# An Improved Approach for Alternative Wires Identification *

Yung-Chih Chen   Chun-Yao Wang
Department of Computer Science, National Tsing Hua University, HsingChu, Taiwan R.O.C.
{phi@nthucad.cs.nthu.edu.tw; wcyao@cs.nthu.edu.tw}

## Abstract

*Redundancy Addition and Removal (RAR) is a restructuring technique used in the synthesis and optimization of logic designs and physical designs. It finds alternative wires to replace a given target wire without changing the functionality of the circuit. Previous approaches apply two-stage algorithms for this problem. First, they build up a set of candidate wires for the target wire. Second, they perform redundancy test on each candidate wire to determine if it is an alternative wire. Recently, a one-stage algorithm RAMFIRE [1] is proposed. It conducts three logic implications to identify backward alternative wires without trial-and-error redundancy tests. However, the number of alternative wires it can find is smaller than that obtained by the previous two-stage approaches. Here, we propose an improved one-stage algorithm, which only conducts two logic implications. The experimental results show that compared to RAMFIRE, our approach only requires 83% cpu time on average, while obtaining the same number of backward alternative wires. As extending to finding both backward and forward alternative wires, on average our approach gets 157% improvement with 32% cpu time overhead.*

## 1. Introduction

Redundancy Addition and Removal (RAR) or rewiring is a process of adding an alternative wire and removing a target wire in a circuit without changing the circuit's functionality. The objective of RAR is to add a redundant wire such that the target wire becomes redundant and, hence, can be removed. Many applications based on the RAR technique have been developed over last decade [2] [6] [8]. With this technique, designers can optimize a circuit to achieve certain objectives by performing a sequence of rewiring.

In general, the techniques for RAR can be divided into two categories. One is Automatic Test Pattern Generation (ATPG)-based approaches [1]∼[5] [7], and the other is graph-based approach [14]. The former first constructs a set of candidate wires by finding the Mandatory Assignments (MAs) for the stuck-at fault test of the target wire. Then it performs redundancy test on each candidate wire to determine if it is an alternative wire. The latter depends on a set of pre-defined graph patterns and each pattern has a pair of target wire and alternative wire. It only requires a pattern matching process between the local sub-circuit and the graph pattern to identify alternative wires. However, the rewiring capability is limited to the number of pre-defined graph patterns. A quantitative comparison and analysis on these approaches are presented in [13].

The traditional ATPG-based approaches [2]∼[5] [7] are two-stage algorithms. They conduct redundancy tests in the second stage on each candidate wire which are suggested in the first stage. Thus, they would spend much effort in the redundancy tests when the candidate set is large. One way to reduce the computation effort is to prune the candidate set. In [5], REWIRE proposes theorems as filters to eliminate those wires that cannot be redundant in the candidate set. Nevertheless, the redundancy tests are still required for the remaining candidate wires. On the other hand, RAMFIRE [1] uses FIRE [9], which is a redundancy identification algorithm, to identify redundant wires such that it can identify alternative wires in one stage. Although RAMFIRE has much improvement on cpu time, about 15 times reduction on average, its rewiring capability is not as good as that of previous two-stage approach [5]. Furthermore, RAMFIRE only finds the backward alternative wires. RAMFIRE contains three logic implications for the backward alternative wire identification.

In this paper, we propose an ATPG-based and improved one-stage algorithm for alternative wires identification. It only requires two logic implications and is applicable to both backward and forward alternative wires. It should be noted that we do not propose an algorithm that targets a specific optimization objective, but we introduce a new approach that improves and complements existing technique [1].

This paper is organized as follows. Section 2 introduces some notations and reviews related concepts. Section 3 describes the sufficient conditions for forward and backward alternative wires. Section 4 describes the alternative wires identification with the addition of redundant gates. Section 5

*This work was supported in part by ROC National Science Council under Grant NSC93-2220-E-007-012.

discusses the complexity analysis and complexity reduction by our approach. Section 6 presents the experimental results. Finally, Section 7 concludes this work.

## 2. Notations and background

In this work, we only consider circuits consisting of AND, OR and INV gates. Complex gates can be decomposed into these gates. Also, the circuits are irredundant, i.e., no wire in the circuits is redundant. This is because the redundant circuits can be restructured directly by removing redundant wires. These circuits are not considered by the RAR technique.

