SPFD-Based Effective One-to-Many Rewiring (OMR) for Delay Reduction of LUT-based FPGA Circuits

Katsunori Tanaka
Grad. School of Informatics,
Kyoto University,
606-8501, Kyoto, Japan
ktanaka@db.soc.i.kyoto-u.ac.jp

Shigeru Yamashita
Grad. School of Information Science, NAIST
Ikoma, Nara, 630-0101, Japan
ger@is.nara-aist.ne.jp

Yahiko Kambayashi
Grad. School of Informatics,
Kyoto University,
606-8501, Kyoto, Japan
yahiko@db.soc.i.kyoto-u.ac.jp

ABSTRACT

This paper proposes an innovative method for SPFD-based rewiring in Look-Up-Table-based (LUT-based) FPGA circuits. The new method adds new input wires to two or more LUT's in order to remove or to replace a target wire. There have been a few rewiring methods for FPGA circuits so far, such as the original SPFD-based optimization sometimes called Local Rewiring (LR), SPFD-based Global Rewiring (GR) and SPFD-based Enhanced Rewiring (ER). However, all of them replace one wire with other new input wire to one LUT but not with those to two or more LUT's. Moreover, the LR removes or replaces input wires with new one to the same LUT only, and the GR and ER topologically limit the LUT's where new input wires are added. Our new method, called One-to-Many Rewiring (OMR), loosens such topological constraints for more flexible FPGA circuit transformation so that it is easier to import constraints on physical design to the logic optimization. The experimental results show our OMR can transform FPGA circuits more flexibly than the LR, GR and ER, by introducing the new manipulation, wire addition. The OMR can rewire 1.2 times as many wires as the existing methods, especially, the ER. The computation time is as short as the existing methods.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Optimization*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate Arrays*

General Terms

Algorithms, Design

Keywords

SPFD, Logic Optimization, One-to-Many Rewiring (OMR), Global Rewiring (GR), Collaboration of Logic and Physical Design

1. INTRODUCTION

FPGA circuits have been used mainly for prototyping, but now are often employed for practical implementations. While their performance is still lower than full/semi-custom-designed circuits', the technology is adopted partially or fully in circuit design and production due to its advantageous feature, "reconfigurable". This is expected to be the key feature of new computer models in the future.

FPGA is classified into two types, Multiplexer-based (MUX-based) and Look-Up-Table-based (LUT-based).

The former type of circuits consist of MUX's and are designed isomorphically to the binary decision diagrams (BDD's) for the functions to realize.

The latter type of circuits consist of logic blocks, called LUT's, and interconnections among them. Each LUT in an FPGA circuit can realize any function with respect to a specified number of variables. When the number is K, then the FPGA circuit is said to be K-feasible and the function stored inside of the LUT is called the internal function. In such a K-feasible circuit, each LUT can work similarly to a K-input gate. The function realized by the LUT is analogous to the logic operations realized by the gate. Therefore, logic optimization methods for gate networks have been applied or extended for LUT-based FPGA.

One of the most powerful logic optimization methods is the Transduction Method [1], which is based on the concept of permissible functions (PFs). This method utilizes the concept of permissible functions (PFs) in order to transform logic circuits, which can be represented by an directed acyclic graph. A set of PFs (SPF) is expressed with an incompletely specified function (ISF). The method was originally invented for NOR logic circuits and has been extended for more types of logic gate circuits [2] and LUT-based FPGA logic optimization [7]. The biggest advantage of the Transduction Method is to be easier to import constraints on physical design into logic design. Thus, while the physical design like placement and routing is getting more important for high-performance FPGA circuit design, transformation-based logic optimizations, such as the Transduction Method as well as ATPG (Automatic Test Pattern Generation)-based ones [5] [6] [8] [9] [10] [11] [12], are also getting more significant, since such optimization methods can transform logic circuits so flexibly that the circuits satisfy the specifications, such as the speed and area.

In the case of use for LUT-based FPGA, there is a great advantage of logic flexibility utilized for logic optimization: modification of the internal function. When SPF's repre-

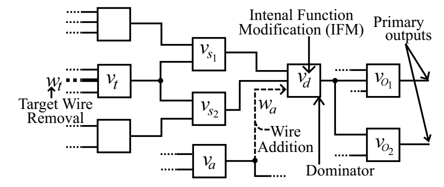

Figure 1: A Dominator considered in GR and ER

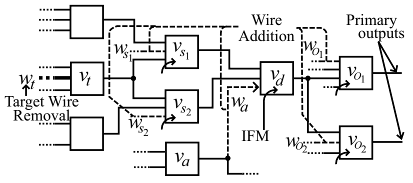

Figure 2: Relaxation of Topological Constraints by OMR

sented as ISF's are used in the logic optimization, the representation takes an implicit undesired constraint that the internal function is not changed when the procedure is applied to manipulation of input wires of LUT's.

