Single-Pass Redundancy Addition And Removal

Chih-Wei (Jim) Chang and Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering,
University of California, Santa Barbara, CA 93106, USA

Abstract

Redundancy-addition-and-removal is a rewiring technique which for a given target wire w_t finds a redundant alternative wire w_a. Addition of w_a makes w_t redundant and hence removable without changing the overall circuit functionality. Incremental logic restructuring based on this technique has been used in many applications. However, the search for valid alternative wires requires trial-and-error redundancy testing of a potentially large set of candidate wires. In this paper, we study the fundamental theory behind this technique and propose a new reasoning scheme which directly identifies alternative wires without performing trial-and-error tests. Experimental results show up to 15 times speedup in comparison to the best techniques in literature.

1. Introduction

Redundancy Addition and Removal (RAR) is a powerful combinational logic restructuring technique introduced in [5]. First a redundant wire is added to the circuit. As a result, some previously irredundant wires become redundant and hence can be removed while the overall functionality of the circuit remains unchanged. The underlying engine is based on logic implication. Many applications of this technique have been developed in the past, including technology independent literal minimization [5][1][2], FPGA routing [4], and post-layout timing optimization [9][7]. The major advantage of the redundancy addition and removal technique is that only wires are reconnected while logic gates are preserved. This property is especially desirable in the deep-submicron age, when timing estimation obtained during logic synthesis can not be justified after placement and routing. Timing can be incrementally corrected through sequence of rewiring steps guided by the accurate physical information. Rewiring minimally perturbs layout and helps in achieving timing closure. There are two atomic operations which are essential for any RAR-type applications and are stated below:

Operation 1: Given a target wire w_t to be removed, which redundant wires, when added, will make w_t redundant?

This operation has been used to remove unroutable wires by adding routable alternatives[4], and in post-layout timing optimization to remove timing critical wires by adding non-critical alternatives[9].

Operation 2: After the addition of a redundant wire w_a, which wires become redundant and hence removable?

Operation 2 is primarily used in technology-independent literal minimization[5][1][2][11]. The algorithm searches for redundancy additions that can remove more than one wire and therefore minimize the literal count.

All the existing techniques use trial-and-error approaches when performing Operation 1 and Operation 2. First, a set of candidate wires is identified by finding the mandatory assignments (MA). Then, on each candidate wire, a redundancy test based on MA is performed to determine if it is indeed redundant[5][1][11]. To reduce the number of candidates, filters are proposed [2] to eliminate the impossible candidates. Redundancy test is performed on the remaining candidates. In [6], a set of candidate sinks of the added wire is first identified. For each sink, redundancy test is performed to derive redundant wires that can be added. In either approach, since the number of candidates can potentially be very large, the trial-and-error approaches pose a serious efficiency bottleneck for RAR-based optimization.

In this paper, we propose a single pass technique which can efficiently identify redundant wires without first building a candidate set. That is, our new technique identifies redundant wires in one pass without trial-and-error search. Tremendous speedup is observed using our approach.

2. Preliminaries

The *controlling value* of a gate *g*, denoted *cv*(*g*), is the logic value which when applied to any input of *g*, uniquely determines its output value regardless of logic values on other inputs. For example, when *type*(*g*) = AND, *cv*(*g*) = 0. Gates of XOR type do not have a controlling value since no single input can determine its output value. Gates of type BUF or INV have only single input and therefore do not have controlling values. The *non-controlling value* of *g*, denoted as *ncv*(*g*), is the opposite logic value of *cv*(*g*).

