

# Enhanced SPFD Rewiring on Improving Rewiring Ability

Jason Cong\*, Joey Y. Lin\* and Wangning Long<sup>+</sup>

\*Computer Science Department, UCLA

<sup>+</sup>Aplus Design Technologies, Inc.

{cong, yizhou}@cs.ucla.edu, longwn@applus-dt.com

## Abstract

This paper presents an in-depth study of the theory and algorithms for the *SPFD*-based (*Set of Pairs of Functions to be Distinguished*) rewiring, and explores the flexibility in the *SPFD* computation. Our contributions are in the following two areas: (1) We present a theorem and a related algorithm for more precise characterization of feasible *SPFD*-based rewiring. Extensive experimental results show that for LUT-based FPGAs, the rewiring ability of our new algorithm is 70% higher than *SPFD*-based local rewiring algorithms (*SPFD-LR*) [19][21] and 18% higher than the recently developed *SPFD*-based global rewiring algorithm (*SPFD-GR*)[20]. (2) In order to achieve more rewiring ability on certain selected wires used in various optimizations, we study the impact of using different atomic *SPFD* pair assignment methods during the *SPFD*-based rewiring. We develop several heuristic atomic *SPFD* pair assignment methods for area or delay minimization and show that they lead to 10% more *selected rewiring ability* than the random (or arbitrary) assignment methods. When combining (1) and (2) together, we can achieve 38.1% higher general rewiring ability.

## 1. Introduction

Rewiring is a technique that replaces a wire with another wire in a given Boolean network for area or/and delay reduction while maintaining functional equivalence. Recently, it has received increased attention due to the need for closer synthesis and layout interaction in timing closure. Using layout information, rewiring is efficient in post-layout logic synthesis. The rewiring problem has been widely studied. Existing approaches include the automatic test pattern generation (*ATPG*) based redundancy addition and removal [2] [5] [6] [7] [12] [14] [15], symmetry detection [4], and the *SPFD* (*Set of Pairs of Functions to be Distinguished*) [19] [17] [20] based algorithms.

*SPFD* is a recently proposed method to represent functional permissibility [19]. It can be understood as a method to express “don’t-care” conditions [3]. It provides a great flexibility in rewiring and other areas. The authors of [19] proposed an *SPFD*-based rewiring algorithm. The authors of [17] applied *SPFD* to the technology-independent logic synthesis. *SPFD* has also been applied to floorplanning and placement of multi-level PLAs [8] and low-power designs for FPGAs [13].

The rewiring algorithm proposed in [19] tries to replace the target wire with an alternative wire based on *SPFD* computation. Although it achieves a good synthesis result for FPGA area minimization, it has an inherent limitation

in that it can only perform local rewiring, that is, the destination node of the alternative wire has to be the same as that of the target wire. In this paper, this approach is referred as *SPFD-LR*. In [20], an *SPFD*-based global rewiring algorithm, *SPFD-GR*, is proposed with application to area minimization of FPGA designs. *SPFD-GR* successfully overcomes the limitation of *SPFD-LR* and is capable of finding a *global* alternative wire that might be *far away* from the target wire.

Rewiring ability defined in [20] is a useful measurement of rewiring algorithms. It is described as the number of wires having alternative wires. The higher the rewiring ability, the more rewiring choice an algorithm has. Previous work shows that the rewiring ability of *SPFD-GR* is already 44% better than that of *SPFD-LR*. However, our study shows that the *SPFD* computation in both *SPFD-LR* and *SPFD-GR* did not explore the full flexibility in *SPFD* computation, and both methods over-constrained the rewiring conditions.

In this paper, we focus on the in-depth study of the *SPFD*-based rewiring algorithm and explore the flexibilities in *SPFD* computation. Our results can be used to improve both *SPFD-LR* and *SPFD-GR*. Our main contributions are as follows:

1. We develop a new theorem and an efficient algorithm for more precise characterization of feasible *SPFD*-based rewiring. (Section 4)
2. We develop several heuristic algorithms for atomic *SPFD* pair assignment for improving the rewiring ability of certain selected wires which are critical for various optimization. (Section 5)
3. We integrate these methods into our enhanced *SPFD*-based rewiring algorithm (*SPFD-ER*). (Section 6)

## 2. Terminologies and Definitions

This section reviews some terminologies and definitions used in this paper. The circuits referred in this paper are combinational circuits. We assume that the input circuit is a mapped *K-LUT* network, meaning that each **logic cell** (or **node**) in the circuit has a single **out-pin**  $p_0$  and up to  $K$  **in-pins** ( $p_1, p_2, \dots, p_n, n \leq K$ ). Each node has an internal logic function  $p_0 = f(p_1, p_2, \dots, p_n)$  that defines the logic relationship between the out-pin and the in-pins of the node, and can be any  $K$ -input function.