Sets of pairs of functions to be distinguished (SPFD's) were derived for optimization of LUT-based FPGA logic networks [13] to take full advantage of the flexible change of the internal function, and are a more generalized representation of functional permissibility than ISF's. At each time in the logic optimization, the original SPFD-based transformation is applied to one LUT as removal of input wires and replacement of existing input wires with new ones to the LUT. That is why this transformation is called SPFD-based Local Rewiring (LR) also.

In order to take physical design into consideration during logic design phase, SPFD-based Global Rewiring (GR) was proposed in [14]. The GR assumes the following scenario:

1. Netlists are logically synthesized and optimized.
2. They are technology-mapped, placed and routed onto a given LUT-based FPGA architecture. Then, delay information of the designed circuit is obtained.
3. Based on the delay information, rewiring procedure is applied to the netlists.

In order to explain the idea of GR in contrast to LR, we think of an example shown in Fig. 1. Let the delay of a wire, w_t, be denoted by $D(w)$. Suppose that $D(w_t)$ is critically long. In this case, w is called the **target wire**. In the LR, w is just removed or replaced with a new input wire to v_t, if the SPFD-based condition is satisfied. On the other hand, the GR takes a trial-and-error-like strategy. Without any SPFD calculation, w is removed temporarily at the first step of the algorithm. As a consequence, the primary output functions may be undesirably changed. In order to restore them, the GR looks for a new wire added to v_d called the dominator, which is shown by w_a in Fig. 1, and tries to modify the internal function of v_d. Thus, the GR is considered to be **1-to-1 rewiring**. If $D(w_a)$ is far smaller than $D(w_t)$, then the total delay of the circuit is expected to be reduced.

The enhanced SPFD-based rewiring (ER) was proposed in [15] and inherits the transformation concept of the GR. In order words, the ER is also 1-to-1 rewiring. The advantage of the ER to the GR is that the function at v_d can be more flexibly changed, keeping the primary output functions unchanged.

Compared with the traditional logic optimization methods, the GR and ER have the following undesirable constraints, which are explained by using Fig. 1:

1. The wires substituted for w are added to one of the dominators, such as v_d, only.
2. w is removed only, but not replaced with a new input wire to v_t with addition of wires to other LUT's.

In order to loosen the constraints, we propose, in this paper, an innovative method, **SPFD-Based One-to-Many (1-to-m) Rewiring (OMR)**, where m may be equal to or larger than 2. The advantage of the OMR is illustrated in Fig. 2. This OMR performs rewiring by adding two or more wires for removal or replacement of the target wire. In the case of the example in Fig. 1, w can be removed with addition of two or more of new input wires to v_{s_1}, v_{s_2}, v_{O_1} and v_{O_2} as well as v_d, which are shown by w_{s_1}, w_{s_2}, w_{O_1} and w_{O_2} as well as w_a in Fig. 2, respectively. Moreover, w_t is not only removed but also may be replaced with a new input wire to v_t. In other words, the constraints 1. and 2. are loosened. That is advantages of our OMR over the LR, GR and ER. Although the wire count is increased, if $D(w_t)$ is smaller than any of the added wires, then the total delay is expected to be reduced more than the LR, GR and ER.

In fact, wire addition to an LUT appears not to be helpful, since the internal function of the LUT can ignore the new variable for the added wire, i.e., the variable can regarded as a dummy. However, there are several cases where wire addition affects transformation of the FPGA circuit later than the addition. In this paper, we present a sufficient condition to be satisfied for the helpful wire addition.

The OMR exceeds the conventional rewiring with respect to two points; one point is that any wire can be a candidate of rewiring independently of the topology and the other point is that even if the wire can be a candidate of GR or ER, the removal of the target wire can be compensated for by addition of two or more new wires but of not only one. The experimental results show the above advantages over the LR, GR and ER, and that 1.2 times as many wires as the ER can be removed or replaced by the OMR. With respect to the computation time, the OMR copes with the existing rewiring methods. This results show that compared with the conventional methods, the OMR is enough useful for the delay-reducing rewiring by utilizing the information imported from the results of placement and routing.

This paper is organized as follows: Section 2 introduces terminologies and notations to describe our OMR algorithm. Section 3 describes the basic calculation method of the conventional SPFD's. Section 4 proposes our OMR algorithm and conditions to perform the OMR efficiently with an example to illustrate the effects. Section 5 shows experimental results to claim our OMR algorithm's benefit over GR and ER. Section 6 concludes this paper.