A Boolean network is a Directed Acyclic Graph (DAG) where each node $n_i$ is associated with a Boolean function $f_i$. A wire $w(n_i \rightarrow n_j)$ is a connection directed from a node $n_i$ to a node $n_j$. An input of a gate $g$ has a $controlling\ value\ cv(g)$ if this value determines the output of the gate $g$ regardless of the other inputs. The inverse of the controlling value is called $noncontrolling\ value\ ncv(g)$ of gate $g$. For example, if $g$ is an AND gate, the $cv(g)$ is 0 and the $ncv(g)$ is 1. The $dominators$ [10] of a wire $w$ is a set of gates $G$ such that all paths from $w$ to any primary outputs have to pass through all gates in $G$. Consider the dominators of a wire $w$, the $fault\ propagating\ inputs$ of a dominator are its inputs in the transitive fanout of the wire $w$, and the other inputs are $side\ inputs$ of the dominator. For stuck-at 1 $\{0\}$ fault test on a wire $w(n_i \rightarrow n_j)$, a test vector must generate 0 $\{1\}$ at the source $n_i$ of the wire $w$ to activate the fault effect and generate noncontrolling values for all side inputs of $w$'s dominators to propagate the fault effect. If no such test vector exists, the wire $w$ is stuck-at 1 $\{0\}$ untestable and can be replaced by a constant 1 $\{0\}$.

The $mandatory\ assignments$ (MAs) are the unique value assignments to nodes required for a test to exist. The MAs for a test can be obtained by performing $logic\ implication$. Logic implication is a process of computing unique logic values based on known logic values of wires in a Boolean network. Given a logic value assigned at one wire, the value can be propagated forward or backward until no more logic values can be determined. Consider computing MAs for a stuck-at fault test on a wire $w(n_i \rightarrow n_j)$, the MA on $n_i$ is set to fault-activating value and the MAs on the side inputs of dominators $n_d$ are set to their corresponding $ncv(n_d)$. The MAs then can be propagated forward or backward to infer more MAs. Recursive learning [11], which is a learning method for ATPG, can be used to find more MAs.

*Forced MA* [5] is a kind of MA such that violating it causes the target fault untestable. The MAs obtained by setting side inputs of dominators $n_d$ to $ncv(n_d)$ and activating the target fault effect are forced. Besides, the MAs obtained by backward implications are also forced. Based on the idea of violating Forced MA, a theorem developed in [5] shows a necessary and sufficient condition for a redundant wire to be an alternative wire for the target wire $w_t$.

**Theorem 1**: (Theorem 13 in [5]) A redundant wire $w_r(n_s \rightarrow n_d)$ is an alternative wire for $w_t$, if and only if an AND $\{$OR$\}$ gate $n_d$ has a forced MA 1 or D $\{0$ or $\overline{D}\}$ and $n_s$ has a MA 0 $\{1\}$ for the stuck-at fault test of $w_t$.

Theorem 1 also shows how to construct a set of candidate wires that can make the target wire redundant. First, compute the MAs for the $w_t$ stuck-at fault test. Then, collect a set of candidate wires according to the MAs and the types of gates.

In this work, we analyze Theorem 1 and then derive two conditions for identifying wires that make the target wire redundant. Furthermore, we also propose two corresponding conditions for justifying which candidate wires are redundant.

Note that for simplicity, the target wire is denoted as $w_t$ and its alternative wire is denoted as $w_a$. Furthermore, the stuck-at fault of $w_t$ is called the $target\ fault$ in this paper. In the following examples, we always explain our approach using stuck-at 1 fault test for convenience. In fact, it works for stuck-at 0 fault test as well.

## 3. Single alternative wire

The idea of *Single Alternative Wire (SAW)* is proposed in [3]. For a wire to be a SAW $w_a$ of a given target wire $w_t$, two requirements have to be held. (1) The addition of $w_a$ makes the target fault untestable and, hence, $w_t$ is redundant. (2) $w_a$ itself is a redundant wire in the original circuit. For example, in Figure 1(a) (taken from [5]), adding $w(g_1 \rightarrow g_5)$ makes $w(c \rightarrow g_2)$ stuck-at 1 fault untestable, and $w(g_1 \rightarrow g_5)$ is a redundant wire in the circuit. Therefore, $w(g_1 \rightarrow g_5)$ is a SAW for $w(c \rightarrow g_2)$. Since $w(g_1 \rightarrow g_5)$ is an alternative wire for $w(c \rightarrow g_2)$, $w(c \rightarrow g_2)$ is an alternative wire for $w(g_1 \rightarrow g_5)$ as well.