Iyer et. al. have proposed a novel fault-independent combinational redundancy identification (FIRE) technique[8]. The algorithm is based on a simple concept that a fault which requires a conflict as a necessary condition for its detection is undetectable and hence redundant. Faults are identified by the uncontrollability and unobservability analysis. Here, $\overline{0}$($\overline{1}$) denotes the status of a line that is uncontrollable for the value 0(1). A stuck-at-*v* fault on a wire w is redundant if w cannot assume the value $\overline{v}$ (or when w is uncontrollable for the value $\overline{v}$). Fig. 1 illustrates the propagation rules of uncontrollability indicators. Uncontrollability can propagate forward and backward, as shown by arrows. For example, the output of a NAND gate is $\overline{0}$ if at least one input is $\overline{1}$, and is $\overline{1}$ if all its inputs are $\overline{0}$. Similar rules apply to other gate types. Propagation of uncontrollability may result in some wires becoming unobservable. If a gate's input cannot be set to the gate's non-controlling value, all the other inputs become unobservable. For example, in Fig. 1, a = $\overline{1}$ implies that b is unobservable. The unobservability status, denoted ∗, propagates from a gate's output backward to all its inputs. This can be seen as b being unobservable makes both c and d unobservable as well. More details can be found in [8].

Let us consider the uncontrollability/unobservability propagation starting from a wire w. When w is set to $\overline{0}$ and propagation is launched, a set of wires may have value implied. For example, some wire w_i may get $\overline{1}$ implied. This means the stuck-at-0 fault (requires a 1 to activate) at w_i is untestable if the value at w cannot assume a 0. By contraposition, this fault is testable only if w is set to 0. Hence, the propagation of $\overline{0}$ from w results in a list of faults whose testability requires a logic 0 at w. We denote this list as F(w=$\overline{0}$). Similarly, F(w=$\overline{1}$) can be found by setting w to $\overline{1}$ and repeating the propagation. The intersection of F(w=$\overline{0}$) and F(w=$\overline{1}$) gives the set of faults that require w to be both 0 and 1 to be testable. This is clearly a conflicting situation and we can immediately conclude that faults in the intersection are untestable and hence are redundant. In [8], it has been shown that identifying redundancy using this method is much more efficient than Automatic Test Pattern Generation (ATPG) targeting each fault individually.

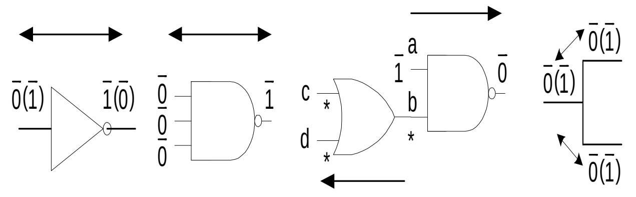

Fig. 1: Propagation of uncontrollability and unobservability [8]

3. Alternative Wires Identified Without Search

In this section, we discuss algorithms to implement Operation 1 and 2 in a non-trial-and-error manner. We start with Operation 2. As explained in the previous section, the existing RAR-type algorithms use filters to screen out wires that cannot possibly become redundant as a result of adding a particular redundant wire. For each of the wires that are not filtered out, ATPG is applied to verify redundancy. For efficiency reasons, ATPG only tries to justify consistency of the mandatory assignments. ATPG tests on candidate wires tend to collect repeatedly the same implications in a local region. In other words, information from individual ATPG runs are not used properly. This is a typical drawback of trial-and-error approaches.

3.1 Analysis of Operation 2

First two theorems developed in [3] will be presented as a theoretical background for our work. We use the D-notation[13]. D value on a wire represents the case when in a good circuit this wire has a value 1 and in the faulty circuit it has a value 0. $\overline{D}$ denotes the opposite case.

Definition 1: Mandatory assignments (MA)[3] of a target stuck-at-fault are the value assignments on nodes required for a test to exist and must be satisfied by any test vector. Forced Mandatory Assignment is the MA which when violated causes the target fault to be untestable. MAs obtained by setting side inputs of dominators to non-controlling values, MAs to activate the fault site, and MAs obtained by backward implication of the previous two MAs are forced.

Theorem 1: (Theorem 11 in [3]) If $w_a(n_s \rightarrow n_d)$ is an alternative wire of a wire w_t, and n_d is an AND (OR) gate, then n_s must have a mandatory assignment 0(1) for the w_t stuck-at-fault test.

