

# Perturb and Simplify: Multi-level Boolean Network Optimizer

Shih-Chieh Chang, and Malgorzata Marek-Sadowska

Electrical and Computer Engineering Department,

University of California Santa Barbara, CA 93106

## Abstract

In this paper, we discuss the problem of optimizing a multi-level logic combinational Boolean network. Our techniques apply a sequence of local perturbations and modifications of the network which are guided by the automatic test pattern generation ATPG based reasoning. In particular, we propose several new ways in which one or more redundant gates or wires can be added to a network. We show how to identify gates which are good candidates for local functionality change. Furthermore, we discuss the problem of adding and removing two wires, none of which alone is redundant, but when jointly added/removed they do not affect functionality of the network. We also address the problem of efficient redundancy computation which allows to eliminate many unnecessary redundancy tests. We have performed experiments on MCNC benchmarks and compared the results to those of misII[4] and RAMBO[6]. Experimental results are very encouraging.

## 1 Introduction

In this paper, we discuss the problem of multi-level logic optimization for a combinational network. Previous multi-level optimization approaches can be categorized into two classes. The first class locally collapses and optimizes a circuit using techniques like factorization, decomposition, kernel extraction, cube extraction, etc. (e.g.: misII[4]). The second class introduces a perturbation, usually in a form adding wires, to a network which results in potential removal of some redundant gates or wires (e.g.: [2],[12], and RAMBO [8]). Our proposed approach falls into the second class.

Among the second class, RAMBO [6], [8] proposed an efficient automatic test pattern generation(ATPG) based approach to optimize a network. The idea was that the perturbation-simplification process of network optimization can be viewed as redundancy addition and removal, which can be efficiently computed using ATPG techniques [10] [13]. In RAMBO, a heuristic of adding one redundant wire at a time and removing redundant wires caused by such a perturbation was proposed. In [5] we applied the ideas of ATPG guided wire additions and removals to alleviate FPGA routing.

In this paper we carry further the idea of perturbing-simplifying a circuit applying ATPG techniques. In particular, we propose several new ways in which one or more redundant gates or wires can be added to the circuit. We also show how to identify gates which are good candidates for local functionality change. In addition, we discuss the problem of adding and removing two wires, none of which alone is redundant,

but when jointly added/removed they do not affect the functionality of a network. We also address the problem of efficient redundancy computation which allows us to eliminate many unnecessary wire redundancy tests.

## 2 Redundancy identification procedure

In a combinational circuit, a wire is redundant if and only if the corresponding stuck-at fault is untestable. We review an approach [13] of identifying redundant wires using the concept mandatory assignments.

The *absolute dominators* (dominators) [10] of a wire  $W$  is a set of gates  $G$  such that all paths from the wire  $W$  to any primary output have to pass through all the gates in  $G$ . An input to a gate has a *controlling* value if it determines the output of the gate regardless of the other inputs. The inverse of the controlling value is called a *non-controlling* value. A gate is in the *transitive fanin* (*fanout*) of a wire, if there is a path from the gate to the wire (from the wire to the gate).

Consider the absolute dominators of a wire  $W$ . The *side inputs* of the absolute dominators are their inputs not in the transitive fanout of the wire  $W$ . A test pattern for a stuck-at fault on wire  $W$  must set all the side inputs of the absolute dominators of  $W$  to their non-controlling values.

*Mandatory assignments* are the unique values which must be present at certain nodes for a test to exist. For a given stuck-at fault  $f$ , the set of mandatory assignments, denoted as  $SMA(f)$ , can be computed using the 9 value implication approach [10] [13]. If the mandatory assignments implied by a stuck-at fault on a wire can not be *consistently justified*, the stuck-at fault is *untestable* and therefore the wire is *redundant*.

## 3 Redundancy addition and removal

In this section, we discuss how to make a wire redundant by adding to the network another *redundant* wire or gate.