For wire  $p_1 \rightarrow p_2$ , as shown in Figure 1,  $G_2$  is its **source node** and  $G_3$  is its **destination node**. **Transitive fanout nodes of pin**  $p_i$  are the nodes on the paths from  $p_i$  to a primary output (**PO**). **Transitive fanout nodes of a node** are the transitive fanout nodes of the node’s out-pin. A **dominator** of pin  $p_i$  is a transitive fanout of  $p_i$  through which all the paths from  $p_i$  to POs must pass. A **dominator of a**

wire is the dominator of the wire's destination pin. For example, in Figure 1,  $G_5$  is the only dominator of pin  $p_1$ , while both  $G_3$  and  $G_5$  are the dominators of pin  $p_2$  as well as wire  $p_1 \rightarrow p_2$ .

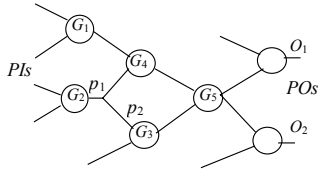


Figure 1. Illustration of dominator

We measure the timing criticality of a wire based on its time slack. When we do rewiring to improve timing performance, we usually try to replace a more critical wire with a less critical wire.

Given a **function pair**  $(\pi_1, \pi_0)$ ,  $\pi_1 \neq 0$ ,  $\pi_0 \neq 0$  and  $\pi_1 \pi_0 \equiv 0$ , function  $f$  is said to **distinguish**  $(\pi_1, \pi_0)$  if either one of the following two conditions is satisfied [17]:

$$\pi_1 \leq f \leq \bar{\pi}_0 \quad \text{or}$$

$$\pi_0 \leq f \leq \bar{\pi}_1$$

where  $\pi_1 \leq f \leq \bar{\pi}_0$  can be understood as  $f = 1$  when  $\pi_1 = 1$ , and  $f = 0$  when  $\pi_0 = 1$  [19]. The second condition can be interpreted in the same way.

An *SPFD* is a Set of Pairs of Functions to be Distinguished, e.g.,  $P = \{(\pi_{11}, \pi_{10}), (\pi_{21}, \pi_{20}), \dots, (\pi_{m1}, \pi_{m0})\}$ . Function  $f$  is said to satisfy an *SPFD* if  $f$  distinguishes all the function pairs in the *SPFD* set. An **atomic SPFD pair** is a function pair in which the two functions are the minimal product terms of the input functions of a node. An **atomic SPFD** is an *SPFD* containing only one or several atomic *SPFD* pair(s) [19][20].

### 3. Review of SPFD Calculation

First, we briefly review the method proposed in [19] for *SPFD* calculation. For an existing logic network, the calculation of the *SPFD*s usually consists of two steps:

- 1) Traverse the entire circuit from primary inputs (PIs) to primary outputs (POs) and calculate the global logic functions\* at all pins.
- 2) Calculate the *SPFD*s backward from POs to PIs.

At each pin, the *SPFD* calculation is carried out in accordance with the following 3 cases:

- a) At a PO,  $O_i$ , the *SPFD* has only one function pair,  $P = \{f_{i1}, f_{i0}\}$ , where  $f_{i1}$  is the on-set of the global function of  $O_i$ , and  $f_{i0}$  is the off-set function of  $O_i$ .
- b) At a node's out-pin, the *SPFD* is the union of its fanout pins' *SPFD*s.
- c) For the in-pins of a node, once its out-pin *SPFD* has been obtained, the in-pin *SPFD*s are obtained by decomposing its out-pin *SPFD* into an atomic *SPFD* and assigning the function pairs backwards to in-pins.

\* When the network is very large, it may be partitioned into several sub-networks for rewiring, as done in [20]. In this case, we shall compute the global function of each node in its sub-network.

### 4. A Precise Characterization of Feasible SPFD-based Rewiring

The first contribution of our work is to provide an exact formulation of rewiring and present a procedure which can find all possible alternative wires under *SPFD* constraints.