Theorem 1 can be proved by contradiction. Suppose n_d is an AND gate and n_s has no mandatory assignment 0. This implies that there exists a test vector *t* for w_t stuck-at-fault such that *t* results in 1 at n_s. After adding w_a, *t* remains a test for w_t since 1 is the non-controlling value of AND. This contradicts the assumption that w_t is redundant after adding w_a.

Theorem 2: (Theorem 12 in [3]) If $w_a = n_s \rightarrow n_d$ is an alternative wire for the wire w_t, the AND(OR) gate n_d must have a forced mandatory assignment 1 or $\overline{D}$ (0 or D).

proof: From Theorem 1 we know that n_s has to have a mandatory assignment. If there is no mandatory assignment on n_d, a test for w_t stuck-at-fault before the addition is still a test after the addition. So n_d must have a mandatory assignment. By definition, the mandatory assignment on n_d must be forced as its violation leads to an untestable target fault.

Our major contribution is based on the observation that wires which become redundant after adding a redundant wire w_a must have a conflict of mandatory assignments on w_a. This is formally stated in the following theorem:

Theorem 3: Let w_t be an irredundant wire in a Boolean network *C* and w_a $(n_s \rightarrow n_d)$ is a wire which when added to *C*, is redundant. w_t becomes

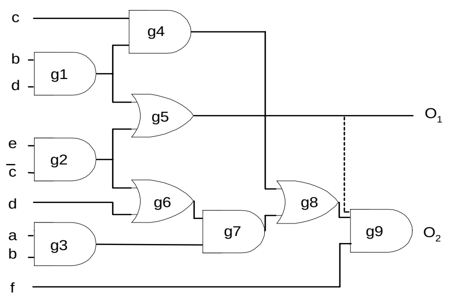

Fig. 2: Finding redundant wires after the addition of g5⟶g9.

redundant after adding w_a if and only if the redundancy test on w_t in the new circuit ($C \cup w_a$) leads to a conflict at w_a.

proof: The "if" part is trivial. The "only if" part is a direct consequence of Theorem 1 and Theorem 2. Without loss of generality, assume n_d is an AND gate. From Theorem 1 and Theorem 2, the stuck-at-fault test on w_t must result in a mandatory assignment 0 at n_s, and either 1 or $\overline{D}$ at n_d. A conflict obviously occurs when n_d is 1 because a 1 at the output of an AND gate means all its inputs must be 1, which conflicts with n_s=0 when w_a is added. When n_d has a mandatory assignment $\overline{D}$, it means n_d is on the fault propagation path and all the side inputs of n_d should be set to the non-controlling value 1. This also causes a conflict with n_s=0 when w_a is added. In summary, the stuck-at-fault test of w_t must create a conflict on w_a. This proves the "only if" part. QED.

Theorem 3 implies that instead of testing each candidate wire to decide redundancy, we simply start both $\overline{0}$ and $\overline{1}$ propagation on the added redundancy w_a. The intersection of $F(w_a=\overline{0})$ and $F(w_a=\overline{1})$ immediately gives a set of redundant faults caused by the addition. The major difference between our application and the original FIRE[8] is that the latter was used as a fault-independent way to identify redundancy: each of the wires in the netlist was targeted by FIRE to launch the uncontrollability/unobservability propagation. In our approach, we only launch FIRE propagation on the added redundancy since by Theorem 3, we know that all wires which are redundant as a result of adding w_a have to have conflict on w_a. This is a drastically different approach as compared to [11] where the whole circuit is tested for redundancy after each wire addition, or as compared to [5][2] where the candidate set is built and filtered.

