Perturb and Simplify: Optimizing Circuits with External Don't Cares

Shih-Chieh Chang

Synopsys Inc.

Malgorzata Marek-Sadowska

Electrical and Computer Engineering Department, University of California at Santa Barbara

Abstract

Earlier optimization techniques based on Automatic Test Pattern Generation [3], [4], and [7] could not handle external don't cares. We propose a technique that uses external don't cares during the ATPG guided logic optimization. This technique transforms external don't cares into internal don't cares. Thus, the optimization can utilize the external don't cares to obtain better results.

Additionally, we also discuss some perturbation techniques to improve further results of logic optimizers. Based on a careful analysis of don't cares migration during incremental changes of an optimized circuit, we have developed a strategy to guide optimization.We have performed experiments on MCNC and ISCAS benchmarks and the results are very encouraging.

1 Introduction

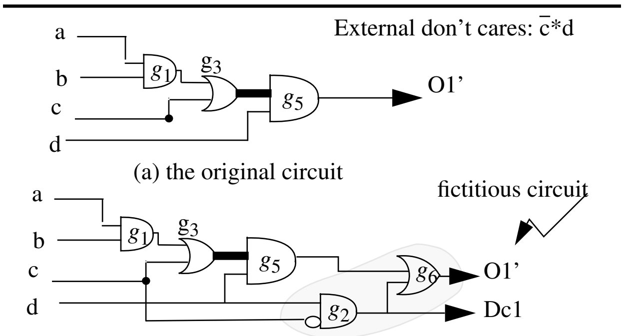
External don't cares are primary input combinations for which primary outputs are of no interest. These don't cares may originate from designers' specifications or be caused by the unreachable states of sequential circuits. Recent automated test pattern generation (ATPG) based logic optimizers [3] [4] [7] have proven very successful both in reducing circuit size and run-time memory usage. Since external don't cares can be very useful for logic optimization, we propose an approach that uses external don't cares during the ATPG guided logic optimization.

The ATPG based techniques [3] [4] [7] iteratively add redundant wires(gates) to remove other wires(gates) in a circuit. A crucial procedure in these techniques is to perform wire redundancy checking. A wire is redundant if a test pattern does not exist for a stuck-at fault test at this wire. On the other hand, the presence of external don't cares has the effect that some irredundant wires (without considering external don't cares) may become redundant. For example, in Fig 1a, the wire g_3->g_5 (bold in the figure) is not redundant because there exist test patterns {(a,b,c,d)=(0,0,0,1), (1,0,0,1), (1,0,0,1)} for g_3->g_5 stuck-at-1 test. However, if the circuit has external don't cares $\overline{c}*d$, the wire g_3->g_5 is redundant. This is because all test patterns are contained in the external don't cares $\overline{c}*d$. Therefore, the wire g_3->g_5 can be removed together with the gates g_1 and g_3.

One straightforward approach to identify whether test patterns are included in the external don't cares is to verify the set containment conditions. It can be performed using BDD based techniques. However, set operations are computationally expensive and potentially result in exponential complexity. Furthermore, the redundancy checking procedure is intensively applied in the ATPG based logic optimization techniques. Consequently, such a costly containment checking to utilize external don't cares cannot be of practical use. In this paper, we propose a technique that can implicitly calculate the containment condition. Our idea is to build a fictitious side circuit which can implicitly determine if test patterns are contained in external don't cares. By adding this fictitious side circuit to the original circuit, any wire that is redundant under external don't care conditions becomes untestable in the newly built circuit. Therefore, we can smoothly apply ATPG based techniques to optimize a combinational circuit with external don't cares. A simple demonstration of such a fictitious circuit is shown in Fig 1b (highlighted in the figure). It can be easily checked that the irredundant wire g_3->g_5 in Fig 1a becomes untestable and is therefore redundant in Fig 1b.

