Logic Synthesis for Look-Up Table based FPGAs using Functional Decomposition and Support Minimization

Hiroshi Sawada, Takayuki Suyama and Akira Nagoya
NTT Communication Science Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, JAPAN
{sawada, suyama, nagoya}@cslab.kecl.ntt.jp

Abstract

This paper presents a logic synthesis method for look-up table (LUT) based field programmable gate arrays (FPGAs). We determine functions to be mapped to LUTs by functional decomposition. We use not only disjunctive decomposition but also nondisjunctive decomposition. Furthermore, we propose a new Boolean resubstitution technique customized for an LUT network synthesis. Resubstitution is used to determine whether an existing function is useful to realize another function; thus, we can share the common function among two or more functions. The Boolean resubstitution is effectively carried out by solving a support minimization problem for an incompletely specified function. We can also handle satisfiability don't cares of an LUT network using the technique.

1 Introduction

Look-up table (LUT) based field programmable gate arrays (FPGAs) consist of an array of programmable logic blocks (LUTs) and a programmable routing network to connect the LUTs. Each LUT can realize any Boolean function with m (typically 4 or 5) inputs.

In logic synthesis, it is important to extract adequate sub-expressions from a large expression. Kernel extraction [1] is a supreme method for extractions when expressions are in sum-of-product forms. As for synthesis of an LUT network, functional decomposition [2, 3] can be considered as one of the methods, and many researchers have used it [4, 5, 6, 7, 8]. Furthermore, several researchers [5, 6, 7] proposed functional decomposition methods for functions represented by an ordered binary decision diagram (OBDD or simply BDD) [9]. We also use a BDD-based functional decomposition procedure to extract functions to be mapped to LUTs. Many of the researchers have used only disjunctive decomposition for LUT network synthesis. We use not only disjunctive decomposition but also nondisjunctive decomposition.

It is also important to identify common sub-expressions. Only functional decompositions for each of single output functions does not allow sharing LUTs among several functions. Resubstitution is a technique to check whether a function is useful to realize another function. By resubstituting a function into several functions, we can determine whether or not the function is common among several functions. Resubstitution techniques for a multi-level network of sum-of-product form can be found in [1, 10]. In this paper, we propose a new Boolean resubstitute technique customized for an LUT network. The Boolean resubstitution is effectively carried out based on support minimization for an incompletely specified function [11, 12, 13]. We can also handle satisfiability don't cares of an LUT network using the technique.

This paper is organized as follows. In Section 2, we introduce some notation about Boolean functions and BDDs and review previous works on functional decomposition and support minimization. In Section 3, we discuss our strategies for generating functions to be mapped to LUTs using functional decomposition. In Section 4, we discuss a new Boolean resubstitution technique for an LUT network, which is carried out by support minimization. Section 5 shows the experimental results and our observations on them. We conclude this paper in Section 6.

2 Preliminaries and Previous Works

2.1 Boolean function and BDD

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function over variables $(x_1, \ldots, x_n)$. Let $f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $f_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$. The **support** $sup(f)$ of a function f is the set of variables that the function depends on: $\forall x \in sup(f), f_{\overline{x}} \neq f_x$ and $\forall x \notin sup(f), f_{\overline{x}} = f_x$. A function f is called m-**feasible** if $|sup(f)| \leq m$; otherwise, f is called m-**infeasible**.

An ordered binary decision diagram (BDD) [9] is a directed acyclic graph representing Boolean functions (Figure 1). A BDD has two kinds of nodes: variable nodes and constant nodes. A constant node represents the Boolean constant 0 or 1. A variable node is associated with a Boolean variable and has two outgoing edges labeled 0 and 1, respectively. Traversing from any variable node to a constant node according to the assignment to variables, Boolean variables must occur at most only once and in a given order. We define the **level** of a variable node as follows. If there exists an edge from a variable node v_i to another variable node v_j, the level of v_i is smaller than that of v_j. We define a **variable order** π as a one-to-one mapping from levels to indexes of Boolean variables. The form of a BDD depends not only on a Boolean function but also on a variable order.

