PAPER *Special Issue on Synthesis and Verification of Hardware Design*

Logic Synthesis for Look-Up Table Based FPGAs Using Functional Decomposition and Boolean Resubstitution

Hiroshi SAWADA[†], ***Member***, **Takayuki SUYAMA**[†], ***Nonmember, and*** **Akira NAGOYA**[†], ***Member***

SUMMARY 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 for each of single-output functions. To share LUTs among several functions, we use a new Boolean resubstitution technique. Resubstitution is used to determine whether an existing function is useful to realize another function; thus, we can share common functions among two or more functions. The Boolean resubstitution proposed in this paper is customized for an LUT network synthesis because it is based on support minimization for an incompletely specified function. Experimental results show that our synthesis method produces a small size circuit in a practical amount of time.

key words: *FPGA, look-up table (LUT), functional decomposition, Boolean resubstitution, support minimization*

1. Introduction

Field programmable gate arrays (FPGAs) are logic devices that can be programmed by users. The look-up table (LUT) based FPGA is a popular one. It consists of an array of programmable logic blocks, which contain LUTs, and a programmable routing network to connect them. Each LUT can realize any Boolean function with m (typically 4 or 5) inputs.

Functional decomposition [1], [2] is widely used in logic synthesis methods for LUT based FPGAs [3]–[10]. The form of a functional decomposition is $f(X) = g(\vec{\alpha}(X^B), X^F)$. If we let the size of X^B be the number m of inputs of an LUT, functions $\vec{\alpha}$ can be mapped to LUTs. Several researchers [4]–[6] have proposed efficient decomposition methods based on an ordered binary decision diagram (OBDD or simply BDD) [11], which enable us to decompose larger functions. We also use a BDD-based functional decomposition to extract functions to be mapped to LUTs.

Only functional decompositions for each of single-output functions does not allow sharing LUTs among several functions. Recently, several researchers [8]–[10] have proposed functional decomposition algorithms for multiple-output functions to share LUTs among them. Since these algorithms try to share common functions among the overall multiple-output function, the problem of selecting an appropriate set of single-output functions has to be solved. Moreover they discussed only the case of disjunctive decomposition, where X^B and X^F have no common variable; the case of nondisjunctive decomposition, where X^B and X^F have at least one common variable, was not discussed.

In this paper, we propose a new Boolean resubstitution technique to share LUTs among several functions. Resubstitution is used 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 them. Resubstitution techniques for a multi-level network of sum-of-products forms can be found in [12], [13]. The Boolean resubstitution proposed in this paper is customized for an LUT network because it is based on support minimization for an incompletely specified function [14]–[16].

The logic synthesis method presented in this paper iterates the following two steps.

1. For each of single-output functions, apply functional decomposition to generate candidate subfunctions to be mapped into LUTs.
2. Resubstitute each of candidate subfunctions into each of single-output functions, to find the best subfunction for the whole multiple-output function.

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

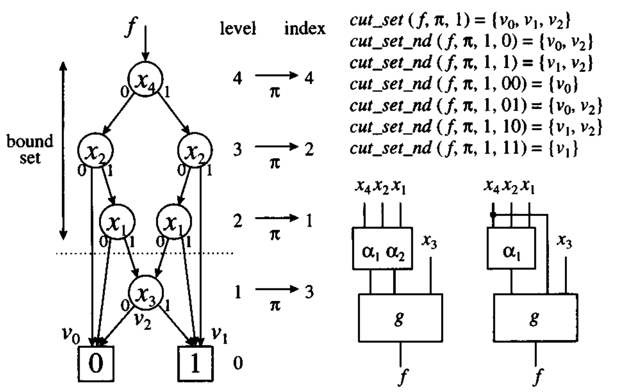
2. Preliminaries

2.1 Boolean Function and BDD

Let $f(x_1, \ldots, x_n)$: $\{0,1\}^n \rightarrow \{0,1\}$ be a completely specified Boolean function. The **cofactors** of f with respect to $x_i = 1$ and $x_i = 0$ are $f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $f_{\bar{x}_i} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$, respectively. The **support** $sup(f)$ of a function f is the set of variables that the function depends on: $\forall x \in$

Manuscript received February 3, 1997.
Manuscript revised April 14, 1997.

[†]The authors are with NTT Communication Science Laboratories, Kyoto-fu, 619–02 Japan.