2. TERMINOLOGIES AND NOTATIONS

This paper discusses an LUT-based (Look-Up-Table-based) FPGA (Field Programmable Gate Array) network. It is given n primary input variables, $x_1, x_2, \ldots, x_n$, but since the function at any point is with respect to the variables, they are always handled as a vector, $\boldsymbol{x}$. A network consists

Table 1: Functional Expression for an m-Pair SPFD

	$(h_{i,1}, h_{i,0})$					
$\boldsymbol{x}$	$i=1$	$i=2$	$\cdots$	$i=m$	$\bigvee$	g
000	$(0,1)$	$(0,0)$		$(0,0)$	1	$\overline{a}_1$
001	$(0,0)$	$(1,0)$		$(0,0)$	1	a_2
010	$(0,0)$	$(0,0)$		$(0,1)$	1	$\overline{a}_m$
011	$(0,0)$	$(0,0)$		$(0,0)$	0	$*$
100	$(0,0)$	$(0,1)$	$\cdots$	$(0,0)$	1	$\overline{a}_2$
101	$(0,0)$	$(0,0)$		$(0,0)$	0	$*$
110	$(1,0)$	$(0,0)$		$(0,0)$	1	a_1
111	$(0,0)$	$(0,0)$		$(1,0)$	1	a_m

"$\bigvee$" means $\bigvee_{i=1}^{m}(h_{i,1} \vee h_{i,0})$

of **look-up-table**s (**LUT**'s) which can realize any but only one function with respect to a limited number of variables. In this paper, the function is called the **internal function** of the LUT. Although the number of input wires is changed by logic optimization, the number of variables is supposed to be K initially. Each of the K input wires are given an ordinal number, 1, 2, …, K, and the corresponding variable of the internal function is denoted by y_1, y_2, …, y_K. The function at each input wire of an LUT, v, with respect to the primary input variables, x_1, x_2, …, x_n, is denoted by $f_{1/v}$, $f_{2/v}$, …, $f_{K/v}$. The internal function is denoted by $u(y_1, y_2, \ldots, y_k)$, and the argument, $(y_1, y_2, \ldots, y_K)$, is often omitted. The function at the output of the LUT is denoted by $f_{0/v}$. The K variables, y_1, y_2, …, y_K, is handled as a vector, denoted as $\boldsymbol{y}$.

In this paper, the logic AND, OR and NOT operations are denoted as $\wedge$, $\vee$ and an over-line, respectively. In the case of AND, $\cdot$ also may be used as the operator. (For example, the K-input OR internal function is represented by $u = y_1 \vee y_2 \vee \ldots \vee y_K$.) The constant function whose value is 0 or 1 for any primary input vector, $\boldsymbol{x}$, is denoted as $\mathbf{0}$ or $\mathbf{1}$, respectively.

3. BASIC CALCULATION OF SPFD'S

An SPFD is the concept to represent a set of completely specified functions (CSF's), corresponding to an incompletely specified function (ISF), which represents maximum/compatible set of permissible functions (MSPF/CSPF) in the Transduction Method.

An ISF is expressed with a pair of CSF's, for example, the ON-set and OFF-set functions. Let an ISF be f. The ON-set and OFF-set, f^{ON} and f^{OFF}, are defined as follows:

- $f^{ON}(\boldsymbol{x}) = 1$ only for every $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = 1$.
- $f^{OFF}(\boldsymbol{x}) = 1$ only for every $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = 0$.
- ($f^{ON}(\boldsymbol{x}) = f^{OFF}(\boldsymbol{x}) = 0$ for every $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = *$ (*don't-care*).)

Then, for every CSF, g, satisfying the ISF, f, the following relationship holds:

$$f^{ON} \leq g \leq \overline{f^{OFF}} \tag{1}$$

As an extension of ISF-form representation, an SPFD is expressed in the following form:

$$\{(h_{1,1}, h_{1,0}), (h_{2,1}, h_{2,0}), \ldots, (h_{m,1}, h_{m,0})\}. \tag{2}$$

where m is a non-negative integer, $h_{i,1} \cdot h_{i,0} = \mathbf{0}$, but $h_{i,1} \neq \mathbf{0}$ and $h_{i,0} \neq \mathbf{0}$ hold for every $i = 1, 2, \ldots, m$. For each i, the corresponding set consists of CSF's, g's, satisfying the following condition:

$$h_{i,1} \leq g \leq \overline{h}_{i,0} \quad \text{or} \quad h_{i,0} \leq g \leq \overline{h}_{i,1} \tag{3}$$

When this condition is satisfied, g is said to **distinguish** $h_{i,1}$ and $h_{i,0}$. Moreover, if g distinguishes all the m pairs, g is said to **satisfy** the SPFD. In the case of $m = 1$, the SPFD is very similar to an ISF since $h_{1,1}$ and $h_{1,0}$ are regarded as the on-set and off-set functions of the ISF, respectively. However, unlike in the ISF, they can be regarded conversely. In other words, it represents a pair of ISF's. If $(h_{i,1} \vee h_{i,0})(h_{j,1} \vee h_{j,0}) = \mathbf{0}$ holds for any (i, j) where $i \neq j$, $i = 1, \ldots, m$ and $j = 1, \ldots, m$, then the SPFD represents 2^m ISF's. The ISF's can be represented a functional expression in Table 1 where 0 or 1 can be assigned to a_i in g for each $i = 1, 2, \ldots, m$.

