FGMap: Mapping for FPGAS using Function Decmposition

Yung-Te Lai, Kuo-Rueih R. Pan,
Massoud Pedram & Sarma Vrudhula

CENG Technical Report 93-10

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4458

April 1993

FGMap: Mapping for FPGAs using Function Decomposition

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

Sarma B.K. Vrudhula (a.k.a. Sarma Sastry)
Dept. of ECE
Univ. of Arizona
Tuscon, AZ 85721

April 12, 1993

Contents

List of Figures

List of Tables

Abstract

This paper presents a BDD-based algorithm for performing multiple output function decomposition. As far as we know, this is the first such algorithm for extracting *common subfunctions* from a set of Boolean functions using the BDD representation. This algorithm provides a new way of extracting common subfunctions. Using the BDD representation, we also present a function decomposition based technology mapping algorithm for look-up table type FPGAs. Our algorithm has been implemented in C under the SIS environment and the results are encouraging.

1 Introduction

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. FPGAs can be viewed as an evolution of PALs where size is increased by an order of magnitude, or a refinement of mask-programmed gate arrays, where the reprogramming time and cost are drastically reduced.

There are mainly two types of FPGA architecture: Look-Up-Table (LUT) based and multiplexor based. The LUT-based FPGA is a popular architecture used by several FPGA manufactures, including Xilinx and AT&T. In a LUT-based FPGA device, 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 [16, 17, 18, 9, 10, 11, 2, 21, 5]. The *mispga* [16, 17, 18] mapper produces a circuit of lookup tables as an intermediate result and then uses a covering algorithm to reduce the number of CLBs. The *chortle* [9] mapper divides the input network into a forest of trees and determines an optimal mapping for each tree. Later versions of *chortle* [10, 11] use a bin packing approach to select gate level decompositions, and address performance issues. The *Flow_map* [5] uses network flow techniques to minimize the depth of LUT-based network. [2, 21] maximize the routability of LUT-based networks. These works use either algebraic or topological operations to obtain the mapped network. In contrast, in this paper, we propose a BDD-based decomposition algorithm called FGMap that decomposes a Boolean equation into functions that can be directly mapped into the CLBs of a Xilinx XC4000 device. The preliminary result are very promising.

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). One key issue in synthesis for XC4000 device is to obtain maximal utilization of the CLBs provided on the device. The function decomposition theory developed in [15] provides an efficient and effective solution to this problem.

The paper is organized as follows. In Section 2, we give basic definitions, terminology and notations used thereafter. In Section 3, we describe BDD-based decomposition algorithm for multiple output functions. We present techniques for detecting various patterns of a XC4000 device and the basic algorithm of FGMap in Sections 4 and 5. Experimental results and conclusions are given in Sections 6 and 7.

2 Definitions

In this section, we give some useful definitions.

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* $\{x_{i-k}, \ldots, x_{n-1}\}, 0 < i < n, 0 \leq k$ if f can be transformed to $f'(g_0(x_0, \ldots, x_{i-1}), \ldots, g_{j-1}(x_0, \ldots, x_{i-1}), x_{i-k}, \ldots, x_{n-1})$, where $0 < j < i - k$. If k equals 0 then it is *disjunctive decomposable*; otherwise, it is *nondisjunctive decomposable*. If j equals 1 then it is *simple decomposable*. Function f' is referred as the *F-function*, each g_i is referred as a *G-function*, and $i - (j + k)$ is referred as the variable support reduction (VSR).

Definition 2.2 A BDD is a directed acyclic graph consisting of two types of nodes. A *nonterminal* node $\mathbf{v}$ 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 $\mathbf{v}$ is either **0** or **1**. A BDD is ordered if there exist an index function $index(x) \in \{0, \ldots, n-1\}$ such that for every nonterminal node $\mathbf{v}$, 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 $index(variable(\mathbf{v})) < index(variable(child_r(\mathbf{v})))$. A BDD is reduced if there is no nonterminal node $\mathbf{v}$ such that $child_l(\mathbf{v}) = child_r(\mathbf{v})$, and there are no two nonterminal nodes $\mathbf{u}$ and $\mathbf{v}$ such that $\mathbf{u} = \mathbf{v}$. The function denoted by $\langle x, \mathbf{v}_l, \mathbf{v}_r\rangle$ is $xf_l + \bar{x}f_r$ 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.

Definition 2.3 Given a BDD node $\mathbf{v}$ 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

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

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

Definition 2.4 Given a BDD $\mathbf{v}$ representing $f(x_0, \ldots, x_{n-1})$ with variable ordering $x_0, \ldots, x_{n-1}$ and bound set $B = \{x_0, \ldots, x_{k-1}\}$, define

$$\begin{aligned} cut\_vector(\mathbf{v}, B) &= \langle eval(\mathbf{v}, 0), \ldots, eval(\mathbf{v}, 2^k - 1)\rangle, \\ cut\_set(\mathbf{v}, B) &= \{\mathbf{u} \mid \mathbf{u} = eval(\mathbf{v}, i), 0 \leq i < 2^k\}. \end{aligned}$$

Each element in $cut\_vector(\mathbf{v}, B)$ corresponds to a column in Ashenhurst-Curtis *decomposition chart* [1, 6]. Each element in $cut\_set(\mathbf{v}, B)$ corresponds to a distinct column in decomposition chart and a *compatible class* of Roth-Karp's method [19]. 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 [15].

A BDD based algorithm for computing the F- and G-functions can be found in [15]. Usually, there is more than one way to carry out a decompostion. Different encoding of the nodes of cut_set generates different decomposition. For a given decomposition, we use $decomp\_f(f, B)$ to note the F-function and $decomp\_g(f, B)$ to note the set of G-functions.