In this paper, we also discuss some perturbation techniques to further improve logic optimization results. The purpose of such perturbations is to avoid premature termination of the optimization process resulting in local minimum. The basic operation in our technique is to perform single-wire substitution (replace a wire by another wire) as discussed in [2]. This kind of single-wire substitution is very useful to alter circuit structure without additional costs. In our technique, instead of randomly applying such single-wire substitutions, we carefully guide the wire replacement procedure. It is based on an analysis of how internal don't cares are redistributed when a wire is replaced by another wire.

This paper is organized as follows. Section 2 reviews the techniques introduced in [4] [7][3]. Section 3 presents our approach of utilizing external don't cares. In section 4, we discuss some perturbation related issues. Finally, the experimental results are shown, followed by conclusions.



(a) the original circuit

(b) the original circuit+ the fictitious circuit

Fig 1 Redundant wire caused by external Dc

2 Background

2.1 Redundancy identification based on mandatory assignments

Here we briefly summarize the procedure proposed in [4] to identify redundancies (stuck-at faults) using the concept of mandatory assignments.

The *absolute dominators* (dominators) [9] 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 gates in G. When we perform ATPG for a wire stuck-at fault, a test vector must satisfy both conditions: to activate the fault and to propagate the fault to at least one primary output. *Mandatory Assignments* MA are value assignments required for a test to exist and must be satisfied by any test vectors. All mandatory assignments can be computed via implication (nine value implication) [9] [13] and recursive learning [10].If the mandatory assignments of a stuck-at fault test cannot be consistently justified, the fault is redundant.

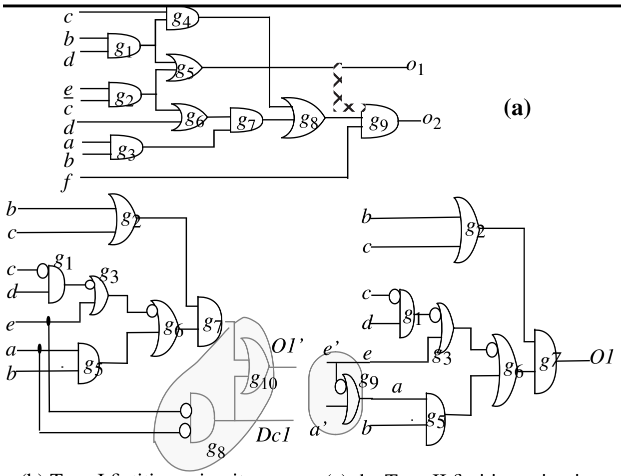
2.2 Wire substitution procedure

In this section, we discuss the process of replacing one wire by another wire(gate), which we call a selection of an alternative wire [3] [4] [7]. For example, in Fig 2a., we can replace g_1->g_4 by g_5->g_9 without changing circuit functionality. We say g_5->g_9 is an alternative wire of g_1->g_4. Though additional general transformations are possible to remove a particular wire [3], for clarity in the following we consider only a simplified frame work of finding single-wire alternatives.

First the MA 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 MA(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. For example, consider the circuit in Fig 2a. Let g_1->g_4 be the target wire to be removed. MA(g_1->g_4 s-a-1) = {c=1, g_1=0, $g5$=0, $g2$=0, f=1}. Note that g_5=0 and g_9 is a dominator. If we connect g_5 to g_9, g_5 must have a mandatory assignment of 1 for g_1->g_4 s-a-1 fault. We then choose g_5->g_9 as a candidate connection. Next, we check if g_5->g_9 is redundant. It is redundant therefore, we can indeed add the wire g_5->g_9 and remove the wire g_1->g_4.

3 Perturbation and simplification of a circuit with external don't cares

In this section, we propose a technique that naturally extends [3] [4] [7] to include external don't cares. Our idea is to built a fictitious side circuit which transforms external don't cares into internal don't cares. In the following, we first discuss the effects that external don't cares have on the ATPG based optimizers. We then show how to construct a fictitious side circuit to include external don't care conditions.



