A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications

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

Abstract

This paper presents a new method to express functional permissibilities for look-up table (LUT) based field programmable gate arrays (FPGAs). The method represents functional permissibilities by using sets of pairs of functions, not by incompletely specified functions. It makes good use of the properties of LUTs such that their internal logics can be freely changed. The permissibilities expressed by the proposed method have the desired property that at many points of a network they can be simultaneously treated. Applications of the proposed method are also presented; a method to optimize networks and a method to remove connections that are obstacles at the routing step. Preliminary experimental results are given to show the effectiveness of our proposed method.

1 Introduction

Because of their low cost, re-programmability and rapid turnaround times, field programmable gate arrays (FPGAs) have emerged as an attractive means to implement low volume applications and prototypes[1]. FPGAs also offer the possibility to design digital systems that can be easily reconfigured. There are many types of commercially available FPGAs[1]. One particularly popular type, look-up table (LUT) based FPGAs, consist of an array of programmable logic blocks which contain LUTs and a programmable routing network to connect the LUTs. Each LUT can realize any boolean function with m (typically 4 or 5) inputs.

The traditional design flow for LUT based FPGAs is as follows[1]. In the first step, a logic optimizer performs technology independent optimization[2, 3]. Next, a technology mapper[4, 5, 6, 7, 8, 9, 10, 11] maps networks to LUTs. Finally, placement and routing are done. Networks are optimized in the first step using the number of literals for the cost function expected to be realized with gate arrays or standard cells. In the second step, most of the technology mappers start from multi-level networks whose nodes are represented by sum-of-product forms obtained in the first step. Thus LUT networks obtained by the technology mappers usually have some functional redundancies. On the other hand, some technology mappers[7, 9, 11] directly generate LUT networks from primary output functions in terms of primary inputs represented by an ordered binary decision diagram (or simply BDD)[12]. These methods are not effected by intermediate sum-of-product forms and usually generate better LUT networks than the former technology mappers. However, they can not treat large networks because of limited BDD power. Large networks, therefore, must be divided and recombined to apply these methods. In such cases, there are some functional redundancies around the boundaries between divided networks. Either way, there are some functional redundancies in LUT networks after technology mapping. Such redundancies can be expressed by incompletely specified functions such as **satisfiability don't cares (SDCs)**, **observability don't cares (ODCs)**[13] or **Compatible Sets of Permissible Functions (CSPFs)**[3]. CSPFs have been used to express redundancies and optimize LUT networks[14]. Although expressing redundancies by incompletely specified functions is very efficient for logic optimization[15], it does not utilize such flexibility as freely changing the internal logic of an LUT.

In this paper, we propose a new method to express functional permissibilities for LUT based FPGAs. The method utilizes not incompletely specified functions, but sets of pairs of functions. The sets are called "**Sets of Pairs of Functions to be Distinguished (SPFDs)**". SPFDs represent functional permissibilities utilizing properties of LUTs such that their internal logics can be changed, and are suitable for expressing functional permissibilities in LUT networks. SPFDs can be calculated as efficiently as CSPFs. Furthermore, the permissibilities expressed by SPFDs have the desired property of being able to be simultaneously treated at many points of a network as CSPFs (or compatible observability don't care sets). As an application of SPFDs, a method to change connections in LUT networks is proposed. This method can be applied to optimize LUT networks and to remove unroutable connections in the routing step.

This paper is organized as follows. In Section 2, we explain basic terminology and give an example of expressing don't care sets. In Section 3, the notion of SPFDs is introduced. The way to calculate SPFDs and their comparison with CSPFs are also described in Section 3. In Section 4, we discuss applications of SPFDs. Section 5 gives preliminary experimental results and our observations. Finally, Section 6 concludes this paper and mentions future work.

2 Preliminaries

2.1 Terminology

In this section, we provide the terminology for the rest of this paper. We treat loop-free multi-level combinational