$cut\_set(f, \pi, 1) = \{v_0, v_1, v_2\}$
$cut\_set\_nd(f, \pi, 1, 0) = \{v_0, v_2\}$
$cut\_set\_nd(f, \pi, 1, 1) = \{v_1, v_2\}$
$cut\_set\_nd(f, \pi, 1, 00) = \{v_0\}$
$cut\_set\_nd(f, \pi, 1, 01) = \{v_0, v_2\}$
$cut\_set\_nd(f, \pi, 1, 10) = \{v_1, v_2\}$
$cut\_set\_nd(f, \pi, 1, 11) = \{v_1\}$

Fig. 1 A BDD and its functional decomposition forms.

$sup(f), f_{\bar{x}} \neq f_x$ and $\forall x \notin sup(f), f_{\bar{x}} = f_x$. A function f is called **m-feasible** if $|sup(f)| \leq m$; otherwise, f is called **m-infeasible**.

An ordered binary decision diagram (BDD)[11] is a directed acyclic graph representing Boolean functions (Fig. 1). A BDD has two kinds of nodes: variable nodes and constant nodes. A constant node represents a Boolean constant 0 or 1. A variable node v is associated with a Boolean variable x_i and represents a function f^v. It has two outgoing edges labeled with 0 and 1, which point the nodes representing functions $f^v_{\bar{x}_i}$ and $f^v_{x_i}$, respectively. When traversing from any variable node to a constant node according to the cofactors of variables, each variable must occur at most only once and in a given order. We define a **level** of a node as follows: if there exists an edge from a node v_i to another node v_j, the level of v_i is more than that of v_j. We also define a **variable order** π to be a one-to-one mapping from levels to indexes of variables.

2.2 Functional Decomposition

Functional decomposition of a function f is

$$\begin{aligned} f(x_1,\dots,x_n) &= g(\alpha_1(X^B),\dots,\alpha_t(X^B),X^F) \quad (1)\\ &= g(\vec{\alpha}(X^B),X^F), \end{aligned}$$

where X^B and X^F are sets of variables such that $X^B \cup X^F = \{x_1,\dots,x_n\}$. 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 $\alpha_1(X^B),\dots,\alpha_t(X^B)$ the **subfunctions** of a decomposition.

The fundamental concept of a functional decomposition was studied by Ashenhurst[1] and Roth and Karp[2]. Recently, several researchers[4]–[6] have proposed BDD-based algorithms for functional decompositions. We use the following definitions and propositions similar to ones found in[6].

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 less than or equal to l and that have edges from nodes of level greater than 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 $X^B = \{x_{\pi(n)},\dots,x_{\pi(l+1)}\}$ and $X^F = \{x_{\pi(l)},\dots,x_{\pi(1)}\}$. □

Definition 2: In the BDD of a function f with a variable order π, let $cut\_set\_nd(f,\pi,l,b_1\cdots b_k)$ mean $cut\_set(f(x_{\pi(n)} = b_1,\dots,x_{\pi(n-k+1)} = b_k),\pi,l)$, where $b_i \in \{0,1\}$. □

Proposition 2 (nondisjunctive decomposition): For an n-variable function f with a variable order π, if $\forall\, b_1\cdots b_k \in \{0,1\}^k$, $|cut\_set\_nd(f,\pi,l,b_1\cdots b_k)| \leq 2^t$, there exists a decomposition of the form (1) where $X^B = \{x_{\pi(n)},\dots,x_{\pi(l+1)}\}$ and $X^F = \{x_{\pi(n)}\cdots, x_{\pi(n-k+1)}, x_{\pi(l)},\dots,x_{\pi(1)}\}$. □

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

2.3 Support Minimization

Let the relation of two functions $f \cdot \bar{g} = 0$ be denoted by $f \leq g$. An incompletely specified function $\hat{f} : \{0,1\}^n \to \{0,1,*\}$ ($*$ means don't care) can be given by an **interval** $[f^L, f^U]$, where f^L and f^U are completely specified functions satisfying $f^L \leq f^U$. The sets of minterms that map to 0, 1 and $*$ (**on-set, off-set** and **dc-set**) are given by $\{Y \mid f^L(Y) = 1\}$, $\{Y \mid f^U(Y) = 0\}$ and $\{Y \mid f^L(Y) = 0, f^U(Y) = 1\}$, respectively. We will use the notation $\hat{f}$ instead of f to represent that the 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[14]–[16]. 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 $\hat{f} = [f^L = \bar{x}_1 x_2 + x_1 \bar{x}_2 \bar{x}_3,\ f^U = x_1 + x_2 + x_3]$ shown in