(b) Type I fictitious circuit

(c) the Type II fictitious circuit

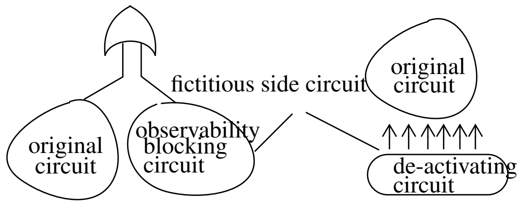
Fig 2 Additional alternative wire caused by external don't cares (exdc = $\bar{a}$*$\bar{e}$)

3.1 Two effects: more redundant wires and more alternative wires

The external don't cares create two effects of interest to us. First, some irredundant wires may become redundant when external don't cares are considered. For example in Fig 1a, the wire g_3->g_5 becomes redundant when $\bar{c}$*d is an external don't care. The other effect is that external don't cares introduce more mandatory assignments for testing a wire w_r stuck-at fault. These additional mandatory assignments are very helpful in finding more alternative wires for a target wire w_r. Since finding alternative wires is an essential operation in ATPG based logic optimizers, the increase of wire alternatives may in fact improve the results of circuit optimization. These two effects caused by external don't cares (more redundant wires and more alternative wires) are very helpful for the ATPG based logic optimizers.

3.2 Building a fictitious side circuit

We propose to build a fictitious side circuit which transforms the external don't care into internal don't care conditions. In Fig 3, two types of fictitious circuits are depicted. The Type I fictitious circuit in Fig 3a makes the original circuit unobservable for the input combinations included in external don't cares. The circuit is directly implemented from the Boolean function describing the external don't cares. For example, Fig 2b shows the original circuit along with the Type I fictitious circuit (highlighted). Here $\bar{a}*\bar{e}$ is the external don't care and the side circuit realizes the Boolean function $\bar{a}*\bar{e}$. Therefore, $\bar{a}*\bar{e}$ whenever evaluates to 1, the fictitious circuit outputs a value '1' and makes the original circuit unobservable. Mandatory assignments for g_1->g_3 s-a-1 test in the Fig 2b are MA={g_1=0, e=0, g_5 =0, g_2=1, g_8=0, ***a=1, b=0, c=1***}. g_8=0 is obtained because g_8 is a side input to a dominator g_{10}. As a result, inclusion of the fictitious circuit creates additional mandatory assignments (in bold font) caused by the external don't care conditions and leads to the recognition that *b*->g_7 is an alternative wire for g_1->g_3.



(a) Type I **(b)Type II**

Fig 3 Fictitious side circuits

The Type II fictitious circuit in Fig 3b forces any input combination in the external don't care to never occur at the inputs of the original circuit. For example, Fig 2c shows the Type II fictitious side circuit (highlighted). Since $\bar{a}*\bar{e}$ is the external don't care, the presence of Type II circuit causes that (a, e) = (0, 0) never occurs in the Fig 2c. It can be easily shown that the MA(g_1->g_3 s-a-1) = {g_1=0, e=0, g_5 =0, g_2=1, ***a=1, b=0, c=1***} and has the same additional mandatory assignments as the SMA(g_1->g_3 s-a-1) in Fig 2b.

Both Type I and Type II fictitious circuits have the same effect of transforming the external don't cares into internal don't cares. However, the Type I fictitious circuit is more flexible than Type II. The reason is that when several primary outputs have different don't care conditions, the Type I circuit can be still applied but Type II cannot. In addition, the Type I circuit can be directly constructed from the external don't cares of the function.

We now discuss some issues of implementing fictitious circuits. Since the only connection between the original circuit with the fictitious side circuit is through the primary inputs, it may cause inefficiency in finding mandatory assignments (MA) process. To tackle this inefficiency problem, we pull redundant wires which originate from the original circuit to the fictitious side circuit. Then, we remove as many redundant wires in the fictitious side circuit as possible. In this way, the connection between the original circuit and the fictitious side circuit is stronger, which in turn improves the process of computing mandatory assignments.

