# Error Injection & Correction : An Efficient Formal Logic Restructuring Algorithm

Ching-Yi Huang, Daw-Ming Lee, Chun-Chi Lin, and Chun-Yao Wang
Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan, R.O.C.

***Abstract*—Redundancy Addition and Removal (RAR) and ATPG/Diagnosis-based Design Rewiring (ADDR) are both logic restructuring techniques used in the synthesis and optimization of logic designs. However, not every irredundant target wire can be successfully removed due to some limitations in these two approaches. Therefore, this paper proposes an efficient restructuring technique, Error Injection & Correction (EIC), which formally constructs a corresponding rectification network at feasible locations without verification efforts for the injected errors of the wire removal, addition, or gate replacement. We use the EIC to serve as a logic perturbation engine and combine other logic optimization engines to optimize the circuit. The experimental results show that the size of highly optimized circuits can be further reduced with this EIC technique as compared with the configuration using only logic optimization engines.**

## I. Introduction

Logic synthesis is a process in VLSI design. In this process, gate-level design is often restructured to achieve different optimization objectives. Existing approaches to logic restructuring can be classified into two categories: *Redundancy* Restructuring and *Error Injection-based* Restructuring. The Redundancy Restructuring approach keeps a circuit's functionality intact in *every* operation. *Node merging* [4] [12] ~ [15] [18] [23] [34], *Rewriting* [21], and *Redundancy Addition and Removal* (RAR) [5] ~ [11] [16] [20] [24] [31] belong to this category. Although these redundancy restructuring techniques efficiently transform a circuit, a shortcoming shared among them is that some wires/gates cannot be adjusted/removed when certain specific conditions are not met.

On the other hand, the Error Injection-based Restructuring approach transforms a circuit by first injecting an error/errors and then correcting it/them [19] [25] [26] [32] [33]. This approach usually provides more restructuring abilities than the Redundancy Restructuring approach.

In this work, we propose a new technique, Error Injection & Correction (EIC) for circuit restructuring, which also belongs to the error injection-based category. The EIC can directly remove, add, or replace a desired irredundant target wire/gate and rectify the erroneous functionality due to this removal, addition, and replacement by adding some other wires/gates, which is named as a *rectification network* The main contribution of this paper is that it is the first error-injection based restructuring technique that can simultaneously deal wit removal, addition, and replacement of wire/gates of logic circuits without needing a verification procedure [1] [27] ~ [30]. Moreover, the EIC can be served as a logic perturbation engine, which fast restructures a circuit without changing its functionality, and combined with other logic optimization engines to further optimize the overall circuit.

## II. Error Injection & Correction

### A. Overview

**Definition 1:** *Error Injection-based Restructuring* is a problem where given an original circuit, $C_{ori}$, and an error that is modeled by a set of error effects $E_e = \{e_1, e_2, ..., e_i\}$, we add rectification networks which are modeled by another set of error effects $E_r = \{e_{i+1}, e_{i+2}, ..., e_n\}$ into the faulty circuit, so that the rectified circuit, $C_{rec}$, becomes $C_{ori} + E_e + E_r$, and the combined error effects $E_e + E_r = \{e_1, e_2, ..., e_n\}$ are redundant to $C_{ori}$. The error effect is a situation that the different logic values are on the same wire of the original and faulty circuits.

This work is supported in part by the National Science Council of R.O.C. under Grant NSC 101-2628-E-007-005 and NSC 100-2628-E-007-031-MY3.

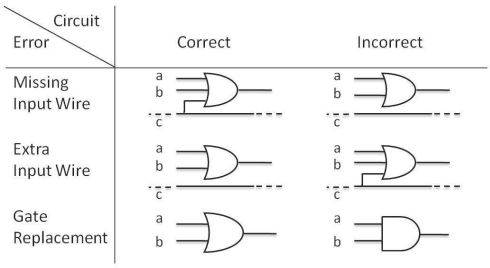


Figure 1. Design error types illustrated by using OR and AND gates [1].

There are three issues to be dealt with in the algorithm of the error injection-based approach:

1) Where to add the rectification network for the error?
2) What to add into the rectification network for the error?
3) How to construct the corrected function with the rectification networks at the selected locations?

We will discuss these three issues of the proposed algorithm in the next subsection.