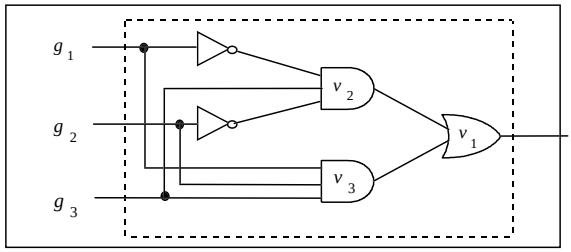

Figure 1: A part of a network

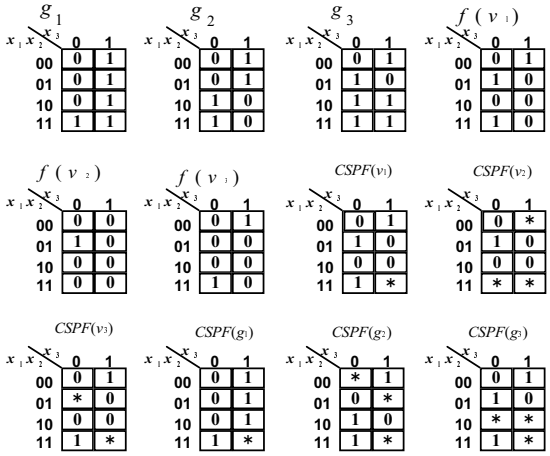

Figure 2: Calculation of CSPFs

networks that consist of LUTs and connections between them, and call such networks as LUT networks. The maximum number of inputs of an LUT is fixed (typically 4 or 5). Let $f(L_i)$ (or $f(c_j)$) be the logic function in terms of primary inputs realized at LUT L_i (or connection c_j). The **set of permissible functions**[3] of an LUT (or a connection) is the set of functions; we can change the function realized at the LUT (or the connection) to a member of the set of the functions without changing the functionalities of the primary outputs of the network. **CSPF**[3] is one of the **sets of permissible functions** with the property that we can change functions of many LUTs to their CSPFs at the same time. The CSPF of an LUT (or a connection) is represented with an incompletely specified function whose values are 1, 0 or $*$ (means **don't care**).

2.2 Expressing Functional Permissibilities by CSPFs

Here, how to express functional permissibilities by CSPFs is explained by the following example.

Figure 1 shows a part of a network. Let v_1 be an output gate of the network and g_1, g_2 and g_3 be intermediate functions that are expressed by the primary inputs x_1, x_2 and x_3. g_1, g_2 and g_3 are shown in Fig. 2. Here, the functions realized at gates v_1, v_2 and v_3 are $f(v_1)$, $f(v_2)$ and $f(v_3)$ in Fig. 2, respectively. If the bit of $f(v_1)$ is don't care when (x_1, x_2, x_3) is $(1, 1, 1)$ (external don't care), the CSPF of v_1 is expressed as "$CSPF(v_1)$" in Fig. 2. If a bit of $f(v_1)$ is 1, the corresponding bit of either $f(v_2)$ or $f(v_3)$ must be 1, but the other's is don't care since v_1 is an OR gate. For example, when (x_1, x_2, x_3) is $(0, 0, 1)$, the bit of $f(v_3)$ is 1; therefore, the bit of $f(v_2)$ is don't care. In this way, the CSPFs of input functions of v_1 are calculated as "$CSPF(v_2)$" and "$CSPF(v_3)$" in Fig. 2. For other kinds of gates, the CSPFs of the gates' inputs are calculated by the same notion. The functional permissibilities expressed by CSPFs are calculated from primary outputs in the direction of primary inputs of a network. If a gate has more than two fanouts, the CSPF of the gate is calculated by the intersection of the fanouts' CSPFs. In this example, the CSPFs of the connections concerning g_1, g_2 and g_3 are calculated as "$CSPF(g_1)$", "$CSPF(g_2)$" and "$CSPF(g_3)$" in Fig. 2, respectively. "$CSPF(g_1)$", "$CSPF(g_2)$" and "$CSPF(g_3)$" will be compared with our new expression in Section 3.

3 Functional Permissibility Expression for LUT based FPGAs

3.1 Sets of Pairs of Functions to be Distinguished

The CSPF of L_i is represented by an incompletely specified function. This means that the alternative functions for $f(L_i)$ must be 1 or 0 when the CSPF is 1 or 0, respectively. Such expression of the conditions of alternative functions is very useful in networks where the internal logic of each node is fixed. In an LUT network, however, the internal logic of each node can be changed; therefore, another method can be used for expressing the conditions of alternative functions. To explain such a method, some definitions are introduced here.

Definition 1 A function f is said to **distinguish** a pair of functions g_1 and g_2 if either one of the following two conditions is satisfied.

condition 1 ($f = 1$ when $g_1 = 1$) and ($f = 0$ when $g_2 = 1$).

condition 2 ($f = 0$ when $g_1 = 1$) and ($f = 1$ when $g_2 = 1$).

Note that $(g_1 \cdot g_2)$ must be the function that is constantly 0.

For example, function f (in Fig. 3) distinguishes a pair of functions f_{1a} and f_{1b} (in Fig. 3) because $f = 1$ when $f_{1a} = 1$, and $f = 0$ when $f_{1b} = 1$. f also distinguishes a pair of functions f_{2a} and f_{2b}.

Definition 2 A set of pairs of functions $\{(f_{1a}, f_{1b}), (f_{2a}, f_{2b}), \cdots, (f_{na}, f_{nb})\}$ represents the conditions of functions such that the functions must distinguish f_{ia} and f_{ib} for each pair (f_{ia}, f_{ib}) in the set.

For example, function f (in Fig. 3) satisfies the conditions represented by $\{(f_{1a}, f_{1b}), (f_{2a}, f_{2b})\}$ because f distinguishes f_{1a} and f_{1b}, and f_{2a} and f_{2b}.

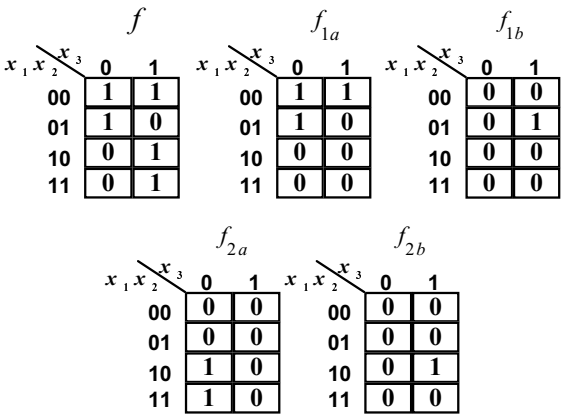

Figure 3: Functions to be distinguished

In LUT networks, we propose using a set of pairs of functions to represent the conditions of alternative functions instead of incompletely specified functions and introduce the following definition.