4 Perturbations

In this section, we discuss some perturbation techniques. These perturbations are performed to avoid premature termination of the optimization process resulting in local minimum. Here, we discuss the single-wire substitution technique to accomplish the perturbation purpose. Replacing one wire by another wire has advantages of altering the circuit's structure without increasing its size. Repeatedly applying single wire substitutions may potentially lead to a better circuit structure for subsequent optimization. In our technique, instead of randomly applying such single-wire substitutions, a carefully guided process is performed.

The degree of difficulty to propagate a fault effect from a wire(gate) to any primary output is expressed in terms of observability don't cares. Intuitively, it is easy (difficult) to propagate a fault effect to any primary output when the observability don't care set is small (large). We use the term "observable" to describe how difficult it is to propagate the fault. If a gate has a fanout to a primary output, the gate is fully observable and has no observability don't cares. Typically, if a gate has more fanouts, it is easier observed. This is because there may be many paths to primary outputs. However, it is not true in general, because some multiple faults may cancel each other due to reconvergence of fanouts. In the following, we discuss how observability don't cares are redistributed when a wire is replaced by another wire.

In Fig 4 two types of single wire substitutions are depicted. They are referred to as fanout and fanin substitution. We assert that fanout substitution occurs when a fanout wire *(g_a->g_d)* of a gate is replaced by another fanout *(g_a->g_b)*. This case is shown in Fig 4a. Here, we restrict our discussion to the case when g_b is a dominator of g_d and g_b is a gate of the same type as g_d. We discuss how the internal don't cares are redistributed during fanout substitution. Referring to Fig 4a, let Cone II contain all the wires in the intersection of the transitive fanout of g_d and in the transitive fanin of g_b. We have the following theorem.

Theorem 1: Suppose that during the fanout substitution the wire g_a->g_d is replaced by g_a->g_b and *w* is a wire in

Cone II. A test pattern which can test w stuck-at fault for the circuit after replacing g_a->g_d by g_a->g_b, is also a test pattern to test w stuck-at fault for the circuit before replacement.

Proof:. Let us consider testing w stuck-at fault in the circuit after replacement. Because g_b is a dominator for w, the side input g_a must be assigned to an uncontrolling value. The uncontrolling value of g_a makes the two wires g_a->g_d and g_a->g_b unimportant. Therefore, any test pattern which can test w stuck-at fault for the circuit after replacing g_a->g_d by g_a->g_b is also a test pattern to test w stuck-at fault for the circuit before replacement. Q.E.D.

Corollary: The size of the don't care set for the wires in the Cone II increases after replacing g_a->g_d by g_a->g_b.

According to this corollary, the don't cares for wires in the Cone II increase. As a result, it is beneficial for those wires when performing circuit size optimization. However, this replacement may also potentially reduce the don't cares for other parts of a circuit. Let us still consider the situation of replacing g_a->g_d by g_a->g_b. We define Cone I (in Fig 4a) to include all the wires that are in the transitive fanin of g_a. The wires in Cone I may potentially decrease the size of their don't care sets. The intuition behind this observation is as follows. For simplicity, assume that g_a->g_d is the only fanout and is replaced by g_a->g_b. Since g_b is a dominator of g_d, the "distance" from g_a to the primary output is now shorter. It is therefore easier to "observe" the information from g_a after replacement. Since g_a is easier to observe, also all the gates in the input cone are easier to observe. Therefore the size of observability don't cares for the input cone of g_a may decrease.

To summarize effects of the fanout substitution replacement, the size of don't care set increases for the wires in the Cone II and potentially decreases in Cone I. If g_a is a primary input, the input cone of g_a is empty. The replacement g_a->g_d by g_a->g_b is beneficial in logic optimization. Suppose now that g_a has a fanout to a primary output. Replacing g_a->g_d (another fanout for g_a) with g_a->g_b is also good for optimization. This is because g_a is already "fully observable" since one of its fanout is a primary output. It is not possible to increase its observability anymore. Intuitively, we can gain some benefit for logic optimization if g_a is close to the primary inputs or primary outputs and g_b is far away from g_d.