Recall case b) of the *SPFD* calculation in Section 3. For a node with multiple fanouts, its out-pin's *SPFD* is the union of its fanout pins' *SPFD*s. The union operation is performed as follows: Suppose its fanout pins' *SPFD*s are  $P = \{(\pi_{11}, \pi_{10}), (\pi_{21}, \pi_{20}), \dots, (\pi_{m1}, \pi_{m0})\}$ . First, we will find a function  $f$  that can distinguish all of them. For example, the original function of this out-pin is such a function. Without a loss of generality, we can assume  $\pi_{i1} \leq f \leq \bar{\pi}_{i0}$ ,  $i=1, 2, \dots, m$ . Then we will

union  $\pi_1 = \bigcup_{i=1}^m \pi_{i1}$  and  $\pi_0 = \bigcup_{i=1}^m \pi_{i0}$ , and set the *SPFD* of

the outpin to be  $(\pi_1, \pi_0)$ . In turn, the resulting union  $(\pi_1, \pi_0)$ , is used to calculate the *SPFD*s of its in-pins. This union operation is an important step for *SPFD* calculation. The *SPFD* method may not be able to create the node function without this step. Such operation is used in many previous rewiring algorithms, including those in [19] and [20]. In this paper, we will refer to  $P$  as the *pre-union SPFD* and  $\{(\pi_1, \pi_0)\}$  as the *post-union SPFD*.

The union operation here is obviously a sufficient condition for rewiring, however, our study shows that it is not a necessary condition for successful rewiring. In many cases, it will create unnecessary constraints between disjunctive *SPFD* pairs.

For example, suppose that  $(\pi_{11}, \pi_{10}), (\pi_{21}, \pi_{20}), \dots, (\pi_{m1}, \pi_{m0})$  are the *SPFD*s of the fanout branches of outpin of node  $G$  and they are disjunctive. If we union them with

$\pi_1 = \bigcup_{i=1}^m \pi_{i1}$  and  $\pi_0 = \bigcup_{i=1}^m \pi_{i0}$ , it will create such

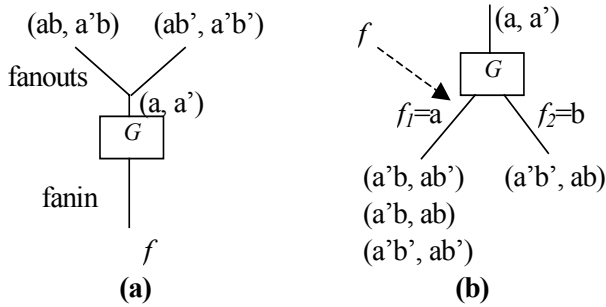
constraints like  $(\pi_{11}, \pi_{20}), (\pi_{m1}, \pi_{10})$  which are not among the original *SPFD*s. This will limit the rewiring ability of node  $G$ . We perform a small experiment on the benchmarks we used in Section 6, and find that about 25% of the fanouts in the circuits which have disjunctive *SPFD* pairs\*\*. The *SPFD* rewiring ability may significantly increase after we eliminate the unnecessary constraints introduced by the union operation. Here is a simple example to illustrate this problem.

**Example:** In Figure 2(a), node  $G$  has two fanouts whose *SPFD* requirement are  $(ab, a'b)$  and  $(ab', a'b')$ , and the *SPFD* constraint becomes  $(a, a')$  after the union. Now the wires which can replace  $f$  are only the wires with function  $a$  or  $a'$ , since no other function can distinguish  $(a, a')$ . However function  $a \oplus b$ , and function  $a \oplus \bar{b}$ , can also replace  $f$  and

\*\* It was stated in [17] that disjoint *SPFD* pairs occurs rarely because they map the *SPFD* to the local inputs, while we always map the *SPFD* to the primary inputs. Therefore our *SPFD* calculation is less redundant and may have more disjunctive pairs.

satisfy the *SPFD* constraints  $(ab, a'b)$  and  $(ab', a'b')$ . Therefore we can increase the likelihood of finding an alternative wire with the *SPFDs* prior to union.

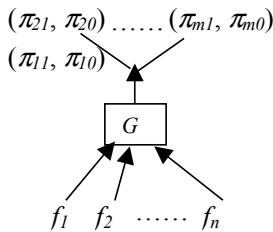
Moreover, we noticed that even without a union, the *SPFD* assigned to each input pin is not a necessary condition. In Figure 2(b), the *SPFD* requirement of  $(a, a')$  can be assigned to the two input pins  $f_1$  and  $f_2$  as shown. We will find that the *SPFD* assigned to input pin  $f_1$  can only be satisfied by function  $a$  or  $a'$ . This means that only  $f=a$  or  $f=a'$  can replace  $f_1$  in conventional *SPFD* methods. However  $f=a\oplus b$  and  $f=a\oplus \bar{b}$  can also satisfy the *SPFD* constraints  $(a, a')$  at the output pin and be able to replace  $f_1$ .