Like MSPF's and CSPF's in the Transduction Method, SPFD's are used to determine whether wires can be removed or replaced. In the case of SPFD's, this removal and replacement considers the modification of internal functions, which are analogous to gate types, such as AND or OR. Thus, SPFD's can be utilized particularly for optimization of LUT-based FPGA networks.

In this section, we introduce how to calculate and use SPFD's for the optimization.

3.1 SPFD Calculation at Input Wires

Let us consider how to calculate SPFD's at all K input wires of an LUT, v, which are denoted as $SPFD_{1/v}$, $SPFD_{2/v}$, …, $SPFD_{K/v}$. We suppose that SPFD at the output of an LUT is obtained beforehand so as to consist of only one pair, i.e., $SPFD_{0/v} = \{(h_{1/v}, h_{0/v})\}$, as will be mentioned in Section 3.4. Then, the SPFD's at the K input wires, $SPFD_{1/v}$, $SPFD_{2/v}$, …, $SPFD_{K/v}$, are computed as follows:

3.1.1 Step 1:

For every $(y_1 y_2 \ldots y_k) = (00\ldots0)$, …, $(11\ldots1)$, the following function is calculated:

$$h_{\boldsymbol{y}/v} = h_{y_1 y_2 \cdots y_k/v} = (h_{1/v} \vee h_{0/v}) \bigwedge_{i=1}^{K} \hat{f}_{i/v}. \tag{4}$$

where $\hat{f}_i$ is a CSF defined for each $i = 1, 2, \ldots, K$ as follows:

$$\hat{f}_{i/v} = \begin{cases} f_{i/v} & \text{if } y_i = 1 \\ \overline{f}_{i/v} & \text{if } y_i = 0. \end{cases} \tag{5}$$

As a result, 2^K functions are obtained in total.

3.1.2 Step 2:

Pairs of these CSF's are made up to be contained in the SPFD's by constructing the following two sets,

$$\begin{aligned} H_{ON/v} = \{h_{\boldsymbol{y_{ON}}/v} | \quad & h_{\boldsymbol{y_{ON}}/v} \leq h_{1/v}, \\ & h_{\boldsymbol{y_{ON}}/v} \neq \mathbf{0}\}. \\ H_{OFF/v} = \{h_{\boldsymbol{y_{OFF}}/v} | \quad & h_{\boldsymbol{y_{OFF}}/v} \leq h_{0/v}, \\ & h_{\boldsymbol{y_{OFF}}/v} \neq \mathbf{0}\}. \end{aligned} \tag{6}$$

and by calculating the direct product as follows:

$$P_v = H_{ON/v} \times H_{OFF/v}. \tag{7}$$

where P_v is to be the union of all the SPFD's at the input wires to be computed in Step 3.

Table 2: Example without Wire Addition

(a) Functions

$\boldsymbol{x}$	$f_{i/v_{s_1}}$			$f_{i/v_{s_2}}$			f_{i/v_t}			
	i									
	1	2	0	1	2	0	1	2	3	0
000	0	0	0	0	0	1	0	0	0	0
001	1	0	1	0	0	1	1	0	0	0
010	1	1	0	1	1	1	0	0	1	1
011	1	1	0	0	1	0	1	1	1	1
100	0	0	0	1	0	0	1	0	1	0
101	1	0	1	1	0	0	1	1	0	0
110	0	1	1	0	1	0	0	0	1	1
111	0	1	1	1	1	1	1	1	1	1

(b) Expression of Functions Satisfying SPFD's

$\boldsymbol{x}$	$g_{i/v_{s_1}}$			$g_{i/v_{s_2}}$			g_{i/v_t}			
	i									
	1	2	0	1	2	0	1	2	3	0
000	$\overline{a}_1$	$\overline{a}_2$	0	$\overline{a}_3$	$\overline{a}_4$	1	$\overline{a}_5$	$*$	$\overline{a}_7$	0
001	a_1	$\overline{b}_2$	1	$\overline{a}_3$	$\overline{a}_4$	1	b_5	$\overline{a}_6$	$*$	0
010	b_1	b_2	0	b_3	b_4	1	$\overline{b}_5$	$*$	a_7	1
011	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
100	$\overline{a}_1$	$\overline{a}_2$	0	$*$	$*$	$*$	b_5	$\overline{a}_6$	$*$	0
101	$*$	$*$	$*$	a_3	$\overline{b}_4$	0	b_5	$*$	$\overline{b}_7$	0
110	$\overline{b}_1$	a_2	1	$\overline{b}_3$	a_4	0	$\overline{b}_5$	$*$	a_7	1
111	$\overline{b}_1$	a_2	1	b_3	b_4	1	a_5	a_6	b_7	1

3.1.3 Step 3:

Each $(h_{\boldsymbol{y_{ON}}/v}, h_{\boldsymbol{y_{OFF}}/v}) \in P_v$ is assigned into $SPFD_j$ such that

$$\begin{array}{c} h_{\boldsymbol{y_{ON}}/v} \leq f_{j/v} \leq \overline{h}_{\boldsymbol{y_{OFF}}/v} \\ \text{or} \\ h_{\boldsymbol{y_{OFF}}/v} \leq f_{j/v} \leq \overline{h}_{\boldsymbol{y_{ON}}/v}. \end{array} \tag{8}$$

holds. There may exist two or more j's such that Eq.(8) holds. In this case, one can be chosen from them, based on the given priorities set up by heuristics. As a result of this assignment for all the pairs, $SPFD_{j/v}$ is obtained for every $j = 1, 2, \ldots, K$.