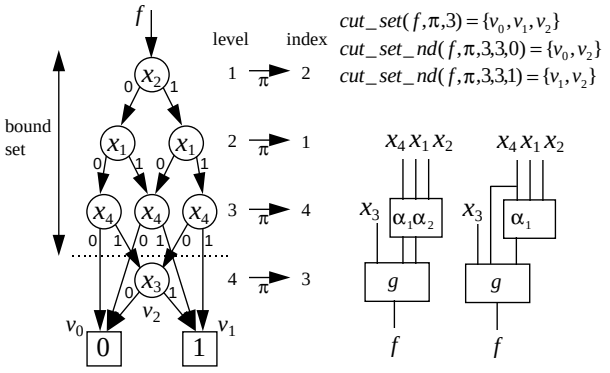

Figure 1: A BDD and its Functional Decompositions

2.2 Functional Decomposition

Functional decomposition of a function $f(x_1, \ldots, x_n)$ is of the form

$$f = g(\alpha_1(X^B), \ldots, \alpha_t(X^B), X^F) = g(\vec{\alpha}(X^B), X^F), \quad (1)$$

where X^B and X^F are sets of variables such that $X^B \cup X^F = \{x_1, \ldots, x_n\}$. The sets X^B and X^F are called the **bound set** and the **free set**, respectively. If $X^B \cap X^F = \emptyset$, the form is called **disjunctive decomposition**; otherwise, it is called **nondisjunctive decomposition**. g is called the **image** of a decomposition. In this paper, we will call $\alpha_1(X^B), \ldots, \alpha_t(X^B)$ the **subfunctions** of a decomposition.

The fundamental concept of a functional decomposition was studied by Ashenhurst [2] and Roth and Karp [3]. Recently, several researchers [5, 6, 7] have proposed BDD-based algorithms for functional decompositions. We use the following definitions and propositions that Lai, Pedram and Vrudhula [7] have proposed.

Definition 1 In the BDD of a function f with a variable order π, let $cut\_set(f, \pi, l)$ denote the set of nodes whose levels are greater than l and that have edges from nodes of level less than or equal to l. □

Proposition 1 (disjunctive decomposition) For an n-variable function f with a variable order π, if $|cut\_set(f, \pi, l)| \leq 2^t$, there exists a decomposition of the form (1) where $X^B = \{x_{\pi(1)}, \ldots, x_{\pi(l)}\}$ and $X^F = \{x_{\pi(l+1)}, \ldots, x_{\pi(n)}\}$. □

Definition 2 Let $s \leq l$ and $i \in \{0,1\}^{l-s+1}$. In the BDD of a function f with a variable order π, let $cut\_set\_nd(f, \pi, l, s, i)$ mean $cut\_set(f_i, \pi, l)$, where f_i is the function resulting from assigning i to f at the variables from level s to level l. □

Proposition 2 (nondisjunctive decomposition) For an n-variable function f with a variable order π, if $\forall i \in \{0,1\}^{l-s+1}$, $|cut\_set\_nd(f, \pi, l, s, i)| \leq 2^t$, there exists a decomposition of the form (1) where $X^B = \{x_{\pi(1)}, \ldots, x_{\pi(l)}\}$ and $X^F = \{x_{\pi(s)}, \ldots, x_{\pi(n)}\}$. □

Figure 1 shows the concepts of *cut_set* and *cut_set_nd* and their relations to decomposition forms. There exists a decomposition of the form $f = g(\alpha_1(x_2, x_1, x_4), \alpha_2(x_2, x_1, x_4), x_3)$ because $|cut\_set(f, \pi, 3)| = 3 \leq 2^2$. The form requires three 3-input LUTs. There also exists a decomposition of the form $f = g(\alpha_1(x_2, x_1, x_4), x_4, x_3)$ because $|cut\_set\_nd(f, \pi, 3, 3, 0)| = 2 \leq 2^1$ and $|cut\_set\_nd(f, \pi, 3, 3, 1)| = 2 \leq 2^1$. The form requires two 3-input LUTs.