### 3.1 Wire substitution procedure

Suppose the objective is to remove a wire  $w_r$  from a network. We attempt to add a redundant wire(gate)  $w_a$  such that the originally irredundant wire  $w_r$  becomes redundant. Since  $w_r$  is irredundant, the  $SMA(w_r, \text{stuck-at fault})$  is consistent. If the  $SMA(w_r, \text{stuck-at fault})$  is inconsistent under the change (adding  $w_a$ ), the stuck-at fault becomes untestable and we can conclude  $w_r$  is redundant. In the following, we show the wire substitution procedure: adding a redundant wire(gate) to make an irredundant wire redundant.

First the SMA of the target wire  $w_r$  stuck-at fault is calculated. Then, a set of candidate connections is identified. Each candidate connection when added to the circuit causes inconsistency of the SMA( $w_r$  stuck-at fault) and thus makes the stuck-at fault untestable. However, adding such a candidate connection may change the circuit's behavior. Therefore, a redundancy check is needed to verify whether a candidate connection is redundant or not. If a candidate connection is redundant, it can be added to remove the target wire  $w_r$ .

Consider the circuit in Fig. 1 [8]. Let  $g_{I \rightarrow g_4}$  be the target wire to be removed.  $\text{SMA}(g_{I \rightarrow g_4} \text{ s-a-1}) = \{c=1, g_1=0, g_5=0, g_2=0, f=1\}$ . Note that  $g_5$  is outside transitive fanout of  $g_{I \rightarrow g_4}$  and has a mandatory assignment 0. Since  $g_9$  is an absolute dominator of  $g_{I \rightarrow g_4}$ , if we connect  $g_5$  to  $g_9$  (the dotted wire in Fig. 1),  $g_5$  must have a mandatory assignment of 1 for  $g_{I \rightarrow g_4}$  s-a-1 fault. This is inconsistent with the original  $g_5=0$ , and therefore, the presence of  $g_5 \rightarrow g_9$  makes  $g_{I \rightarrow g_4}$  redundant. We then choose  $g_5 \rightarrow g_9$  as a candidate connection. Finally, we check if  $g_5 \rightarrow g_9$  is redundant by examining the SMA( $g_5 \rightarrow g_9$  s-a-1). The SMA( $g_5 \rightarrow g_9$  s-a-1) is inconsistent. Therefore, we can add the wire  $g_5 \rightarrow g_9$  and remove the wire  $g_{I \rightarrow g_4}$ .

The above example shows that a good candidate connection can be a wire between a gate with a mandatory assignment ( $g_5=0$ ) and a dominator ( $g_9$ ). We generalize this observation as follows.

### 3.2 Only two among all transformations are necessary

We define a wire  $w_f$  as a *fault propagating wire* if there is a path from the target wire  $w_r$  under stuck-at fault test to the wire  $w_f$ . The patterns of all possible logic transformations that we consider in this section are: (1) the source gate  $g_s$  is in SMA( $w_r$  stuck-at fault) but not in the transitive fanout of  $w_r$ , (2) the destination gate  $g_d$  is a dominator of the target wire  $w_r$  (for simplicity, only consider when  $g_d$  is a 2-input AND/OR gate), and (3) the 2-input gate  $g_d$  is replaced by a certain 3-input gate, whose inputs are  $g_1, g_2$ , and  $g_s$  (the dotted box in Fig. 5). Since our goal is to remove the target wire  $w_r$ , we term a logic transformation *adequate* if, after the transformation, SMA( $w_r$  stuck-at fault) becomes inconsistent. For example, consider the Type 1 transformation in Fig. 7. Let  $g_I$  be the fault propagating wire and  $g_s$  be a wire with mandatory assignment 0 outside the transitive fanout of  $w_r$ . Because  $g_d$  is a dominator of  $w_r$ , the added gate  $g_n$  is also a dominator of  $w_r$ . Since  $g_s$  is a side input to the dominator  $g_n$ ,  $g_s$  must be assigned a non-controlling value 1, which causes a conflict with the original mandatory assignment of 0. Therefore, this Type 1 transformation is an adequate logic transformation.