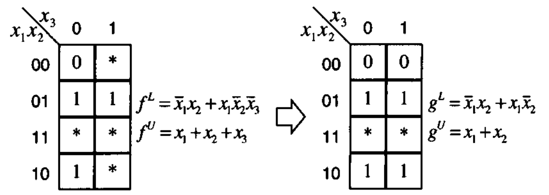

$f^L = \bar{x}_1x_2 + x_1\bar{x}_2\bar{x}_3$
$f^U = x_1 + x_2 + x_3$
$g^L = \bar{x}_1x_2 + x_1\bar{x}_2$
$g^U = x_1 + x_2$

Fig. 2 Support minimization.

Fig. 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 obtain 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 $\{x_1, x_2\}$. 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 $\hat{f}$.

Definition 3 ([14]): Let $f(x_1, \ldots, x_n)$ be a Boolean function and let R and S be nonempty subsets of $\{x_1, \ldots, x_n\}$. **Disjunctive eliminant** $edis(f, R)$ and **conjunctive eliminant** $econ(f, R)$ are defined as follows:

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

Proposition 3 ([14]): 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 $sup(\hat{f}') = sup(\hat{f}) - E$. $\square$

According to Proposition 3, a support minimization problem can be solved by finding one of the largest subsets E of eliminated variables. In Fig. 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

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 will be discussed in Sect. 4.

3.1 Decomposition Forms and Their Costs

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.

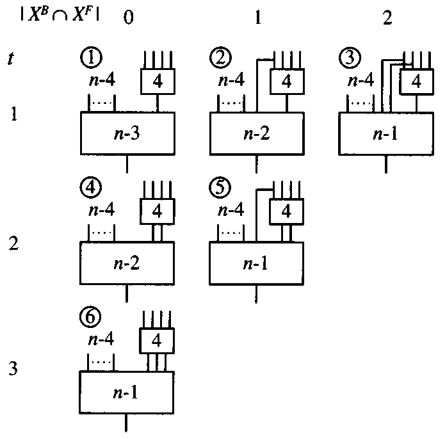


Fig. 3 Decomposition forms and their costs.

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 Fig. 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 Fig. 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 Fig. 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 bound sets X^B of size m and find the least cost decomposition form. If $sup(f)$ is n, the number of the bound sets is ${}_nC_m$. According to Propositions 1 and 2, the variables in a bound set should be located from level n to level $n - m + 1$. 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 increases the levels of all variables from level $i - 1$ to level j by 1. Figure 4 shows a recursive algorithm

```
/* global variables that store the best solution */
$mincost$;
$min\pi$;
/* $f$ is a BDD and $\pi$ 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 */
$bound\_set(f,\ \pi,\ Nin,\ Nout)$ {
    if ( $Nin = m$ or $Nout = n - m$ ) { /* terminal case */
        $cost\ =\ least\_cost\_decomposition(f,\ \pi)$;
        if ( $mincost > cost$ ) {
            $mincost\ =\ cost$;
            $min\pi\ =\ \pi$;
        }
    }
    else { /* non-terminal case */
        /* include the variable of level $n - Nin$ in bound set */
        $bound\_set(f,\ \pi,\ Nin+1,\ Nout)$;
        /* exclude the variable of level $n - Nin$ from bound set */
        $(newf,\ new\pi) = jump\_down(f,\ \pi,\ n - Nin,\ Nout+1)$;
        $bound\_set(newf,\ new\pi,\ Nin,\ Nout+1)$;
    }
}
```

Fig. 4 Decomposition tests for all bound sets of size m.

to examine decomposition forms and their costs for all bound sets of size m. The computation starts by calling $bound\_set(f,\pi,0,0)$. For nondisjunctive decompositions, the variables in both bound set and free set should be located from level n to level $n-k+1$. This is also done by changing the variable order of a BDD.

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 [4], [7]. We encode *cut_set* in a straightforward way: assigning the binary representation of i to the i-th element. For example, in Fig. 1 the elements of $cut\_set(f,\pi,1)$ are encoded in $\alpha_2\alpha_1 = \{v_0 : 00,\ v_1 : 01,\ v_2 : 10\}$. In case of *cut_set_nd*'s, we pay attention to assign the same codes to the same BDD nodes if possible.

In the above example, 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 $\forall\ b_1\cdots b_k \in \{0,1\}^k$, $|cut\_set\_nd(f,\pi,l,b_1\cdots b_k)| = 2^t$, 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 to Share LUTs

Only the procedure presented in Sect. 3 does not allow sharing LUTs among several functions. This section discusses a new Boolean resubstitution technique to share LUTs among them. Resubstitution, discussed in [12], [13], 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.