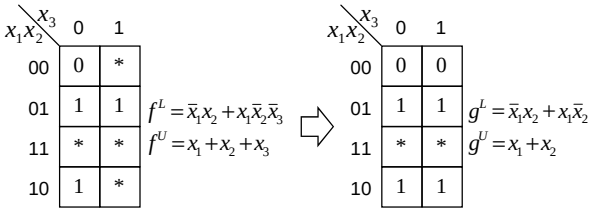

Figure 2: Support Minimization

2.3 Support Minimization

Let the relation of two functions $f \cdot \overline{g} = 0$ be denoted by $f \leq g$. An incompletely specified function $\hat{f} : \{0,1\}^n \rightarrow \{0, 1, *\}$ ($*$ means don't care) can be given by an **interval** $[f^L, f^U]$, where f^L and f^U are complete specified functions satisfying $f^L \leq f^U$. The set of minterms that map to 0, 1 and $*$ (**on-set**, **off-set** and **dc-set**) is given by $\{X \mid f^L(X) = 1\}$, $\{X \mid f^U(X) = 0\}$ and $\{X \mid f^L(X) = 0, f^U(X) = 1\}$, respectively. We will use the notation $\hat{f}$ instead of f to represent that a function may be incompletely specified. The support of an incompletely specified function $\hat{f}$ is given by $sup(\hat{f}) = sup(f^L) \cup sup(f^U)$.

A completely specified function f is said to be **compatible** with an incompletely specified function $[f^L, f^U]$, denoted by $f \prec [f^L, f^U]$, if $f^L \leq f \leq f^U$. In the same manner, an incompletely specified function $[g^L, g^U]$ is said to be compatible with an incompletely specified function $[f^L, f^U]$ if $f^L \leq g^L \leq g^U \leq f^U$.

Support minimizations for incompletely specified functions were discussed in [11, 12, 13]. We address a support minimization problem as follows: given an incompletely specified function $[f^L, f^U]$, find a compatible function $\hat{g}$ whose support $sup(\hat{g})$ is the smallest. For example, consider the incompletely specified function $\hat{f} = [f^L = \overline{x_1}x_2 + x_1\overline{x}_2\overline{x}_3,\ f^U = x_1 + x_2 + x_3]$ shown in Figure 2. The support $sup(\hat{f})$ is $\{x_1, x_2, x_3\}$. By replacing the dc-set with on-set or off-set, we can get a compatible function $\hat{g} = [g^L = \overline{x_1}x_2 + x_1\overline{x_2},\ g^U = x_1 + x_2]$ whose support $sup(\hat{g})$ is $\{x_1, x_2\}$. All the completely specified functions compatible with $\hat{g}$, $\overline{x_1}x_2 + x_1\overline{x_2}$ and $x_1 + x_2$, are also compatible with $\hat{f}$.

We use the following definition and proposition found in [11].

Definition 3 Let $f(x_1, \ldots, x_n)$ be a Boolean function and let R and S be subsets of $\{x_1, \ldots, x_n\}$.

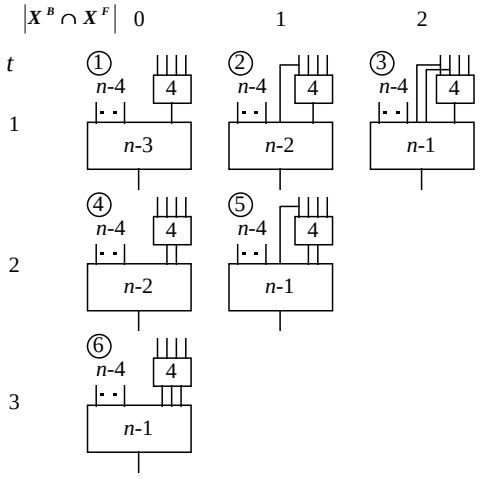

Figure 3: Decomposition Forms and their Costs

```
/* global variables that store the best solution */
mincost;
minπ;
/* f is a BDD and π represents the variable order of f */
/* Nin is the number of variables included in bound set */
/* Nout is the number of variables excluded from bound set */
bound_set(f, π, Nin, Nout) {
    if ( Nin = m or Nout = n−m ) { /* terminal case */
        cost = least_cost_decomposition(f, π);
        if ( mincost > cost ) {
            mincost = cost;
            minπ = π;
        }
    }
    else { /* non-terminal case */
        /* include the variable of level Nin+1 in bound set */
        bound_set(f, π, Nin+1, Nout);
        /* exclude the variable of level Nin+1 from bound set */
        (newf, newπ) = jump_down(f, π, Nin+1, n−Nout);
        bound_set(newf, newπ, Nin, Nout+1);
    }
}
```

Figure 4: Decomposition Tests for All the Bound Sets

disjunctive eliminant $edis(f,R)$ and **conjunctive eliminant** $econ(f,R)$ are defined as follows.