Let us discuss the requirement (1) first, i.e., how to make a target fault untestable by adding a wire to the circuit. There are two methods for this. (a) Blocking the fault effect propagation to the primary outputs. (b) Causing MAs inconsistent. These two methods also classify SAWs into forward and backward. We will discuss them in Section 3.1 and 3.2, respectively. The requirement (2), checking if $w_a$ itself is a redundant wire in the original circuit, will be discuss in the corresponding subsections as well.

### 3.1. Forward single alternative wire

We say that a SAW is a *Forward* Single Alternative Wire (FSAW) of $w_t$ if its addition can block the fault effect propagation of the target fault test. We can add a wire according to the type of the dominator $n_d$ and the MAs at $n_s$, such that the target fault is untestable. The sufficient condition of an

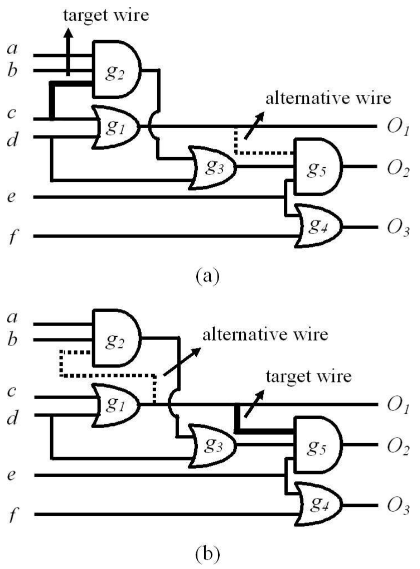


**Figure 1. Examples of single alternative wire. (a) Forward: blocking the fault effect propagation to the primary outputs. (b) Backward: causing the MAs inconsistent.**

added wire to block the fault effect propagation of the target fault test is stated in Condition 1.

**Condition 1**: If there exists a gate $n_s$ with a MA $cv(n_d)$ for the target fault test, where $n_d$ is a dominator of $w_t$, adding $w(n_s \rightarrow n_d)$ can block the fault effect propagation of the target fault test.

Next, let us consider requirement (2), determining if the added wire is a redundant wire or not. Suppose $w(n_s \rightarrow n_d)$ is a wire added to the circuit, where $n_d$ is a dominator of $w_t$ and $p$ is the fault propagating input of $n_d$. For the stuck-at fault test of $w(n_s \rightarrow n_d)$, $p$ has to be $ncv(n_d)$ to propagate fault effect through $n_d$. However, if $p = cv(n_d)$ is necessary to activate the fault effect, the fault is untestable and, hence, the added wire $w(n_s \rightarrow n_d)$ is a redundant wire. The following condition is a sufficient condition for a wire connecting to a dominator is a redundant wire.

**Condition 2**: If $w(n_s \rightarrow n_d)$ requires fault propagating input $p$ of $n_d$ to be $cv(n_d)$ to activate fault effect for $w(n_s \rightarrow n_d)$ stuck-at fault test, $w(n_s \rightarrow n_d)$ is redundant.

**Theorem 2**: If there exists a wire $w(n_s \rightarrow n_d)$ which satisfies Condition 1 and Condition 2, it is a FSAW of $w_t$.

We can modify Condition 1 for identifying a SAW $w(n_s \rightarrow n_d)$ with an INV between $n_s$ and $n_d$. Figure 2 summarizes the configurations of $n_s$ values and $n_d$ types such that $w(n_s \rightarrow n_d)$ is a $w_a$. In each configuration, there is a pair of values, $x/y$, for $n_s$. $x$ is the MA of $n_s$ for the target fault test and $y$ is the result of $n_s$ after performing logic

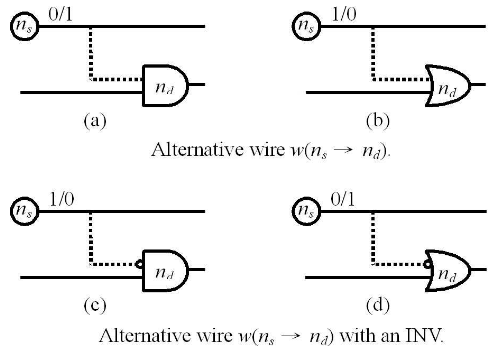


**Figure 2. FSAW addition example.**

implication of $p = ncv(n_d)$.

The procedure for FSAWs identification is conducted as follows. First, computing the MAs for the target fault test. Second, selecting a dominator $n_d$ of $w_t$ and then performing logic implication of $p = ncv(n_d)$. Finally, checking if there exists a wire $w(n_s \rightarrow n_d)$ that satisfies Condition 1 and Condition 2.