### B. The Proposed Approach

Logic *design errors* are functional mismatches between the specification and the implementation [3]. A list of typical design error types, *missing input wire*, *extra input wire*, and *gate replacement*, that compose most of complex design errors, taken from [1], is shown in Fig. 1.

Let us discuss the requirements for rectifying the functionality of a circuit after introducing an error. If the error is redundant, the error effects cannot be propagated to any POs. On the other hand, if the error is irredundant, the error effects will be propagated to one or more POs. However, if these error effects can be blocked in *all* propagating paths from the error location to the POs by adding rectification networks, the circuit's functionality will be corrected. These selected locations for adding the rectification networks are called *destination gates* hereafter.

**Definition 2:** A gate $g$ is in the *Transitive FanOut Cone* (TFOC) of a gate $g_s$ if there exists a path from $g_s$ to $g$. The *dominators* [17] of a wire $w$ (or a gate $g$) is the set of gates $G$ such that all paths from $w$ (or $g$) to any POs have to pass through all gates in $G$.

After modeling an irredundant design error as error effects, these error effects will be propagated to at least one PO through the TFOC of the error location. Therefore, to block the error effects from being propagated out, the gates in the TFOC are the destination gates for the

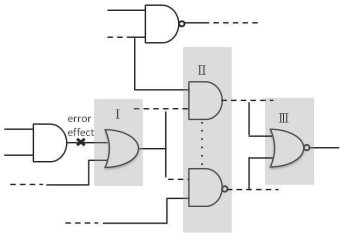


Figure 2. Three possible regions within the TFOC of the error location designated for correction.

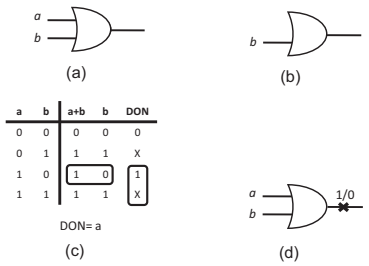


Figure 3. The modeling of a missing input wire from an OR gate. (a) The original circuit. (b) The removal of the target wire $a$. (c) The truth table. (d) The modeling with an error effect $1/0$.

rectification intuitively. Fig. 2 shows three possible regions within the TFOC of the error location for correction. We can choose any one of them as the destination gate. Note that if we choose region II, the error effects may be duplicated and then propagated out through different paths. In such a case, the rectification networks have to be added at more than one location.

After determining the location of the rectification network, we then deal with the second issue of the error injection-based approach: "What to add into the rectification network for the error?". Here we define two networks that are used to construct the rectification network.

**Definition 3:** Given a Boolean network, an error effect, and a destination gate $gd$ in the TFOC of the error effect, the *Definite ON-set network (DON)* {*Definite OFF-set network (DOFF)*} at $gd$ is defined as the network having the on-set minterms that change from the on-set {off-set} of a good circuit to the off-set {on-set} of a faulty circuit, and having the don't-care set of minterms that are in the on-set {off-set} of both good and faulty circuits.

The *DON* and *DOFF* are exactly the rectification networks for the error effects $1/0$ and $0/1$, respectively.

For example in Fig. 3, the function of the good circuit is $a+b$, and that of the faulty one is $b$. An error effect $1/0$ is modeled for the error of wire $a$ removal. The on-set of *DON* is $\{a\bar{b}\}$, because this minterm is in the on-set of the good circuit, and is in the off-set of the faulty one. The don't-care set of *DON* is $\{\bar{a}b, ab\}$ because these minterms are in the on-set of both good and faulty circuits. After including the don't-care minterm $ab$ in the *DON*, the *DON* can be simplified from $a\bar{b}$ to $a$.

Next, we discuss the third issue of the error injection-based approach: "How to construct the corrected function with the rectification networks at the selected locations?".

**Theorem 1:** Given a Boolean network, one or more than one destination gate $gd$, and the error effects at $gd$, if the error effect propagated to $gd$ is $1/0$ $\{0/1\}$, then the corrected function for the error effect is $gd + DON$ $\{gd \cdot \overline{DOFF}\}$. If both $1/0$ and $0/1$ are propagated to $gd$, the corrected function for the error effects is either $(gd + DON) \cdot \overline{DOFF}$ or $(gd \cdot \overline{DOFF}) + DON$.