$edis(f,\emptyset) = f$
$edis(f,\{x_i\}) = f_{\overline{x_i}} + f_{x_i}$, $i \in \{1,\ldots,n\}$
$edis(f,R \cup S) = edis(edis(f,R),S)$
$econ(f,\emptyset) = f$
$econ(f,\{x_i\}) = f_{\overline{x_i}} \cdot f_{x_i}$, $i \in \{1,\ldots,n\}$
$econ(f,R \cup S) = econ(econ(f,R),S)$ □

Proposition 3 Let $\hat{f} = [f^L, f^U]$ be an incompletely specified function and E be a subset of $sup(\hat{f})$. If $edis(f^L,E) \leq econ(f^U,E)$, $\hat{f}'$=$[edis(f^L,E), econ(f^U,E)]$ is compatible with $\hat{f}$ and $sup(\hat{f}') = sup(\hat{f})-E$. □

According to Proposition 3, a support minimization problem can be solved by finding one of the largest subsets E of eliminated variables. In Figure 2, $g^L = edis(f^L,\{x_3\})$ and $g^U = econ(f^U,\{x_3\})$. If we let E be $\{x_1\}$ or $\{x_2\}$, the inequality $edis(f^L,E) \leq econ(f^U,E)$ is not satisfied. Thus, $\{x_3\}$ is the largest subset of eliminated variables.

3 Generating m-feasible Functions using Functional Decomposition

3.1 Decomposition Forms and their Costs

We assume that every LUT in a network can realize any Boolean function with m ($m \geq 3$) inputs and 1 output. Our synthesis procedure iterates functional decompositions to break a function into new functions having fewer supports until the supports of all functions are less than or equal to m. Functional decompositions are applied not to multiple output functions but to each of the single output functions. How to share common LUTs among several functions will be discussed in Section 4.

Given an m-infeasible function, we try to decompose the function such that the size of a bound set X^B is equal to m. The subfunctions $\alpha_1(X^B),\ldots,\alpha_t(X^B)$ of the decomposition can be allocated to LUTs because they are m-feasible. If the image g of the decomposition is m-feasible, it can also be allocated to an LUT; otherwise, it becomes a new m-infeasible function.

We are only interested in a decomposition whose image has fewer supports than the original function; therefore, an inequality $t+|X^F| < |X^B|+|X^F|-|X^B \cap X^F|$ is given as the condition for decomposability. From this inequality and $|X^B| = m$, we can derive $t + |X^B \cap X^F| < m$. Because $t \geq 1$ and $|X^B \cap X^F| \geq 0$, we can consider $m(m-1)/2$ kinds of decomposition forms. For example, if $m = 4$, the 6 kinds of decomposition forms shown in Figure 3 can be considered. We evaluate the costs of decomposition forms as follows. Decompositions of fewer t have less cost, and in decompositions of equal t, those of fewer $|X^B \cap X^F|$ have less cost. In Figure 3, the number in a circle represents the cost of decomposition for $m = 4$.

If a function f is not decomposable in any of the forms in Figure 3, we apply an expansion $f = \overline{x_i} \cdot f_{\overline{x_i}} + x_i \cdot f_{x_i}$ using a variable $x_i \in sup(f)$ to the function. Consequently, a function $\overline{x_1} \cdot x_2 + x_1 \cdot x_3$ can be realized by an LUT and $f_{\overline{x_i}}$ and f_{x_i} become new m-infeasible functions.

