets most frequently.
5. For each pg-set that contains g, decrease its $count$ by 1. If $count = 0$, remove this set.
6. For each pg-set that contains g, perform $bp = \mathcal{M}(bp, g)$ and remove any pg that is not compatible with $\mathcal{M}(bp, g)$.
7. Repeat steps 1-3 until every pg-set is removed. Then, return the set of bp's, which defines the encoding for each pg-set.

4 Application to LUT-Based FPGA Synthesis

FPGAs are ASICs that can be configured by the user. They combine the logic integration benefits of custom VLSI with the design, production, and time-to-market advantages of standard logic ICs. One important class of FPGAs is Look-Up-Table (LUT) based FPGAs. In a LUT-based FPGA device (e.g., XC3000 device from Xilinx Inc [26]), the basic programmable logic block is a K-input lookup table (K-LUT) which can implement any Boolean function of up to K variables.

The technology mapping problem for LUT-based FPGA designs is to transform a Boolean network into a functionally equivalent network of K-LUTs. There are several different approaches for solving the FPGA mapping problem [10, 17, 18]. All of these works are based on the algebraic decomposition method.

4.1 Mapping for XC3000 Device

Based on the theory presented in the previous sections, we have developed Boolean methods for LUT-based FPGA synthesis. We describe an OBDD-based decomposition program, called FGSyn, that

integrates the technology-independent and technology-mapping processes for LUT-based FPGAs. FGSyn is based on the multiple-output function decomposition theory and is carried out recursively as follows:

Algorithm $fg\_synthesis$:

1. *For each bound set B of size k (for k-input LUT's);*
2. *Compute the column_vector $\mathcal{V}^{f_i}$ of each output f_i;*
3. *Perform output grouping using $\mathcal{V}^{f_i}$'s;*
4. *Perform subfunction extraction on each group of outputs;*
5. *Compute gain for performing the decomposition;*
6. *Choose bound set with the maximum gain and perform decomposition to generate F' for next iteration.*

In step 1, we choose the bound set size as the input size k of a LUT so that each g-function can be directly mapped to a single LUT. Steps 2 and 3 are carried out as described in Sec. 3.1 while step 4 is carried out by column encoding or shared subfunction encoding as described in Sec. 3.2. In step 6, the bound set with maximum gain is chosen where the gain is defined as:

$$\sum_{f_i \in F}(supp(f_i) \cap B) -$$ the number of g-functions required.

An important issue is how to come up with the input $F = \langle f_0, \ldots, f_{m-1} \rangle$ to the $fg\_synthesis$ algorithm. One option is to use the whole Boolean network as F. This is not an attractive solution as size of the OBDD representing F may become too large. In addition, the output partitioning performed in step 3 may lead to a large number of output groups, and thus, a lot of logic may be duplicated among various output groups. A better option is to run the rugged script [22] on the network, and then do some node clustering where each cluster is a complex multi-input, multi-output Boolean function. Each such node then constitutes an input to the $fg\_synthesis$ algorithm.

Node clustering is currently performed by the following greedy algorithm (this problem is obviously NP-hard):

Algorithm $node\_clustering$:

1. Start with a seed node N_s and insert it into the first cluster C^0.
2. Find a new node N_i that maximizes the $\mid supp(N_i) \cap supp(N_s) \mid$ and minimizes $\mid supp(N_i) \cup supp(N_s) \mid$.

3. If $\mid supp(N_i) \cup supp(C^0) \mid \leq \alpha$ and $1+ \mid C^0 \mid \leq \beta$ for some specified parameters α, β, then merge N_i with C^0; otherwise pick a new seed node and initialize C^1;

4. Repeat the above until all nodes are assigned to some node cluster.

Example 4.1 Given a Boolean network

$f_1 = ab + x$;
$f_2 = b + gx$;
$x = c + ad$;
$f_3 = uv + wy$;
$f_4 = uy + v$;
$y = cd + ae$;

The node_clustering algorithm leads to the following clusters for $\alpha = 5, \beta = 5$.
$C^0 = \{f_1, f_2\}$,
$C^1 = \{x, y\}$ and
$C^3 = \{f_3, f_4\}$. □

4.2 Mapping for XC4000 Device

The latest generation of the Xilinx FPGA devices, i.e., XC4000, contains a number of architectural and technological improvements that allows densities up to 20K equivalent gates and support clock rates up to 60MHz. Among the important architectural improvements that contribute to the XC4000 family's increased logic density and performance is a more powerful and flexible configurable logic block (CLB). A simplified block diagram of the combinational logic part of this CLB is shown in Figure 6. One key issue in synthesis for XC4000 device is to obtain maximal utilization of the CLBs provided on the device.

Figure 6 goes here.