Definition 3 A set of pairs of functions that represents the conditions of the alternative functions for $f(L_i)$ (or $f(c_j)$) is called the "**Set of Pairs of Functions to be Distinguished (SPFD)**" of L_i (or c_j). If a function g satisfies the conditions represented by an SPFD, g is said to "**satisfy**" the SPFD.

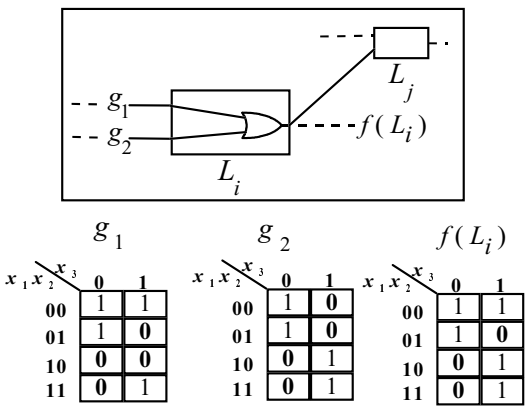

Figure 4: An LUT in a network

An intuitive explanation of SPFDs is given as follows. In Fig. 4, LUT L_i has two inputs whose functions are g_1 and g_2, which are shown in Fig. 4. The internal logic of L_i is OR; therefore, $f(L_i)$ is expressed as shown in Fig. 4. If $f(L_i)$ is used as an input of another LUT L_j, the role of $f(L_i)$ is to distinguish the two functions f_a and f_b, which are expressed as shown in Fig. 5. f_a and f_b are the ON-set and the OFF-set of $f(L_i)$, respectively. Therefore, the SPFD of L_i is $\{(f_a, f_b)\}$. Only $f(L_i)$ and $\overline{f(L_i)}$ satisfy the SPFD of L_i. For example, $\overline{f(L_i)}$ distinguishes f_a and f_b because $\overline{f(L_i)}$ is 1 when f_b is 1 and $\overline{f(L_i)}$ is 0 when f_a is 1. If $f(L_i)$ is changed to $\overline{f(L_i)}$, we only modify the internal logic of L_j to negate the input function from L_i.

g_1 (in Fig. 4) can distinguish g_{1a} and g_{1b} (in Fig. 5) which are the ON-set and the OFF-set of g_1, respectively. If the internal logic of L_i can be freely changed, $f(L_i)$ can be changed to distinguish functions that can be distinguished by using both g_1 and g_2 (how to change $f(L_i)$ is shown in Section 4.1). Thus, if the SPFD of L_i and function g_1 are not changed, g_2 must distinguish g_{2a} and g_{2b}; therefore, the SPFD of the input connection concerning g_2 is $\{(g_{2a}, g_{2b})\}$. This is because the conditions expressed by $\{(f_a, f_b)\}$ are the same as those expressed by $\{(g_{1a}, g_{1b}), (g_{2a}, g_{2b})\}$. This is an intuitive way to calculate the SPFD of g_2. The formal way to calculate SPFDs is shown in Section 3.2.

Function g_2' in Fig. 5 distinguishes g_{2a} and g_{2b}; therefore, g_2 can be replaced with g_2'. If g_2 is replaced with g_2', the internal logic of L_i must be modified to $(g_1 + \overline{g_2'})$. How to modify the internal logic of an LUT is shown in Section 4.1.

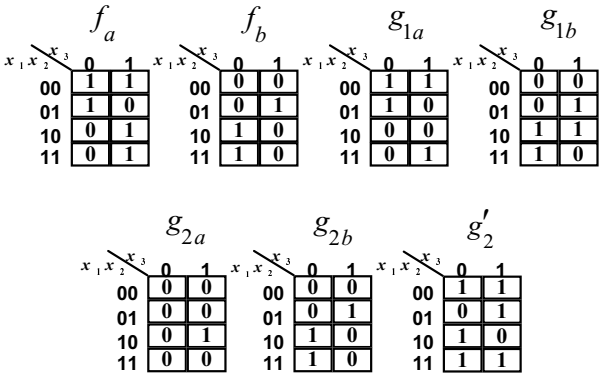

Figure 5: Functions in SPFDs

In this example, the permissibility of g_2 expressed by the SPFD is the same as that expressed by the CSPF. In more complicated cases, however, the permissibilities expressed by SPFDs are different from those expressed by CSPFs, and this is shown in Section 3.2.

3.2 Calculating SPFDs

3.2.1 Calculating the SPFDs of the Input Connections of an LUT

The procedure to calculate the SPFDs of the input connections of LUT L is formally stated as follows. L has n input connections $(c_1, \cdots, c_n)$, and the functions realized

at $(c_1, \cdots, c_n)$ are $(g_1, \cdots, g_n)$, respectively. Let the SPFD of L be $\{(f_1, f_0)\}$, which has already been calculated.