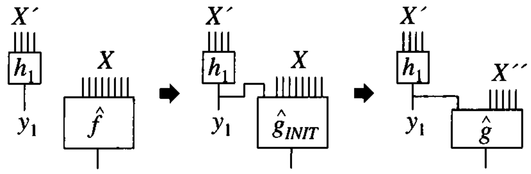


Fig. 5 Boolean resubstitution based on support minimization.

4.1 Problem Formulation

In this paper, 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 $\hat{f}$, and $sup(\hat{g})$ is the minimum. □

In the case of LUT network synthesis, support size can be considered as one of the costs of a Boolean function. The image of a functional decomposition sometimes becomes an incompletely specified function as shown in Sect. 3.3.

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 $y \not\equiv h(X')$. Such don't cares are called satisfiability don't cares (SDCs) [17]. Boolean resubstitution can be carried out by support minimization for an incompletely specified function generated by considering SDCs.

We will show our procedure to solve Problem 1. Figure 5 shows the case of $s = 1$.

1. Let $y_1,\ldots,y_s$ be variables such that $y_i = h_i(X')$, $i \in \{1,\ldots,s\}$. Consider $D = \sum_{i\in\{1,\ldots,s\}} y_i \not\equiv h_i(X')$ 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 to $\hat{g}_{INIT}$ and find a

```
/* 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 ( $elim \leqq 0$ ) return; /* terminal case */
    if ( $Nsup - elim \geqq minNsup$ ) return;
    $newf^L = edis(f^L, \{x_{elim}\})$;
    $newf^U = econ(f^U, \{x_{elim}\})$;
    if ( $newf^L \leqq newf^U$ ) { /* 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 $x_{elim}$ */
}
```

Fig. 6 Support minimization algorithm.

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[16] gave a BDD-based algorithm to find all valid supports. Although our method also uses BDD representations, it finds one of minimal supports. Figure 6 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 $newf^L \leqq newf^U$ is not satisfied, no compatible function can be found from the search state; also, if $Nsup - elim \geqq minNsup$, 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 now show our synthesis procedure using not only functional decomposition but also Boolean resubstitution. The procedure iterates the following steps until all 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 Sect. 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 was a combinational (multi-level or two-level) circuit. The circuit

Table 1 Experimental results (5-input 1-output LUTs).

circuit			without resub.			with resub.			resub. PO			[3]	[4]	[18]
name	in	out	size	dep	time	size	dep	time	size	dep	time		size	
5xp1	7	10	14	2	0.14	10	2	0.14	9	3	0.14	18	12	27
9sym	9	1	6	3	0.27	6	3	0.27	6	3	0.26	7	6	59
alu2	10	6	47	5	4.38	47	5	4.67	47	5	4.71	109	54	116
apex7	49	37	118	5	40.77	58	5	41.62	54	6	6.51	60	56	64
b12	15	9	16	3	0.22	16	3	0.26	16	3	0.25			
b9	41	21	50	4	2.22	39	4	2.28	33	5	1.28	39	35	
clip	9	5	16	3	0.46	11	3	0.49	13	8	0.50	28		
cordic	23	2	16	5	10.19	10	4	9.99	11	5	6.15			
count	35	16	52	4	0.39	31	6	1.16	31	6	1.28	31	32	31
f51m	8	8	11	3	0.19	9	3	0.21	7	4	0.14	17	12	
misex1	8	7	12	2	0.11	10	2	0.13	10	4	0.11	11	11	19
misex2	25	18	40	3	0.36	36	4	0.47	36	4	0.50	28	29	
misex3c	14	14	76	9	25.00	70	9	27.13	64	10	20.29			
rd73	7	3	8	2	0.13	6	3	0.14	6	3	0.12	6	7	
rd84	8	4	11	3	0.22	7	3	0.26	8	5	0.22	10	12	73
sao2	10	4	22	3	1.53	20	3	1.57	20	3	1.57	28	46	
t481	16	1	5	3	1.76	5	3	1.76	5	3	1.76			
vg2	25	8	45	4	28.42	20	4	28.96	16	5	0.87	20		21
z4ml	7	4	6	2	0.12	5	2	0.12	4	2	0.12	5	5	6
total			571	68	116.9	416	71	121.6	396	87	46.8			

description was transformed to BDD representations of primary outputs in terms of primary inputs, with an initial variable order obtained by heuristics [19]. The procedure then constructed a network of m-input LUTs.