3.2 Decomposition Tests

For a function f to be decomposed, we examine decomposition forms and their costs for all the bound sets X^B of size m and find the least cost decomposition form. If $sup(f)$ is n, the number of all the bound sets of size m is ${}_nC_m$. According to Propositions 1 and 2, the variables in a bound set should be ordered from level 1 to level m in the BDD representation. Thus, we need to construct ${}_nC_m$ BDDs of different variable orders. We change the variable order of a BDD by $jump\_down$ operations. $jump\_down(i,j)$ moves the variable at level i to level j ($i < j$) and decreases the levels of all the variables from level j to level $i-1$ by 1. Figure 4 shows a recursive algorithm to

examine decomposition forms and their costs for all the bound sets of size m. The computation starts by calling $bound\_set(f, \pi, 0, 0)$.

3.3 Encoding and Don't Cares

Even if the bound set and free set that give the least cost decomposition are found, the image g and the subfunctions $\alpha_1, \ldots, \alpha_t$ are not uniquely determined. Different encoding of *cut_set* or *cut_set_nd*'s yield different functions g and $\alpha_1, \ldots, \alpha_t$. Discussions of encoding problems were found in [5, 8]. However, in our implementation up to now, we encode *cut_set* or *cut_set_nd*'s in a straightforward way: assigning the binary representation of i to the i-th element. For example, in Figure 1 the elements of $cut\_set(f, \pi, 3)$ are encoded in $\alpha_2\alpha_1 = \{v_0 : 00, v_1 : 01, v_2 : 10\}$.

Since $\alpha_2\alpha_1$ never has the value 11, the minterms of the image, $g(1, 1, 0)$ and $g(1, 1, 1)$, can be handled as don't cares. Unless $|cut\_set(f, \pi, l)| = 2^t$ or $\forall i \in \{0, 1\}^{l-s+1}$, $|cut\_set\_nd(f, \pi, l, s, i)| = 2^t$, we can encode *cut_set* and *cut_set_nd*'s such that the image g has don't cares. The Boolean resubstitution technique discussed in the next section can identify such don't cares because it uses satisfiability don't cares.

4 Boolean Resubstitution based on Support Minimization

4.1 Problem Formulation

Only the procedure presented in Section 3 does not allow sharing LUTs among several functions. Resubstitution, discussed in [1, 10], is a technique to check whether an existing function is useful to realize other functions. For example, let $y_1 = x_1x_2 + x_1x_3 + x_4$ and $y_2 = x_2 + x_3$. If we resubstitute y_2 into y_1, y_1 can be represented as $y_1 = x_1(x_2 + x_3) + x_4 = x_1y_2 + x_4$, which costs less than the original.

The image of a functional decomposition sometimes becomes an incompletely specified function as shown in Subsection 3.3. If a function that plans to utilize other functions is incompletely specified, we will find a resubstitution form such that the resultant function of the substitution is compatible with the original function. In the case of LUT network synthesis, support size can be considered as one of the costs of a Boolean function. Therefore, we formulate Boolean resubstitution problem as follows.

Problem 1 Let $\hat{f}$ be an incompletely specified function whose support is X and let $h_1, \ldots, h_s$ be completely specified functions whose supports are X', $(X' \subseteq X)$. Find a function $\hat{g}$ such that $\hat{g}(h_1(X'), \ldots, h_s(X'), X'')$, $(X'' \subset X)$ is compatible with $\hat{f}$, $sup(\hat{g}) < sup(\hat{f})$, and $sup(\hat{g})$ is the minimum. □

4.2 An Algorithm based on Support Minimization

Let y be a variable such that $y = h(X')$. If y is utilized by another function, we do not care about the minterms represented by $y \neq h(X')$. Such don't cares are called satisfiability don't cares (SDCs) [14]. Boolean resubstitution can be carried out by support minimization for an incompletely specified function that is generated by considering SDCs.

```
/* global variables that store the best solution */
minNsup;
minf^L;
minf^U;
/* the function is given by [f^L, f^U] */
/* Nsup is the size of the support of the function */
/* elim is the index of the variable to be eliminated */
support_min(f^L, f^U, Nsup, elim) {
    if ( elim ≤ 0 ) return; /* terminal case */
    if ( Nsup−elim ≥ minNsup ) return;
    newf^L = econ(f^L, x_elim);
    newf^U = edis(f^U, x_elim);
    if ( newf^L ≤ newf^U ) { /* exclude x_elim */
        if ( Nsup−1 < minNsup ) {
            minNsup = Nsup−1;
            minf^L = newf^L;
            minf^U = newf^U;
        }
        support_min(newf^L, newf^U, Nsup−1, elim−1);
    }
    support_min(f^L, f^U, Nsup, elim−1); /* include x_elim */
}
```

Figure 5: Support Minimization

We will show our procedure to solve Problem 1.