Below, a function f is said to be **included** in a function g if $f \cdot \overline{g}$ is the function that is constantly 0. Initially, let the SPFDs of $(c_1, \cdots, c_n)$ be empty.

step 1 Calculate 2^n logical products for all possible combinations of $(g_1, \cdots, g_n)$, where each g_i is negated or not (e.g., $(\overline{g_1} \cdot \overline{g_2} \cdot, \cdots, \cdot \overline{g_n})$, $(g_1 \cdot g_2 \cdot, \cdots, \cdot g_n)$, and so on). Let these products be $(b_{0\cdots0}, \cdots, b_{1\cdots1})$, where the index of b_k is an n bits binary number that satisfies the following condition.

- The i-th bit (from the left) of k is 0 or 1, depending on whether g_i is negated or not in b_k.

For example, when $n = 3$, $b_{011} = \overline{g_1} \cdot g_2 \cdot g_3$.

step 2 For all b_i, calculate $a_i = b_i \cdot (f_1 + f_0)$.

step 3 From $(a_{0\cdots0}, \cdots, a_{1\cdots1})$, select all of the functions that are included in f_1 and not constantly 0. Let the set of these functions be $F1$. From $(a_{0\cdots0}, \cdots, a_{1\cdots1})$, select all of the functions that are included in f_0 and not constantly 0. Let the set of these functions be $F0$.

step 4 Calculate the cartesian product $F = F1 \times F0$.

step 5 Select an element in F as (a_i, a_j) one by one, and go to step 6. If there is no element to select, halt.

step 6 If the different bits of i and j are the $k_1, k_2, \cdots, k_s$-th bits (from the left), select an arbitrary k_t from them, add (a_i, a_j) to the SPFD of c_{k_t}, and go to step 5.

If the SPFD of L has more than two elements, the above procedure is applied for all elements in the SPFD of L, and the SPFD of c_i is calculated by the union of all the calculated SPFDs for c_i. Although the procedure consumes $O(2^{2n})$ time for the worst case where n is the number of inputs of an LUT (even if the calculation time of logic functions is thought to be constant), in practice it does not consume so much time because n is typically a small number (4 or 5). Therefore, this procedure is very suitable for LUT networks whose nodes have a small number of fanins.

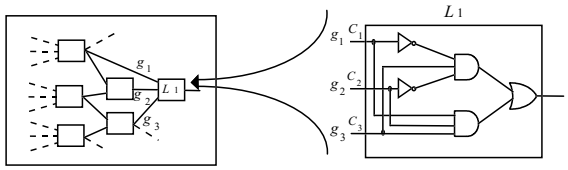

Figure 6: An LUT network

The procedure is explained using the following example. Figure 6 shows an LUT network. LUT L_1 has three input connections c_1, c_2 and c_3, and the functions realized at c_1, c_2 and c_3 are g_1, g_2 and g_3, respectively. g_1, g_2 and g_3 are shown in Fig. 2. The internal logic of L_1 is expressed as $(g_1 \cdot g_2 \cdot g_3 + \overline{g_1} \cdot \overline{g_2} \cdot g_3)$ (shown in Fig. 6). The SPFD of L_1 has already been calculated as $\{(f_1, f_0)\}$, where f_1 and f_0 are the ON-set and the OFF-set of "$CSPF(v_1)$" shown in Fig. 2. f_1 and f_0 are shown in Fig. 7. The internal logic of L_1 has the same functionalities as the network shown in Fig. 1. Therefore, the conditions of this example are the same as those of the example in Section 2.2. Here, the SPFDs of c_1, c_2 and c_3 are calculated as follows.

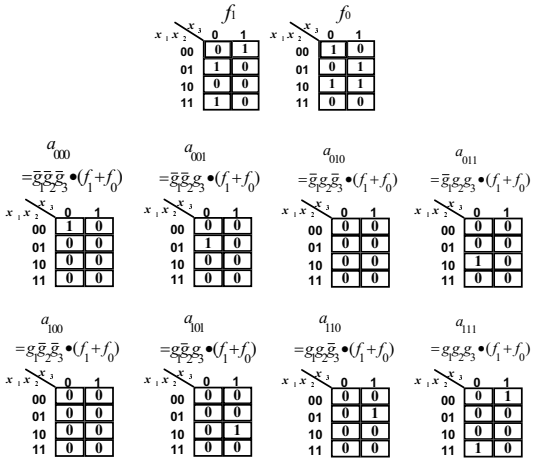

Figure 7: Calculating SPFDs (1)

step 1 Calculate 2^3 logical products for all possible combinations of g_1, g_2 and g_3, where each g_i is negated or not (e.g., $b_{000} = \overline{g_1} \cdot \overline{g_2} \cdot \overline{g_3}$, $b_{111} = g_1 \cdot g_2 \cdot g_3$, and so on).

step 2 For $(b_{000}, \cdots, b_{111})$, calculate $a_i = b_i \cdot (f_1 + f_0)$. $(a_{000}, \cdots, a_{111})$ are shown in Fig. 7. This step removes don't care bits of the SPFD of L_1 from b_i.

step 3 Since a_{001} and a_{111} are included in f_1, let $F1$ be $\{a_{001}, a_{111}\}$. Since a_{000}, a_{011}, a_{101} and a_{110} are included in f_0, let $F0$ be $\{a_{000}, a_{011}, a_{101}, a_{110}\}$. There is no need to consider a_{010} and a_{100} because they are functions that are constantly 0.