For example, in Figure 1(a), suppose we would like to remove $w(c \rightarrow g_2)$ and $w(g_1 \rightarrow g_5)$ is not present in the circuit now. First, after computing $w(c \rightarrow g_2)$ stuck-at 1 fault test , we have the MAs:$\{c$=0, $b$=1, $a$=1, $d$=0, $g_1$=0, $e$=1, $g_4$=1$\}$. Then in the second step, we select a dominator $g_5$ of $w(c \rightarrow g_2)$. Since $w(g_3 \rightarrow g_5)$ is the fault propagating input of $g_5$, we perform logic implication of $w(g_3 \rightarrow g_5)$=$ncv(g_5)$=1 and then have $g_1$=1 by recursive learning. Finally, we find that adding $w(g_1 \rightarrow g_5)$ to the circuit makes $w(c \rightarrow g_2)$ stuck-at 1 fault untestable, and the stuck-at 1 fault of $w(g_1 \rightarrow g_5)$ is untestable as well. Therefore, $w(g_1 \rightarrow g_5)$ is a FSAW of $w(c \rightarrow g_2)$.

## 3.2. Backward single alternative wire

We say that a SAW is a *Backward* Single Alternative Wire (BSAW) of $w_t$ if its addition makes the MAs of the target fault test inconsistent. We can make the target fault untestable by adding a wire which violates a forced MA. The sufficient condition of an added wire to make the target fault untestable is stated in Condition 3.

**Condition 3**: If there exists a gate $n_s$ with a MA 0 $\{1\}$ and an AND $\{$OR$\}$ gate $n_d$ with a forced MA 1 $\{0\}$ for the target fault test, adding $w(n_s \rightarrow n_d)$ makes the target fault untestable and, hence, $w_t$ is redundant.

Let us consider adding a redundant wire violating the forced MA for the target fault test. Theorem 5 of [1] shows that if $w_a$ is an alternative wire of $w_t$, after adding $w_a$ to the circuit, the redundancy test on $w_a$ results in a conflict at $w_t$. Therefore, we can find a redundant wire by checking if its redundancy test results in a conflict at $w_t$. The following condition is a sufficient condition for a wire to be a redundant wire whose redundancy test results in a conflict at $w_t$.

**Condition 4**: Suppose the source of $w_t$ is $n_{ts}$ and the sink is $n_{td}$. If $w(n_s \to n_d)$ requires $n_{ts} = cv(n_{td})$ to activate the fault effect and $n_{ts} = ncv(n_{td})$ to propagate the fault effect for $w(n_s \to n_d)$ stuck-at fault test, $w(n_s \to n_d)$ is redundant.

**Theorem 3**: If there exists a wire $w(n_s \to n_d)$ which satisfies Condition 3 and Condition 4, it is a BSAW of $w_t$.

We can also modify Condition 3 for identifying BSAWs with inverted polarity. If there exists a gate $n_s$ with a MA 0 $\{1\}$ and an OR $\{\text{AND}\}$ gate $n_d$ with a forced MA 0 $\{1\}$, adding $w(n_s \to n_d)$ with an INV between $n_s$ and $n_d$ makes the target fault untestable and, hence, $w_t$ is redundant.

The procedure for BSAWs identification is conducted as follows. Suppose the source of $w_t$ is $n_{ts}$ and the sink is $n_{td}$. First, computing the MAs for the target fault test. Second, performing logic implication of $n_{ts} = ncv(n_{td})$. Finally, checking if there exists a wire $w(n_s \to n_d)$ which satisfies Conditions 3 and Condition 4.

For example, in Figure 1(b), suppose we would like to remove $w(g_1 \to g_5)$ and $w(g_1 \to g_2)$ is not present in the circuit now. First, after computing $w(g_1 \to g_5)$ stuck-at 1 fault test, we have the MAs:$\{g_1$=0, $c$=0, $d$=0, $g_3$=1, $e$=1, $g_2$=1, $a$=1, $b$=1, $g_4$=1$\}$. Among the MAs, $\{g_1$=0, $c$=0, $d$=0, $g_3$=1, $e$=1, $g_2$=1, $a$=1, $b$=1$\}$ are the forced MAs. Then in the second step, we perform logic implication of $w(g_1 \to g_5)$=1 and then have $g_1$=1. Finally, we find that adding $w(g_1 \to g_2)$ violates the forced MA on $g_2$. This is because $g_1$ has a MA 0 and $g_2$ is a AND gate with a forced MA 1. For the stuck-at 1 fault test of $w(g_1 \to g_2)$, $g_1$ has to be 0 for activating the fault effect and $g_1$ has to be 1 for propagating the fault effect through $g_5$. There is a conflict on $g_1$ for $w(g_1 \to g_2)$ stuck-at 1 test and, hence, $w(g_1 \to g_2)$ is a redundant wire. Therefore, $w(g_1 \to g_2)$ is a BSAW of $w(g_1 \to g_5)$.