Nine different patterns of XC4000 device are recognized for mapping to diffenert types of functions (Figure 7). Among these patterns, the first two patterns are the most interesting and cost effective. Note that the part enclosed by dotted box in the second pattern of Figure **??** can

be interpreted as an instance of non-disjunctive decomposition. To have such an interpretation, consider the following decompositions:

Figure 7 goes here.

$$\begin{aligned} f(X_f, X_g, x_h, \ldots) &= f_1(F(X_f), G(X_g), x_h, \ldots) \\ &= f_1(x_f, x_g, x_h, \ldots) \\ &= f_2(H(x_f, x_g, x_h), x_f, \ldots) \end{aligned}$$

In the first decomposition (f to f_1), variables X_f and X_g are bound variables with respect to the functions F and G. In the second line of the above equation, we replace $F(X_f)$ and $G(X_g)$ by variables x_f and x_g. Then, in the second decomposition (f_1 to f_2), variable x_f is both a bound variable and free variable.

Techniques introduced in the previous sections can be used to map Boolean functions to various patterns of Figure 7. This mapping can however be performed more efficiently (viz. for patterns (b) and (d)) as described next.

4.2.1 Two-Layer Decomposition

Definition 4.1 Given a function $f(X, Y, Z)$ and a decomposition $f'(g(X), Y, Z)$, if f' is simply decomposable under bound set $\{g(X), Y\}$ and free set $\{g(X), Z\}$ (e.g., $f' = f''(h(g(X), Y), g(X), Z)$), then f is *type I two-layer decomposable*.

The graphical representation of type I two-layer decomposition is shown in Fig. 8 and is also the pattern (d) in Fig. 7.

Figure 8 goes here.

Theorem 4.1 Given a function f represented by OBDD $\mathbf{v_f}$ and two bound sets X and Y, the necessary and sufficient conditions for f to be type I two-layer decomposable with respect to X and Y are:

1. $\mid cut\_set(\mathbf{v_f}, X) \mid \leq 2$ (let $cut\_set(\mathbf{v_f}, X) = \{\mathbf{u}, \mathbf{v}\}$), and

2. $\mid cut\_set(\mathbf{u}, Y) \mid \leq 2$ and $\mid cut\_set(\mathbf{v}, Y) \mid \leq 2$.

Definition 4.2 Given a function $f(X, Y, Z)$ and a decomposition $f'(g_0(X), \ldots, g_{i-1}(X), h_0(Y), \ldots, h_{j-1}(Y), Z)$, if f' is simply decomposable under bound set $\{g_k(X), h_l(Y)\}$, $0 \leq k < i$, $0 \leq l < j$, and free set consisting of Z and all g and h functions except for g_k and h_l, then f is *type II two-layer decomposable*.

The graphical representation of type II two-layer decomposition is shown in Fig. 9 and is also the pattern (c) in Fig. 7.

Figure 9 goes here.

One way to see if a function $\mathbf{v_f}$ is type II two-layer decomposable under the bound set $X \cup Y$ is the following: If $\lceil \log_2 \mid cut\_set(\mathbf{v_f}, X) \mid \rceil + \lceil \log_2 \mid cut\_set(\mathbf{v_f}, Y) \mid \rceil > \lceil \log_2 \mid cut\_set(\mathbf{v_f}, X \cup Y) \mid \rceil$ and there exists an encoding of $cut\_set(\mathbf{v_f}, X \cup Y)$ such that each g-function g satisfies the following conditions:

1. g is a function of variables X, or

2. g is a function of variables Y, or

3. g is simply decomposable under bound sets X and Y.

Therefore, to detect type II two-layer decomposition under bound sets X and Y, one must first compute:

$$\begin{array}{lcl} C_X & = & cut\_set(\mathbf{v_f}, X) = \{\mathbf{u_0}, \ldots, \mathbf{u_{k-1}}\}, \\ V_i & = & cut\_vector(\mathbf{u_i}, Y) = \langle \mathbf{v_{i,0}}, \ldots, \mathbf{v_{i,2^{|Y|}-1}} \rangle, \mathbf{u_i} \in C_X, \\ C_Y & = & column\_encode(V_0, \ldots, V_{k-1}), \text{ and} \\ C_{XY} & = & cut\_set(\mathbf{v_f}, X \cup Y) = \{\mathbf{w_0}, \ldots, \mathbf{w_{l-1}}\}. \end{array}$$

If $\lceil \log_2 \mid C_X \mid \rceil + bit\_size(C_Y) > \lceil \log_2 \mid C_{XY} \mid \rceil$, then type II two-layer decomposition is possible. We then compute a set of compatible bit-partitions of C_{XY} such that for each bit-partition $S^1 = \{S_0, S_1\}$ and associated permissable g-function g satisfies one of the following conditions: Let $W_i = \langle b_{i,0}, \ldots, b_{i,2^{|Y|}-1} \rangle$ where

$$b_{i,j} = 0 \quad \text{if } \mathbf{v_{i,j}} \in S_0 \text{ and}$$
$$b_{i,j} = 1 \quad \text{if } \mathbf{v_{i,j}} \in S_1,$$

1. $W_i = \langle 0, \ldots, 0 \rangle$ or $W_i = \langle 1, \ldots, 1 \rangle$, $0 \leq i < k$. Then, g is a function of variables X (Fig. 10 (a)).