step 4 The set F is calculated from the cartesian product $F1 \times F0$. $F1$, $F0$ and F are shown in Fig. 8. F is a set of pairs of functions that must be distinguished by any one of the input functions of L_1. At step 5 and step 6, the pairs in F are divided into the SPFDs of the input connections of L_1.

step 5 For the first element in F (i.e., (a_{001}, a_{000})), go to step 6.

step 6 The third bits of (001) and (000) are different from each other, which means that g_3 can distinguish a_{001} and a_{000}. This is because the difference between

$(\overline{g_1} \cdot \overline{g_2} \cdot g_3)$ and $(\overline{g_1} \cdot \overline{g_2} \cdot \overline{g_3})$ is whether g_3 is negated or not. Therefore, (a_{001}, a_{000}) is added to the SPFD of c_3.

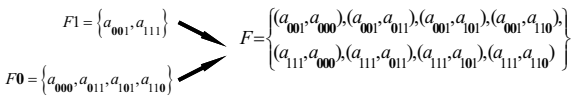

Figure 8: Calculating SPFDs (2)

For the remaining elements in F, step 5 and step 6 are done in the same way. For (a_{111}, a_{000}), all bits of (111) and (000) are different, which means that all of g_1, g_2 and g_3 can distinguish a_{111} and a_{000}. Therefore, (a_{111}, a_{000}) can be added to any one of the SPFDs of c_1, c_2 or c_3 (this selection corresponds to the selection of k_i from $k_1, k_2, \cdots, k_s$ at step 6 in the above procedure). If c_1 is selected in such cases, the SPFDs of c_1, c_2 and c_3 are calculated as "SPFD(c_1)", "SPFD(c_2)" and "SPFD(c_3)" in Fig. 9. For example, the conditions of alternative functions for g_1 are represented by the SPFD of c_1 as follows.

- The bits of g_1 corresponding to the bits of "A" and "$\overline{A}$" in "Conditions by $SPFD(c_1)$" (in Fig. 9) must be different from each other.
- The bits of g_1 corresponding to the bits of "B" and "$\overline{B}$" in "Conditions by $SPFD(c_1)$" (in Fig. 9) must be different from each other.

Thus, the bits of either "A" or "$\overline{A}$" must be 1, but the bits of the other must be 0. There are two assignments of 1 and 0 to "A" and "$\overline{A}$". In the same way, there are two assignments of 1 and 0 to "B" and "$\overline{B}$". There are two assignments of 1 and 0 to "$*$" in "Conditions by $SPFD(c_1)$" in Fig. 9. Therefore, the number of functions that satisfy the SPFD of c_1 is 8. The CSPF of c_1 is shown as "$CSPF(c_1)$" in Fig. 9 (It is the same as "$CSPF(g_1)$" in Fig. 2). The number of functions that satisfy "$CSPF(c_1)$" is 2 because the CSPF has one don't care bit. In this example, we can find more alternative functions for g_1 using SPFDs than using CSPFs. Comparisons of the SPFDs and the CSPFs of c_1, c_2 and c_3 are shown in Fig. 9. In this example, the internal logic of L_1 is not redundant. If the internal logic of an LUT is redundant, the difference between SPFDs and CSPFs becomes larger.

3.2.2 Calculating the SPFD of an LUT

The SPFD of LUT L_i is obtained by the union of all the SPFDs of the output connections of L_i. For example, if L_i has two fanouts c_1 and c_2, and the SPFDs of c_1 and c_2 are $\{(f_{11}, f_{10})\}$ and $\{(f_{21}, f_{20})\}$ respectively, the SPFD of L_i is calculated as $\{(f_{11}, f_{10}), (f_{21}, f_{20})\}$ unless (f_{11}, f_{10}) and (f_{21}, f_{20}) are the same. There are some cases where the number of elements in the SPFD of L_i obtained by such a calculation becomes too large. In such cases, the SPFD

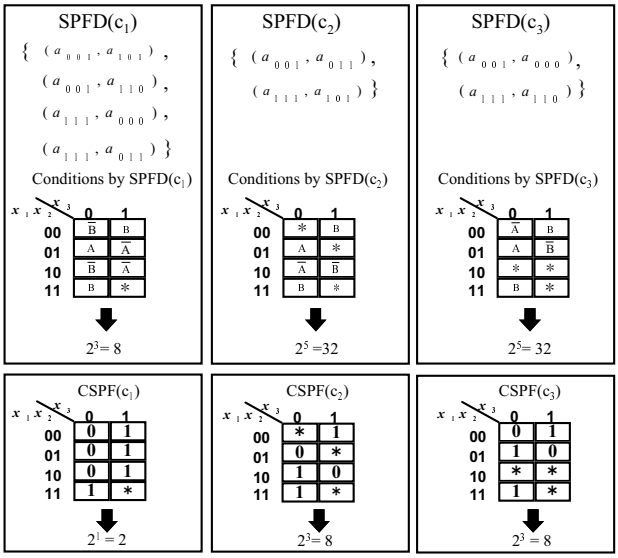

Figure 9: Comparisons of SPFDs with CSPFs

of L_i can be filtered to have less elements. For example, the SPFD of L_i in the above example can be filtered to be $\{(f_{11} + f_{21}, f_{10} + f_{20})\}$ in the following cases (we can also do the same type of filtering in other cases).