# 4. Alternative wire with gate

Sometimes, a $w_t$ does not have a SAW, however it could be replaced by a redundant wire with a redundant gate [1]∼[4]. We extend our SAW identification approaches with the addition of a redundant gate. If a SAW does not exist due to the sufficient conditions are violated, a redundant gate $g_r$ is added such that a wire $w_r$ could satisfy the sufficient conditions. The wire $w_r$ and the gate $g_r$ are the alternative wire and gate for the $w_t$.

## 4.1. Forward alternative wire with gate

If a FSAW cannot be found for the $w_t$, we could add a gate $n_{d\_new}$ at the fanout of $n_d$ for finding another FSAW. Since $n_{d\_new}$ is located at the fanout of $n_d$, $n_{d\_new}$ is also a dominator of $w_t$. For the target fault test, it is possible to block the fault effect propagation by adding a wire connecting to $n_{d\_new}$. For example, in Figure 3(a), suppose we would like to remove $w(b \to g_1)$ and the dotted lines and nodes are not present in the circuit now. First, we compute the MAs for $w(b \to g_1)$ stuck-at 1 fault test. We have the MAs:$\{b$=0, $a$=1, $g_2$=0$\}$. In the second step, we perform logic implication of $w(g_1 \to g_3)$=0. Finally, we cannot find a FSAW for $w(b \to g_1)$. However, we can add an AND gate $g_4$ at the fanout of $g_3$ as a new dominator for $w(b \to g_1)$ and then we have a new fault propagating input $w(g_3 \to g_4)$. After performing logic implication of $w(g_3 \to g_4)$=1, we find that $w(b \to g_4)$ which satisfies Condition 1 and Condition 2 is a $w_a$ for $w(b \to g_1)$. Therefore, $w(b \to g_1)$ can be replaced by $w(b \to g_4)$ with $g_4$.

## 4.2. Backward alternative wire with gate

Review Condition 3 for BSAW identification. If there exists a gate $n_s$ with a MA 0 $\{1\}$ and an AND $\{\text{OR}\}$ gate $n_d$ with a forced MA 1 $\{0\}$, adding $w(n_s \to n_d)$ makes the target fault untestable. If we cannot find alternative wires due to unmatched type $n_d$ in Condition 3, such as an OR $\{\text{AND}\}$ gate $n_d$ with a forced MA 1 $\{0\}$, we can add an AND $\{\text{OR}\}$ gate $n_{d\_new}$ at the fanout of $n_d$. Because $n_{d\_new}$ is located at the fanout of $n_d$, it also has a forced MA 1 $\{0\}$. After adding $n_{d\_new}$ , we get a BSAW $w(n_s \to n_{d\_new})$ for $w_t$. Therefore, $w(n_s \to n_{d\_new})$ and $n_{d\_new}$ are the alternative wire with gate for the $w_t$. For example, in Figure 3(b), suppose we would like to remove $w(g_1 \to g_3)$ and the dotted lines and node are not present in the circuit now. First, we compute the MAs for $w(g_1 \to g_3)$ stuck-at 1 fault test. We have the MAs:$\{g_1$=0, $a$=0, $b$=0, $g_2$=1, $g_4$=1, $e$=1$\}$ and they are all forced MAs. In the second step, we perform logic implication of $w(g_1 \to g_3)$=1 and we have $g_1$=1. The stuck-at 1 fault test of a wire with a source $g_1$ requires $g_1$=0 to activate fault effect. Furthermore, the stuck-at fault test of a wire with a sink $g_4$ requires $g_1$=1 to propagate fault effect through $g_3$. However, $g_4$ is not an AND gate and, hence, $w(g_1 \to g_4)$ is not an alternative wire. But we can add an AND gate $g_5$ between $g_4$ and $g_2$. Since $g_4$ has a forced MA 1, $g_5$ also has a forced MA 1, adding $w(g_1 \to g_5)$ violates the forced MA. Next, consider the stuck-at 1 test of $w(g_1 \to g_5)$, the test requires $g_1$=0 to activate the fault effect and $g_1$=1 to propagate the fault effect. Therefore, $w(g_1 \to g_5)$ is a redundant wire. Consequently, $w(g_1 \to g_3)$ can be replaced by $w(g_1 \to g_5)$ with $g_5$.