We can view all possible logic transformations as replacing  $g_d$  by a 3-input gate fed by  $g_s, g_1$  and  $g_2$  (in Fig. 5). We enumerated all possible 256 3-input functions, only sixteen of them are adequate logic transformations, which have the desired property of making SMA( $w_r$  stuck-at fault) become inconsistent. In Fig. 6,7. we list two of these sixteen 3-input functions, which we term Type 0 and Type 1 transformations.

In the subsequent discussion, we show that the other fourteen adequate transformations are unnecessary when Type 0 and Type 1 transformations are performed. For simplicity, in Fig. 8, we list another possible adequate transformation, which we term Type 2 transformation, and prove that it is not necessary to consider it independently. We omit the similar discussion when  $g_d$  is an OR gate, or when  $g_s$  has a mandatory assignment 1, and the cases of the remaining thirteen adequate logic transformations.

The transformations in Fig. 6-8 guarantee only that an addition of the new wire (dotted wire there) will cause inconsistency in the SMA( $w_r$  stuck-at fault) and therefore make the target wire  $w_r$  redundant. Still, it is essential to verify if the added wire itself is redundant. The following theorem guarantees that if the added wire/gate in the Type 1 transformation are irredundant, then the added wire/gate in the Type 2 transformation, when applied to the same  $g_s$  and  $g_d$ , are also irredundant. Therefore, the Type 2 transformation is unnecessary when the Type 1 transformation is performed.

**Theorem 1.** Consider the transformation of Type 1 and Type 2 in Fig. 6,7 applied to the same  $g_s$  and  $g_d$ . If a new wire added to the network as suggested by the Type 1 transformation is irredundant, then so is the wire added by the Type 2 transformation.

### 3.3 Further exclusion of unnecessary redundancy checks

To make a target wire  $w_r$  redundant, the discussion in the last section tells us that we only need to consider the Type 0 and Type 1 transformations. In the following, we show that, depending on the mandatory assignment of the dominator gate  $g_d$ , the added wire/gate in either Type 0 or Type 1 transformations is always irredundant. Therefore we can prune the space of redundancy checks on these a priori known irredundant transformations.

Let  $w_r$  be a wire under stuck-at 0(1) test. A *faulty* circuit is the circuit in which  $w_r$  is replaced by a constant 0(1). As we noted earlier, a wire is irredundant if the corresponding stuck-at fault is testable. Therefore, if we can find an input vector which can differentiate between the faulty and the good networks, the corresponding stuck-at fault is testable and the wire is irredundant. If no such a test vector exists, then the wire under stuck-at fault test is redundant. We define  $b_g(g_i)$  to be a binary value at the output of gate  $g_i$  in the *good* network, and  $b_f(g_i)$  be the value at  $g_i$  in the *faulty* network, both under the same excitation vector applied to the primary inputs. For example, considering  $g_{I \rightarrow g_4}$  s-a-1 in Fig. 1, when a vector  $(a,b,c,d,e,f) = (0,0,1,0,0,1)$  is applied, we have  $b_g(g_4)=0$ ,  $b_f(g_4)=1$ ,  $b_g(g_8)=0$ ,  $b_f(g_8)=1$ ,  $b_g(g_9)=0$ , and  $b_f(g_9)=1$ .

**Theorem 2.** Let  $w_r$  be an irredundant wire in the network. Consider the transformations of Type 0 and Type 1 applied to the network to make  $w_r$  stuck-at fault untestable. Let  $g_d$  be a dominator of  $w_r$ . For  $w_r$  stuck-at fault, if there exists a test vector  $v_t$  which causes that  $b_g(g_d)=1$  and  $b_f(g_d)=0$ , then, the

candidate connection suggested by the Type 0 transformation is irredundant. When test vector  $v_t$  causes  $b_g(g_d)=0$  and  $b_f(g_d)=1$ , then the candidate connection suggested by the Type 1 transformation is irredundant.