**Figure 2. (a) Unnecessary constraints by union  
(b) Not a necessary condition**

Based on these observations, we formulate the general form of the rewiring problem and provide a procedure that defines an exact characterization for the rewiring based on the fanouts' *SPFDs*. Our procedure avoids the union operation and, thus, does not have to create additional constraints. We formulate our **rewiring problem** as follows:

Given a node  $G$ , the *SPFD* of its out-pins as  $P=\{(\pi_{11}, \pi_{10}), (\pi_{21}, \pi_{20}), \dots, (\pi_{m1}, \pi_{m0})\}$ , the input functions of this node are  $f_1, f_2, \dots, f_n$  (Figure 3). We find a necessary and sufficient condition for  $f_1, f_2, \dots, f_n$  which satisfies that there exist a  $g=f(f_1, f_2, \dots, f_n)$  that can distinguish  $(\pi_{11}, \pi_{10}), (\pi_{21}, \pi_{20}), \dots, (\pi_{m1}, \pi_{m0})$ .



**Figure 3. Illustration of the rewiring problem**

We will build a graph to solve this problem. First let

$$\begin{aligned} \pi &= \pi_{11} + \pi_{10} + \pi_{21} + \pi_{20} + \dots + \pi_{m1} + \pi_{m0}, \text{ and} \\ \beta_{00\dots 0} &= f_1' f_2' \dots f_n' \pi \\ \beta_{00\dots 1} &= f_1 f_2' \dots f_n \pi \\ &\dots \\ \beta_{11\dots 1} &= f_1 f_2 \dots f_n \pi \end{aligned}$$

We can build a graph  $S$  which has  $2^n$  vertices, each vertex corresponding to one of these  $\beta$ 's. We construct the edge of the graph in the following way: If  $\exists i, \beta_a \cap \pi_{i1} \neq \emptyset$  and  $\beta_b \cap \pi_{i0} \neq \emptyset$ , we add an edge between vertices  $a$  and  $b$ . Now we have

**Theorem:**  $S$  is a bi-partite graph if and only if there exists  $g=f(f_1, f_2, \dots, f_n)$  which can distinguish  $(\pi_{11}, \pi_{10}), (\pi_{21}, \pi_{20}), \dots, (\pi_{m1}, \pi_{m0})$ .

Proof is omitted. Reader may refer to [22] for detail.

With this equivalent condition, the *SPFDs* of the input pins do not need to satisfy the redundant constraints brought by the union operation. Thus we can find more alternative wires and increase the rewiring ability.

The work in [17] also represents *SPFD* in a bi-partite graph which reflects the nature of the *SPFD* technique. The difference between our method and previous work is that we distinguish between the *SPFD* used for identifying rewiring (*pre-union SPFD*) and the *SPFD* used for distributing backward (*post-union SPFD*). Instead, [17] distributes the *pre-union SPFDs* to its fanins. In their method, when a pin has more than one disjunctive *SPFDs*, a non-bipartition situation may occur and new nodes or multi-valued nodes need be inserted in the circuit, causing trouble for post-layout rewiring. [19] and [20] use the *post-union SPFD* for rewiring, therefore it may not find all the possible alternative wires. Our method uses both *SPFDs* properly and achieves the maximum rewiring ability under current *SPFD* assignment.

Moreover, the rewiring algorithm in [17] only allows one wire to be changed at a time. While our problem formulation is a more general form of rewiring, since the input functions  $f_1, \dots, f_n$  could be any possible function. It accommodates changing several input functions simultaneously, as required by *SPFD-GR*.

Given target wire  $w_r$  which is the input of  $G_1$ , for its dominator  $G_D$ , our rewiring algorithm is as follows:

- 1) Temporarily remove  $w_r$  from  $G_1$  and re-calculate the output function of  $G_1$ ;
- 2) Propagate  $G_1$ 's new output function through its transitive fanouts until reaching  $G_D$ ;
- 3) For the new input functions of  $G_D$ , build the graph  $S_D$  by the above theorem. If the graph  $S_D$  is a bi-partite graph, then we can remove  $w_r$  without adding other wires.
- 4) If 3) fails, try to add another wire  $w_a$  as an input of  $G_D$ . Then build the graph  $S_D$  and check whether it is a bi-partite graph. If it is, then we can use  $w_a$  to replace  $w_r$ . If  $w_a$  fails, we may try another wire as a candidate.
- 5) If both 3) and 4) fail, we will recover wire  $w_r$ .

Since we can determine if a graph is a bi-partite graph in linear time by DFS, we do not spend much time checking this equivalent condition. This algorithm will cost similar CPU time with *SPFD-GR*, while providing more rewiring ability.