1. Let $y_1, \ldots, y_s$ be variables such that $y_i = h_i(X')$, $i \in \{1, \ldots, s\}$. Consider $D = \sum_{i \in \{1,\ldots,s\}} y_i \neq h_i(X')$ as the SDC among X' and $y_1, \ldots, y_s$.
2. Let $\hat{f}$ be expressed by an interval $[f^L, f^U]$. Consider an incompletely specified function $\hat{g}_{INIT}$ given by an interval $[f^L \cdot \overline{D}, f^U + D]$. The support of $\hat{g}_{INIT}$ is $\{y_1, \ldots, y_s\} \cup X$. Because D becomes 0 by substituting $h_i(X')$ for y_i $(\forall i \in \{1, \ldots, s\})$, $\hat{f} = \hat{g}_{INIT}(h_1(X'), \ldots, h_s(X'), X)$.
3. Apply a support minimization procedure to $\hat{g}_{INIT}$ and find a compatible function $\hat{g}$ that has a minimal support. If $sup(\hat{g}) < |X|$, the resubstitution has succeeded; otherwise the resubstitution has failed. Let E be the set of eliminated variables in the support minimization. Then, $\hat{f} \succ \hat{g}(h_1(X'), \ldots, h_s(X'), X'')$, where $X'' = X - E$.

Lin [13] gave a BDD-based algorithm to find all the supports. Although our method is also based on BDD representation, it only finds just one of minimal supports. Figure 5 shows our recursive algorithm. For an interval $\hat{f} = [f^L, f^U]$ and $sup(\hat{f}) = \{x_1, \ldots, x_n\}$, the computation starts by calling $support\_min(f^L, f^U, n, n)$. The algorithm is not time consuming because the search space can be pruned in the following two cases. If $newf^L \leq newf^U$ is not satisfied, no compatible function can be found from the search state; also, if $Nsup{-}elim \geq minNsup$, any compatible function whose support size is less than $minNsup$ cannot be found from the search state.

4.3 Resubstitution of m-feasible functions

We will now show our synthesis procedure using not only functional decomposition but also Boolean resubstitution. The procedure iterates the following steps until all the functions become m-feasible.

1. Let $\hat{f}_1, \ldots, \hat{f}_k$ be functions that are m-infeasible, and let f_i be a completely specified function compatible with $\hat{f}_i$, $(i \in \{1, \ldots, k\})$.
2. Find the least cost decomposition form for each of functions $f_1, \ldots, f_k$ by the procedure described in Section 3. Let $\vec{\alpha}_i(X_i^B)$ be the subfunctions generated in the least cost decomposition form of f_i $(i \in \{1, \ldots, k\})$. Note that $\vec{\alpha}_i(X_i^B)$ are m-feasible functions.
3. For all $i, j \in \{1, \ldots, k\}$, try to resubstitute $\vec{\alpha}_i(X_i^B)$ into $\hat{f}_j$. Let $success_i$ be a subset of $\{1, \ldots, k\}$ such that $j \in success_i$ if and only if the resubstitution of $\vec{\alpha}_i(X_i^B)$ into $\hat{f}_j$ is successful. Let $\hat{g}_{ij}$ be the function generated by the resubstitution of $\vec{\alpha}_i(X_i^B)$ into $\hat{f}_j$.
4. We calculate $gain_i = \sum_{j \in success_i} |sup(\hat{f}_j)| - |sup(\hat{g}_{ij})|$, which means how many fanins of functions are reduced if $\vec{\alpha}_i(X_i^B)$ is used. Find the best subfunction $\vec{\alpha}_b(X_b^B)$ among $\vec{\alpha}_1, \ldots, \vec{\alpha}_k$ such that $gain_b$ is the maximum.
5. Allocate LUTs for $\vec{\alpha}_b(X_b^B)$. For all $j \in success_b$, replace $\hat{f}_j$ with $\hat{g}_{bj}$. If there exist m-feasible functions among $\hat{f}_1, \ldots, \hat{f}_k$, allocate an LUT for each of them.