For example, let us assume that a redundant wire $w_a(g_5 \rightarrow g_9)$ has been added to the circuit in Fig. 2. Based on Theorem 3, wires that can become redundant as a result of adding w_a will have implication conflicts on w_a for stuck-at-fault tests. So, we start FIRE by first assigning $\overline{0}$ on w_a and g_5 is implied with $\overline{0}$. Since there are no more direct implications, we apply recursive learning[10] on gate g_5. First, if g_1 is $\overline{0}$, the following values are implied: $\{b=\overline{0}, d=\overline{0}, g_6=\overline{0}, g_2=*\}$. This also implies the set of faults which are testable only if g_1 is 0. In particular, $\{g_1 \rightarrow g_4$ s-a-1, $g_6 \rightarrow g_7$ s-a-1$\}$ are in the set of implied faults. Second, if $\overline{0}$ is assigned to g_2 during recursive learning on g_5 the implied values are: $\{e=\overline{0}, c=\overline{1}, g_1 \rightarrow g_4=*\}$. Similarly, the faults $\{g_1 \rightarrow g_4$ s-a-1, $g_6 \rightarrow g_7$ s-a-1$\}$ are implied. Since $\{g_1 \rightarrow g_4$ s-a-1, $g_6 \rightarrow g_7$ s-a-1$\}$ are in the intersection of both implications, they are testable only if g_5 is 0. That is, $F(w_a=\overline{0}) = \{g_1 \rightarrow g_4$ s-a-1, $g_6 \rightarrow g_7$ s-a-1$\}$. Now we assign $\overline{1}$ on w_a. Since 1 is the non-controlling value of g_9, the inability of setting w_a to 1 means that faults which are only observable through g_9 are blocked. $\{g_1 \rightarrow g_4$ s-a-1, $g_6 \rightarrow g_7$ s-a-1$\}$ are among them. Now, it is clear that $\{g_1 \rightarrow g_4$ s-a-1, $g_6 \rightarrow g_7$ s-a-1$\}$ are in the intersection of $F(w_a=\overline{0})$ and $F(w_a=\overline{1})$, and therefore redundant after addition of w_a. The same faults are found by the existing trial-and-error methods.

3.2 The Quality of Results

Let w_t be an irredundant wire in a Boolean network *C* and w_a ($n_s \rightarrow n_d$) is a wire which when added to *C*, is redundant. Theorem 3 says that w_t is redundant after adding w_a if and only if the redundancy test on w_t in the new circuit ($C \cup w_a$) results in a conflict at w_a. We identify those w_t-wires that become redundant by performing FIRE on the added redundancy w_a. We ask if the solution returned by FIRE contains all the newly created redundancies? In other words, if the stuck-at-fault test on some wire w_t creates a conflict on w_a, we want to know if performing FIRE on w_a will always find this w_t?

Lemma 1: Let w_m and w_n be two wires in the circuit. f_m denotes the stuck-at-fault on w_m. Suppose that the stuck-at-fault test on w_m results in a conflict on w_n. Let Ω be the set of redundant faults found by performing FIRE on w_n. Then $f_m \in \Omega$.

proof: We prove by contradiction. Let f_m be a redundant fault not contained in Ω. Let $F(w_n=\bar{0})$ and $F(w_n=\bar{1})$ be the sets of faults that are collected by setting w_n to $\bar{0}$ and $\bar{1}$ respectively. By definition,

$$\Omega = F(w_n = \bar{0}) \cap F(w_n = \bar{1})$$

Since $f_m \notin \Omega$ by assumption, f_m is either not in $F(w_n=\bar{0})$ or not in $F(w_n=\bar{1})$. If f_m is not in $F(w_n=\bar{0})$, it means $w_n=0$ is not a necessary condition for f_m to be testable. Similarly, if f_m is not in $F(w_n=\bar{1})$, it means $w_n=1$ is not a necessary condition for f_m to be testable.

Since the stuck-at-fault test of w_m has a conflict on w_n, the following relations are true.

$$w_m \text{ is testable} \Rightarrow w_n = 1 \qquad \text{(EQ1)}$$

$$\text{and} \quad w_m \text{ is testable} \Rightarrow w_n = 0 \qquad \text{(EQ2)}$$

EQ1 and EQ2 contradict the previous conclusions that either $w_n=0$ or $w_n=1$ is not a necessary condition for f_m to be testable. We conclude that f_m has to be contained in Ω. QED.