This precise characterization can determine all the possible alternative wires which can satisfy the constraints at the fanout pins. Thus we can achieve the maximum rewiring ability under current *SPFD* constraints. This method can be

added to any current algorithms which use *SPFD* technique to do rewiring.

## 5. Atomic *SPFD* Pair Assignment Methods

With our problem formulation, we can improve the rewiring ability for all the wires in the circuits. This rewiring ability is referred to *general rewiring ability*. In fact, not all the wire replacements have the same criticality during circuit optimization. For example, when we optimize the performance, we only focus on replacing the wires in the critical path; when we optimize the routability, we want to replace the wires which cause congestion. In other words, we want to improve the rewiring ability for a subset of wires which are important for optimization. We refer to this as *selected rewiring ability*.

More selected rewiring ability can be achieved in the atomic *SPFD* pair assignment. Recall case c) of the *SPFD* calculation in Section 3. Given a node's out-pin *SPFD*, to calculate the *SPFDs* of the node's in-pins, we first decompose the out-pin's *SPFD* into an atomic *SPFD*, then assign the function pairs backwards to the in-pins. In this step, a function pair can be assigned to any in-pin if the *SPFD* can be satisfied. In the example of Figure 3, if the function pair  $P = (\alpha_{00}, \alpha_{11})$ ,  $P$  can be satisfied by both  $p_1$  and  $p_2$ . In other words, we can assign  $P$  to either  $p_1$  or  $p_2$ . As we know, the impact of *SPFD* assignment has not been fully studied.

Random (or arbitrary) assignment methods have been used. In this paper, we proposed two types of heuristics depending on the optimization objectives, one is the criticality-oriented heuristic for delay optimization, and the other is the fanout-oriented heuristic for area optimization.

The basic idea is that if a wire has fewer function pairs in its *SPFD*, it is easier to satisfy. As the result, the wire has a greater chance to be replaced because it has fewer constraints. Therefore if removing a wire will benefit certain objective function, we will assign fewer function pairs to this wire.

For different purposes, we should use different heuristics to guide the atomic *SPFD* pair assignment. In this paper, we focus on two different optimization objectives. One is the delay optimization, and the other is the area minimization.

For the delay optimization, our goal is to reduce the longest path delay in the circuits. We use the criticality-oriented heuristics to guide the atomic *SPFD* pair assignment. We first sort the in-pins according to their timing criticality in ascending order (i.e. the least critical in-pins are at the beginning of the list). This way, the most critical nodes are at the end of the list and tend to receive fewer function pairs.

For the area minimization, we use a fanout-related heuristic. For example, when a node has only one fanout  $p$ , if we can replace  $p$  by another wire, the node can be removed since it has no fanout. On the other hand, when a node has many fanouts, it is almost impossible to remove the node because it is very hard to replace all the fanouts. Based on this consideration, we try to assign fewer function pairs to the input wires with smaller fanout numbers so that they have higher opportunity to be removed.

In order to achieve more rewiring ability, we also combined these two heuristics with other heuristics. In [21], an edge distribution scheme is applied to ignore atomic pairs that have already been assigned. In this paper, we extend the idea and propose an *SPFD* size oriented heuristic. Consider an atomic function pair  $P=(\alpha, \beta)$  at the output of a node and two input wires  $p$  and  $q$  which can distinguish  $P$ . Suppose  $p$ 's *SPFD* so far is  $R$  and  $q$ 's *SPFD* so far is  $S$ . If  $R$  covers  $\alpha$  while  $S$  does not cover any of  $\alpha$  and  $\beta$ , we assign  $P$  to wire  $p$ . The reason is that after the assignment,  $R$ 's size will not change. On the other hand, if we assign  $P$  to  $q$ ,  $S$ 's size will increase by one.

The experimental results in Section 6 will show that the above optimized heuristic assignment methods lead to better results than the random assignment method in general, while the differences due to different assignment heuristics appear to be small.

## 6. Experimental Results

In our study, we use *rewiring ability* to compare different rewiring algorithms, which is defined as the *number of wires* having at least one alternative wire. The higher the rewiring ability, the more rewiring choices an algorithm has. It usually reflects the potential of a rewiring algorithm in performing optimization.

We combined the new equivalence condition and some speed up techniques into our enhanced *SPFD*-based rewiring algorithm (*SPFD-ER*). We applied *SPFD-ER* in *LUT*-based *FPGA* synthesis and integrated it into the *RASP* system [11]. We did the following experiments: 1) Comparison of *general rewiring ability* among *SPFD-ER*, *SPFD-LR* and *SPFD-GR*; and 2) *Selected rewiring ability* comparison between different atomic *SPFD* assignment methods. 3) A primitive flow is presented to show that a larger rewiring ability helps the performance optimization.