- when f_{11} and f_{21} are both included in $f(L_i)$, and f_{10} and f_{20} are both included in $\overline{f(L_i)}$.
- when f_{11} and f_{21} are both included in $\overline{f(L_i)}$, and f_{10} and f_{20} are both included in $f(L_i)$.

Note that f_{11}, f_{10}, f_{21} and f_{20} are always included in either $f(L_i)$ or $\overline{f(L_i)}$. By this filtering, the number of elements in SPFDs becomes smaller, which contributes toward reducing the calculation time.

4 Applications of SPFDs

In this section, we discuss about applications of SPFDs. SPFDs at many points of a network can be simultaneously treated as CSPFs. Therefore, SPFDs can be used in the same way as CSPFs.

4.1 Changing Connections Using SPFDs

In LUT networks, we can exchange connection c_i with the output of L_j, if $f(L_j)$ satisfies the SPFD of c_i.

In the example mentioned in Section 3.2.1, the conditions represented by the SPFD of c_3 are expressed as "Conditions by $SPFD(c_3)$" in Fig. 9. g_3' shown in Fig. 10 satisfies the conditions because the bits of g_3' corresponding to the bits of "A" and "$\overline{A}$" in "Conditions by $SPFD(c_3)$" are 0 and 1, and the bits of g_3' corresponding to the bits of "B" and "$\overline{B}$" are 1 and 0. Therefore, if LUT L_j realizes function g_3', c_3 can be replaced with the output of L_j. If CSPFs are used, c_3 can not be replaced

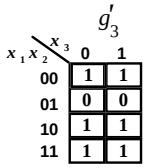

Figure 10: An alterative function for g_3

because g_3' does not satisfy "$CSPF(g_3)$" in Fig. 2. When c_3 is replaced with the output of L_j, $f(L_1)$ is changed and does not satisfy the SPFD of L_1 anymore. Therefore, the internal logic of L_1 must be modified as follows so that $f(L_1)$ still satisfies the SPFD of L_1. Observing $F1$ and $F0$ obtained at step 3 in the example of calculating the SPFDs of input connections of L_1 in Section 3.2.1 ($F1 = \{a_{001}, a_{111}\}$, $F0 = \{a_{000}, a_{011}, a_{101}, a_{110}\}$), we can see that $f(L_1)$ must satisfy the following two conditions.

condition 1 $f(L_1) = 1$ when $a_{001} = 1$, and $f(L_1) = 0$ when ($a_{000} = 1$, $a_{011} = 1$, $a_{101} = 1$ and $a_{110} = 1$).

condition 2 $f(L_1) = 1$ when $a_{111} = 1$, and $f(L_1) = 0$ when ($a_{000} = 1$, $a_{011} = 1$, $a_{101} = 1$ and $a_{110} = 1$).

The internal logic of L_1 must be modified so that $f(L_1)$ satisfies the above two conditions. $\overline{g_2} = 1$ when $a_{001} = 1$, and $\overline{g_2} = 0$ when ($a_{011} = 1$ and $a_{110} = 1$). $\overline{g_3'} = 1$ when $a_{001} = 1$, and $\overline{g_3'} = 0$ when ($a_{000} = 1$ and $a_{101} = 1$). Therefore, the function $(\overline{g_2} \cdot \overline{g_3'})$ satisfies condition 1. In the same way, the function $(g_1 \cdot g_2 \cdot g_3')$ satisfies condition 2. Therefore, the function $(g_1 \cdot g_2 \cdot g_3' + \overline{g_2} \cdot \overline{g_3'})$ satisfies the above two conditions, which is the modified internal logic of L_1.

The procedure to modify the internal logic of LUT L is formally stated as follows. L has n input connections whose functions are $(g_1, \cdots, g_n)$. Note that $(g_1, \cdots, g_n)$ may be changed by replacing connections. Let $F1$ and $F0$ be the sets obtained at step 3 of the procedure to calculate the SPFDs of input connections of L (mentioned in Section 3.2.1). The modified internal logic is calculated as a sum-of-product form, and the i-th product is l_i. Initially, let all l_i be the function that is constantly 1.

step 1 Select the i-th element in $F1$ as a_i, and go to step 2. If there is no element to select, halt.

step 2 Select an element in $F0$ as a_j one by one, and go to step 3. If there is no element to select, go to step 1.

step 3 From $(g_1, \cdots, g_n)$, select the function that distinguishes a_i and a_j as g_k, and go to step 4.

step 4 If g_k is 1 when a_i is 1, modify l_i to $(l_i \cdot g_k)$. If g_k is 0 when a_i is 1, modify l_i to $(l_i \cdot \overline{g_k})$. go to step 2.

Finally, the modified logic is obtained as $(l_1 + l_2, \cdots, + l_m)$, where m is the number of elements in $F1$.

The output functions of the LUTs that are the transitive fanouts of L_1 may also change. The internal logics of such LUTs must be changed in the same way.