The quality of results of our proposed approach is formally stated in the following theorem:

Theorem 4: Let w_t be an irredundant wire in the circuit and adding a redundant wire w_a makes w_t redundant. Applying FIRE on w_a finds all such w_t.

proof: By Theorem 3, the stuck-at-fault test of w_t must result in a conflict on w_a. Lemma 1 guarantees that all redundancies which have conflicts on w_a are always captured by FIRE performed on w_a. We conclude that the theorem is true. QED.

3.3 Analysis of Operation 1

Recall that Operation 1 finds an alternative wire w_a such that adding it to the circuit makes the target wire w_t redundant. The existing state-of-the-art solution is proposed by Chang et al in [3] using Theorem 1 and Theorem 2 as necessary conditions to build a candidate set. Then the members of the candidate set are tested to determine if they are indeed redundant. Let the original circuit be denoted by *C* and $w_t \in C$ be the target wire. We observe that if w_a is a redundant alternative wire for w_t in *C*, then w_t is also an alternative wire for w_a in the circuit $C + w_a - w_t$ (i.e. the circuit where w_a is added and w_t is removed). That is, w_a and w_t are mutually alternative wires. This leads to the following theorem:

Theorem 5: Let w_t be an irredundant wire in a Boolean network *C*, and w_a ($n_s \rightarrow n_d$) is a redundant wire which when added to *C* makes the target wire w_t redundant. The redundancy test on w_a in the new circuit ($C + w_a$) results in a conflict at w_t.

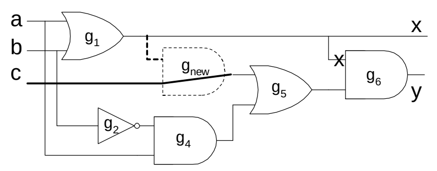

Fig. 3: Finding alternative wire for w_t(g1$\rightarrow$g6)

proof: The proof directly follows from the fact that not only w_a is an alternative wire of w_t, but w_t can also be viewed as an alternative wire of w_a. Based on Theorem 3, if w_t is an alternative wire to w_a, the redundancy test on w_a will result in a conflict at w_t. QED.

Theorem 6: $w_a(n_s \rightarrow n_d)$ is a redundant alternative wire of a wire w_t in the circuit C if and only if

(a) Let n_d be an AND(OR) gate. For the w_t stuck-at-fault test, n_s has a mandatory assignment 0(1) and n_d has a forced mandatory assignment 1 or $\bar{D}$ (0 or D).

(b) The stuck-at-fault test on w_a leads to a conflict on w_t.

proof: The proof is a direct consequence of *Theorem 1, Theorem 2,* and *Theorem 5*.

Theorem 6 gives a basis to implement Operation 1 in a non-trial-and-error manner. Recall that Theorem 1 and Theorem 2 give a set of wires which when added, make the target wire w_t redundant. These wires need to be tested to determine which ones among them are indeed redundant. However, with the help of Theorem 5, we launch FIRE on w_t and directly locate the set of wires which have conflicts on w_t. The intersection of these two sets gives exactly the alternative wires that can replace w_t. Operation 1 can now be performed without any trial-and-error search.

We formally prove the quality of result using our approach for Operation 1 in the following theorem:

Theorem 7: Let S be the set of new wires satisfying condition (a) of Theorem 6. Let T be the set of new wires obtained by the application of FIRE on w_t. Let w_a be a redundant alternative wire of w_t. Then $w_a \in S \cap T$.

proof: We prove by contradiction. Assume $w_a \notin S \cap T$. That is, $w_a \notin S$ or $w_a \notin T$. By Theorem 1 and Theorem 2, if w_a is an alternative wire, w_a must be contained in S. By Theorem 4, w_a must be contained in T. These contradict the assumption $w_a \notin S \cap T$. We conclude that $w_a \in S \cap T$. QED.