The circuits used in our experiments are the combinational networks with 4-*LUTs*, obtained through *script.rugged*, *Cutmap* [9], *Red Removal* and *Greedy Pack*. These routines are available in *SIS* [16] and *RASP* [11].

### 6.1 Comparison of General Rewiring Ability

Table 1 compares the *general rewiring ability* of our enhanced method *SPFD-ER* with *SPFD-GR* and *SPFD-LR*. We implemented the *SPFD-LR* algorithm according to [19] and *SPFD-GR* algorithm according to [20]. All three programs traverse the entire circuit once, selecting every wire as a target wire and trying to find alternative wires that satisfy the circuit depth restriction. For the purpose of collecting statistical data, we did not make real changes to the circuit, even when rewiring is possible. Column 2 lists the number of wires in each circuit. Column 3 shows the rewiring ability of *SPFD-LR*, Columns 4 and 6 show the rewiring ability of *SPFD-GR* and *SPFD-ER*. Columns 5 and 7 show the improvement of the rewiring ability of *SPFD-GR* and *SPFD-ER* over *SPFD-LR* respectively. The results show that *SPFD-ER* has 70% more target wires that have alternative wires when compared with *SPFD-LR* has, and 18% more when compared with *SPFD-GR*.

The runtime of our method is 12.5X that of *SPFD-LR* and 4X that of *SPFD-GR*. This is due to the equivalent condition

	Total #wires	SPFD -LR	SPFD -GR	Impr. over SPFD-LR	SPFD -ER	Impr. over SPFD-LR
C1908	423	74	99	33.8%	114	54.1%
C432	538	154	183	18.8%	208	35.1%
alu4	939	277	419	51.3%	522	88.4%
apex6	1025	270	345	27.8%	410	51.9%
dalu	1338	468	704	50.4%	739	57.9%
example2	433	85	136	60.0%	169	98.8%
term1	244	71	99	39.4%	114	60.6%
x1	557	164	222	35.4%	271	65.2%
x3	958	154	319	107.1%	366	137.7%
alu2	510	179	253	41.3%	306	70.9%
C5315	1772	500	607	21.4%	757	51.4%
Average				44.3%		70.2%

**Table 1. Comparison of general rewiring ability for 4-LUT FPGA designs under circuit depth restriction** test for the rewiring, and it is a trade-off between synthesis quality and CPU time. By using the partition method as [20], we may use small partitions and let the CPU time increase proportionally to the circuit size.

## 6.2 Comparison among Atomic SPFD Pair Assignment Methods

In this sub-section, we compare the selected rewiring ability among different atomic *SPFD* pair assignment methods. We will consider delay optimization and area minimization respectively. Since we want to study the effect of atomic *SPFD* assignment only, we apply these methods on *SPFD-GR*, not *SPFD-ER*.

Table 2 shows the selected rewiring ability among the wires in the 20%  $\epsilon$ -network. We list the result for Methods A ~ E. Given a function pair  $P = (a, b)$  at a node's out-pin, these methods work as follows: Method A randomly assigns  $P$  to an input edge whose function distinguishes it. Method B randomly sorts the input edges and then assigns  $P$  to the first edge that can distinguish it. Method C uses a delay oriented heuristic, which sorts the edges according to their criticality (with the least critical edge in the first) and assigns  $P$  to the first edge which can distinguish it; Method D is the combination of Method C and the *SPFD* size oriented heuristic (see Section 5); Method E is the combination of Method C and the edge assignment scheme of [21]. The criticality-oriented method scan achieve about 10% more selected rewiring ability.

The atomic *SPFD* assignment methods will also help the wires of the non-critical network. After the selected rewiring ability is improved near its limitation, it is hard to increase any more. At this point, the assignment methods will do more improve on general rewiring ability. Experimental results show that when we combine *SPFD-ER* and criticality-oriented heuristic together (*SPFD-ER* + Method C), the average general rewiring ability improvement over *SPFD-GR* + Method A is 38.1%. This experiment is under the circuit depth restriction as Section 6.1. We do not list the data here due to page limitation.