4.2 Optimization Using SPFDs

The following procedure optimizes an LUT network using SPFDs.

step 1 Calculate the SPFDs of all LUTs and connections in the network.

step 2 Select a connection as c_i one by one, and go to step 3. If there is no connection to select, halt.

step 3 If the SPFD of c_i is empty, remove c_i and go to step 5. Otherwise, go to step 4.

step 4 If c_i can be replaced with the output of L_j, replace it and go to step 5. Otherwise, go to step 2.

step 5 If the logic functions of some LUTs are changed owing to removing or replacing c_i with the output of L_j, change the internal logics of such LUTs properly (by the method mentioned in Section 4.1). Go to step 2.

The order to select c_i in step 2 and the order to select L_j in step 4 are based on heuristics. For example, if the optimized networks should have lower levels, the LUT whose number of levels is the smallest is selected as L_j in step 4. If the optimized networks should have less LUTs, the output of the LUT that has one fanout is selected first as c_i in step 2. This is because if the output of the LUT that has one fanout is replaced with another LUT's output, the LUT can be immediately removed.

4.3 Removal of Unroutable Connections Using SPFDs

Because of limited routing resources in FPGAs, automatic routing may fail in a congested area, even though routing resources are available in a non-congested area. In such a case, the design is usually modified manually and the routing is tried again. We propose removing unroutable connections or replacing them with other connections by using our method mentioned in Section 4.1, and routing again automatically. This approach raises the possibility of successful automatic routing.

5 Experimental Results

We have implemented the methods presented here and performed preliminary experiments on MCNC[16] benchmark circuits. BDD was used for representing functions, and the maximum number of usable BDD nodes was limited to 1,000,000. Therefore, some large circuits, e.g., C3540, C7552, C2670, etc., could not be treated. In the experiments, a 5-input LUT architecture was assumed. The SIS (**A System for Sequential Circuit Synthesis** of UC Berkeley) technology mapper commands were used to generate initial networks, i.e., eliminate 2, gkx -ac, simplify -d, xl_part_coll -m -g 2, xl_coll_ck, xl_partition -m, simplify, xl_imp, xl_partition -t, xl_cover -e 30 -u 200, xl_coll_ck -k. These commands are recommended by the SIS package document.

Two preliminary experiments were done on the initial networks to check the effectiveness of the proposed applications of SPFDs. One was to optimize networks and the other was to count the number of connections that could be removed or replaced with other connections. In

Table 1: Results of the optimization methods

Circuits	Initial			Area				Level			
	LUT	conn	lev	LUT	conn	lev	CPU	LUT	conn	lev	CPU
C1908	103	429	13	98	389	13	48.54	98	393	11	32.95
C432	66	275	17	63	257	16	12.25	63	257	16	10.03
alu2	109	482	19	97	378	17	1.75	97	378	17	1.45
alu4	208	862	24	192	723	26	21.23	198	745	22	5.5
apex6	194	894	10	181	803	10	5.23	181	803	10	4.72
apex7	73	292	6	67	247	11	1.19	68	255	6	0.78
cordic	17	76	8	12	52	8	0.17	12	52	8	0.17
dalu	331	1393	16	286	1115	10	15.79	287	1125	9	10.97
des	1118	4663	11	1104	4407	22	3585.08	1111	4508	11	681.79
example2	105	451	5	100	396	8	4.14	101	415	5	2.04
frg2	339	1307	8	278	1019	9	22.08	278	1039	8	15.21
i9	138	679	5	137	675	5	6.88	137	675	5	4.75
k2	536	2325	9	528	2144	13	156.04	533	2267	9	19.66
lal	36	142	4	30	102	8	0.46	31	121	3	0.17
rot	192	753	14	187	707	13	502.3	187	707	13	409.9
t481	404	1738	21	379	1505	21	13.75	379	1505	21	11.75
term1	69	303	7	45	186	6	0.68	45	186	6	0.68
too_large	188	882	12	179	805	12	36.65	179	805	12	36.65
ttt2	53	237	4	46	189	5	0.28	47	196	4	0.36
vda	246	1043	8	239	941	25	280.19	246	992	8	3.42
x1	111	455	6	96	383	7	2.4	99	393	6	3.19
x2	13	56	3	12	48	3	0.04	12	48	3	0.04
x3	205	938	6	189	825	6	7.47	189	830	5	3.43
x4	140	598	4	110	441	4	1.2	110	441	4	1.2
total	4994	21273	240	4655	18737	278	4723.79	4688	19136	222	1260.8
ratio	1.00	1.00	1.0	0.93	0.88	1.1		0.94	0.90	0.9	

this section, “LUT”, “conn” and “lev” mean the number of LUTs, the number of connections and the number of network levels.

5.1 Results of the Optimization Methods

We did an experiment to check the effectiveness of the optimization method proposed in Section 4.2. Table 1 shows the results of this experiment. In Table 1, “CPU” shows the CPU run-time (sec.) on a SPARC station 20. For the procedure mentioned in Section 4.2, two kinds of heuristics were tried. In one, the output of the LUT having one fanout was selected first as c_i in step 2; the objective was to reduce “LUT”. In the other, the LUT whose number of levels was the smallest was selected as L_j in step 4; the objective was to reduce “lev”. The results are shown in the columns “Area” and “Level” in the table, respectively. The row “total” shows the total numbers of “LUT”, “conn”, “lev” and “CPU”. The row “ratio” shows the ratios of both “Area” and “Level” to “Initial”. Comparing the columns “Area” and “Level”, we can observe the following. The method “Area” increases “lev” in some cases, while the method “Level” does not increase “lev”. In addition, the method “Level” consumes less CPU time than the method “Area”.