When the variable ordering of a BDD does not correspond to a given bound set, we use a *rotate* operation to move the bound variables to the top.

3 Multiple Output Function Decomposition

In this section, we first present an algorithm *decomp_mo* for decomposing a multiple output function with respect to a given bound set, and prove its correctness. We then discuss its application to FPGA mapping.

3.1 General Theory

The cut_set of a single output function corresponds to distinct columns of *decomposition chart* [1, 6] of the function. We extend this concept to multiple output functions by viewing each output variable as a variable in the free set. That is, we stack up the decomposition charts of each function and then check for distinct columns in the resulting multiple output decomposition chart. In the following, we define the cut_set for multiple output functions and an operator $\circ$ which not only computes the multiple-output cut_set, but also gives each element in the cut_set a unique encoding.

Definition 3.1 Given a multiple output function $f_0, \ldots, f_{m-1}$ represented by BDD nodes $\mathbf{v}_0, \ldots, \mathbf{v}_{m-1}$ and a bound set B, let $cut\_vector(\mathbf{v}_k, B) = \langle \mathbf{u}_{k,0}, \ldots, \mathbf{u}_{k,2^{|B|}-1} \rangle$, for $0 \leq k < m$, then the cut_set of the multiple output function is the set $\{\langle \mathbf{u}_{0,i}, \ldots, \mathbf{u}_{m-1,i} \rangle \mid 0 \leq i < 2^{|B|}\}$.

Definition 3.2 Operator $\circ$ with function type $S_1^m \times \ldots \times S_k^m \rightarrow I^m$ (where $k \geq 1, m \geq 1$, S_i is an arbitrary set, S_i^m is a vector of elements of S_i of size m, and $I^m = \{0, \ldots, m-1\}$), is defined as:

$$\circ(\langle a_{1,0}, \ldots, a_{1,m-1} \rangle, \ldots, \langle a_{k,0}, \ldots, a_{k,m-1} \rangle) = \langle c_0, \ldots, c_{m-1} \rangle,$$

where $c_i = j$ if $\langle a_{1,i}, \ldots, a_{k,i} \rangle$ is the j^{th} distinct k-tuple of $\langle a_{1,0}, \ldots, a_{k,0} \rangle$, ..., $\langle a_{1,m-1}, \ldots, a_{k,m-1} \rangle$ and the origin of j is 0. Operator $\circ^{-1}$, the inverse of $\circ$, with function type $I \times S_1^m \times \ldots \times S_k^m \rightarrow S_1 \times \ldots \times S_k$ is defined as

$$\circ^{-1}(i, \langle a_{1,0}, \ldots, a_{1,m-1} \rangle, \ldots, \langle a_{k,0}, \ldots, a_{k,m-1} \rangle) = \langle a_{1,j}, \ldots, a_{k,j} \rangle,$$

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

Definition 3.3 Vectors $\{V_1, \ldots, V_k\}$ *contains* W *with respect to* $\circ$, denoted by $W \subseteq_\circ \{V_1, \ldots, V_k\}$ if $\circ(W, \circ(V_1, \ldots, V_k)) = \circ(V_1, \ldots, V_k)$. If k equals 1, it is called *single containment*; otherwise, it is called *multiple containment*. When $W \subseteq_\circ V$, it also means that for all $i \neq j$, $v_i = v_j$ implies $w_i = w_j$.

Example 3.1 Let $V_0 = \langle a, a, a, b, b, b, c, c \rangle$, $V_1 = \langle c, d, d, e, e, e, c, c \rangle$, and $V_2 = \langle e, f, f, f, f, f, g, g \rangle$ then we have:

$\circ(V_0) = \langle 0, 0, 0, 1, 1, 1, 2, 2 \rangle$, $\circ(V_1) = \langle 0, 1, 1, 2, 2, 2, 0, 0 \rangle$,
$\circ(V_0, V_1) = \langle 0, 1, 1, 2, 2, 2, 3, 3 \rangle$,
$\circ(V_2, \circ(V_0, V_1)) = \circ(V_2, V_0, V_1) = \langle 0, 1, 1, 2, 2, 2, 3, 3 \rangle$,
$\circ^{-1}(0, V_0, V_1, V_2) = \langle a, c, e \rangle$,
$\circ^{-1}(1, V_0, V_1, V_2) = \langle a, d, f \rangle$, and
$V_2 \subseteq_\circ \{V_0, V_1\}$.

□

Note that one way to obtain $\circ(V_0, V_1)$ is to write V_0 above V_1 and then identify the distinct column patterns.

Lemma 3.1 Let $\circ(\langle a_{1,0}, \ldots, a_{1,m-1} \rangle, \ldots, \langle a_{k,0}, \ldots, a_{k,m-1} \rangle) = \langle c_0, \ldots, c_{m-1} \rangle$, if $c_i = c_j$, $0 \leq i, j < m$, then $a_{l,i} = a_{l,j}$ for $1 \leq l \leq k$.

Proof by contradiction. If $a_{l,i} \neq a_{l,j}$ for some l, then $\langle a_{1,i}, \ldots, a_{l,i}, \ldots, a_{k,i} \rangle$ is distinct from $\langle a_{1,j}, \ldots, a_{l,j}, \ldots, a_{k,j} \rangle$ which implies that $c_i \neq c_j$.

□

Given a multiple output function $F = f_0, \ldots, f_{m-1}$ and a bound set $\{x_0, \ldots, x_{i-1}\}$, algorithm *decomp_mo* 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 $0 \leq k < m$. To carry out the above transformation, we first compute the cut_set of F to determine the number of G-functions j required. We then encode each element in cut_set and construct the G- and F-functions as follows.