2. $W_i = W_j$, $0 \leq i, j < k$. Then, g is a function of variables Y (Fig. 10 (b)).

3. There exist only two distinct W's, say W_i and W_j, and $bit\_size(coding(W_i, W_j)) = 1$. Then, g is a function of variables $X \cup Y$ and is simple decomposable under bound sets X and Y (Fig. 10 (c)). The former condition ensures that it is simple decomposable under X and the latter condition ensures that it is simple decomposable under Y.

Figure 10 goes here.

A straightforward way to detect type II two-layer decomposability under two bound sets X and Y is the following: We start with the decomposition of f given by $f = f'(g_0(X), \ldots, g_{i-1}(X), h_0(Y), \ldots h_{j-1}(Y), \ldots) = f'(y_0, \ldots, y_{i-1}, z_0, \ldots, z_{j-1}, \ldots)$. We then test if f' satisfies simple decompositions under bound set $\{y_k, z_l, V_f\}$, for $0 \leq k < i$ and $0 \leq l < j$. The result of this test depends on the binary encodings for y_k and z_l. Using a wrong encoding causes the test to fail when indeed type II two-layer decomposition was possible. Trying all possible encodings is clearly nonviable. On the other hand, using an arbitrary encoding may cause too many false failures.

Given a function f and a bound set $\mid B \mid \leq 4$, we can use $\mid decomp\_g(f, B) \mid$ LUTs to map the g-functions. However, this may not be the best results we can achieve. For example, let $\mid decomp\_g(f, B) \mid = 2$, under one encoding we may have two g-functions g_0 and g_1 such that the true support of each function is 4 and 2. In this case, it is possible that we can map g_1 and two other free variables to a single LUT. Furthermore, the new LUT may be combined with the one of g_0 to match the first two patterns. The effect of this process has two different interpretations. First, it is viewed as the size of bound set is 6 and the number of g-functions for this bound set is 2 or 1. Second, it is viewed as a case of nondisjunctive decomposition: supporting variables in g_1 are both in bound set B and in free set.

5 Experimental Results

The OBDD-based decomposition procedure described in Section 2 has been implemented and compared with the Roth_Karp decomposition algorithm implemented in SIS [24]. In particular, we used "xl_k_decomp -n 4 -e -d -f 100" which for every node in the Boolean network finds the best bound set of size ≤ 4 that reduces the node's variable support after decomposition, and then decomposes the node and modifies the network to reflect the change.

Our OBDD-based decomposition approach obtains significant speed-up over Roth_Karp approach by an average factor of 28.5.

The LUT-based FPGA synthesis algorithm described in Sections 3 and 4, called FGSyn, has been implemented in C and incorporated into the SIS environment. We used FGSyn to synthesize and map a number of benchmark circuits to the XC3000 device. In Table 1, we present the results obtained by using different options of FGSyn [15]: column encoding (-g), shared subfunction encoding (-s), non-disjunctive decompositions (-n) and node clustering (-c). The best results are obtained with the -csn option.

In Table 2, we compare FGSyn with Chortle-crf [10], ASYL [1] and mis-pga (new) [18]. We do not compare our results with those of FlowMap [7] as that program focuses on generating a set of mapping solutions with area and depth trade-off while our program (as well as Chortle-crf, ASYL and mis-pga (new)) targets minimum area with no depth constraints. FGSyn and mis-pga (new) were run under SPARC station II (28.5 MIPS) with 64 MB of memory.

As seen in Table 2, FGSyn does 20.6% better than Chortle-crf, 16.8% better than ASYL and 13.0% better than mis-pga (new). The memory requirement of FGSyn is only 30% more than that of the mis-pga (new). while its run time is about 2 times less than that of the mis-pga (new).

In Table 3, we present the results obtained by using direct decomposition, type I two-layer decomposition (-T 1) and type II two-layer decomposition (-T 2). In general, type I decomposition produces somewhat better results.

In Table 4, we compare FGSyn results on XC4000 with ASYL [1] and PPR [26]. For these benchmarks, we achieved 13.4% CLB reduction over PPR and 12.4% CLB reduction over ASYL.