In our synthesis procedure, LUTs are allocated from the side of primary inputs to the side of primary outputs. Thus, the SDCs of the circuit can be used to simplify the functions that have not been mapped to LUTs. The resubstitution technique helps us to easily handle the SDCs. In step 3, if $i = j$, it is clear that the resubstitution is successful. However, we actually resubstitute $\vec{\alpha}_i(X_i^B)$ into $\hat{f}_i$ and generate $\hat{g}_{ii}$ to easily identify the don't cares caused by encoding of *cut_set* or *cut_set_nd*'s.

4.4 Resubstitution of Another Primary Output

There exists a case where an primary output function can be realized simply by resubstituting another primary output function into the function. Such a case may not be detected by the procedure described so far.

We apply resubstitution of another primary output at the beginning of our synthesis process in the following manner. Let $f_1, \ldots, f_k$ be output functions. For all $i, j \in \{1, \ldots, k\}$, try to resubstitute f_i into f_j. In fact, we apply the resubstitution according to the following priority to restrain the depth of a circuit from increasing.

1. In the case that the function generated by the resubstitution is m-feasible: only one additional LUT is needed to realize the function.
2. In the case that f_i is m-feasible.
3. All other cases.

5 Experimental Results

The logic synthesis procedure presented so far has been implemented. The input to the program is a combinational (multi-level or two-level) circuit. The circuit description is transformed to BDD representations of primary outputs in terms of primary inputs. The synthesis procedure is then carried out to construct a network of m-input LUTs.

Table 1 shows the experimental results for several of the MCNC [16] benchmark circuits listed in the column "circuit". The columns "size" and "dep" show the number of 5-input and 1-output LUTs and the depth of the circuit, respectively. The column "time" shows CPU time in seconds on a SPARCstation 10/51. We limited the maximum number of usable BDD nodes to 1,000,000.

Three experiments were performed on each benchmark circuit. The column "without resub." means that Boolean resubstitutions were not carried out. In the case, no LUT is shared among two or more primary outputs. The column "with resub." means that resubstitutions of m-feasible functions were carried out. The column "resub. PO" means that resubstitution of a primary output into another primary output was also applied at the beginning of the synthesis process.

In each experiment, functional decompositions for all the bound sets of size m were carried out. Thus the execution time of circuits having a primary output of many supports tended to be enormous. For some larger circuits, e.g., C880, the method failed due to memory overflow.

Comparing the columns "without resub." and "with resub.", we can observe the following. Boolean resubstitution is very effective because it reduces the number of LUTs sharing common LUTs among several functions without increasing the circuit depth in many cases. Furthermore, the execution time of Boolean resubstitution, most of which is spent in support minimization, is not expensive. Comparing the columns "with resub." and "resub. PO", we can observe the following. Although resubstitution of a primary output generally reduces the number of LUTs and execution time, it tends to increase the depth of a circuit. In order to compare our results with other LUT network synthesizers, we pick up the results found in [4, 15, 5]. We observe that our method gives good results for most of the circuits.

6 Conclusion

We have presented a logic synthesis method for an LUT network using functional decompositions and Boolean resubstitutions based on support minimizations. Functional decompositions are used to enumerate candidates of m-feasible functions to be mapped to LUTs. After the enumeration, the best m-feasible functions are determined by resubstituting the candidates into all the m-infeasible functions.

In each of synthesis steps, we generate only one m-feasible function as a candidate from each of m-infeasible functions. To synthesize LUT networks of higher quality, methods to enumerate more candidates will be required. The Boolean