In the implemented method, the optimization method was applied only once and all of the SPFDs of LUTs were filtered to have one element (because of simplicity in implementing the program). Therefore, we expect that better results can be obtained if the method is applied many times and no filter is used.

5.2 Possibility of Removing a Connection

We did another preliminary experiment to check the effectiveness of the method proposed in Section 4.3. In the experiment, the number of connections that could be removed or replaced with other connections (called “changeable connections”) was counted. The column “Num.” in Table 2 shows the number of changeable connections in a network, and the column “Ratio” in Table 2 shows the ratio(%) of changeable connections to all connections in the network. The mean value of the ratios was 73.8%. The column “CPU” in Table 2 shows the CPU run-time (sec.) on a SPARC station 20 to check all connections in the network. From this experiment, we could observe that most of the connections could be removed or replaced with other connections by our method. We plan to integrate our method into routing tools and check the effectiveness of our method in the routing step.

6 Conclusion and Future Work

We have presented a new method to express functional permissibilities for LUT based FPGAs. The method utilizes “sets of pairs of functions” that are called **SPFDs**. The SPFD of an LUT (or a connection) is a set of pairs

Table 2: The number of changeable connections

Circuits	Initial			Num.	Ratio	CPU
	LUT	conn	lev			
C1908	103	429	13	357	83.2	32.7
C432	66	275	17	245	89.1	8.79
alu2	109	482	19	464	96.2	1.18
alu4	208	862	24	831	96.4	3.46
apex6	194	894	10	447	50.0	4.08
apex7	73	292	6	159	54.4	0.15
cordic	17	76	8	69	90.7	0.05
dalu	331	1393	16	1335	95.8	7.8
des	1118	4663	11	3780	81.1	422.32
example2	105	451	5	171	37.9	1.31
frg2	339	1307	8	883	67.5	11.89
i9	138	679	5	364	53.6	3.43
k2	536	2325	9	2184	93.9	13.17
lal	36	142	4	77	54.2	0.06
rot	192	753	14	380	50.4	334.48
t481	404	1738	21	1735	99.8	10.56
term1	69	303	7	268	88.4	0.17
too_large	188	882	12	868	98.4	32.01
ttt2	53	237	4	148	62.4	0.1
vda	246	1043	8	917	87.9	2.35
x1	111	455	6	345	75.8	2.48
x2	13	56	3	30	53.5	0.02
x3	205	938	6	497	52.9	1.97
x4	140	598	4	342	57.1	0.4

of functions that must be **distinguished** by the function realized at the LUT (or the connection). SPFDs make good use of properties of LUTs such that their internal logics can be changed. We have also proposed applications of SPFDs and presented preliminary experimental results to show the effectiveness of SPFDs. SPFDs used in large networks could not be calculated because of limited BDD power. Therefore, we plan to treat larger networks by methods such as network division. We also plan to integrate the method of removing connections into routing tools and check the effectiveness of SPFDs in the routing step.

References

[1] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *FIELD-PROGRAMMABLE GATE ARRAYS*. Kluwer Academic Publishers, 1992.

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

[3] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The Transduction Method-Design of Logic Networks Based on Permissible Functions," *IEEE Trans. Computers*, vol. 38, pp. 1404–1424, Oct. 1989.

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

[5] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," in *28th ACM/IEEE Design Automation Conference*, pp. 227–233, June 1991.

[6] K. Karplus, "Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays," in *28th ACM/IEEE Design Automation Conference*, pp. 240–243, June 1991.

[7] M. Tsai, T. Hwang, and Y. Lin, "Technology Mapping for Field Programmable Gate Arrays Using Binary Decision Diagram," in *Proc. of the Synthesis and Simulation Meeting and International Interchange*, pp. 84–92, 1992.

[8] S. Chang and M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graphs," in *International Conference on Computer Design*, pp. 159–162, Oct. 1992.

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

[10] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1–11, Jan. 1994.

[11] H. Sawada, T. Suyama, and A. Nagoya, "Logic Synthesis for Look-up Table Based FPGAs Using Functional Decomposition and Support Minimization," in *International Conference on CAD*, pp. 353–358, Nov. 1995.

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

[13] K. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, and C. R. Wang, "Multi-level Logic Minimization Using Implict Don't Cares," in *International Conference on CAD*, pp. 723–740, June 1988.

[14] S. Yamashita, Y. Kambayashi, and S. Muroga, "Optimization Methods for Lookup-Table-Based FPGAs Using Transduction Method," in *ASP-DAC '95*, pp. 353–356, Aug. 1995.

[15] H. Savoj and R. K. Brayton, "The Use of Observability and External Don't Cares for Simplification of Multi-Level Networks," in *27th ACM/IEEE Design Automation Conference*, pp. 297–301, June 1990.

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