Algorithm *decomp_mo*
Given a vector of BDDs $\langle \mathbf{v}_0, \ldots, \mathbf{v}_{m-1} \rangle$ 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 $V_k = \mathit{cut\_vector}(\mathbf{v}_k, B) = \langle \mathbf{u}_{k,0}, \ldots, \mathbf{u}_{k,2^i-1} \rangle$, $0 \leq k < m$.

2. Compute the multiple output cut_set $C = o(V_0, \ldots, V_{m-1})$. Let $C = \langle c_0, \ldots, c_{2^i-1} \rangle$, $2^{j-1} < max(C) + 1 \leq 2^j$, encode c_p, $0 \leq p < 2^i$ by j bits $d_{p,0} \ldots d_{p,j-1}$ such that $c_p = 2^{j-1}d_{p,0} + \ldots + 2^0 d_{p,j-1}$.

3. Construct each G-function $g_q(x_0, \ldots, x_{i-1})$, $0 \leq q < j$, as
$$g_q(x_0, \ldots, x_{i-1}) = [d_{0,q} \ldots d_{2^i-1,q}] \quad \text{(truth table of } g_q\text{)}$$
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 $o^{-1}(r, V_0, \ldots, V_{m-1}) = \langle \mathbf{u}_{0,s_r}, \ldots, \mathbf{u}_{m-1,s_r} \rangle$, $0 \leq r < 2^j$, $0 \leq s_r < 2^i - 1$, s_r is any l such that $c_l = r$.

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

We give an example of how *decomp_mo* works next.

Example 3.2 Fig. 1 (a) is a 3-output function. The cut_sets under bound set $B = \{x_0, x_1, x_2\}$ of each output are {**a, b, c**}, {**c, d, e**} and {**e, f, g**}. We compute the following values.

1. $\mathit{cut\_vector}(f_0, B) = \langle \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{c} \rangle = V_0$,
$\mathit{cut\_vector}(f_1, B) = \langle \mathbf{c}, \mathbf{d}, \mathbf{d}, \mathbf{e}, \mathbf{e}, \mathbf{e}, \mathbf{c}, \mathbf{c} \rangle = V_1$,
$\mathit{cut\_vector}(f_2, B) = \langle \mathbf{e}, \mathbf{f}, \mathbf{f}, \mathbf{f}, \mathbf{f}, \mathbf{f}, \mathbf{g}, \mathbf{g} \rangle = V_2$,

2. $o(V_0, V_1, V_2) = \langle 0, 1, 1, 2, 2, 2, 3, 3 \rangle \Rightarrow \langle 00, 01, 01, 10, 10, 10, 11, 11 \rangle$,