3.2 Transformations based on SPFD's at Input Wires

3.2.1 Removal of an input wire

The j-th input wire of an LUT, v, can be removed if the following condition is satisfied:

$$SPFD_{j/v} = \emptyset. \tag{9}$$

3.2.2 Replacement of an input wire

Suppose that $SPFD_j$ is obtained as $\{(h_{\boldsymbol{y_{ON_i}}}, h_{\boldsymbol{y_{OFF_i}}}) \mid i = 1, \ldots, m\}$ where $h_{\boldsymbol{y_{ON_i}}}$'s or $h_{\boldsymbol{y_{OFF_i}}}$'s may be identical for some of the i's. Then, the j-th input wire can be replaced with a new one from another LUT, v', whose output function is $f_{0/v'}$ with respect to $\boldsymbol{x}$ if the following condition is satisfied:

$$\begin{array}{rc} & h_{\boldsymbol{y_{ON_i}}/v} \leq f_{0/v'} \leq \overline{h}_{\boldsymbol{y_{OFF_i}}/v} \\ \text{or} & h_{\boldsymbol{y_{OFF_i}}/v} \leq f_{0/v'} \leq \overline{h}_{\boldsymbol{y_{ON_i}}/v} \\ & \text{for every } i = 1, \ldots, m. \end{array} \tag{10}$$

3.3 Internal Function Modification

The concept of SPFD assumes that the internal function can be changed within a specified number of variables. Thus, after input wires of an LUT, v, is removed or replaced, the internal function must be changed, i.e., **modified**, in order to keep all the primary output functions unchanged. In this section, we show how to modify the internal function.

The modified internal function is obtained in a disjunctive form (i.e., a sum of products). Assume that $f_{i/v}$ is replaced with $g_{i/v}$ for each $i = 1, 2, \ldots, K$ by the FPGA circuit transformation. Each term, $T_{j/v}$, in the form is obtained, corresponding to $h_{\boldsymbol{y_{ON_j}}/v} \in H_{ON/v}$ for each $j = 1, 2, \ldots, p$ as follows, when H_{ON} consists of p pairs containing $h_{\boldsymbol{y_{ON_1}}/v}$, $h_{\boldsymbol{y_{ON_2}}/v}$, $\ldots$, $h_{\boldsymbol{y_{ON_p}}/v}$:

1. The term contains a literal y_i if $SPFD_{i/v}$ contains a pair with $h_{\boldsymbol{y_{ON_j}}/v} \in H_{ON/v}$ such that $h_{\boldsymbol{y_{ON_j}}/v} \leq g_i$,
2. The term contains a literal $\overline{y}_i$ if $SPFD_{i/v}$ contains a pair with $h_{\boldsymbol{y_{ON_j}}/v} \in H_{ON/v}$ such that $h_{\boldsymbol{y_{ON_j}}/v} \leq \overline{g}_i$,
3. The term contains no literal if $SPFD_{i/v}$ contains no pair with $h_{\boldsymbol{y_{ON_j}}/v} \in H_{ON/v}$

After $T_{j/v}$'s are obtained for all $j = 1, 2, \ldots, p$, the sum is assigned as the modified internal function, $u = T_{1/v} \vee T_{2/v} \vee \ldots \vee T_{p/v}$.

3.4 SPFD Calculation at the Output

Let us consider how to compute SPFD at the output of an LUT, v, given SPFD's at all its t output wires, $SPFD_{0_1/v}$, $SPFD_{0_2/v}$, $\ldots$, $SPFD_{0_t/v}$. First, the union of all the SPFD's is calculated as follows:

$$\bigcup_{i=1}^{t} SPFD_{0_i/v} \tag{11}$$

However, the union is not used as SPFD at the output, $SPFD_{0/v}$, in the conventional calculation if the pairs in the union are not disjoint. If the function at the output of v were changed from $f_{0/v}$ to $g_{0/v}$, then $g_{0/v}$ would have to satisfy the union. In other words, all the pairs in the union would have to be distinguished by $g_{0/v}$. Although the internal function would have to be modified so that each pair is distinguished, v can realize only one internal function. If the union consists of p pairs, the LUT would have to realze at most p internal functions, but actually not. Hence, the union is transformed into one-pair SPFD.