**Corollary 1:** Suppose that all test vectors for  $w_r$  stuck-at fault cause  $b_g(g_d)=0$  and  $b_f(g_d)=1$ , then only the Type 0 transformation should be tried. If  $b_g(g_d)=1$  and  $b_f(g_d)=0$ , then, only the Type 1 transformation should be tried. If some test vectors cause  $b_g(g_d)=0$  and  $b_f(g_d)=1$  and others cause  $b_g(g_d)=1$  and  $b_f(g_d)=0$ , then both Type 0 and Type 1 should not be attempted.

For example, after performing a s-a-0 test on the wire  $g_2 \rightarrow g_6$  in Fig. 1, we have  $SMA=\{g_1=0, c=0, \dots\}$ . Since  $g_9$  is a dominator and  $b_g(g_9)=1$  and  $b_f(g_9)=0$ . Therefore, from the corollary 1, we conclude that the Type 0 transformation of  $g_1 \rightarrow g_9$ ,  $c \rightarrow g_9$  are irredundant and need not be tested. Same situation can be derived for the Type 0 transformation of  $g_1 \rightarrow g_7$ ,  $c \rightarrow g_7$ .

The intuition behind theorem 2 is as follows. The test vectors  $V_t$  for  $w_r$  stuck-at fault are the input vectors that can distinguish between the good circuit and the faulty circuit. If a wire  $w_a$  could indeed replace the wire  $w_r$ , then adding  $w_a$  should at least be able to compensate the discrepancies produced by  $V_t$  between the good and the faulty networks. The above theorem says that if a transformation can not correct discrepancies for one test vector, then it must be irredundant.

#### 4 Multiple-wire addition and gate function substitution

In this section, we extend the idea of adding *one* wire(gate) to adding *multiple* wires (gates). In addition, for the purpose of removing a wire, we also allow gates to change their functionality.

##### 4.1 Multiple-wire addition

In a Boolean network, there exist wires which when deleted may trigger a sequence of other reductions. For example, when  $g_6 \rightarrow g_7$  dotted in Fig. 2 is removed, the 3-input gate  $g_6$  can be also removed. This in turn leaves the gate  $g_7$  with a single input, therefore a direct connection  $g_3 \rightarrow g_8$  is possible and  $g_7$  can be deleted. If deleting a wire can result in the removal of more than 2 wires or gates from the network, we refer to such a wire as a *large\_reduction* wire. When optimizing a circuit, we give higher priority to removing large\_reduction wires. In case that adding one redundant wire(gate) can not remove a large\_reduction wire, we may add more than one wires (gates) to delete the wire in question. For example, in Fig. 2, we add a 2-input gate  $g_m$  and a wire  $g_m \rightarrow g_9$  to remove the large\_reduction wire  $g_6 \rightarrow g_7$ . However, arbitrarily adding many wires as in [5] to remove a large\_reduction wire is computationally expensive. We have developed an efficient approach that limits the search space and that still has much more power in comparison to the one-wire addition.

Our basic philosophy of adding multiple wires is to cause an originally irredundant candidate connection to become