3. $g_0(x_0, x_1, x_2) = [00011111]$,
$g_1(x_0, x_1, x_2) = [01100011]$,

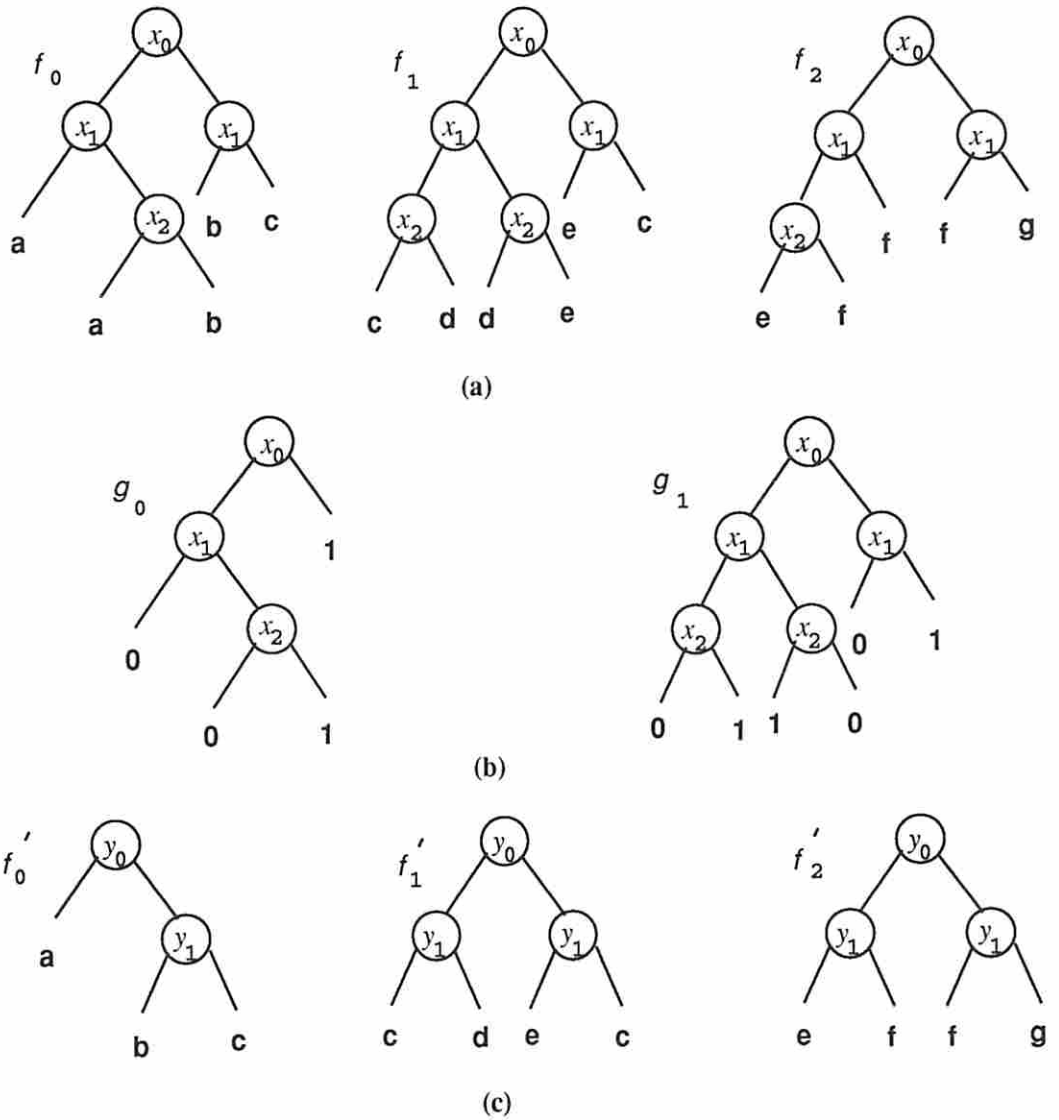

Figure 1: An example of multiple output decomposition

4. $o^{-1}(0, V_0, V_1, V_2) = \langle \mathbf{a}, \mathbf{c}, \mathbf{e} \rangle$,
   $o^{-1}(1, V_0, V_1, V_2) = \langle \mathbf{a}, \mathbf{d}, \mathbf{f} \rangle$,
   $o^{-1}(2, V_0, V_1, V_2) = \langle \mathbf{b}, \mathbf{e}, \mathbf{f} \rangle$,
   $o^{-1}(3, V_0, V_1, V_2) = \langle \mathbf{c}, \mathbf{c}, \mathbf{g} \rangle$,

5. $f'_0(g_0, g_1, x_3, \ldots) = [\mathbf{aabc}]$,
   $f'_1(g_0, g_1, x_3, \ldots) = [\mathbf{cdec}]$, and
   $f'_2(g_0, g_1, x_3, \ldots) = [\mathbf{effg}]$.

The resulting G-functions and F-functions are shown in Fig. 1 (b) and (c) respectively.

□

The following lemma proves the correctness of *decomp_mo*.

Lemma 3.2 Algorithm *decomp_mo* 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 $0 \leq k < m, 0 < i < n$.

Proof: What we need to show is for all $b_0, \ldots, b_{i-1} \in \{0, 1\}$, $0 \leq k < m$,

$f_k(b_0, \ldots, b_{i-1}, x_i, \ldots, x_{n-1}) = f'_k(g_0(b_0, \ldots, b_{i-1}), \ldots, g_{j-1}(b_0, \ldots, b_{i-1}), x_i, \ldots, x_{n-1})$.

Let $cut\_vector(\mathbf{v}_k, \{x_0, \ldots, x_{i-1}\}) = \langle \mathbf{u}_{k,0}, \ldots, \mathbf{u}_{k,2^i-1} \rangle$ and $p = 2^{i-1}b_0 + \ldots + 2^0 b_{i-1}$ for an arbitrary bit vector $b_0, \ldots, b_{i-1}$.

From Def. 2.3, $f_k(b_0, \ldots, b_{i-1}, x_i, \ldots, x_{n-1}) = eval(\mathbf{v}_k, \langle b_0, \ldots, b_{i-1} \rangle) = \mathbf{u}_{k,p}$.

From step 3 of *decomp_mo*:

$g_q(b_0, \ldots, b_{i-1}) = d_{p,q}, 0 \leq q < j$, and

$2^{j-1}g_0(b_0, \ldots, b_{i-1}) + \ldots + 2^0 g_{j-1}(b_0, \ldots, b_{i-1})$

$= 2^{j-1}d_{p,0} + \ldots + 2^0 d_{p,j-1}$

$= c_p$.

From step 5 of *decomp_mo*:

$f'_k(g_0(b_0, \ldots, b_{i-1}), \ldots, g_{j-1}(b_0, \ldots, b_{i-1}), x_i, \ldots, x_{n-1})$

$= f'_k(d_{p,0}, \ldots, d_{p,j-1}, x_i, \ldots, x_{n-1})$

$= \mathbf{u}_{k,s_{c_p}} \qquad (2^{j-1}d_{p,0} + \ldots + 2^0 d_{p,j-1} = c_p)$,

where $s_{c_p} = l$ such that $c_l = c_p$.

From Lemma 3.1 $c_l = c_p$ implies that $\mathbf{u}_{k,s_{c_p}} = \mathbf{u}_{k,p}$.

□

Algorithm *decomp_mo* provides a new way of extracting *common subfunctions*. For example, functions g_0 and g_1 are common subfunctions of f_0, f_1, and f_2 in the above example. Common subfunctions extracted in the above manner are restricted by the size of bound set considered but not by the function representation. This method is, as far as we know, the first one that is able to extract common subfunctions from a set of Boolean functions using BDD representation. This is obviously much stronger than algebraic common subexpression extraction.

3.2 Specialization to FPGA Mapping

For the mapping of LUT devices, we define multiple output decomposition in the following way.

Definition 3.4 A multiple output function $F = \langle f_0, \ldots, f_{m-1} \rangle$ is said to be *multiple output decomposable* under bound set B if there exists a partition of F into $P_1, \ldots, P_k$ such that the following transformation holds:

$$f_{i,j}(X) \;=\; f'_{i,j}(\underline{g_i}(B), X - B),$$

where $f_{i,j} \in P_i, \mid \underline{g_i} \mid < \mid B \mid$

The variable support reduction (VSR) is defined to be $m \times \mid B \mid - \sum \mid \underline{g_i} \mid$.

□

Although the total number of variables may increase after a multiple output decomposition, the true support of individual function always decrease. Based on the above definition, we formulate the problem of multiple output decomposition as:

Problem A: Given a multiple output function $F = \langle f_0, \ldots, f_{m-1} \rangle$ and a bound set B, find a partition of $\{f_0, \ldots, f_{m-1}\}$ into $P_1, \ldots, P_k$ such that

$$\sum_{i=1}^{k} S_i \text{ is minimized and } S_i <| B |$$

where $S_i = \lceil \log_2(max(M_i) + 1) \rceil$, (S-value)

$M_i = \circ(cut\_vector(f_{i,1}, B), \ldots cut\_vector(f_{i,j}, B)), \quad f_{i,l} \in P_i.$

The minimization of $\sum S_i$ is performed in order to achieve maximal VSR. Constraint $S_i <| B |$ is imposed in order to guarantee that the true support of each individual function is decreased.