# 5. Complexity and quality analysis

This section reviews our backward alternative wires identification procedure and analyzes the time complexity. At the

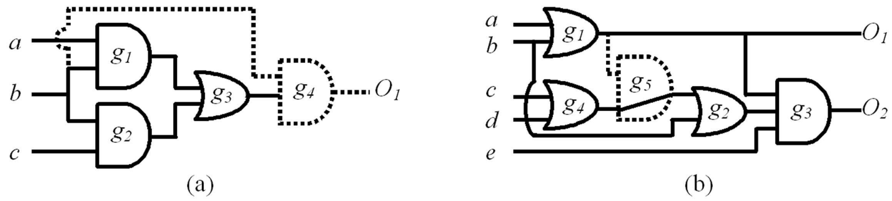


**Figure 3. Examples of alternative wire with gate. (a) Forward: add a new dominator $g_4$. (b) Backward: add a redundant node $g_5$.**

end, we explain why our approach obtains the same number of backward alternative wires as RAMFIRE but costs less computation effort.

### 5.1. The time complexity comparison

Let us consider identifying BSAWs for a target wire $w_t$, we first compute the MAs for the target fault test. Let $M$ denote the time complexity of computing the MAs. After computing the MAs, we perform a logic implication from $w_t$. We use $P$ to denote the time complexity of performing a logic implication. Therefore, the time complexity of finding BSAWs for $w_t$ is $M + P$. Consider adding a redundant gate, because no additional logic implications are needed, the time complexity is also $M + P$.

FIRE [9] is a redundancy identification algorithm that can identify redundant wires in a circuit. RAMFIRE uses FIRE to determine if the added wires are redundant wires. The process of RAMFIRE includes three logic implications. First, computing the MAs for the target fault test (one implication). Then, performing FIRE on $w_t$ to find redundant wires. The process of FIRE involves two logic implications and each logic implication includes uncontrollability and unobservability implications. We use $F$ to denote the time complexity of one logic implication. Thus, the time complexity of RAMFIRE is $M + 2F$. The complexity comparison of $F$ and $P$ will discussed in Section 5.2.

### 5.2. The determination of redundant wires

Let us consider determining if the added wires are redundant wires. In an irredundant circuit, there are two possible situations for a wire to cause a conflict on the target wire $w_t(n_{ts} \rightarrow n_{td})$ for its stuck-at fault test. (1) A wire requires $n_{ts} = ncv(n_{td})$ to activate fault effect and $n_{ts} = cv(n_{td})$ to propagate fault effect. (2) A wire requires $n_{ts} = cv(n_{td})$ to activate fault effect and $n_{ts} = ncv(n_{td})$ to propagate fault effect. However, situation (1) is useless for finding alternative wires.

**Theorem 4**: Suppose the source of $w_t$ is $n_{ts}$ and the sink is $n_{td}$. If a redundant wire $w_r$ that is not yet in the circuit requires $n_{ts} = ncv(n_{td})$ to activate the fault effect, and requires $n_{ts} = cv(n_{td})$ to propagate the fault effect for its stuck-at fault test, then $w_r$ is *not* an alternative wire of $w_t$.

Therefore, for identifying backward alternative wires, we only consider the wires that require $n_{ts} = cv(n_{td})$ to activate fault effect and $n_{ts} = ncv(n_{td})$ to propagate fault effect for their stuck-at fault test. In our approach, our logic implication of $n_{ts} = ncv(n_{td})$ does not involve unobservability implication which FIRE involves. Thus, $F$ is more complicated than $P$. Furthermore, we combine the implication of $n_{ts} = cv(n_{td})$ with the process of computing the MAs for the target fault test. Therefore, as compared with RAMFIRE, our approach costs less computation effort and obtains the same rewiring results.

## 6. Experimental results

The experiments are conducted over a set of ISCAS85 and MCNC benchmarks within SIS [12] environment on a Sun Fire V440 workstation. These benchmarks are in Berkeley Logic Interchange Format (BLIF), which is a netlist level design description. Since the circuits under consideration only consist of AND, OR, and INV gates and are irredundant, we decompose the complex gates into these primitive gates and remove redundant wires by using *decomp_tech_network* and *com_redundancy_removal* functions in SIS, respectively.

Our approach for finding backward alternative wires is named $P_b$, and for finding both backward and forward alternative wires is named $P_{b+f}$. In the experiments, finding more alternative wires is desirable. Recursive learning technique is also applied in these experiments with depth=1.