redundant. Suppose we wish to remove a large\_reduction wire, and all the one-wire candidate connections are irredundant. Then, we consider a possibility of adding multiple wires. The procedure is as follows. We compute and store the  $SMA(w_r \text{ stuck-at fault})$ . Then, we pick a candidate connection  $(g_{s1}, g_d, \text{type})$  and compute the  $SMA(\text{candidate wire stuck-at fault})$ . After that, we look for another gate, call it  $g_{s2}$ , such that it is in both the  $SMA(w_r \text{ stuck-at fault})$  and the  $SMA(g_{s1} \rightarrow g_d \text{ candidate wire stuck-at fault})$  but has different mandatory assignments. In the example of Fig. 2, the gate  $d$  appears with the assignment 0 in the  $SMA(g_6 \rightarrow g_7 \text{ s-a-1})$  and with assignment 1 in the  $SMA(g_5 \rightarrow g_9 \text{ s-a-1})$  so the gate  $d$  is our  $g_{s2}$ . Finally, after finding the gate  $g_{s2}$ , we add a gate  $g_m$  and a wire  $g_m \rightarrow g_d$  where  $g_d$  is the dominator of the candidate connection, using the following rule. The gate  $g_m$  is an OR(AND) gate, if the gate  $g_d$  is an AND(OR) gate. The inputs to  $g_m$  are  $g_{s1}$  and  $g_{s2}$ . If  $g_{s1}=1(0)$  in  $SMA(w_r \text{ stuck-at fault})$  and  $g_m$  is an OR(AND) gate, we invert the input of  $g_{s1}$  to  $g_m$ . The same rule is applied to  $g_{s2}$ . In our example,  $g_m$  is an OR gate and  $g_{s1}=g_5$  and  $g_{s2}=d$ . Both  $d=0$  and  $g_5=0$  in the  $SMA(g_6 \rightarrow g_7 \text{ s-a-1})$  so we do not invert the input phase for  $d$  and  $g_5$  to  $g_m$ .

**Theorem 3.** If in the above procedure, a gate  $g_{s2}$  can be found then the network modification is valid, i.e.  $w_r$  can be deleted and the functionality of the network does not change.

##### 4.2 Changing the gate's functionality

In this section, we discuss how to change a gate's function to remove a particular wire. For example, we can change the gate  $g_5$  (highlighted in Fig. 3) from an AND to an XNOR without changing the circuit functionality. After changing  $g_5$  to an XNOR gate, the wire  $g_6 \rightarrow g_7$  becomes redundant. Two issues need to be addressed, namely, how to check if a given gate can change its functionality, and which gate should be changed to make a target wire redundant.

Consider an AND gate  $g_x(g_{i1}, g_{i2})$  with two inputs  $g_{i1}$  and  $g_{i2}$ . If the output of  $g_x$  is a don't care when  $(g_{i1}, g_{i2})=(0, 0)$ , the function of  $g_x(g_{i1}, g_{i2})$  can change from AND to XNOR. We first show a procedure to verify whether  $(v_1, v_2, \dots, v_n)$  is a don't care minterm for a gate  $g_x(g_{i1}, g_{i2}, \dots, g_{in})$ . The procedure is based on checking consistency of a certain SMA.

The procedure to verify whether a minterm  $(v_1, v_2, \dots, v_n)$  is a don't care to  $g_x(g_{i1}, g_{i2}, \dots, g_{in})$  first sets  $g_{i1}=v_1, g_{i2}=v_2, \dots, g_{in}=v_n$  and includes this assignment in a SMA. Then, we treat the output of  $g_x$  as stuck-at the value produced by that minterm and compute the appropriate SMA. In the following theorem, we show that if the  $SMA(f)$  is inconsistent, the minterm is a don't care.

**Theorem 4** Consider the SMA induced by setting the gate's inputs to a minterm in question and treating the gate's output as stuck-at the value produced by the minterm. If this SMA cannot be consistently justified, then the minterm  $(g_{i1}, g_{i2}, \dots, g_{in})$  is a don't care of the  $g_x(g_{i1}, g_{i2}, \dots, g_{in})$  embedded in the network.

Theorem 4 suggests how to verify quickly if a gate can switch its functionality without affecting the network's behavior.