Example 3.3 Let $| B |= 3$, and

f_0 : C_0 = 0, 0, 1, 2, 1, 3, 0, 0
f_1 : C_1 = 0, 0, 0, 0, 1, 1, 1, 1
f_2 : C_2 = 0, 0, 1, 1, 1, 0, 0, 0
f_3 : C_3 = 0, 0, 0, 1, 0, 2, 0, 0

where C_i is the cut_vector of f_i.

Because $\circ(C_0, C_1) = 0, 0, 1, 2, 3, 4, 5, 5$, $max(\circ(C_0, C_1)) + 1 = 6$, $\lceil \log_2 6 \rceil = 3 \not< 3$, we cannot put f_0 and f_1 into the same set. Putting f_2 or f_3 into the same set of f_1 will increase S-value, while putting them into the same set of f_0 will not change S-value. The solution for this example is $\{\{f_0, f_2, f_3\}, \{f_1\}\}$.

□

In the following, we propose a greedy algorithm for solving problem A.

Algorithm *output_partition*
Assume each $S_i <| B |$ for each i.

1. For all $cut\_vector(\mathbf{v}_i, B) \subseteq_\circ cut\_vector(\mathbf{v}_j, B)$ put f_i in the same set of f_j (single containment).
2. Order the result from step 1 into nonincreasing order of $max(\circ(cut\_vector(\mathbf{v}_j, B)))$.
3. Starting from the first element of the above list, put as many set k as possible into the same set of j, $j < k$, such that $\lceil log_2(max(\circ(C_j, C_k)) + 1) \rceil <| B |$.

With proper backtracking in step 3, we can develop an algorithm which finds the optimal solution of the above problem. The computation cost, of course, may become exponential.

4 Pattern Matching in Xilinx XC4000

In this section, we present a basic algorithm for matching patterns of XC4000 based on the decomposition theory developed in [15]. We also describe a new Boolean transformation called *two-layer decomposition* which is used for improving our basic algorithm. A simplified block diagram of the combinational logic part of XC4000 is shown in Figure 2.

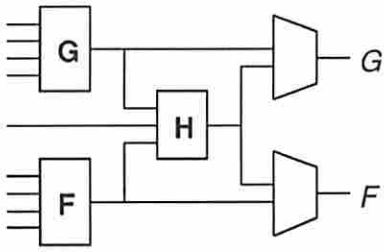

Figure 2: The Xilinx XC4000

4.1 Basic Pattern Matching

Nine different patterns of XC4000 device are recognized for mapping to different types of functions (Fig. 3). Among these patterns, the first two patterns are the most interesting and cost effective. We show how to use the decomposition theory to map Boolean functions to the first two patterns of Figure 3. The remaining ones are sub-patterns of these two patterns.

Given a BDD v representing $f(x_0, \ldots, x_{n-1})$, two sets of variables X_f and X_g each containing at most 4 variables, and a variable x_h, the following algorithm $match\_pattern(\mathbf{v}, X_f, X_g, x_h)$ returns 1 if $\{X_f, X_g, x_h\}$ can be mapped to the pattern in Fig. 3 (a); returns 2 if it can be mapped to the pattern in Figure 3 (b); otherwise it returns 0. The sets of variables X_f and X_g are computed by the method described in [15].

The algorithm *match_pattern* is straightforward. It first moves x_h to the top of BDD. It then moves variables X_g to the top and computes the cut_set with respect to variables X_g. If the *cut_set* size is greater than 2, X_g cannot be mapped to a single LUT (lines 2-5). It then carries out the same operation for variables X_f (lines 6-9). After that, it computes the *cut_set* with respect to variables X_f , X_g and x_h (line 10). If the *cut_set* size is greater than 4, then it requires more than two outputs. Neither pattern can be mapped (line 11). On the other hand, if the *cut_set* size is less than or equal to 2, the pattern (a) is detected (line 12). Otherwise, it checks for pattern (b) (lines 13-21).