Table 1 summarizes the experimental results of our approaches as compared with the reimplemented RAMFIRE algorithm. Column 1 lists the name of the benchmarks. Column 2 lists the total number of wires in each benchmark, $N_t$. These wires are all considered as target wires in the experiments. Column 3, 5, and 7 list the number of target wires having alternative wires. Column 4, 6, and 8 list the cpu time of corresponding approaches measured in second.

According to Table 1, the number of target wires which have alternative wires are the same for $P_b$ and RAMFIRE, which is shown in row Ratio 1. However, $P_b$ only requires 83% cpu time of that of RAMFIRE on average, which is shown in row Ratio 2. In particular, for circuits

**Table 1. Comparison of experimental results between RAMFIRE [1] and our approaches.**

| | | Back. | | | | Back.+For. | |
|---|---|---|---|---|---|---|---|
| | | [1] | | $P_b$ | | $P_{b+f}$ | |
| Circuit | $N_t$ | $N_a$ | CPU | $N_a$ | CPU | $N_a$ | CPU |
| c432 | 285 | 73 | 2.24 | 73 | 2.1 | 109 | 2.84 |
| c880 | 640 | 149 | 2.22 | 149 | 2.07 | 183 | 3.77 |
| c1908 | 1046 | 147 | 31.24 | 147 | 29.58 | 321 | 55.7 |
| c2670 | 1167 | 201 | 14.24 | 201 | 12.73 | 279 | 18.66 |
| c3540 | 2060 | 396 | 135.44 | 396 | 125.71 | 952 | 204.73 |
| c5315 | 3398 | 414 | 31.59 | 414 | 29.56 | 1186 | 51.83 |
| c7552 | 4425 | 659 | 199.94 | 659 | 166.47 | 2128 | 225.11 |
| 9symml | 387 | 40 | 6.37 | 40 | 5.88 | 152 | 7.84 |
| alu2 | 598 | 57 | 28.26 | 57 | 24.37 | 205 | 60.61 |
| alu4 | 1239 | 87 | 156.04 | 87 | 119.4 | 399 | 276.93 |
| apex6 | 1216 | 75 | 7.84 | 75 | 5.07 | 240 | 13.16 |
| apex7 | 411 | 24 | 1.81 | 24 | 1.35 | 119 | 3.03 |
| b9 | 230 | 54 | 0.37 | 54 | 0.31 | 131 | 0.56 |
| c8 | 335 | 63 | 1.06 | 63 | 0.91 | 157 | 1.94 |
| cc | 131 | 29 | 0.21 | 29 | 0.16 | 78 | 0.25 |
| cm85a | 88 | 28 | 0.12 | 28 | 0.11 | 56 | 0.17 |
| comp | 159 | 54 | 0.38 | 54 | 0.36 | 94 | 0.47 |
| cu | 109 | 44 | 0.17 | 44 | 0.15 | 71 | 0.22 |
| dalu | 2708 | 797 | 576.61 | 797 | 481.44 | 1754 | 657.16 |
| example2 | 487 | 44 | 2.94 | 44 | 2.38 | 115 | 5.31 |
| frg2 | 2604 | 717 | 196.86 | 717 | 156.08 | 1544 | 261.65 |
| go | 111 | 30 | 0.13 | 30 | 0.12 | 79 | 0.2 |
| i10 | 3808 | 459 | 537.21 | 459 | 444.31 | 1401 | 663.43 |
| lal | 304 | 118 | 1.08 | 118 | 0.94 | 192 | 1.21 |
| mux | 82 | 8 | 0.12 | 8 | 0.1 | 24 | 0.22 |
| pair | 2770 | 509 | 19.88 | 509 | 16.3 | 1177 | 29.13 |
| pcler8 | 125 | 7 | 0.2 | 7 | 0.16 | 24 | 0.44 |
| pm1 | 111 | 30 | 0.12 | 30 | 0.11 | 79 | 0.2 |
| rot | 1579 | 106 | 24.92 | 106 | 23.43 | 500 | 51.72 |
| sct | 299 | 66 | 1.89 | 66 | 1.72 | 189 | 2.75 |
| term1 | 789 | 196 | 9.63 | 196 | 8.93 | 538 | 14 |
| ttt2 | 558 | 126 | 3.78 | 126 | 3.24 | 369 | 6.19 |
| unreg | 208 | 16 | 0.51 | 16 | 0.4 | 64 | 0.79 |
| x3 | 2084 | 377 | 24.69 | 377 | 20.2 | 1154 | 42.18 |
| x4 | 1025 | 260 | 14.61 | 260 | 12.19 | 513 | 19.48 |
| Total | 37576 | 6460 | 2034.72 | 6460 | 1698.34 | 16576 | 2683.88 |
| Ratio 1 | 1 | 0.1719 | | 0.1719 | | 0.4411 | |
| Ratio 2 | | | 1 | | 0.83 | | 1.32 |
| Ratio 3 | | 1 | | 1 | | 2.57 | |