The SMA computed to justify particular conditions in the network depend on gates' functionality. We may change some gates' functionality to achieve SMA's inconsistency, and therefore achieve redundancy. In Fig. 3, the SMA( $g_6 \rightarrow g_7$  s-a-1) =  $\{g_1=0, g_2=0, g_5=0, \dots\}$ . Changing  $g_5$  from an AND to an XNOR causes that  $g_5=1$ . But on the other hand  $g_5=0$  as a side input of a dominator. Therefore, we conclude that if  $g_5$  is changed to an XNOR gate, then SMA( $g_6 \rightarrow g_7$  s-a-1) is inconsistent and  $g_6 \rightarrow g_7$  is redundant.

## 5 Simultaneous addition and removal of two wires (gates)

We say that two wires ( $w_a, w_b$ ) are simultaneously redundant if each wire is irredundant but simultaneously adding/removing  $w_a$  and adding/removing  $w_b$  does not change the circuit's functionality.

Intuitively, the existence of two simultaneously redundant wires is not obvious. It can be explained as follows. Let  $V_f(w_r$  stuck-at fault) denote all the input vectors that can test the  $w_r$  stuck-at fault. That is any vector in  $V_f(w_r$  stuck-at fault) can distinguish the original (good) circuit from the faulty one. Consider another wire  $w_a$ . If both sets,  $V_f(w_r$  stuck-at fault) and  $V_f(w_a$  stuck-at fault) are the same, and the faults propagate through the same XOR(XNOR) gate, they cancel out at the output of XOR(XNOR) gate.

The scenario in which the two simultaneously redundant wires transform is used, is best explained using example in Fig. 4. Applying the transformation of two simultaneously redundant wires, we can replace the wire  $g_6 \rightarrow g_7$  by another wire  $g_5 \rightarrow g_7$ . As a result the 2-input gate  $g_6$  can be removed. The following shows a stuck-at fault test that can verify if two wires are simultaneously redundant.

**Theorem 5.** Let  $g_r$  be a fanin of XOR(XNOR) gate  $g_x$  and let  $g_a$  be another gate. If  $(g_a, g_r) = (0,1)$  and  $(g_a, g_r) = (1,0)$  are don't cares in the network, we can replace  $g_r \rightarrow g_x$  by  $g_a \rightarrow g_x$ .

Note that when a dominator  $g_d$  is an XOR(XNOR) gate, the network transformations in Fig. 6-8 are no longer valid and can not be used to remove a particular wire. It is because some test vectors cause  $g_d$  to have  $b_g/b_f = 1/0$ , and others cause  $g_d$  to have  $b_g/b_f = 0/1$ . According to the Corollary 1, none of the transformations should be applied. Therefore, the redundancy addition and removal technique can not be applied when a dominator is an XOR(XNOR) gate.

## 6 The algorithm

We have implemented the transformations described in the previous sections. The overall algorithm is shown in Fig 9. The subroutine *Add\_one\_gate\_to\_remove\_other\_wires()*, which besides adding a wire also adds gates, is an extended algorithm of RAMBO [6]. For each gate  $g_d$  in the circuit, we

try to apply the transformations in Fig. 6,7 to remove wires that are dominated by  $g_d$ . The second subroutine, *Remove\_large\_reduction\_wires()*, identifies the large\_reduction wires and attempts to remove them using more expensive techniques like adding multiple gates and changing gate's functionality. Finally, the last subroutine, *Perturb\_circuit\_more()*, perturbs the circuit to jump out of a locally minimal solution before the next iteration. In *Perturb\_circuit\_more()*, for each wire  $w_i$  in the circuit, we attempt to replace it with another wire or gate.

## 7 Experimental results

In this section, we present experimental results for combinatorial benchmark circuits. We implemented the algorithm in Fig. 9 and we choose k\_iteration to be 2. In our experiments, the optimization objective is to reduce the number of two-input gates. We compare our results with misII [4] and RAMBO[6]. Note that when our operation involves *adding a wire* to an AND (OR) gate, we actually *added an extra AND(OR) gate* to the circuit.