Table 1 shows the experimental results for several of the MCNC [20] 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 Sun Ultra 1 Model 170E. 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 was 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.

Comparing the columns "without resub." and "with resub.," we can observe the following. Boolean resubstitution is very effective because it reduced 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, was not expensive. Comparing the columns "with resub." and "resub. PO," we can observe the following. Although resubstitution of a primary output generally reduced the number of LUTs and execution time, it tends to increase the depth of a circuit. To compare our results with other LUT network synthesizers, we pick up the results found in [3], [4], [18]. 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 resubstitution technique proposed in this paper is not time consuming, which will allow effective identification of common LUTs from large amount of candidates.

References

[1] R.L. Ashenhurst, "The decomposition of switching functions," Proc. of an International Symposium on the Theory of Switching, April 1957.

[2] J.P. Roth and R.M. Karp, "Minimization over Boolean graphs," IBM Journal, pp.227–238, April 1962.

[3] R. Murgai, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," Proc. ICCAD, pp.564–567, Nov. 1991.

[4] S. Chang and M. Marek-Sadowska, "Technology mapping via transformations of function graphs," Proc. ICCD, pp.159–162, Oct. 1992.

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

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

[7] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli,

"Optimum functional decomposition using encoding," Proc. DAC, pp.408–414, June 1994.

[8] Y.-T. Lai, K.-R.R. Pan, and M. Pedram, "FPGA synthesis using function decomposition," Proc. ICCD, pp.30–35, Oct. 1994.

[9] C. Scholl and P. Molitor, "Communication based FPGA synthesis for multi-output Boolean functions," Proc. ASP-DAC, pp.279–287, Jan. 1995.

[10] B. Wurth, K. Eckl, and K. Antreich, "Functional multiple-output decomposition: Theory and an implicit algorithm," Proc. DAC, pp.54–59, June 1995.

[11] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, pp.667–691, Aug. 1986.

[12] 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.

[13] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean resubstitution with permissible functions and binary decision diagrams," Proc. DAC, pp.284–289, June 1990.

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

[15] 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," Proc. ICCAD, pp.560–563, Nov. 1991.

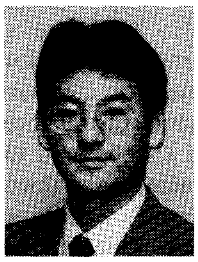
[16] B. Lin, "Efficient symbolic support manipulation," Proc. ICCD, pp.513–516, Oct. 1993.

[17] H. Savoj, R.K. Brayton, and H.J. Touati, "Extracting local don't cares for network optimization," Proc. ICCAD, pp.514–517, Nov. 1991.

[18] R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," Proc. DAC, pp.227–232, June 1991.

[19] M. Fujita, Y. Matsunaga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level logic synthesis," Proc. EDAC, pp.50–54, Feb. 1991.

[20] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," MCNC, Jan. 1991.



Hiroshi Sawada was born in Osaka, Japan, on October 31, 1968. He received the B.E. and M.E. degrees in information science from Kyoto University, Kyoto, Japan, in 1991 and 1993, respectively. In 1993, he joined NTT Communication Science Laboratories, where he has been engaged in research of computer aided design of digital systems and computer architecture. He is a member of the Information Processing Society of Japan.

Takayuki Suyama is a researcher in the NTT Communication Science Laboratories. Since 1992, he has been with the NTT Electrical Communications Laboratories, where he has worked on FPGA mapping and its applications. Mr. Suyama received the B.E. and M.E. degrees in mechanical engineering from Osaka University, in 1990 and 1992, respectively. He is a member of the Information Processing Society of Japan.

Akira Nagoya received his B.E. and M.E. degrees in electronic engineering from Kyoto University in 1978 and 1980, respectively. He joined the NTT Electrical Communication Laboratories in 1980 where he is now a Research Group Leader at the Communication Science Laboratories. From 1980 to 1987, he was engaged in research and development of mainframe CPU architecture. Since 1989, he has been engaged in research of CAD system for ASIC design. From 1990 to 1991, he was with the Department of Computer Science, University of Illinois at Urbana-Champaign, as a visiting scholar. His areas of research interest are logic synthesis, high-level synthesis, and computer architecture design. He is a member of the Information Processing Society of Japan. Mr. Nagoya received the Okochi Memorial Technology Prize in 1992.