	in	out	bx		bx-cg			bx-csn		
			CLB	Time	CLB	% Red.	Time	CLB	% Red.	Time
5xp1	7	10	16	3.8	10	37.5	3.8	9	43.8	13.0
9symml	9	1	6	6.4	6	0.0	6.3	7	-16.7	25.0
alu2	10	6	65	90.0	59	9.2	96.2	55	15.4	123.3
alu4	14	8	59	29.2	59	0.0	30.1	56	5.1	108.1
apex2	39	3	60	42.3	60	0.0	41.8	60	0.0	167.9
apex6	135	99	189	170.7	182	3.7	212.4	181	4.2	509.7
apex7	49	37	55	21.1	47	14.5	27.9	43	21.8	65.5
b9	41	21	28	1.9	28	0.0	1.9	28	0.0	2.0
bw	5	28	27	2.2	27	0.0	2.1	27	0.0	2.1
C499	41	32	54	3.9	54	0.0	3.7	54	0.0	3.8
C880	60	26	93	47.8	91	2.2	53.1	87	6.5	149.6
C1908	33	25	75	17.9	74	1.3	17.9	73	2.7	31.2
C2670	233	140	136	132.0	128	5.9	149.5	122	10.3	394.0
C5315	178	123	364	341.1	335	8.0	488.4	316	13.2	977.1
C7552	207	108	348	450.5	346	0.6	523.5	317	8.9	1263.1
clip	9	5	23	39.6	20	13.0	46.5	18	21.7	180.7
cm162a	14	5	9	1.0	9	0.0	0.9	9	0.0	1.5
count	35	16	29	4.8	29	0.0	6.9	23	20.7	14.6
duke2	22	29	87	48.2	86	1.1	49.1	85	2.3	162.7
e64	65	65	44	2.7	44	0.0	2.6	44	0.0	2.6
f51m	8	8	12	4.8	9	25.0	5.7	8	33.3	26.7
misex1	8	7	9	1.6	10	-11.1	7.1	8	11.1	21.9
misex2	25	18	22	4.0	22	0.0	4.2	22	0.0	6.4
rd73	7	3	7	2.5	6	14.3	2.3	5	28.6	2.4
rd84	8	4	12	7.1	9	25.0	6.4	8	33.3	15.0
rot	135	107	150	187.7	161	-7.3	227.9	136	9.3	689.3
sao2	10	4	26	40.2	33	-26.9	62.2	25	4.0	263.5
vg2	25	8	27	19.0	23	14.8	19.1	17	37.0	83.4
z4ml	7	4	5	2.0	4	20.0	1.7	4	20.0	4.7
Total	–	–	2037	1726.0	1971		2101.2	1847		5310.8
Average	–	–				5.2			11.6	

Table 1: Experimental results of FGSyn for XC3000 device

	in	out	Chortle-crf	ASYL	mis-pga(new)		FGSyn (bx -csn)	
			CLB	CLB	CLB	Time	CLB	Time
5xp1	7	10	20	13	17	16.6	9	13.0
9symml	9	1	41	8	7	207.6	7	25.0
alu2	10	6	83	60	84	304.1	55	123.3
alu4	14	8	138	254	149	2381.8	56	108.1
apex2	39	3	93	69	54	491.1	60	167.9
apex6	135	99	161	156	147	144.3	181	509.7
apex7	49	37	42	44	43	16.0	43	65.5
b9	41	21	–	18	27	10.5	28	2.0
bw	5	28	–	27	27	3.3	27	2.1
C499	41	32	50	–	66	738.0	54	3.8
C880	60	26	69	–	78	648.8	87	149.6
C1908	33	25	–	–	85	264.8	73	31.2
C2670	233	140	–	–	111	796.9	122	394.0
C5315	178	123	–	–	306	1285.7	316	977.1
C7552	207	108	–	–	340	2288.0	317	1263.1
clip	9	5	–	33	23	86.6	18	180.7
cm162a	14	5	–	–	10	3.0	9	1.5
count	35	16	27	28	28	8.5	23	14.6
duke2	22	29	89	82	108	325.0	85	162.7
e64	65	65	54	54	55	12.7	44	2.6
f51m	8	8	–	14	11	14.5	8	26.7
misex1	8	7	14	13	9	1.7	8	21.9
misex2	25	18	–	24	23	3.2	22	6.4
rd73	7	3	–	8	7	19.6	5	2.4
rd84	8	4	53	14	12	119.5	8	15.0
rot	135	107	131	–	139	175.6	136	689.3
sao2	10	4	–	30	28	58.8	25	263.5
vg2	25	8	18	20	20	5.7	17	83.4
z4ml	7	4	3	4	6	6.0	4	4.7
Total	–	–			2020	10437.9	1847	5310.8

Table 2: Comparison with other software programs (XC3000 device)