We will focus now on the other type of single wire substitution. First, we have the following theorem.

Theorem: In Fig 4c, let g_a->g_d be a fanin wire of g_d. Suppose the fanout cone g_a->g_d does not intersect the fanout cone of g_a's all other fanouts. Adding/deleting g_a->g_d decreases/increases the don't cares for wires that are dominated by g_a.

Proof: omitted.

The intuition behind the above theorem is as follows. If a gate has more fanouts, it is easier to observe. As a result, the size of the don't care sets associated with this gate decreases and so do the wires dominated by this gate. The restriction on the empty intersection of the fanout cone is to preclude the possibility of reconvergence. Reconvergent fanouts may potentially cause multiple fault cancellation and invalidate the above theorem. However, intuitively, when a gate adds/deletes one of its fanouts, the don't cares of this gate decrease/increase.

Based on the above theorem, we consider the following single wire substitution in Fig 4b. Let a fanin wire g_a->g_d of the gate g_d be replaced by another fanin wire g_b->g_d. According to the above theorem, since one fanout wire of the gate g_a is deleted, the don't cares of those wires (marked as Cone III in the figure) dominated by g_a possibly increase. And since one fanout wire of the gate g_b is added, the don't cares of those wires (marked as Cone IV) dominated by g_b possibly decrease. For logic optimization purposes, we would like the area of Cone III to be larger than the area of Cone IV.

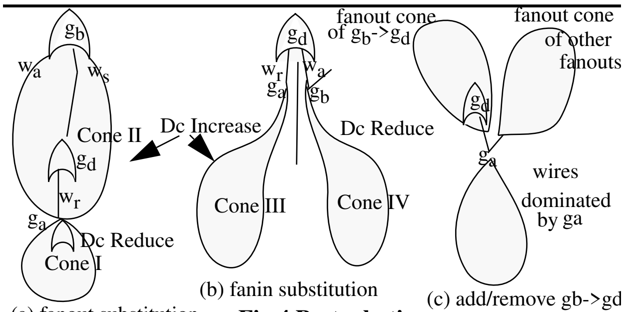


Fig 4 Perturbation

The discussion of single wire substitution and internal don't cares migration can be used to guide wire selection for replacement to improve the optimization results. As shown in the Fig 4ab, wires in Cones II and III increase the don't care sets and are better for logic optimization and wires in Cones I and IV decrease the don't cares and are poorer for logic optimization. In general, keeping track of how much don't cares increase/decrease is quite time consuming. Our heuristic uses the number of dominators of a gate as cost functions. After substitution, if a gate has more(less) dominators, we expect that it has less(more) don't cares than before.

5 Experimental results

We have performed sets of experiments to show the usefulness of our algorithm. The overall algorithm is shown in Fig 5. In the experiment, we applied the algorithm to sequential circuits in the MCNC benchmark suite. We first obtained the external don't cares of sequential circuits from the unreachable states. Then, the circuits were minimized by using script.algebraic in misII [1].

After that, we applied our logic optimizer to the resulting circuits. The results of some MCNC benchmarks are shown in TABLE 1. We compared our results with the results of script.rugged in misII [1]. (Script.rugged applies the algorithm [12] which utilizes external don't cares for optimization.) The results of misII were obtained as follows. After finding the unreachable states and treating them as don't cares, script.rugged was run. Next, we mapped the circuits into 2-input gates by invoking the map command in misII. The entry num1/num2 in the second and third columns show the number of 2-input gate count and literal count. The average improvement of our results relative to misII's is around 21%.

```
perturb_simplify(n_iterations)
{
    if (sequential_circuit) {
        cal_unreachable_states();
        construct_external_dont_cares();
    }
    if (external_dont_cares_exist)
        build_ficticiouts_side_circuit();
    for (i =0 to n_iterations) {
        add_gates_remove_more_gates();
        perturb_the_circuit();
    }
}
```

Fig 5 Perturbation algorithm

Acknowledgment.