Circuits	Method A	Method B	Method C	Method D	Method E
C1908	31	34	35	35	35
C432	26	22	26	26	25
alu4	50	43	55	56	55
apex6	34	32	35	34	34
dalu	59	50	57	57	57
example2	11	14	13	13	13
term1	4	4	4	4	4
x1	14	17	18	18	18
x3	21	25	25	25	25
alu2	48	50	56	53	56
C5315	N/A*	49	72	75	74
Average ** Improvement over Method A		3.1%	10.5%	9.8%	9.8%

**Table 2. Comparison of selected rewiring ability for different atomic SPFD assignment methods**

\*It fails after 8 hours runtime limitation

\*\*The average value does not contain C5315, since one result is N/A

We also did experiments to show the delay optimization results of various function-pair assignment heuristics. We use Quartus (Version II 1.0) [1] for placement and routing, then report longest delay. We can achieve 0.5%, 0.9%, 4.6%, 3.4% and 5.1% delay reduction by Methods A~E respectively.

The methods used for area minimization are similar to those in delay optimization. The only difference is that we use the fanout-number oriented heuristics instead of the delay-oriented heuristics in C, D and E. Due to the page limitation we only list the final average improvement of area minimization as 12.7%, 12.9%, 13.0%, 13.6% and 13.2% for Methods A~E respectively.

From these results, we know that the criticality-oriented methods (Methods C~E) are much better than the random methods (Methods A~B) in delay reduction with Method E performing the best. However, the difference due to different delay optimization heuristics is quite small. Moreover, from area minimization results, we see that random assignments work almost as well as optimized heuristics. The fanout-oriented methods perform only slightly better than the random methods. From these experimental data we conclude that our criticality-oriented methods are effective for delay minimization, while function-pair assignments have little impact for area minimization.

## 6.3 A Primitive Flow for Post-layout Rewiring

We set up a primitive flow to illustrate that large rewiring ability may help us in circuit performance optimization. We apply our rewiring method for post-layout delay optimization for LUT-based designs. For each benchmark circuit, we use SIS to do the logic synthesis, RASP to do technology mapping and Quartus II 1.0 [1] to do placement and routing.

Our post-layout optimization flow works as follows. 1) After Quartus finishes layout design for a circuit, the rewiring engine reads the placement information from the Quartus' output file. 2) In order to do delay optimization, we build our own delay model. Our model is based on the locations of LUTs in Quartus placement. 3) The engine traverses the circuit for  $M$  passes to do rewiring for the wires on the  $\epsilon$ -critical paths, which will be explained in the following. 4) After rewiring, the engine passes the resulting circuit to Quartus with the original location information for each logic cell.

From Table 1, we know that *SPFD-ER* achieves 70% more rewiring ability than *SPFD-LR*. This means we have more opportunity to replace wires in the  $\epsilon$ -critical path, thus reducing the size of  $\epsilon$ -network. From our experiments, we can find the numbers of paths in the 25%  $\epsilon$ -network under our delay model. And these numbers show that *SPFD-ER* reduces the size of the  $\epsilon$ -network by 45%, and is much better than *SPFD-LR*. After the rewiring, Quartus reports the final layout delay. *SPFD-ER* reduces the delays by 5.8% on average, while *SPFD-LR* reduces them by only 2.6%.

This optimization flow is a primitive flow since we do not get the accurate timing model and routing structure for the hierarchical FPGA device, and this information is important for performance estimation. This experiment is only intended to illustrate the relationship between rewiring ability and the optimization potential.

## 7. Conclusion and Future Work

In this paper we present an in-depth study of the theory and algorithm for the *SPFD*-based rewiring, and explore the flexibility in the *SPFD* calculation. We develop a theorem and an efficient algorithm for a more precise characterization of feasible *SPFD*-based rewiring. Extensive experimental results show that for LUT-based FPGAs, the rewiring ability of our new algorithm is 18% greater than the *SPFD*-based global rewiring algorithm (*SPFD-GR*).

We study the impact of using different atomic *SPFD* pair assignment methods during the *SPFD*-based rewiring. Our study concludes that optimized heuristic assignment methods lead to better results than the random assignment methods in general, while the differences due to different assignment heuristics appear to be small. The improvement on selected rewiring ability is about 10%. And the combination of *SPFD-ER* and atomic *SPFD* assignment brings us 38.1% in general rewiring ability.

For speeding up purpose, a simultaneously *SPFD* rewiring technique which is similar to the compatible wire concept in [8] is also studied. We prove that two *SPFD* global rewiring can be performed simultaneously, if both alternative wires and dominators are not the transitive fanouts of the others target wires (reader may refer to [22]). In our future work, we plan to further study the application of this technique.

## 8. Acknowledgement

This work is partially supported by the Gigascale Silicon Research Center (GSRC) and the California

MICRO program with Actel, Altera, Lucent and Xilinx. We thank S. Yamashita, H. Sawada and A. Nagoya of NTT Corp., Japan, for their binary code of the original *SPFD* program [19] for experimental purpose; and Prof. Robert Brayton of UC Berkeley for his stimulating discussions on the *SPFD* technique during various GSRC workshops.