**Proof:** An error effect of $1/0$ $\{0/1\}$ at $gd$ means that there are some minterms changing from the on-set {off-set} of a good circuit to the off-set {on-set} of a faulty one after injecting the error. Since *DON* {*DOFF*} at least contains these minterms by Definition 3, we can correct the function by ORing the *DON* {ANDing the $\overline{DOFF}$}. If both $1/0$ and $0/1$ are propagated to $gd$, we sequentially correct the function. Hence, the rectification network is $(gd + DON) \cdot \overline{DOFF}$. Moreover, since the minterms of *DON* and *DOFF* networks are disjoint, the ordering of connecting *DON* and *DOFF* is irrelevant. Thus, the functionality of $(gd + DON) \cdot \overline{DOFF}$ is equal to $(gd \cdot \overline{DOFF}) + DON$. ■

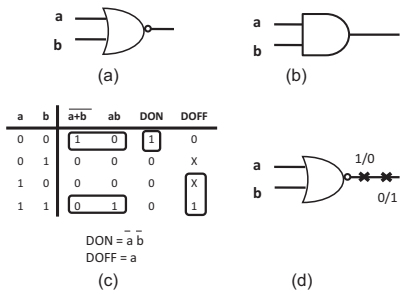


Figure 4. The modeling of a gate replacement error. (a) The original circuit is a NOR gate. (b) The NOR gate is replaced with an AND gate. (c) The truth table. (d) The modeling with the error effects $1/0$ and $0/1$.

Next, we discuss the derivation of *DON* and *DOFF*, which is an important part of the third issue in the error injection-based restructuring approach.

*a) Regions I and II:* Only one path exists from the error location to each destination gate.

**Definition 4:** The *activating assignments* are literals in the smallest rectification network at the output of the error location.

The activating assignments can be derived by examining the truth table of the gate where the error occurs. Since the error gate usually does not involve a large amount of inputs in practical circuits, this operation of activating assignment derivation can be easily achieved. For example in Fig. 4, the on-set of *DON* is $\bar{a}\bar{b}$. Since the don't-care set of *DON* is empty, the smallest rectification network is $\bar{a}\bar{b}$. Hence, the activating assignments are $\bar{a}$ and $\bar{b}$. Similarly, the on-set of *DOFF* is $ab$, the don't-care set of *DOFF* are $\bar{a}b$ and $a\bar{b}$. Hence, the smallest rectification network is either $a$ or $b$, and the corresponding activating assignment is either $a$ or $b$[1].

The DON and DOFF have to be updated when the destination gates are specified elsewhere along the path. Since there is only one propagating path from the error location to the destination gate in this discussion, the side inputs of this propagating path may block the error effect and may influence the update of DON and DOFF. We summarize the operation of this update as follows : If the good value of the error effect is the controlling value of a gate on a propagating path, which is called *controlling error effect*, the side input of this gate is ignored. On the contrary, if the good value of the error effect is the noncontrolling value of a gate on a propagating path, which is called *noncontrolling error effect*, this error effect can be propagated to the next level by setting the side input a noncontrolling value of the gate for updating. We call this side input assignment a *propagating assignment*.

Theorem 2 summarizes the method of constructing the updated rectification networks, *DON* or *DOFF*, at $gd$ in regions I and II.

**Theorem 2:** Given error effects and a destination gate $gd$, if there exists only one path from the error location to $gd$, the updated rectification network at *gd, DON* or *DOFF* is the AND of activating

[1]In Fig. 4, we choose $a$ as the smallest rectification network *DOFF*.

assignments and propagating assignments along the propagating path[2].
**Proof:** The AND of activating assignments represents the smallest rectification network at the output of the error location. Since there is only *one* path from the error location to $gd$, for all side inputs along the propagating path, if one side input assignment is not a propagating assignment, we add don't-care minterms into the rectification network by ignoring this side input assignment. On the other hand, if one side input assignment is a propagating assignment, no additional minterm will be included into the rectification network. Hence, we AND the propagating assignments with the original rectification network. As a result, the updated rectification network at $gd$ is the AND of activating assignments and propagating assignments. ■

*b) Region III:* More than one path exists from the error location to each destination gate.

For the destination gate in region III, error effects propagate through a fanout, and reach a reconvergent gate via multiple paths. Thus, the method of constructing the updated rectification networks *DON* or *DOFF* as stated in Theorem 3 is slightly different from that in regions I and II as stated in Theorem 2.