This work was supported in part by the NSF grant MIP 9419119 and MICRO/LSI/SVR.

6 Conclusions

The ATPG based logic optimizers which have proven very successful for the combinational logic optimization did not make use of the external don't cares. In this paper, we have proposed a technique that utilized external don't cares in the ATPG based logic optimization. Our technique transforms external don't cares into internal don't cares extending the applicability of the ATPG based optimizers to circuits with external don't cares. We also proposed a guided perturbation technique to avoid premature termination of the optimization. Our experimental results show around a 21% improvement compared to the script.rugged from misII.

7 Reference

[1] 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.

[2] S-C Chang, K.T. Cheng, N.S. Woo, and M. Marek-Sadowska, "Layout Driven Logic Synthesis for FPGA," *Proc*. DAC,1994.

[3] S-C Chang and M. Marek-Sadowska, "Perturb and Simplify: Multi-level Boolean Network Optimizer," *Proc*. I CCAD, *pp*. 1-4, 1994.

[4] K.T. Cheng and L.A. Entrena, "Multi-Level Logic Optimization by Redundancy Addition and Removal," in Proc. EDAC, 1993.

[5] K.T. Cheng, "An ATPG-Based Approach to Sequential Logic Optimization," in Proc. ICCAD, pp. 372-375, 1993.

[6] M.Damiani and G.De Micheli, "Don't Care Set Specifications in Combinational and Synchronous Logic Circuits", IEEEE Trans. on CAD, Vol 12, no. 3, pp.365-388, March 1993.

[7] L.A. Entrena and K.T. Cheng, "Sequential Logic Optimization by Redundancy Addition and Removal," Proc. ICCAD, 1993.

[8] G. D. Hachtel, R. M. Jacoby, P. H. Moceyunas, "On Computing and Approximating the Observability Don't Care Set," in Proc. Int. Workshop on Logic Synthesis, Research Triangle Park, May 1989.

[9] T. Kirkland and M.R. Mercer, "A Topological Search Algorithm For ATPG," *Proc. 24th DA C*., pp. 502-508, 1987.

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

[11] H. Savoj and R. K. Brayton, "The Use of Observability and External Don't Cares for the Simplification of Multiple-level Networks," in Proc. DAC, pp. 297-301, 1990.

[12] H. Savoj, R. K. Brayton and H. Touati, "Extracting Local Don't Cares for Network Optimization," in Proc. ICCAD, 1991.

[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

[14] T. Touati, H. Savoj, B. Lin, R. Brayton, and A. Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines using BDD's," Proc. ICCAD, 1990.

TABLE 1 Comparison of benchmarks

circuit	**script.rug (2-input gates/ literals)**	**perturb-simplify (2-input gates/ literals)**	**cpu time perturb-simplify (sec)**
bbara	51/101	43/86	15.7
dk14	90/180	76/152	50.3
dk16	256/512	216/432	591.2
ex2	126/252	103/206	90.0
ex3	67/134	51/102	20.1
ex4	66/129	57/114	23.3
ex5	61/102	44/88	14.5
ex6	77/153	58/116	22.9
ex7	56/112	41/82	16.1
keyb	222/444	178/356	455.1
opus	64/128	49/98	22.5
planet1	445/890	325/650	1260.4
s1	309/618	221/442	534.3
s1a	311/620	240/468	709.1
sand	440/880	377/754	1851.3
scf	665/1319	575/1146	5855.2
sse	105/213	75/150	55.6
styr	475/914	339/678	1452.3
tbk	693/1386	564/1128	7202.7
train4	16/32	10/20	1.0
s1196	475/947	384/766	5864.4
s1238	464/925	389/766	6154.1
s1488	540/1079	409/818	2549.3
s1494	538/1075	409/818	2916.9
s382	115/223	91/177	297.2
s386	92/182	73/146	50.1
s400	112/221	89/172	317.3
s444	112/215	92/178	672.2
s820	247/485	183/366	391.4
s832	239/467	177/354	378.5
total	7290/14471	5761/11475	39456