4. Complexity Analysis

After performing a stuck-at-fault test on the target wire w_t, a set of mandatory assignments are derived. Let M denotes the time complexity of performing the stuck-at-fault test. The mandatory assignments for the test include fault activation value assignment on the source of the target wire and non-controlling value assignments on the side inputs of each dominator. Using only mandatory assignments to derive the test, M is linear in the number of dominators of the target fault. We observe that even though the number of dominators could potentially increase as the overall circuit size increases, M is relatively independent of the circuit size since a linear increase in circuit size does not necessary linearly increase the number of dominators. Because recursive learning can be used for the test, M is also a function of the user specified recursive learning depth. In [4], based on Theorem 1 and Theorem 2, wires which satisfy the necessary conditions are considered to be possible candidate wires to be added to the circuit. Assume K such candidates exist and each of them needs to be tested to determine if it is indeed redundant.

TABLE 1. Experimental Results

	#wire tested	CPU[6]	ratio	CPU[2]	ratio	RAM-FIRE (Ours)	ratio
C432	483	47.4	7.4	57.8	9.0	6.4	1.0
C499	936	151.1	11.8	173.4	13.5	12.8	1.0
C880	792	42.4	7.4	31.1	5.5	5.7	1.0
C1355	1044	289.8	14.7	330.6	16.8	19.7	1.0
C1908	990	461.6	14.1	874.6	26.7	32.8	1.0
C2670	1589	342.8	15.0	864.2	37.7	22.9	1.0
C5315	3846	722.4	6.3	575.0	5.1	113.8	1.0
C6288	5451	2736.6	13.6	1944.8	9.7	200.6	1.0
C7552	5042	4135.7	12.8	3674.5	11.4	323.1	1.0
average			11.5		15.0		1.0

The time complexity of finding alternative wires for w_t is therefore M+KM=(K+1)M. This is the time complexity of the approach proposed in [2].

A different approach described by Entrena et al.[6] is based on the property that only the gates with forced mandatory assignments can possibly be sinks of added wires. Given a gate g, the effort to find all possible wires that are redundant when added to g is the same as that of one stuck-at-fault test. This effort is denoted by M. Assume there are T gates that have forced mandatory assignments. The overall time complexity is then M+TM=(T+1)M.

In our proposed approach, after the initial stuck-at-fault test on w_t, we only need two implications: $\overline{0}$ and $\overline{1}$ from w_t. We use M' to denote the time effort of one such implication. Our implication involves three values: $\overline{0}$, $\overline{1}$, and * (unobservability) which potentially may be more expensive than M. Overall, the new approach has complexity M+2M'. In practice M' is very close to M.

The complexity of the two previously-proposed approaches involve variables K or T. These two parameters are difficult to determine and vary from one target wire to the other. In our approach, just three implications, and always three, are enough to determine all possible alternative wires.

5. Experimental Results

Our prototype tool RAMFIRE (Redundancy Addition and reMoval using Fault-Independent Redundancy idEntification) has been implemented in C++. To measure the benefits of using the proposed one-pass alternative wire identification scheme, we have performed experiments on some publicly available benchmarks from LGSynth93[12]. Especially, we were interested in Operation 1, as it is the foundation of many applications in timing optimization. We did not focus on any specific application, but rather on the fundamental building block of all applications. We performed the following experiment: Each wire in the circuit was targeted once to find its redundant alternatives and the runtime was record. As a comparison, we also implemented the two techniques mentioned in the previous section. Table 1 shows the result. Column 1 lists the names of benchmark circuits. Column 2 lists the total number of wires tested for alternatives in each benchmark. Column 3 shows the run time for the approach proposed in [6] with recursive learning depth set to 1. Column 5 shows the run time using the necessary conditions to build a candidate set followed by redundancy test[2]. The runtime was recorded without using recursive learning because stuck-at-fault test has the advantage of requiring less learning. Column 7 shows the runtime of our approach with recursive learning depth set to 1. This setting leads to the same set of alternative wires found by all three techniques. Column 4, 6, 8 show the ratio normalized with respect to our approach. Runtime is recorded in seconds on an AMD Athlon 650MHz processor.