with more target wires, the cpu time reduction is more significant. Comparing $P_{b+f}$ with RAMFIRE, $P_{b+f}$ has $N_a/N_t$ =44.11% and gets 157% improvement with 32% cpu time overhead on average. This is because $P_{b+f}$ includes the process of finding forward alternative wires that RAMFIRE canonot find. This is shown in row Ratio 2 and 3. Since RAMFIRE shows that it has 15 times speed-up as compared with REWIRE on average, our approach is efficient as well.

## 7. Conclusions

Redundancy Addition and Removal is an important technique for synthesis and optimization of logic designs and physical designs. RAMFIRE has shown that it obtains much speed-up for backward alternative wires as compared with previous approaches. Especially, as compared with REWIRE, the speed-up is 15 times on average. In this paper, we propose an improved approach for identifying *both* forward and backward alternative wires of a target wire. As compared with RAMFIRE, our approach obtains the same results for backward alternative wires with less cpu time. Furthermore, our approach finds forward alternative wires as well. The rewiring capability of our approach is close to that of REWIRE.

## References


[1] C.-W. Jim Chang, M.-F. Hsiao, and M. Marek-Sadowska, "A New Reasoning Scheme for Efficient Redundancy Addition and Removal," *IEEE Trans. Computer-Aided Design*, vol. 22, pp. 945-952, July 2003.

[2] S.-C. Chang, K.-T. Cheng, N.-S. Woo, and M. Marek-Sadowska, "Postlayout Logic Restructuring Using Alternative Wires,"*IEEE Trans. Computer-Aided Design*, vol. 16, pp. 587-596, June 1997.

[3] S.-C. Chang, K.-T. Cheng, N.-S. Woo and M. Marek-Sadowska, "Layout Driven Logic Synthesis for FPGA," in *Proc. Design Automation Conf.*, pp. 308-313, 1994.

[4] S.-C. Chang, M. Marek-Sadowska, and K.-T. Cheng, "Perturb and Simplify: Multi-level Boolean Network Optimizer,"*IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1494-1504, Dec. 1996.

[5] S.-C. Chang, L. P. P. P. Van Ginneken, and M. Marek-Sadowska, "Fast Boolean Optimization by Rewiring," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 262-269, 1996.

[6] David I. Cheng, C.-C. Lin, and M. Marek-Sadowska, "Circuit Partitioning with Logic Perturbation," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 650-655, 1995.

[7] L. A. Entrena, and K.-T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal,"*IEEE Trans. Computer-Aided Design*, vol. 14, pp. 909-916, July 1995.

[8] L. A. Entrena, J. A. Espejo, E. Olias, and J. Uceda, "Timing Optimization by An Improved Redundancy Addition and Removal Technique," in *Proc. Eur. Design Automation Conf.*, pp. 342-347. 1996,

[9] M. A. Iyer and M. Abramovici, "FIRE: A Fault-Independent Combinational Redundancy Identification Algorithm,"*IEEE Trans. Very Large Scale Integrated Systems*, vol. 4, pp. 295-301, June 1996.

[10] T. Kirkland and M. R. Mercer, "A Topological Search Algorithm For ATPG," in *Proc. Design Automation Conf.*, pp. 502-508, 1987.

[11] W. Kunz and D. K. Pradhan, "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation for Digital Circuits," in *Proc. Int. Test Conf.*, pp. 816-825, 1992.

[12] E. M. Sentovich, K. T. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proc. IEEE Int. Conf. Computer Design*, pp. 328-333, 1992.

[13] W.-C. Tang, W.-H. Lo, T.-K. Lam, K.-K. Mok, C.-K. Ho, S.-H. Yeung, H.-B. Fan, and Y.-L. Wu, "A Quantitative Comparison and Analysis on Rewiring Techniques," in *Proc. Int. Conf. on ASIC*, pp. 242-245, 2003.

[14] Y.-L. Wu, W.-N. Long, and H.-B. Fan, "A Fast Graph-based Alternative Wiring Scheme for Boolean Networks," in *Proc. Int. VLSI Design Conf.*, pp. 268-273, 2000.