## References

- [1] Altera, "Quartus II Software Overview," <http://www.altera.com/products/software/quartus2/qts-index.html>.
- [2] L. A. Entrena and K.-T. Cheng. Combinational and Sequential Logic Optimization by Redundancy Addition and Removal. *IEEE Transaction on CAD of ICS*, Vol. 14, No. 7, pp. 909-916, July 1995.
- [3] R. K. Brayton. Understanding *SPFDs*: A New Method for Specifying Flexibility. In *International Workshop on Logic Synthesis*, 1997.
- [4] C.-W. Chang, C.-K. Cheng, P. Suaris, and M. Marek-Sadowska. Fast Post-placement Rewiring Using Easily Detectable Functional Symmetries. In *Design Automation Conference*, p. 286-289, 2000.
- [5] S.-C. Chang, M. Marek-Sadowska, and K-T Cheng. Perturb and Simplify: Multilevel Boolean Network Optimizer. *IEEE Trans CAD of ICAS*, Vol. 15, No. 12, Dec 1996, pp. 1494 - 1504.
- [6] S.-C. Chang, K.-T. Cheng, N.-S. Woo, and M. Marek-Sadowska. Postlayout rewiring using alternative wires. *IEEE Transaction on CAD of ICS*, Vol. 16, No.6, p.587-96, June 1997.
- [7] S.-C. Chang, L. V. Ginneken, and M. Marek-Sadowska. Circuit Optimization by Rewiring. *IEEE Transaction on Computers*, Vol. 48, No. 9, pp. 962-970 September 1999.
- [8] P. Chong, Y. Jiang, S. Khatri, F. Mo, S. Sinha, and R. Brayton. Don't Care Wires in Logical/Physical Design. In *International Workshop on Logic Synthesis*, pp. 1- 9, 2000.
- [9] J. Cong, Y. Hwang. Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping. *Proc. ACM 3rd Int'l Symp. on FPGA*, Feb. 1995, pp. 68-74.
- [10] J. Cong, S. K. Lim. Edge Separability based Circuit Clustering With Application to Circuit Partitioning. *IEEE/ACM Asia South Pacific Design Automation Conference*, p. 429-434, 2000.
- [11] J. Cong, J. Peck, and Y. Ding. RASP: A General Logic Synthesis System for SRAM-based FPGAs. In *Proc. ACM/SIGDA Int'l Symp. on FPGAs*, p. 137-143, Feb. 1996.
- [12] R. Huang, Y. Wang, and K.-T. Cheng. LIBRA-a library-independent framework for post-layout performance optimization. In *International Symposium on Physical Design*, p.135-140, 1998.
- [13] J. - M. Hwang, F. - Y. Chiang, and T.- T. Hwang. A Re-engineering Approach to Low Power FPGA Design Using *SPFD*. In *Design Automation Conference*, p. 722-725, 1998.
- [14] Y.-M. Jiang, A. Krstic, K.-T. Cheng, and M. Marek-Sadowska. Post-layout rewiring for performance optimization. In *Design Automation Conference*, p.662-665, 1997.
- [15] W. Kunz and P. R. Menon. Multilevel Logic optimization by implication Analysis", In *International Conference on Computer Aided Design*, p. 6-13, 1994.
- [16] E. Sentovich, et. al. SIS: A System for Sequential Circuit Synthesis. Memorandum No. UCB/ERL M92/41, Dept. EECS, UC Berkeley, 1992.
- [17] S. Sinha and R. K. Brayton. Implementation and Use of *SPFDs* in Optimizing Boolean Networks. In *International Conference on Computer Aided Design*, p. 103 - 110, 1998.
- [18] Fabio Somenzi. CUDD: CU Decision Diagram Package Release 2.3.0. Technique Report, Dept. of ECE, Univ. of Colorado at Boulder, 1998.
- [19] S. Yamashita, H. Sawada and A. Nagoya. A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications. In *International Conference on Computer Aided Design*, p. 254 - 261, 1996.
- [20] J. Cong, Y. Lin and W. Long. *SPFD*-based Global Rewiring. In *Proc. ACM/SIGDA Int'l Symp. on FPGAs*, p. 77-84, 2002.
- [21] S. Sinha, R.K. Brayton. Improved Robust *SPFD* Computations. In *International Workshop on Logic Synthesis*, p.156-161, 2001.
- [22] J. Cong, Y. Lin and W. Long. New Advances in *SPFD* Rewiring. In *UCLA CSD Tech. Report No.020019*. May 2002.