**Theorem 3:** Given error effects and a destination gate $gd$, if there exists more than one path from the error location to $gd$, then the updated rectification network $DON$ at $gd$ is expressed as

$$AND(assignment) \cdot gd_{g(assignment)} \quad (1)$$

and $DOFF$ at $gd$ is expressed as

$$AND(assignment) \cdot \overline{gd_{g(assignment)}} \quad (2)$$

where *assignment* consists of all the activating assignments and propagating assignments outside the fanout-reconvergent region and $gd_{g(assignment)}$ represents the cofactor of $gd$ with respect to these assignments in the good circuit.

**Proof:** When there is more than one path from the error location to $gd$, enumerating all the propagating assignments in all propagating paths within the fanout-reconvergent region is not practical. However, we can derive the propagating assignments outside the fanout-reconvergent region as we did in Theorem 2. As mentioned in Theorem 2, we can derive the updated rectification network for the circuit not considering the fanout-reconvergent region, and denote it as $AND(assignment)$. Upon considering the fanout-reconvergent region, we use the cofactor of $gd$ with respect to the assignments in the good and faulty circuits to represent the changed minterms. The $gd_{g(assignment)}$ is the network that represents the on-set minterms at $gd$ in the good circuit with respect to these assignments. The $\overline{gd_{f(assignment)}}$ is the network that represents the off-set minterms at $gd$ in the faulty circuit with respect to these assignments. Therefore, $AND(assignment) \cdot gd_{g(assignment)} \cdot \overline{gd_{f(assignment)}}$ can represent the minterms changing from the on-set of a good circuit to the off-set of a faulty one, i.e., the on-set of *DON*. Next, according to Definition 3, *DON* also contains the don't-care set minterms. Thus, we add the don't-care set, which is represented as $AND(assignment) \cdot gd_{g(assignment)} \cdot gd_{f(assignment)}$, to *DON*. As a result, $DON = AND(assignment) \cdot gd_{g(assignment)} \cdot \overline{gd_{f(assignment)}} + AND(assignment) \cdot gd_{g(assignment)} \cdot gd_{f(assignment)} = AND(assignment) \cdot \overline{gd_{g(assignment)}}$ as shown in EQ(1). Similarly, $AND(assignment) \cdot \overline{gd_{g(assignment)}} \cdot gd_{f(assignment)}$ can represent the minterms changing from the off-set of a good circuit to the on-set of a faulty one, i.e., the on-set of *DOFF*. As a result, $DOFF = AND(assignment) \cdot \overline{gd_{g(assignment)}} \cdot gd_{f(assignment)} + AND(assignment) \cdot \overline{gd_{g(assignment)}} \cdot \overline{gd_{f(assignment)}} = AND(assignment) \cdot \overline{gd_{g(assignment)}}$ as shown in EQ(2) when including the don't-care set. ■

[2]The *DON* or *DOFF* also can be expressed as *AND(assignment)* where *assignment* represents the activating assignments and propagating assignments along the propagating path.

## III. EXPERIMENTAL RESULTS

We implemented our algorithm in C language within the ABC [2] environment. The experiments were conducted on a 3.0 GHz Linux platform (CentOS 4.6). The benchmarks are from LGSynth93 circuits and IWLS 2005 suite [35]. Each benchmark was initially transformed into an AIG format and we only considered its combinational portion. The correctness of all results were verified by an equivalence checking tool, *cec* [22], in the ABC package.

The EIC can directly remove, add, and replace a target wire/gate in a circuit and rectify its functionality with some rectification networks. Thus, in the experiment, we served the EIC as a perturbation engine followed by other optimization engines to further achieve smaller circuit size, though the EIC itself increases the circuit size.

The optimization engine we used was a combination of three logic optimizers - Node merging (NMG) [12] [14], Node addition and Removal (NAR) [13], and *resyn2* script [21] in ABC. This is because in [13], the authors mentioned that the combination of NAR, NMG, and resyn2 script can achieve a more reduction in circuit size. Moreover, iteratively using the combined techniques can get an even smaller circuit size. Therefore, we considered the resyn2 script followed by NMG and NAR as one *optimization* process and repeatedly ran it for 30 times - $(resyn2 + NMG + NAR)\times 30 = optimization\times 30$ - over each benchmark. These highly optimized results were used for comparison against the optimization flow embedding the EIC algorithm.