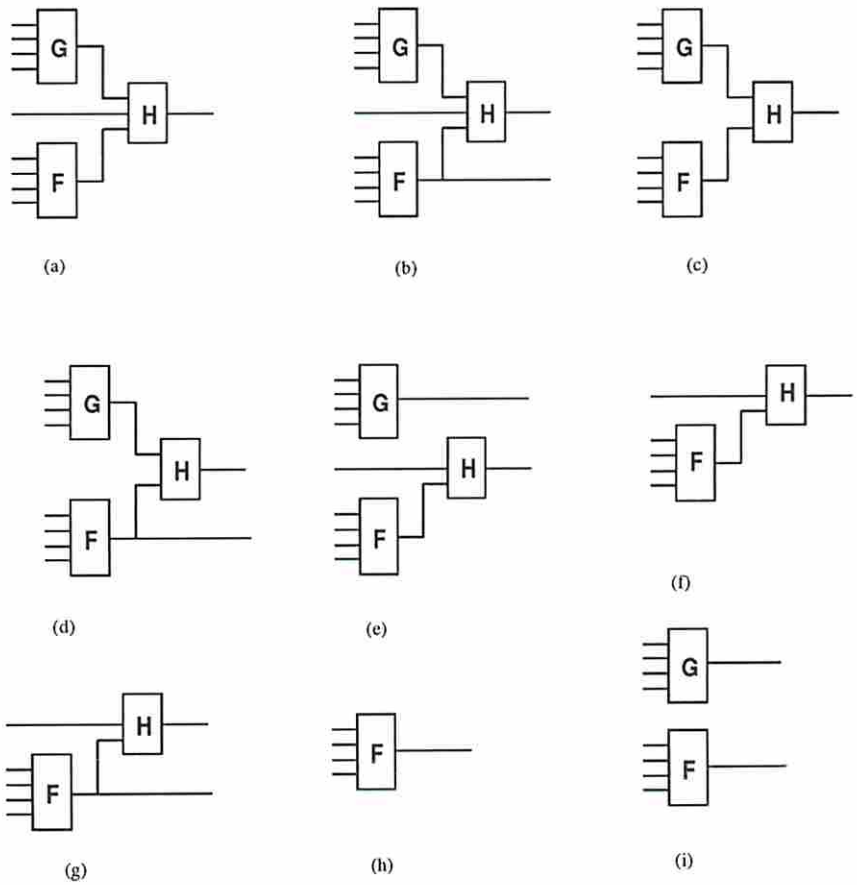

Figure 3: XC4000 patterns

```
match_pattern(v, X_f, X_g, x_h)
{
1    v = rotate(v, x_h);              /* move x_h to the top */
2    for (i = 0; i <| X_g |; i++)     /* move X_g to the top */
3      v = rotate(v, X_{g_i});
4    cset_g = cut_set(v, X_g);        /* compute the cut_set of X_g */
5    if (| cset_g |> 2) return 0;     /* X_g is not simple decomposable */
6    for (i = 0; i <| X_f |; i++)     /* move X_f to the top */
7      v = rotate(v, X_{f_i});
8    cset_f = cut_set(v, X_f);        /* compute the cut_set of X_f */
9    if (| cset_f |> 2) return 0;     /* X_f is not simple decomposable */
10   cset_fgh = cut_set(v, X_f ∪ X_g ∪ {x_h});   /* compute the cut_set of X_f ∪ X_g ∪ {x_h} */
11   if (| cset_fgh |> 4) return 0;   /* X_f, X_g and x_h require more than two outputs */
12   if (| cset_fgh |≤ 2) return 1;   /* pattern (a) is mapped */
13   cset0 = cut_set(cset_f[0], X_g ∪ {x_h});
14   cset1 = cut_set(cset_f[1], X_g ∪ {x_h});
15   if (| cset0 |<= 2 && | cset1 |<= 2) return 2;   /* pattern (b) is mapped */
16   for (i = 0; i <| X_g |; i++)     /* move X_g to the top */
17     v = rotate(v, X_{g_i});
18   cset_g = cut_set(v, X_g);        /* compute the cut_set of X_g */
19   cset0 = cut_set(cset_g[0], X_f ∪ {x_h});
20   cset1 = cut_set(cset_g[1], X_f ∪ {x_h});
21   if (| cset0 |<= 2 && | cset1 |<= 2) return 2;   /* pattern (b) is mapped */
22   return 0;        /* pattern (b) cannot be mapped */
}
```

To see how pattern (b) is detected, consider Fig. 4. Fig. 4 (a) shows a function which satisfies the conditions specified in lines 15 or 21. Fig. 4 (b) shows the two G-functions with the encoding $c = 00, d = 01, e = 10$, and $f = 11$. The reduced BDD of the first G-function is shown in Fig. 4 (c). Thus, the first G-function has the first bound set as support while the second one has the union of the two bound sets as support. Note that lines 16-18 are necessary because when variables X_g are moved to the top for the first time (lines 2-4), positions of X_f are unknown.

It is easy to see that conditions in lines 15 and 21 are not only sufficient but also necessary. If the cardinality of either $cset0$ or $cset1$ is greater than 2, then there is no encoding that produces a reduced BDD as in Fig. 4 (c).

4.2 Two-Layer Decomposition

Although it is easy to detect the first two patterns using the above algorithm, it is often difficult to find a function which satisfies two simple decomposable bound sets X_f and X_g. To increase the possibility for matching the above patterns, we therefore look for *two-layer decomposition* as described below.

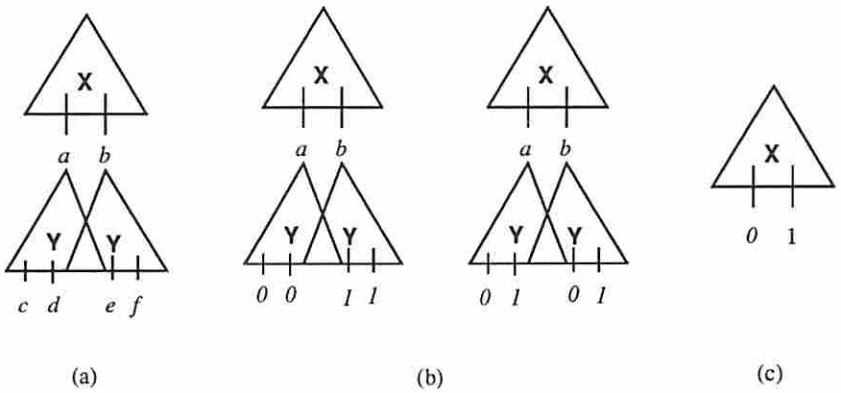

Figure 4: A function satisfying pattern (b) with bound set X and Y

Definition 4.1 Assume that function f can be decomposed as

$f = f'(g_0(B_1), \ldots, g_{i-1}(B_1), h_0(B_2), \ldots, h_{j-1}(B_2), \ldots), 1 \leq i, 1 \leq j,$

under bound sets B_1 and B_2. If there exist $g_k(B_1)$, $h_l(B_2)$, and a set of variables V_f (possibly empty) from free set such that they can be packed to form a single output $p_{kl}(B_1, B_2, V_f)$, then we say that f is *type I two-layer decomposable.* If there exist $g_k(B_1)$ and a nonempty set of variables V_f from free set such that they can be packed to form a single output $p_k(B_1, V_f)$, then we say that f is *type II two-layer decomposable.*