	in	out	bx4		bx4 -T 1			bx4 -T 2		
			CLB	Time	CLB	% Red.	Time	CLB	% Red.	Time
5xp1	7	10	15	1.5	15	0.0	1.5	15	0.0	1.5
9sym	9	1	6	2.3	6	0.0	2.8	6	0.0	2.2
9symml	9	1	6	2.3	6	0.0	2.7	6	0.0	2.3
alu2	10	6	55	38.3	53	3.6	43.9	53	3.6	38.8
alu4	14	8	52	7.2	51	1.9	6.5	51	1.9	6.6
apex2	39	3	55	13.8	55	0.0	13.6	53	3.6	13.1
apex6	135	99	136	157.9	138	-7.0	150.2	128	0.8	153.2
apex7	49	37	39	3.5	37	5.1	3.0	39	0.0	3.5
b9	41	21	25	3.0	24	4.0	2.6	24	4.0	2.9
C880	60	26	75	13.7	78	-4.0	12.7	72	4.0	12.3
C1355	41	32	49	1.6	47	4.1	1.6	47	4.1	1.6
C1908	33	25	68	10.6	67	1.5	10.2	67	1.5	10.0
c8	28	18	20	1.9	20	0.0	1.7	18	10.0	1.6
clip	9	5	26	10.0	24	7.7	10.9	25	3.8	9.7
comp	32	3	18	6.9	18	0.0	6.6	17	0.0	6.8
count	35	16	21	1.3	19	9.5	1.2	19	9.5	1.3
decod	5	16	9	0.2	9	0.0	0.2	9	0.0	0.2
duke2	22	29	84	825.6	77	8.3	806.7	76	9.5	825.9
e64	65	65	43	0.8	43	0.0	0.9	43	0.0	0.7
f51m	8	8	11	1.7	10	9.1	1.7	10	9.1	1.6
misex1	8	7	8	0.4	8	0.0	0.5	9	-12.5	0.5
misex2	25	18	20	1.4	20	0.0	1.5	19	5.0	1.3
mux	21	1	6	0.7	5	16.7	0.7	5	16.7	0.7
rd73	7	3	6	1.2	6	0.0	1.2	7	-16.7	1.1
rd84	8	4	10	2.9	10	0.0	3.0	10	0.0	2.4
rot	135	107	127	205.0	119	6.3	189.5	120	5.5	194.9
sao2	10	4	31	15.3	29	6.5	21.4	31	0.0	16.3
vda	17	39	121	2785.0	116	4.1	2481.7	111	8.3	2267.7
vg2	25	8	16	2.9	15	6.3	3.3	16	0.0	2.7
z4ml	7	4	6	0.7	6	0.0	0.7	7	0.0	0.6
Average	–	–				3.3			2.4	

Table 3: Experimental results of FGSyn options for XC4000 device

	in	out	ASYL	PPR	FGSyn	% Red.	
			CLB	CLB	CLB	ASYL	PPR
5xp1	7	10	13	–	15	-15.4	–
9sym	9	1	9	–	6	33.3	–
9symml	9	1	–	36	6	–	83.3
alu2	10	6	51	71	52	-2.0	26.8
alu4	14	8	211	–	51	75.8	–
apex6	135	99	140	–	128	8.6	–
apex7	49	37	38	38	37	2.6	2.6
b9	41	21	–	20	23	–	-15.0
C1355	41	32	–	91	47	–	48.4
c8	28	18	–	17	18	–	-5.9
clip	9	5	29	–	23	20.7	–
comp	32	3	–	17	17	–	0.0
count	35	16	22	21	19	13.6	9.5
decod	5	16	–	10	9	–	10.0
duke2	22	29	73	–	70	4.1	–
e64	65	65	52	–	43	17.3	–
f51m	8	8	12	–	10	16.7	–
misex1	8	7	9	–	9	0.0	–
misex2	25	18	21	–	19	9.5	–
mux	21	1	–	5	5	–	0.0
rd73	7	3	10	–	7	30.0	–
rd84	8	4	14	–	10	28.6	–
sao2	10	4	23	–	29	-26.1	–
vda	17	39	–	97	109	–	-12.4
vg2	25	8	15	–	16	-6.7	–
Average						12.4	13.4

Table 4: Experimental results for XC4000 device

6 Conclusion

We described techniques for OBDD-based decomposition of Boolean function and presented two Boolean methods for extracting common subfunctions from multiple-output functions. Application of these methods to the synthesis of LUT-based FPGAs was discussed.

It is useful to extend the shared subfunction encoding to include mutli-coding and input variable negation in order to improve the effectiveness of the Boolean extraction methods. Indeed, a hybrid approach where algebraic operations are interleaved with the Boolean extraction operations described here is a promising approach to logic synthesis.

References

[1] P. Abouzeid, B. Babba, M. C. de Paulet and G. Saucier, "Input-Driven Partitioning Methods and Application to Synthesis on Table-Lookup-Based FPGA's," *IEEE Trans. Computer Aided Design*, vol. 12, no. 7, pp. 913-925, July 1993.

[2] R. L. Ashenhurst, "The Decomposition of Switching Functions," *Proc. of the Int'l Symposium on Theory of Switching Functions*, pp. 74-116, 1959.

[3] R. Brayton and C. McMullen, "The Decomposition and Factorization of Boolean Expressions," *Proc. Int'l Sym. on Circuit and System*, pp. 49-54, May 1982.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: Multiple-Level Interactive Logic Optimization System," *IEEE Trans. Computer Aided Design*, vol. CAD-6, no. 6, pp. 1062-1081, November 1987.

[5] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, C-35(8): 677-691, August 1986.

[6] S-C. Chang and M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graph," *Proc. Int'l Conf. on Computer Design*, pp. 159-162, October 1992.

[7] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design," *IEEE Trans. Computer Aided Design*, vol. 13, no. 1, pp. 1-12, January 1994.

[8] H. A. Curtis, "A New Approach to the Design of Switching Circuits," Princeton, N.J., Van Nostrand, 1962.

[9] Y-T. Lai, M. Pedram and S. Sastry, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," *Proc. of 30th Design Automation Conf.*, pp. 642-647, June 1993.