TABLE 1 shows the results for some of the MCNC combinatorial benchmark circuits. misII results were obtained as follows. We use *script.boolean* provided by misII (for consistency, we don't use *script.rugged* because some examples can not be run due to space/time limitation). Then, we map (using the *map* command in misII) the circuit into a circuit with general 2-input gates shown in second column of TABLE 1.. The initial circuits of ours and RAMBO are obtained by running *script.algebra* and then mapping into 2-input gates. We run both RAMBO and our algorithm to optimize the circuit. Since the output of RAMBO may contain gates with more than 2 inputs, we also decompose the result of RAMBO into 2-input gates. In the fourth, fifth, and sixth columns, we show the results of running misII, RAMBO, and our algorithm, respectively. All these three columns are described in terms of the number of 2-input gates and the number of literals. As shown in the table, the results we obtained are, on the average for the listing examples, 19% better than misII and 16% better than RAMBO in terms of number of 2-input gates. Except the superior results of our algorithm, our memory requirement is very inexpensive. For example, C7552 needs only 6 Mbytes. All our results have been verified using the circuit verification command in misII. The experiment was performed on DEC 5000. Note that the RAMBO's results are 4% better than results shown if RAMBO's input circuits are *script.boolean* optimized **first**.

## 8 Conclusions

In this paper, we have proposed several new ways to add one or more redundant gates or wires to remove other gates or wires from the network. We show how to identify gates which are good candidates for local functionality change to achieve network's reduction. Our experimental results have demonstrated usefulness of our approach.

**TABLE 1. Experimental results.**

Circuit	misII gates/literals	RAMBO gate/literals	Pert/Sim gates/literals	CPU of Pert/Sim (sec)
5xp1	117(231)	111(221)	66(131)	34.5
9sym-hdl	96(192)	100(200)	39(78)	19.7
C3540	1073(2145)	988(1976)	938(1876)	5692.8
C5315	1452(2871)	1458(2883)	1321(2631)	2236.7
C6288	2619(5237)	2334(4666)	1883(3766)	2124.8
C7552	1757(3513)	1761(3521)	1426(2851)	3668.6
alu2	383(765)	366(731)	281(562)	1127.4
alu4	687(1373)	700(1399)	555(1110)	4171.5
apex6	632(1260)	647(1291)	543(1086)	568.9
b9_n2	102(200)	96(188)	79(156)	17.4
cm85a	40(80)	40(80)	27(54)	5.1
comp	137(273)	119(273)	84(168)	51.9
des	3048(6095)	3073(6145)	2859(5718)	31507.6
duke2	366(727)	314(626)	246(491)	648.5
f51m	120(239)	116(231)	78(155)	4.7
misex3	434(868)	468(936)	317(634)	978.8
my_adder	160(320)	160(320)	116(232)	29.1
pcler8	80(151)	80(151)	64(128)	29.7
rd53-hdl	36(72)	35(70)	20(40)	2.0
rot	575(1135)	569(1131)	452(902)	256
sao2-hdl	195(390)	199(398)	104(208)	119.6
term1	203(403)	203(404)	113(225)	56.2
ttt2	184(365)	174(247)	118(236)	57.8
x3	629(1253)	617(1231)	552(1104)	472.0
z4ml	37(74)	30(60)	21(42)	1.7
total	15162 (30232)	14758 (29379)	12302 (24584)	53883.0

**9 References**

[1] K.A. Bartlett et al, "Multilevel Logic Minimizing Using Implicit Don't cares," IEEE Trans. on CAD-7(6), pp. 723-740(June 1988).

[2] C. L. Berman and L. H. Trevillyan. "Global Flow Optimization in Automatic Logic Design," IEEE Trans. CAD 10, pp. 557-564(May 1991).

[3] D. Bostick et al, "The Boulder Optimal Logic Design System," Proc. ICCAD, pp. 62-65, 1987.

[4] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: Multi-level Interactive Logic Optimization System," IEEE Trans. on CAD, CAD-6(6), pp. 1062-1081(Nov. 1989).