Table 1: Experimental Results (5-input 1-output LUTs)

circuit			without resub.			with resub.			resub. PO			[4]	[15]	[5]
name	in	out	size	dep	time	size	dep	time	size	dep	time		size	
5xp1	7	10	15	2	0.2	11	2	0.2	10	3	0.2	18	27	12
9sym	9	1	7	3	0.7	7	3	0.8	7	3	0.7	7	59	6
alu2	10	6	48	6	13.8	48	6	16.7	48	6	17.0	109	116	54
alu4	14	8	172	7	291.4	90	7	234.7	56	9	38.4	55	195	
apex4	9	19	374	5	76.7	374	5	230.8	374	5	240.5	412	558	
apex6	135	99	192	6	623.9	161	4	755.0	155	8	208.7	182	212	204
apex7	49	37	120	5	164.1	61	5	126.9	54	5	12.3	60	64	56
b12	15	9	16	3	0.3	16	3	0.3	16	3	0.4			
b9	41	21	53	4	11.3	39	5	12.9	37	5	6.5	39		35
clip	9	5	18	3	3.7	11	3	3.8	14	8	2.6	28		
cordic	23	2	15	5	45.9	9	4	48.0	10	6	27.6			
count	35	16	52	4	5.3	31	6	11.3	31	6	11.0	31	31	32
duke2	22	29	175	7	432.2	155	7	489.4	150	8	451.8	110	120	
f51m	8	8	12	3	0.3	10	3	0.3	8	4	0.1	17		12
misex1	8	7	12	2	0.2	10	2	0.2	10	4	0.2	11	19	11
misex2	25	18	40	3	1.5	36	3	1.8	36	4	2.3	28		29
misex3	14	14	195	9	503.4	213	9	670.4	120	13	136.6			
misex3c	14	14	107	9	132.4	99	9	131.1	92	11	101.6			
rd73	7	3	8	2	0.1	6	3	0.2	6	3	0.2	6		7
rd84	8	4	12	3	0.4	7	3	0.5	8	5	0.3	10	73	12
sao2	10	4	23	4	7.4	21	4	8.2	21	4	7.6	28		46
t481	16	1	5	3	4.4	5	3	4.5	5	3	4.5			
vg2	25	8	44	5	217.5	21	5	253.0	17	4	4.2	20	21	
z4ml	7	4	6	2	0.2	5	2	0.2	4	2	0.1	5	6	5
total			1721	105	2537.3	1446	106	3001.2	1289	132	1275.4			

resubstitution technique proposed in this paper is not time consuming, which will allow effective identification of common LUTs from large amount of candidates.

For functions that do not have many supports, examining decomposition for all the bound sets generates a good m-feasible function to be mapped to LUTs without spending a large amount of time. However, for functions with many supports, the examinations are time consuming and sometimes fail due to memory overflow. Thus, heuristics that avoid the expensive search will be needed for synthesis of larger circuits.

Acknowledgement

We would like to thank Kiyoshi Oguri for his suggestions and encouragements during this reserch.

References

[1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. CAD*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[2] R. L. Ashenhurst, "The Decomposition of Switching Functions," in *Proc. of an International Symposium on the Theory of Switching*, Apr. 1957.

[3] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM journal*, pp. 227–238, Apr. 1962.

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

[5] S. Chang and M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graphs," in *ICCD*, pp. 159–162, Oct. 1992.

[6] T. Sasao, "FPGA design by generalized functional decomposition," in *Logic Synthesis and Optimization* (T. Sasao, ed.), pp. 233–258, Kluwer Academic Publishers, 1993.

[7] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *30th DAC*, pp. 642–647, June 1993.

[8] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," in *31st DAC*, pp. 408–414, June 1994.

[9] R. E. Bryant, "Graph-based algorithm for Boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 667–691, Aug. 1986.

[10] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean Resubstitution With Permissible Functions and Binary Decision Diagrams," in *27th DAC*, pp. 284–289, June 1990.

[11] F. M. Brown, *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer Academic Publishers, 1990.

[12] M. Fujita and Y. Matsunaga, "Multi-level Logic Minimization based on Minimal Support and its Application to the Minimization of Look-up Table Type FPGAs," in *ICCAD*, pp. 560–563, Nov. 1991.

[13] B. Lin, "Efficient Symbolic Support Manipulation," in *ICCD*, pp. 513–516, Oct. 1993.

[14] H. Savoj, R. K. Brayton, and H. J. Touati, "Extracting Local Don't Cares for Network Optimization," in *ICCAD*, pp. 514–517, Nov. 1991.

[15] R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," in *28th DAC*, pp. 227–232, June 1991.

[16] S. Yang, *Logic synthesis and optimization benchmarks user guide version 3.0.* MCNC, Jan. 1991.