[10] R.J. Francis, J. Rose and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", *Proc. of 28th Design Automation Conf.*, pp. 227-233, June 1991.

[11] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.

[12] S. He and M. Torkelson, "Disjoint Decomposition with Partial Vertex Chart," *Int'l Workshop on Logic Synthesis*, pp. p2a 1-5, May 1993.

[13] W-J. Hsu and W-Z. Shen, "Coalgebraic Division for Multilevel Logic Synthesis," *Proc. of 29th Design Automation Conf.*, pp. 438-442, June 1992.

[14] T-T. Hwang, R. M. Owens, and M. J. Irwin, "Efficiently Computing Communication Complexity for Multilevel Logic Synthesis," *IEEE Trans. on Computer Aided Design*, Vol. 11, No. 5, pp. 545-554, May 1992.

[15] Y-T. Lai, K-R. R. Pan and M. Pedram, "FPGA synthesis using Function Decomposition," *Proc. Int'l Conf. on Computer Design*, pp. 30-35, October 1994.

[16] R. M. Karp, "Functional Decomposition and Switching Circuit Design," *J. Soc. Indust. Appl. Math.*, Vol. 11, No. 2, pp. 291-335, June, 1963.

[17] K.Karplus, "Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays," *Proc. of 28th Design Automation Conf.*, pp. 240-243, June 1991.

[18] R. Murgai, N. Shenoy, R.K. Brayton and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *Proc. Int'l Conf. on Computer Aided Design*, November 1991.

[19] J.P. Roth and R.M. Karp, "Minimization Over Boolean Graphs," IBM Journal, pp. 227-238, April 1962.

[20] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Int'l Workshop on Logic Synthesis*, pp. 3a 1-12, May 1993.

[21] T. Sasao, "FPGA Design by Generalized Functional Decomposition," in *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, pp. 233-258, 1993.

[22] H. Savoj and H. Y. Wang, "Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits," *Proc. of Int'l Workshop on Logic Synthesis*, May 1991.

[23] V.Y. Shen and A.C. McKellar, "An Algorithm for the Disjunctive Decomposition of Switching Functions," *IEEE Transaction on Computers*, C-19(3): 239-248, March 1970.

[24] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," *Proc. Int'l Conf. on Computer Design*, October 1992.

[25] J. Vasudevamurthy and J. Rajski, "A Method for Concurrent Decomposition and Factorization of Boolean Expressions," *Proc. Int'l Conf. on Computer Aided Design*, pp. 510-513, November 1990.

[26] Xilinx Inc., 2100 Logic Drive, San Jose, CA 95124.

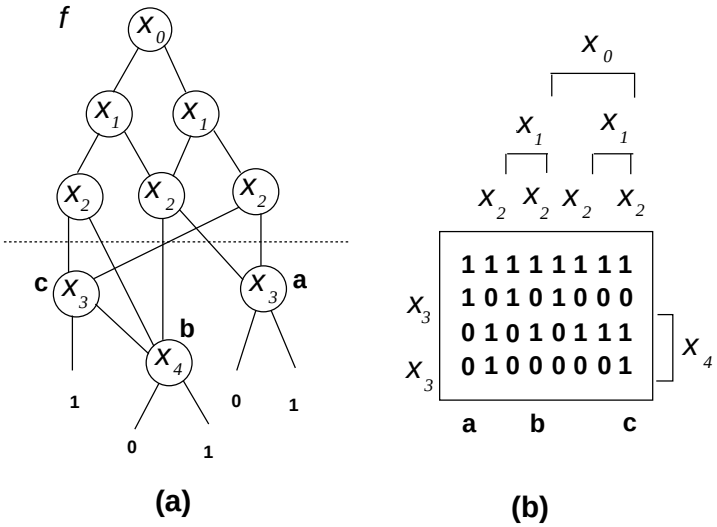

Figure 1: A function represented in (a) OBDD and (b) decomposition chart.

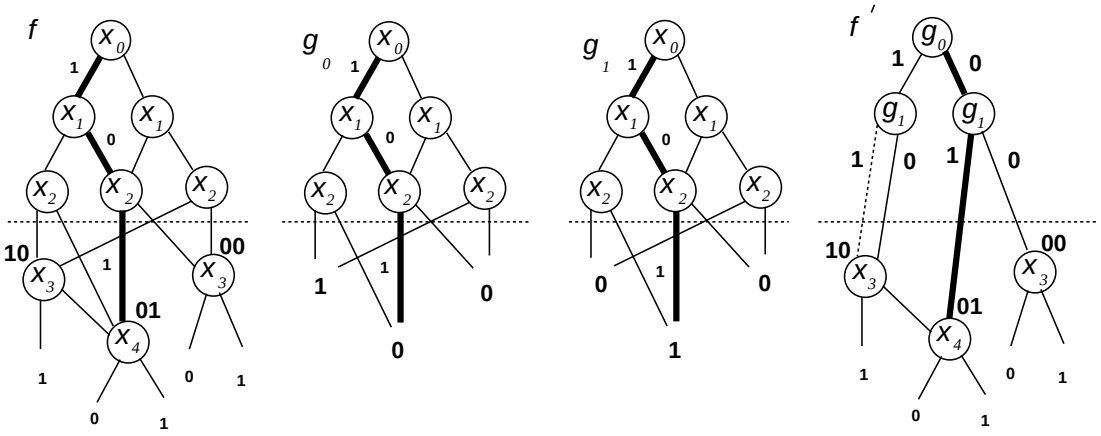

Figure 2: An example of disjunctive decomposition.