On the other hand, we perturbed the circuit structure by our EIC approach after initial optimization processes and then ran the optimization processes again. In each perturbation, the target wire/gate, the injected error type, and the destination gate, were randomly selected in the experiments. Thus, the configuration of the optimization flow with our approach is $optimization\times 5 + (EIC\times 5 + optimization\times 5)\times 5$. Note that the runs of *optimization* are the same for the both two optimization flows.

Table II summarizes the experimental results. Column 1 lists the benchmarks. Column 2 lists the number of nodes $N$ in each benchmark. Columns 3 to 5 list the results by the original optimization flow. They contain the number of nodes $N_{opt}$ after the optimization, the percentage of improvement $\%$ with respect to $N$, and the CPU time $T$ measured in second. Columns 6 to 8 list the corresponding results by our approach.

For example, the benchmark *table5* initially has 1474 nodes. The original optimization flow reduced the node count to 643 nodes or 56.38% improvement. The CPU time was 19.8 seconds. On the other hand, our approach cost 19.7 seconds to reduce the node count to 580 nodes or 60.65% improvement.

Figs. 5 shows the effect of the perturbations during the optimization processes for the benchmark *table5*. In this figure, X-axis represents iteration index and Y-axis represents the node count of the resultant circuit. The dotted line represents the original optimization flow without perturbation and the bold one represents the flow with our approach. For comparison, in the original optimization flow we did not perform the *optimization* in the iterations of 6 to 10, 16 to 20, 26 to 30, 36 to 40, and 46 to 50, in which our approach perturbed the circuit structure such that the circuit restructured and the node count changed as highlighted. In Fig. 5, the result of our approach not only escaped from a local minimum but also kept reducing the area.

Table I
EXPERIMENTAL RESULTS FOR AREA OPTIMIZATION.

| Benchmarks | $N$ | Original Opt. | | | Ours | | |
|---|---|---|---|---|---|---|---|
| | | $N_{opt}$ | % | T(s) | $N_{opt}$ | % | T(s) |
| misex1 | 55 | 43 | 21.82 | 1.10 | 42 | 23.64 | 1.14 |
| sqrt8 | 96 | 48 | 50.00 | 1.12 | 41 | 57.29 | 1.18 |
| rd53 | 103 | 34 | 66.99 | 1.02 | 23 | 77.67 | 1.00 |
| misex2 | 110 | 80 | 27.27 | 1.25 | 77 | 30.00 | 1.29 |
| bw | 173 | 123 | 28.90 | 1.81 | 120 | 30.64 | 1.85 |
| 5xp1 | 210 | 74 | 64.76 | 1.27 | 57 | 72.86 | 1.31 |
| sa02 | 348 | 142 | 59.20 | 2.05 | 134 | 61.49 | 1.98 |
| rd73 | 479 | 136 | 71.61 | 2.00 | 111 | 76.83 | 2.02 |
| clip | 656 | 236 | 64.02 | 3.57 | 218 | 66.77 | 3.58 |
| duke2 | 708 | 331 | 53.25 | 5.75 | 326 | 53.95 | 6.05 |
| rd84 | 908 | 364 | 59.91 | 7.42 | 333 | 63.33 | 7.11 |
| C3540 | 1038 | 917 | 11.66 | 15.92 | 913 | 12.04 | 17.77 |
| rot | 1063 | 690 | 35.09 | 3.96 | 684 | 35.65 | 4.96 |
| misex3c | 1078 | 493 | 54.27 | 10.09 | 462 | 57.14 | 9.91 |
| i2c | 1306 | 864 | 33.84 | 9.88 | 853 | 34.69 | 12.29 |
| e64 | 1436 | 244 | 83.01 | 3.66 | 252 | 82.45 | 4.16 |
| **table5** | **1474** | **643** | **56.38** | **19.80** | **580** | **60.65** | **19.70** |
| dalu | 1740 | 920 | 47.13 | 15.14 | 911 | 47.64 | 17.14 |
| table3 | 1744 | 633 | 63.70 | 19.60 | 617 | 64.62 | 21.08 |
| C5315 | 1773 | 1278 | 27.92 | 13.57 | 1278 | 27.92 | 18.05 |
| C7552 | 2074 | 1554 | 25.07 | 19.47 | 1557 | 24.93 | 26.05 |
| apex1 | 2124 | 901 | 57.58 | 35.50 | 869 | 59.09 | 37.33 |
| ex4p | 2235 | 505 | 77.40 | 5.05 | 434 | 80.58 | 4.76 |
| apex3 | 2364 | 1132 | 52.12 | 71.04 | 1119 | 52.66 | 74.97 |
| i8 | 2673 | 946 | 64.61 | 25.69 | 939 | 64.87 | 27.91 |
| systemcdes | 3190 | 2423 | 24.04 | 76.82 | 2418 | 24.20 | 112.05 |
| ex1010 | 3267 | 1904 | 41.72 | 176.79 | 1864 | 42.94 | 186.82 |
| i10 | 3310 | 1689 | 48.97 | 43.84 | 1643 | 50.36 | 25.69 |
| apex5 | 3666 | 712 | 80.58 | 9.93 | 692 | 81.12 | 11.13 |
| alu4 | 5270 | 1448 | 72.52 | 100.95 | 1398 | 73.47 | 102.30 |
| ex5p | 7429 | 2044 | 72.49 | 442.16 | 2017 | 72.85 | 455.99 |
| misex3 | 8140 | 1395 | 82.86 | 100.05 | 1492 | 81.67 | 115.67 |
| apex2 | 8716 | 1198 | 86.26 | 69.61 | 1195 | 86.29 | 74.63 |
| systemcaes | 13054 | 10218 | 21.73 | 981.00 | 10093 | 22.68 | 1353.57 |
| ac97_ctrl | 14496 | 10310 | 28.88 | 376.72 | 10308 | 28.89 | 701.45 |
| mem_ctrl | 15641 | 7113 | 54.52 | 524.03 | 7113 | 54.52 | 892.34 |
| aes_core | 21513 | 18955 | 11.89 | 1156.61 | 18935 | 11.98 | 1783.47 |