If f is type I two-layer decomposable under $g_k(B_1)$, $h_l(B_2)$, and $V_f = \phi$, then $g_k(B_1)$ and $h_l(B_2)$ can be mapped to pattern (c). If f is type I two-layer decomposable under $g_k(B_1)$, $h_l(B_2)$, and $V_f = \{x\}$, then $g_k(B_1)$, $h_l(B_2)$ and x can be mapped to pattern (a). If f is type II two-layer decomposable under $g_k(B_1)$ and V_f, and $\mid supp(g_k(B_1)) \mid + \mid V_f \mid \leq 4$, then $g_k(B_1)$ and V_f can be mapped to an LUT. This process may be viewed in two ways. First, it is viewed as the size of bound set being $\mid B_1 \mid + \mid V_f \mid$ and the number of G-functions remaining i. Second, it is viewed as a case of nondisjunctive decomposition, that is, supporting variables in $g_k(B_1)$ are both in the bound set B_1 and in the free set.

A straightforward way to detect type I two-layer decomposability under two bound sets B_1 and B_2 is the following: we start with the decomposition of f given by $f = f'(g_0(B_1), \ldots, g_{i-1}(B_1), h_0(B_2), \ldots h_{j-1}(B_2), \ldots) = f'(y_0, \ldots, y_{i-1}, z_0, \ldots, z_{j-1}, \ldots)$. 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 I two-layer decomposition was possible. Trying all possible encodings is clearly nonviable. Using an arbitrary encoding may cause too much false failures. Currently, FGMap uses k arbitrary encodings where k is the size of cut_set.

The detection of type II two-layer decomposition is carried out in the same way as that of type I, that is, by performing decomposition under some encoding followed by the detection of decomposability. Again, FGMap tries k arbitrary encodings for a cut_set size of k.

5 Mapping Algorithm for XC4000

The basic principle of FGMap is to maximize the variable support reduction (SVR) by function decomposition. For two equal size bound sets, the one with smaller cut_set size will be considered first. The first three patterns of Figure 3 will be checked before other patterns because they lead to most input-output reduction. Our algorithm is carried out in a level by level fashion. The procedure for each level is outlined below:

1. *Trivial Cases*: If the number of inputs is 1, do nothing and return nil (0 XC4000 used). If the number of inputs is smaller than or equal to 4, return the circuit (half XC4000 used).

2. *Best Bound Set List*: Compute *cut_sets* for all bound sets with size less than or equal to 4 [15]. Sort these bound sets in non-increasing order of $(2^{|boundset|} - |\ cut\_set\ |)/2^{|boundset|}$ (This metric reflects the percentage reduction of the cut_set size). For example, for a bound set of size 4 with cut_set size of 3, the reduction in cut_set size is 13/16 while for a bound set of size 3 with cut_set of size 2, the reduction is 6/8 (12/16). Thus, the reduction in the former case is more than the latter although both cases give rise to the input-output reduction of 2.

3. *Check Impasse Case*: If the first bound set requires 4 G-functions (4 inputs, 4 outputs), go to step 9. Otherwise, continue.

4. *Find First Bound Set*: Pick the first bound set B which is not mapped, let n be the size of the cut_set under B.

5. *Type II Two-layer Decomposition*: If $n > 2$, try n encodings and choose the one which reduces the variable support of G-functions the most. Perform type II two-layer decomposition.

6. *Find Second Bound Set*: Pick next bound set B' which is not mapped and $supp(B) \cap supp(B') = \phi$. Repeat step 5 for B'.

7. *Type I Two-layer Decomposition*: Carry out type I two-layer decomposition with respect to bound sets B and B'. That is, try to match the first three patterns of Fig. 3.

8. *Pattern Matching*: Try to map the pattern in Fig. 3 (f). Map each G-function resulting from previous steps to an LUT (pattern (h) in Fig. 3). Combine patterns (f) and (h) or pattern (h)s into pattern (e) or (i), if possible. Go to next level.

9. *Impasse Handling Step*: Let $B = \{x_0, \ldots, x_{n-2}\}$, carry out decomposition of f under B. This will produce an F-function with three supporting variables and two G-functions each having at most $n-1$ supporting variables. Map the F-function to an LUT. Go to step 1 for both G-functions.

For the sake of simplicity, some details are omitted from the above algorithm.

In the last step of the above algorithm, there is no bound set which produces input-output reduction. To resolve this **impasse situation**, we use a divide and conquer approach. The

size of cut_set of $\mid B \mid = n - 1$ is at most 4 $\{\mathbf{0}, \mathbf{1}, x, \bar{x}\}$. Thus, the decomposition of f under B requires at most 2 G-functions each having at most $n - 1$ supporting variables and the F-function has $\{y_0, y_1, x_{n-1}\}$ three variables. We then solve the two subproblems corresponding to the decomposition of the G-functions. It may seem that this approach is inefficient because in the worst case, the number of subproblems may grow exponentially. However, according to our experiments, the performance of this approach is reasonably well.

FGMap does not use patterns (d) and (g) in Figure 3. It uses (i) for (d), and (h*) [1] for (g). This is due to the following reasons:

1. The input-output reduction rates for (i) and (h*) are the same as for (d) and (g) respectively.

2. (i) and (h*) use the same number of CLBs as (d) and (g) respectively.

3. Although LUT H in (d) or (g) performs a transformation on the function, we don't know how to make the transformation so as to increase the decomposability of the function in the subsequent stages.