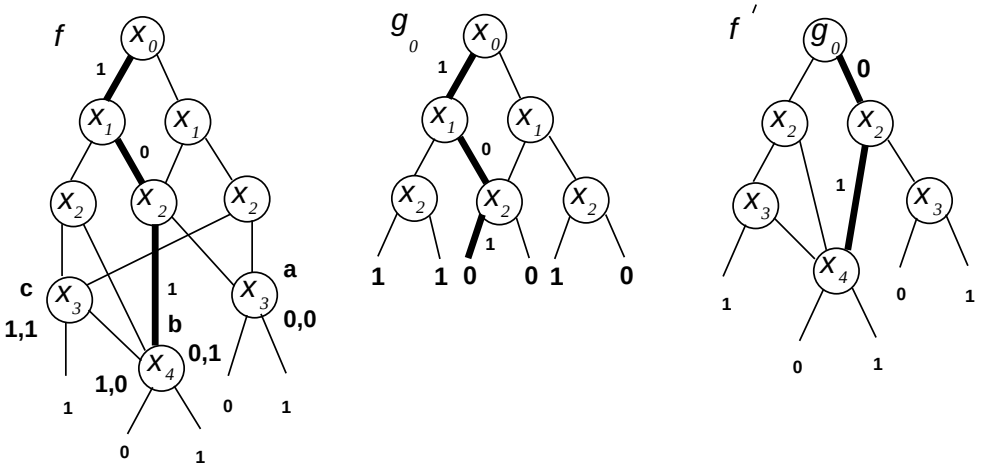

Figure 3: An example of nondisjunctive decomposition.

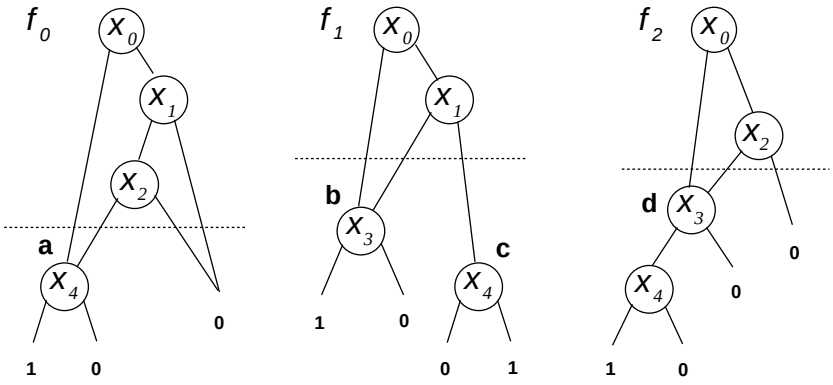

Figure 4: An example for operator cut_vector.

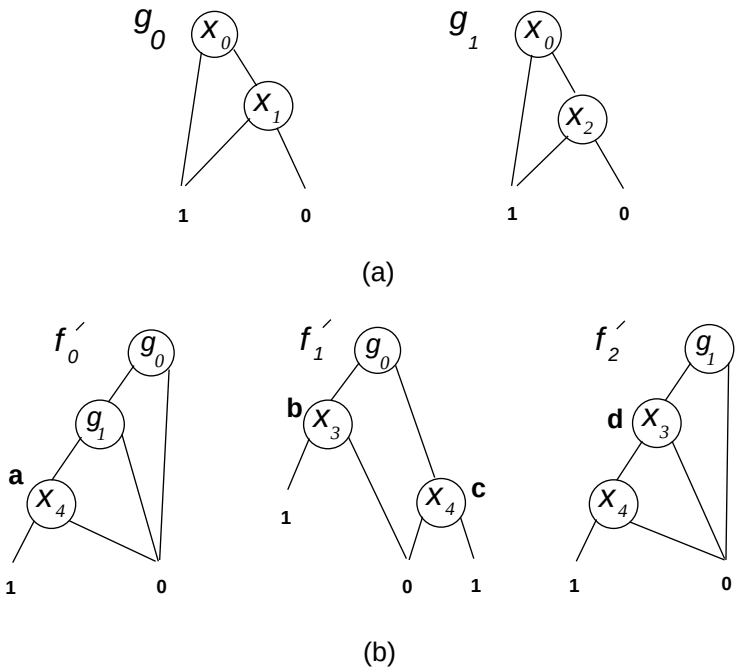

Figure 5: An example of multiple-output decomposition in OBDD representation.