[5] Shih-Chieh Chang and Malgorzata Marek-Sadowska, "Layout Driven Logic Synthesis for FPGA," Proc. Design Automation Conference. pp

[6] K.T. Cheng and L.A. Entrena, "Multi-Level Logic Optimization by Redundancy Addition and Removal," in Proc. European Conference On Design Automation, pp. 373-377, Feb. 1993.

[7] M.Damiani, J.C.Y.Yang and G.De Micheli, "Optimization of Combinational Logic Circuits Based on Compatible Gates", Proc. DAC' 93, pp.631-636, June 1993.

[8] L.A. Entrena and K. T. Cheng, "Sequential Logic Optimization By Redundancy Addition and Removal", Proc. International Conference on Computer Aided Design, Nov. 1993.

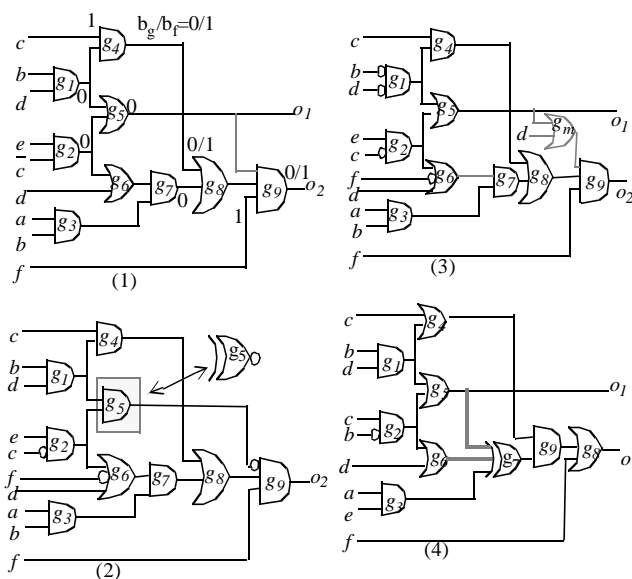
[9] E. Detjens, G. Gannot, R. Rudell, A. L. Sangiovanni-Vincentelli and A. Wang, "Technology Mapping in MIS," Proc. ICCAD, pp. 116-119, 1987.

[10] T.Kirkand and M.R. Mercer, "A Topological Search Algorithm For ATPG," Proc. 24th Design Automation Conf., pp. 502-508, June 1987.

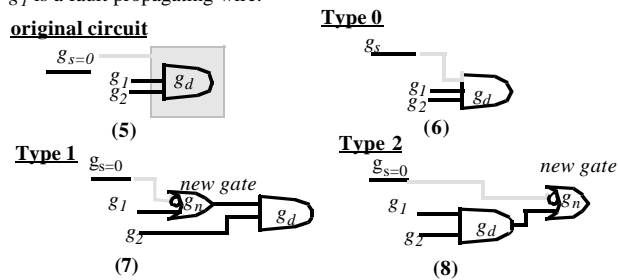
[11] C.E.Leiserson, F.M.Rose, and J.B.Saxe, "Optimizing synchronous circuit by retiming", in Proc. Third Caltech Conf. on VLSI, 1983.

[12] S. Muroga et al, "The Transduction Method-Design of Logic Networks Based on Permissible Functions," IEEE Transaction. on Computer C38(10). pp. 1404-1423 (Oct. 1989).

[13] M.Schulz and E.Auth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," Proc. Fault Tolerant Computing Symposium, pp. 30-34 June 1988.



Let  $g_s$  has mandatory assignment 0,  $g_d$  is a dominator (AND gate) and  $g_j$  is a fault propagating wire.



```

perturb_simplify(k_iteration, network)
int k_iteration;
network_t *network;
{
  For (i =0; i < k_iteration; i++) {
    Add_one_gate_to_remove_other_wires();
    Remove_large_reduction_wires();
    Perturb_circuit_more();
  }
  Add_one_gate_to_remove_other_wires();
}
(9)

```