Suppose that the union consists of p pairs, $(h_{1,1}, h_{1,0})$, $(h_{2,1}, h_{2,0})$, $\ldots$, $(h_{p,1}, h_{p,0})$ where $h_{i,1} \leq f_{0/v} \leq \overline{h}_{i,0}$ holds for every $i = 1, 2, \ldots, p$. SPFD at the output, $SPFD_{0/v}$, is calculated by using the following sum operations:

$$\begin{array}{rcl} SPFD_{0/v} & = & \left\{ \left(\bigvee_{i=1}^{t} h_{i,1}, \bigvee_{i=1}^{t} h_{i,0} \right) \right\} \\ & = & \{(h_{1/v}, h_{0/v})\}. \end{array} \tag{12}$$

It is important to notice that $(h_{1/v}, h_{0/v})$, given in Section 3.1, is obtained by this calculation.

4. SPFD-BASED ONE-TO-MANY REWIRING (OMR)

In this section, we show a condition for effective wire addition and an example of the application.

4.1 Condition for Effective Wire Addition

Let us consider the addition of an input wire from v_i to v_j. The addition is possibly effective if the following condition

Table 3: Example with Wire Addition

(a) Functions

$\boldsymbol{x}$	$f_{i/v_{s_1}}$				$f_{i/v_{s_2}}$				f_{i/v_t}			
	i											
	1	2	3	0	1	2	3	0	1	2	3	0
000	0	0	1	0	0	0	0	1	0	0	0	0
001	1	0	0	1	0	0	1	1	1	0	0	0
010	1	1	1	0	1	1	1	1	0	0	1	1
011	1	1	1	0	0	1	0	0	1	1	1	1
100	0	0	0	0	1	0	1	0	1	0	1	0
101	1	0	0	1	1	0	0	0	1	1	0	0
110	0	1	0	1	0	1	0	0	0	0	1	1
111	0	1	1	1	1	1	1	1	1	1	1	1

(b) Expression of Functions Satisfying SPFD's

$\boldsymbol{x}$	$g_{i/v_{s_1}}$				$g_{i/v_{s_2}}$				g_{i/v_t}			
	i											
	1	2	3	0	1	2	3	0	1	2	3	0
000	$\overline{a}_1$	$\overline{b}_2$	b_3	0	$\overline{b}_4$	$\overline{a}_5$	$*$	1	$*$	$*$	$\overline{a}_9$	0
001	a_1	$*$	$\overline{c}_3$	1	$\overline{b}_4$	$*$	a_6	1	$*$	$*$	$*$	$*$
010	b_1	$*$	c_3	0	a_4	$*$	b_6	1	$*$	$*$	$*$	$*$
011	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
100	$\overline{a}_1$	$\overline{a}_2$	$\overline{a}_3$	0	$*$	$*$	$*$	$*$	a_7	$*$	$*$	0
101	$*$	$*$	$*$	$*$	b_4	$*$	$\overline{b}_6$	0	$*$	$*$	$*$	$*$
110	$\overline{b}_1$	a_2	$\overline{b}_3$	1	$\overline{a}_4$	a_5	$\overline{a}_6$	0	$\overline{a}_7$	$*$	$\overline{a}_9$	1
111	$\overline{b}_1$	b_2	a_3	1	a_4	$*$	b_6	1	$\overline{a}_7$	$*$	a_9	1

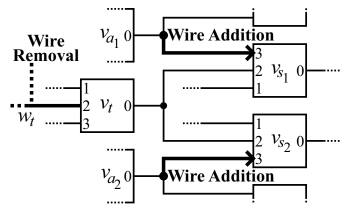

Figure 3: An Example Network

is satisfied:

$$\text{There is at least one } \boldsymbol{hy} \text{ such that } \boldsymbol{hy} \le f_{v_i} \text{ or } \boldsymbol{hy} \le \overline{f}_{v_i} \text{ holds.} \tag{13}$$

4.2 An Example

Let us consider an example network shown in Fig. 3 where v_{a_1} or v_{a_2} is not a successor of any LUT's, v_{s_1}, v_{s_2} and v_t. Assume that the delay of w_t, which is the second input wire of v_t, causes long delay and that we have been obtained information on physical design that the delay of w_{a_1,s_1} and w_{a_2,s_2} is likely to be shorter than the delay of w_t. The functions in the network is shown in the truth table in Table 2 (a), where i is the pin number of LUT's in Fig. 3. SPFD's at the outputs of v_{s_1} and v_{s_2} are obtained to represent the ISF's in the columns, $g_{0/v_{s_1}}$ and $g_{0/v_{s_2}}$, respectively, in Tables 2 (b) and 3 (b), where 0 or 1 can be arbitrarily assigned to a_i, b_i or c_i for each $i = 1, 2, \ldots, 9$ independently.