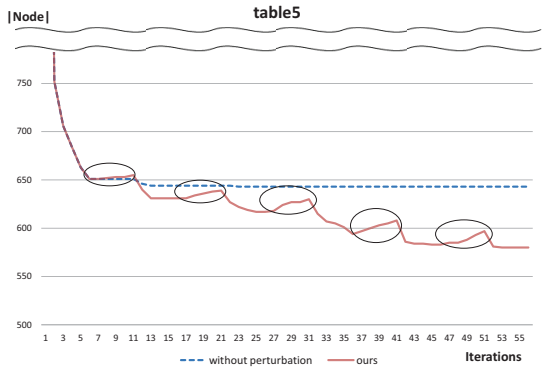


Figure 5. The comparison of node number in *table5* benchmark between the original optimization flow without perturbation and ours.

## IV. CONCLUSION AND FUTURE WORK

This paper presents a fast formal logic restructuring technique, EIC, through the process of injecting errors first and then correcting them. The EIC offers opportunities to the logic restructuring, and it can be served as a logic perturbation engine to further optimize the area of highly optimized circuits. Our future work is to integrate this work with other optimization engines to reach different objectives, e.g., power, delay, or reliability of the VLSI circuits.

## REFERENCES


[1] M. S. Abadir, et al., "Logic Design Verification via Test Generation," *IEEE TCAD*, vol. 7, pp. 138-148, Jan. 1988.

[2] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/~alanmi/abc/

[3] R. D. Blanton, et al., "Defect Modeling Using Fault Tuples," *IEEE TCAD*, vol. 25, pp. 2450-2464, Nov. 2006.

[4] M. Case, et al., "Merging Nodes Under Sequential Observability," in *Proc. DAC*, 2008, pp. 540-545.

[5] C.-W. Chang, et al., "Theory of Wire Addition and Removal in Combinational Boolean Networks," *Microelectronic Engineering*, vol. 84, pp. 229-243, Feb. 2007.

[6] C.-W. Chang, et al., "A New Reasoning Scheme for Efficient Redundancy Addition and Removal," *IEEE TCAD*, vol. 22, pp. 945-952, July. 2003.

[7] S.-C. Chang, et al., "Postlayout Logic Restructuring Using Alternative Wires," *IEEE TCAD*, vol. 16, pp. 587-596, June. 1997.

[8] S.-C. Chang, et al., "Perturb and Simplify: Multi-level Boolean Network Optimizer," *IEEE TCAD*, vol. 15, pp. 1494-1504, Dec. 1996.

[9] S.-C. Chang, et al., "Fast Boolean Optimization by Rewiring," in *Proc. ICCAD*, 1996, pp. 262-269.