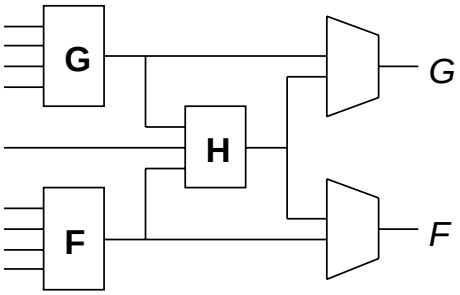

Figure 6: The Xilinx XC4000 CLB.

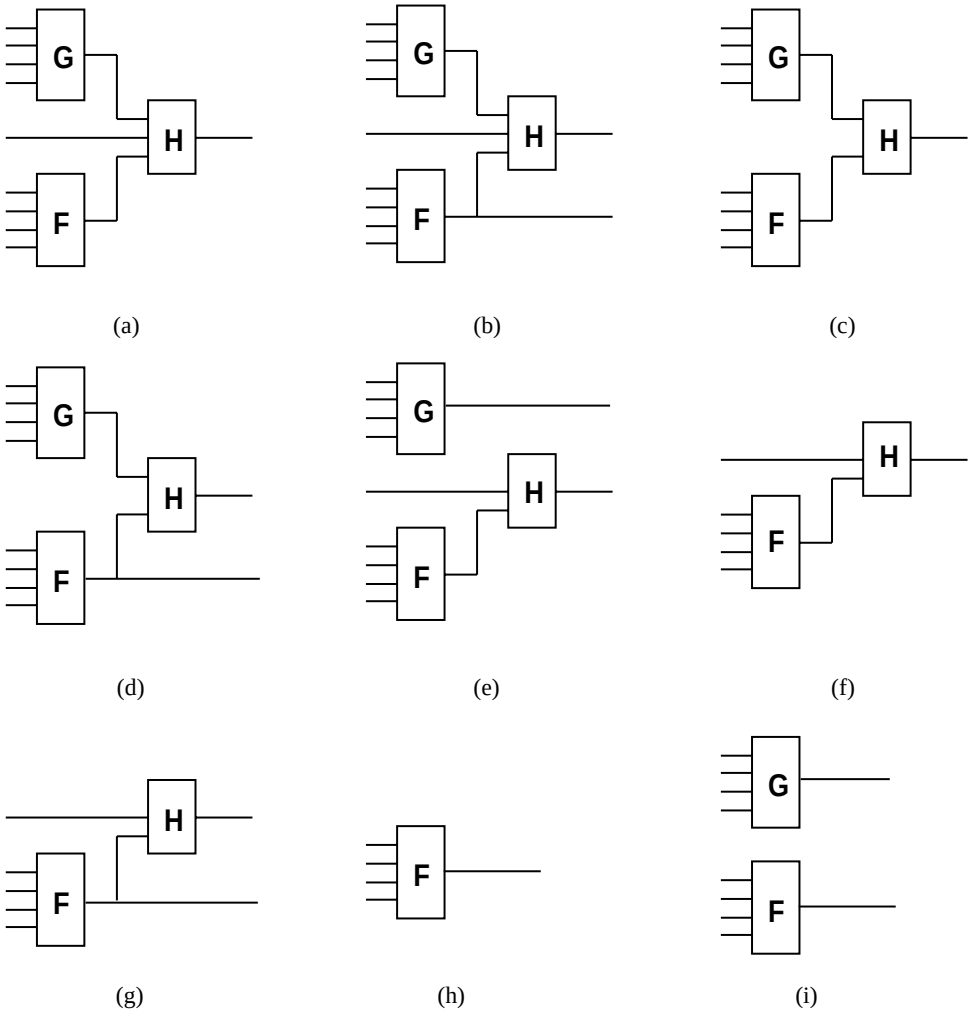

Figure 7: XC4000 patterns.

Figure 8: Graphical representation of type I two-layer decomposition.

Figure 9: Graphical representation of type II two-layer decomposition.

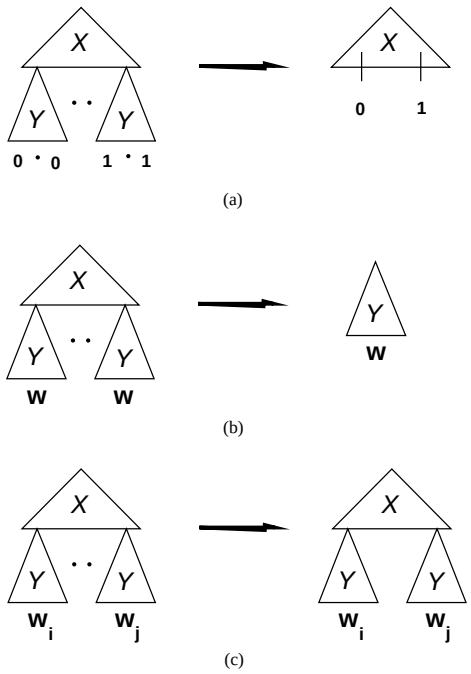

Figure 10: Conditions for type II two-layer decomposition.