The results show that our technique achieves a significant runtime improvement over the other two techniques. Compared with [6], we obtain speedup ranging from 6.3 to 15 times. Compared with [2], speedup is as high as 37.7 times. On average, RAMFIRE outperforms these two approaches in terms of speedup by more than an order of magnitude. Any rewiring application based on RAMFIRE would greatly benefit from this speedup.

6. Conclusion

In this paper, we propose RAMFIRE, a general reasoning scheme for redundancy addition and removal. We studied two fundamental problems: 1) Given a target wire, which are the redundant alternatives that can replace the target wire? 2) When a redundant wire is added, which are the previously irredundant wires that become redundant? Efficient one-pass algorithms are proposed to solve these problems. Compared with existing techniques, no time-consuming trial-and-error searches are needed. This provides tremendous runtime savings, as more than 10 times speedup is observed on average over a wide range of benchmarks with no degradation in quality of results. Theoretical complexity measure of the proposed method is also studied and compared to complexities of the two previous approaches.

Acknowledgment

This work is sponsored by Semiconductor Research Corporation under grant 98-DJ-619. We would like to thank Dr. Ric Chung-Yang Huang of Verplex Systems for providing us the RAMBAS framework[7].

References

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

[2] S.-C. Chang, L.P.P.P. Van Ginneken, and M. Marek-Sadowska, “Circuit Optimization by Rewiring”, *IEEE Transactions on Computers*, vol.48, p.962-970, Sep. 1999.

[3] S.-C. Chang, L.P.P.P. Van Ginneken, and M. Marek-Sadowska, “Fast Boolean Optimization by Rewiring”, *Proc. of Intl. Conf. on Computer-Aided Design*, Nov. 1996, pp. 262-269

[4] S. -C. Chang, K.-T. Cheng, N. S. Woo, and M. Marek-Sadowska, “Post-layout Logic Restructuring Using Alternative Wires”, I*EEE Trans. on Computer-Aided Design*, vol. 6, pp. 587-596, June 1997

[5] L. A. Entrena and K. -T. Cheng, “Combinational and Sequential Logic Optimization by Redundancy Addition and Removal”, I*EEE Trans. on Computer-Aided Design*, pp. 909-916, July 1995

[6] L. A. Entrena, J. A. Espejo, E. Olias, and J. Uceda, “Timing Optimization by an Improved Redundancy Addition and Removal Technique”, in *Proc. of EURO-DAC*, 1996, pp. 342-347.

[7] R. C.-Y. Huang, Y. Wang, and K.-T. Cheng, “Libra - A Library-Independent Framework for Post-Layout Performance Optimization”, *in Proc. of Intl. Symposium on Physical Design*, April 1998, pp. 135-140.

[8] M. A. Iyer and M. Abramovici, “FIRE: A Fault-Independent Combinational Redundancy Identification Algorithm”, *IEEE Trans. on VLSI*, vol. 4, no. 2, pp. 295-301, June. 1996.

[9] Y. -M. Jiang, A. Krstic, K. -T. Cheng, and M. Marek-Sadowska, “Post-Layout Logic Restructuring for Performance Optimization”, in *Proc. of Design Automation Conf.*, 1997, pp. 662-665,

[10]W. Kunz.and D. Pradhan, “Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems: Test, Verification and Optimization”, *IEEE Trans. on Computer-Aided Design*, pp 1143-1158, Sep. 1994.

[11]W. Kunz, D. Stoffel, and P. R. Menon, “Logic Optimization and Equivalence Checking by Implication Analysis”, *IEEE Trans. on Computer-Aided Design*, pp. 266-281, Mar. 1997

[12]K. McElvain, “LGSynth93 Benchmark Set: Version 4.0”, http://zodiac.cbl.ncsu.edu/CBL_Docs/lgs93.html

[13]J. P. Roth, “Diagnosis of automata failures: A calculus and a method,” in *IBM J. Res. Develop.*, vol 10, pp. 278-291, July 1966.