[10] S.-C. Chang, et al., "Circuit Optimization by Rewiring," *IEEE TC*, vol. 48, pp. 962-970, Sep. 1999.

[11] Y.-C. Chen, et al., "An Improved Approach for Alternative Wires Identification," in *Proc. ICCD*, 2005, pp. 711-716.

[12] Y.-C. Chen, et al., "Fast Detection of Node Mergers Using Logic Implications," in *Proc. ICCAD*, 2009, pp. 785-788.

[13] Y.-C. Chen, et al., "Node Addition and Removal in the Presence of Don't Cares," in *Proc. DAC*, 2010, pp. 505-510.

[14] Y.-C. Chen, et al., "Fast Node Merging With Don't Cares Using Logic Implications," *IEEE TCAD*, vol. 29, pp. 1827-1832, Nov. 2010.

[15] Y.-C. Chen, et al., "Logic Restructuring Using Node Addition and Removal," *IEEE TCAD*, vol. 31, pp. 260-270, Feb. 2012.

[16] S.-C. Fu, "On Improved Scheme for Digital Circuit Rewiring and Application on Further Improving FPGA Technology Mapping, in *Proc. ASPDAC* 2009, pp.197-202.

[17] T. Kirkland, et al., "A Topological Search Algorithm for ATPG," in *Proc. DAC*, 1987, pp. 502-508.

[18] A. Kuehlmann, "Dynamic Transition Relation Simplification for Bounded Property Checking," in *Proc. ICCAD*, 2004, pp. 50-57.

[19] C.-C. Lin, et al., "Rewiring Using IRredundancy Removal and Addition," in *Proc. DATE*, 2009, pp. 324-327.

[20] W.-H. Lo, et al., "Improving Single-Pass Redundancy Addition and Removal with Inconsistent Assignments," in *Proc. VLSI-DAT*, 2006, pp. 175-178.

[21] A. Mishchenko, et al., "DAG-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis," in *Proc. DAC*, 2006, pp. 532-536.

[22] A. Mishchenko, et al., "Improvements to Combinational Equivalence Checking," in *Proc. ICCAD*, 2006, pp. 836-843.

[23] S. Plaza, et al., "Node Mergers in the Presence of Don't Cares," in *Proc. ASPDAC*, 2007, pp. 414-419.

[24] W.-C. Tang, et al., "A Quantitative Comparison and Analysis on Rewiring Techniques," in *Proc. Int. Conf. on ASIC*, 2003, pp. 242-245.

[25] A. Veneris, et al., "Design Rewiring Using ATPG," *IEEE TCAD*, vol. 21, pp.1469-1479, Dec. 2002.

[26] A. Veneris, et al., "Design Rewiring Based on Diagnosis Techniques," in *Proc. ASPDAC* 2001, pp. 479-484.

[27] C.-Y. Wang, et al., "Automatic Interconnection Rectification for SoC Design Verification Based on the Port Order Fault Model," *IEEE TCAD*, vol. 22, pp. 104-114, Jan. 2003.

[28] C.-Y. Wang, et al., "On Automatic-Verification Pattern Generation for SoC with Port-Order Fault Model," *IEEE TCAD*, vol. 21, pp. 466-479, April, 2002.

[29] C.-Y. Wang, et al., "An AVPG for SoC Design Verification with Port Order Fault Model," in *Proc. ISCAS*, pp. 259-262, 2001.

[30] S.-C. Wu, et al., "Novel Probabilistic Combinational Equivalence Checking," *IEEE TVLSI*, vol. 16, pp. 365-375, April, 2008.

[31] Y.-L. Wu, et al., "A Fast Graph-based Alternative Wiring Scheme for Boolean Networks," in *Proc. Int. VLSI Design Conf*., 2000, pp. 268-273.

[32] X.-Q. Yang, et al., "ECR:A Low Complexity Generalized Error Cancellation Rewiring Scheme," in *Proc. DAC*, 2010, pp. 511-516.

[33] X.-Q. Yang, et al., "Almost Every Wire Is Removable: A Modeling and Solution for Removing Any Circuit Wire" in *Proc. DATE*, 2012, pp. 1573-1578.

[34] Q. Zhu, et al., "SAT Sweeping with Local Observability Don't Cares," in *Proc. DAC*, 2006, pp. 229-234.

[35] http://iwls.org/iwls2005/benchmarks.html