6 Experimental Results

FGMap has been implemented in C and incorporated into the SIS package. Results of our algorithm on a number of benchmarks are shown in Table 1. Column FGmap indicates the number of XC4000 CLBs required to map a design; Columns PPR and Trimberger are the results from Xilinx and Trimberger and Chene's work [22].

We don't have a full implementation of the multiple output decomposition algorithm and hence results reported here reflect the effectiveness of single output decomposition algorithm only. We apply this algorithm to a Boolean network that has been optimized by script.rugged of SIS.

FGMap can be applied in two different ways: as a technology mapping tool or as a logic synthesis tool. In the former, FGMap is applied to each intermediate node in the optimized Boolean network. In the latter, FGMap is directly applied to a given Boolean network. The advantage of the latter is that FGMap is a Boolean method, the decomposability of a function is checked based on the function itself rather than a subfunction of it.

There are three single output functions in Table 1. Two of them are out perform the PPR and Trimberger's result while the third is the same as that of PPR which implies that PPR has found the optimal solution.

There are three benchmarks alu2, frg1, and vda where our algorithm encounters the impasse situations. As described in Section 5, we resolve impasse situation by creating two subproblems and recursively apply our algorithm on each subproblem. Therefore, common subexpressions are duplicated. We believe that by using multiple output decomposition, we can improve our results not only for impasse situations but also in general cases.

[1] pattern (h) plus a single line.

	in	out	FGmap	PPR	Trimberger	%
9symml	9	1	6	36	37	-83
alu2	10	6	52	71	126	-27
apex7	49	37	37	38	57	-3
b9	41	21	22	20	30	10
c8	28	18	17	17	41	0
C1355	41	32	65	91	—	-29
cc	21	20	8	8	16	0
cht	47	36	23	—	30	-23
cm162a	14	5	6	5	—	20
cmb	16	4	8	—	10	-20
comp	32	3	17	17	30	0
cordic	23	2	7	—	14	-50
count	35	16	16	21	16	0
cu	14	11	8	—	12	-34
dalu	75	16	183	—	350	-48
decod	5	16	9	10	—	-10
example2	85	66	58	—	71	-18
f51m	8	8	9	—	41	-78
frg1	28	3	90	—	161	-44
lal	26	19	13	—	30	-57
mux	21	1	5	5	20	0
my-adder	33	17	16	—	16	0
parity	16	1	3	—	8	-63
pcle	19	9	12	—	17	-29
pcler8	27	17	15	—	19	-21
pm1	16	13	8	—	15	-47
sct	19	15	9	—	33	-73
term1	34	10	20	—	116	-82
ttt2	24	21	35	—	91	-62
unreg	36	16	16	—	16	0
vda	17	39	110	97	—	13
x2	10	7	7	—	12	-42
z4ml	7	4	5	3	29	67

Table 1: Experimental results of FGMap

7 Future Work

In this paper, we present a BDD-based algorithm for multiple output function decomposition. Our algorithm is the first Boolean method for extracting common subfunctions. We also present a technology mapping tool called FGMap for Xilinx XC4000 module. The distinct feature of FGMap is that it is a function decomposition based Boolean method. The initial results has already shown improvement over previously published work.

Our future work will concentrate on the improving the FGMap. Particularly, we will extend our algorithm from single output decompostion to multiple output decomposition.

Acknowledgement – This work was supported in part by the NSF's Research Initiation Award under contracts No. MIP-9111206 and No. MIP-9211668.

References

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

[2] Bhat, N. and D. Hill, "Routable Technology Mapping for FPGAs," First Int'l ACM/SIGDA Workshop on FPGAs, pp. 143-148, Feb. 1992.

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

[4] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization," IEEE Design and Test of Computers, Sep. 1992.

[5] J. Cong, and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," Proc. of ICCAD, pp. 48-53, 1992.

[6] H.A. Curtis, "A new approach to the Design of Switching Circuits," Princeton, N.J., Van Nostrand, 1962.

[7] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "Technology Mapping in MIS," Proc. of ICCAD, 1987.

[8] S. Ercolani and G. D. Micheli, "Technology Mapping for Electrically Programmable Gate Arrays," Proc. 28th Design Automation Conference, 1991

[9] R.J. Francis, J. Rose and K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based FPGAs," Proc. 27th DAC, June 1990, pp. 613-619.

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

[11] R.J. Francis, J. Rose and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance," Proc. ICCAD, pp. 568-571, Nov. 1991.

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

[13] K.Karplus, "Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays," Proc. 28th DAC, pp. 240-243, June 1991.

[14] K. Keutzer, "Dagon: Technology Binding and Local Optimization by DAG Matching," Proc. 24th Design Automation Conference, June 1987.

[15] Yung-Te Lai, Massoud Pedram and Sarma Sastry, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," to appear in 30th ACM/IEEE Design Automation Conference, June 1993.

[16] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," Proc. 27th Design Automation Conference, 1990

[17] R. Murgai, N. Shenoy, R.K. Brayton and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," Proc. of ICCAD, pp. 564-567, Nov. 1991.

[18] R. Murgai, N. Shenoy, R.K. Brayton and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," Proc. ICCAD, pp. 572-575, Nov. 1991.

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

[20] Sawkar, P. and D. Thomas, "Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays," ACM/SIGDA Workshop on FPGAs, pp. 83-88, Feb. 1992.

[21] Schlag, M., J. Kong, and P. K. Chan, "Routability-Driven Technology Mapping for Lookup Table-Based FPGAs," Proc. ICCD, Oct. 1992.

[22] S. Trimberger and M-R. Chene, "Placement-Based Partitioning for Lookup-Table-Based FPGAs", FPGA '92 Workshop.

[23] Woo, N.-S., "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," Proc. 28th Design Automation Conference, pp. 248-251, 1991.