4.2.1 *Without Wire Addition*

When no wires are added to v_{s_1} or v_{s_2}, SPFD's are obtained to represent the functional expressions in Table 2 (b)., where 0 or 1 can be independently assigned to each a_i or b_i for $i = 1, 2, \ldots, 7$. In this calculation of SPFD's at the input wires of v_{s_1} and v_{s_2}, we assign the priority to assign into $SPFD_{2/v_{s_1}}$ and $SPFD_{2/v_{s_2}}$, respectively, as few pairs as possible. As shown in the column of g_{i/v_t}'s for all $i = 1, 2, 3$, no input wire of v_t can be removed, since the expressions mean that the none of SPFD's is empty. Thus, w is not disconnectable without wire addition.

4.2.2 *Effects by Wire Addition*

Table 3 illustrates an example of effective wire addition. Suppose that $f_{0/v_{a_1}}$ and $f_{0/v_{a_2}}$ are equal to $f_{3/v_{s_1}}$ and $f_{3/v_{s_2}}$ in Table 3 (a), respectively. In the case of these functions, the following relationships are satisfied:

$$\begin{aligned} &h_{10/v_{s_1}} \le \overline{f}_{0/v_{a_1}}, h_{11/v_{s_1}} \le f_{0/v_{a_1}}, \\ &h_{01/v_{s_2}} \le \overline{f}_{0/v_{a_2}}, h_{10/v_{s_2}} \le \overline{f}_{0/v_{a_2}}, h_{11/v_{s_2}} \le f_{0/v_{a_2}}. \end{aligned} \tag{14}$$

Hence, we determine that wire addition of w_{a_1,s_1} and w_{a_2,s_2} is likely to be effective.

As a result of the wire addition, the new wires become the third inputs of v_{s_1} and v_{s_2}, as shown in Fig. 3 and Table 3 (a). Based on these functions, we obtain SPFD's represented by expressions in Table 3 (b) where 0 or 1 can be assigned independently to a_i, b_i or c_i for $i = 1, 2, \ldots, 9$. The expression in the column g_{2/v_t} means that SPFD at w_t, which is the second input wire of v_t, is empty and w_t can be removed. Thus, the wire addition of w_{a_1,s_1} and w_{a_2,s_2} makes w_t redundant. Since the delay of these added wires is shorter then the delay of the removed wire, the total delay of this FPGA circuit is expected to be reduced, or even in the worst case, to be kept unchanged.

As shown in this example, the condition in Eq.(13) is useful to determine which new wires should be added for the target wire to be redundant.

5. EXPERIMENTAL RESULTS

As described in the previous sections, our rewiring method, OMR, allows to add two or more new wires substituted for the target wire, and appears to be more powerful and flexible than GR and ER.

In this section, we show the experimental results on MCNC benchmark circuits [3]. The initial circuits are composed of LUT's so that each of them has five or fewer input wires, by applying SIS commands developed at UCB [4]. This sequence of commands yields 5-LUT FPGA networks.

In order to compare our OMR with the LR, GR and ER, we only counted successful rewirings, since a circuit may not be placed and routed well even if its structure is compact at the logic design level. Especially, it is theoretically clear that the OMR and ER cover the LR and GR, respectively. Therefore, we compared the rewiring counts by the OMR and ER. The results are shown in Table 4, where "Init." means the number of wires in an initial benchmark circuit, "ER" and "OMR" mean the numbers of target wires successfully replaced with new ones and the computation time is obtained as seconds for each target wire. In Table 4, although the results for several benchmark circuits are not shown, we present the average results.

As shown in Table 4, the OMR can replace about 1.2 times as many target wires as the ER can. Since the topological constraints on the ER are loosened, it is easier for the OMR to replace target wires. It appears not to be fair

Table 4: Experimental Results: Successful Rewiring Counts

		# of Successful RW			Time/Wire	
Circuit	Init	ER	OMR	Adv.	ER	OMR
C1908	429	88	84	29	0.76	1.13
C432	275	127	141	62	2.15	2.89
alu2	482	310	348	79	0.05	0.15
alu4	885	562	566	135	0.16	0.37
apex6	894	306	224	59	0.07	0.58
apex7	292	93	101	36	0.03	0.15
b9	159	20	37	27	0.01	0.05
example2	451	69	138	91	0.00	0.09
f51m	106	50	33	10	0.01	0.02
i7	511	36	40	4	0.00	0.09
i9	679	22	253	242	4.13	1.39
term1	303	198	205	36	0.02	0.16
ttt2	246	92	122	39	0.00	0.02
x2	56	21	14	4	0.00	0.01
x3	937	187	197	80	0.04	0.96
x4	598	177	275	101	0.02	0.43
(%)	100	30.1	36.0	15.1	100.0	82.45

Adv.: Target Wires that are replaced by using OMR but not by ER.

enough to compare the number of successful rewiring, since the concepts of the OMR and ER are very different. The ER once changes the primary output functions and finally tries to restore them. On the other hand, the OMR always keeps the primary output functions unchanged. In fact, comparing the results on each target wire, 15% of all wires in the initial circuits are successfully rewired by using the OMR but cannot by using the ER. Since the ER and OMR rewire about 30% and 36% successfully, the percentage, 15%, is relatively large. Hence, we can conclude that the OMR is another option for rewiring after the physical design, such as placement and routing.

With respect to the computation time, the OMR takes about 82% as long as the ER does. However, benchmark circuits, 'C432' and 'i9', take very long time relatively to the other ones. Without these circuits, the computation time of the OMR is longer than that of the ER. Considering the procedures of the ER and OMR, both need SPFD calculation once. Since the OMR looks for effectively added wires, it is considered to need additional time proportional to the number of LUT's in a circuit. On the other hand, the ER is applied only to dominators, which are considered to be far fewer than the LUT's in the circuit. Considering the experimental results on the computation time, once the critically long target wire is determined, the computation time of the OMR is practically short. Thus, when the ER cannot modify the netlist well enough to satisfy the specifications, the OMR is useful as a logic design level rewiring method collaborating with the physical design.

6. CONCLUSION

This paper proposed an innovative method for SPFD-based rewiring, **the One-to-Many Rewiring (OMR)**. Topological constraints are not imposed on our OMR, although they are imposed on the existing methods, the LR, GR and ER. Thus, the OMR provides with a more flexible transformation than the existing methods. From our experimental results, the OMR provides with 1.2 times as flexible transformation as the existing rewiring methods presents. By importing the results on the physical design, such as placement and routing, this feature makes it easier to remove or replace wires causing long delay with shorter-delay wires. The OMR is more useful for collaboration of logic and physical design. We are implementing it to link with placement and routing tools.

7. REFERENCES

[1] S.Muroga, Y.Kambayashi, H.C.Lai, J.Niel, Culliney, "The Transduction Method - Design of Logic Networks Based on Permissible Functions," IEEE Transactions on Computers, Vol.38, No.10, pp.1404-1424, Dec. 1989.

[2] X. Xiang and S. Muroga, "Synthesis of Multilevel Networks with Simple Gates," Int'l Workshop on Logic Synthesis, May. 1989.

[3] S.Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," MCNC International Workshop on Logic Synthesis, 1991.

[4] E. Sentovich, et. al., "SIS: A System for Sequential Circuit Synthesis," Memorandum of No. UCB/ERL M92/41, Dept. of EECS, UC Berkeley, 1992.

[5] W. Kunz and P. R. Meron, "Multilevel Logic optimization by implication Analysis," Proc. of Int'l Conf. on Computer-Aided Design (ICCAD), pp. 6-13, Nov. 1994.

[6] L. A. Entrena and K. -T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal," IEEE Trans. on CAD of ICS, Vol. 14, No. 7, pp.909-916, Jul. 1995.

[7] S. Yamashita, Y. Kambayashi and S. Muroga, "Optimization Methods for Lookup-Table-Based FPGAs Using Transduction Method," Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC), pp. 353-356, Aug. 1995.

[8] S. -C. Chang, M. Marek-Sadowska and K.-T Cheng, "Perturb and Simplify: Multilevel Boolean Network Optimizer," IEEE Trans. on CAD of ICAS, Vol. 15, No. 12, pp. 1494-1504, Dec. 1996.

[9] Y. -M. Jiang, A. Kritic, K. -T. Cheng and M. Marek-Sadowska, "Post-layout rewiring for performance optimization," In Proc. of Design Automation Conf. (DAC), pp.662-665, 1997.

[10] S. -C. Chang, K. -T. Cheng, N. -S. Woo and M. Marek-Sadowska, "Postlayout rewiring using alternative wires," IEEE Trans. on CAD of ICS, Vol. 16, No. 6, pp.587-596, Jun. 1997.

[11] R. Huang, Y. Wang and K. -T. Cheng, "LIBRA-a library-independent framework for post-layout performance optimization," In Int'l Symposium on Physical Design, pp.135-140, 1998.

[12] S. -C. Chang, L. V. Ginneken and M. Marek-Sadowska, "Circuit Optimization by Rewiring," IEEE Trans. on Computers, Vol. 48, No. 9, pp.962-970, Sep. 1999.

[13] S. Yamashita, H. Sawada and A. Nagoya, "SPFD: A New Method to Express Functional Flexibility," IEEE Trans. on CAD of ICAS, Vol. 19, No. 8, pp. 840-849, Aug. 2000.

[14] J. Cong, Y. Lin and W. Long, "SPFD-based Global Rewiring," Proc. of Int'l Symposium on Field Programmable Gate Arrays (FPGA), pp.77-84, Feb. 2002.

[15] J. Cong, Y. Lin and W. Long, "A New Enhanced SPFD-based Rewiring Algorithm," Proc. of Int'l Conf. on Computer-Aided Design (ICCAD), pp.672-678, Nov. 2002.