REDI: An Efficient Fault Oriented Procedure to Identify Redundant Faults in Combinational Logic Circuits *

Chen Wang[†], Irith Pomeranz[††] and Sudhakar M. Reddy[†]

[†]Electrical and Computer Engineering Department
University of Iowa, Iowa City, IA 52242
[††]School of Electrical and Computer Engineering
Purdue University , West Lafayette, IN 47907

Abstract

In this work, a new and effective procedure, called REDI, to efficiently identify redundant single stuck-at faults in combinational logic circuits is proposed. The method is fault oriented and uses sensitizability of partial paths to determine redundant faults. It uses only implications and hence may not determine all the redundant faults of a circuit. However, experimental results presented on benchmark circuits show that the procedure identifies nearly all the redundant faults in most of the benchmark circuits. The key features of REDI that make it efficient are: partial path sensitization, blockage learning, dynamic branch ordering and fault grouping. Experimental results on benchmark circuits demonstrate the efficiency of the proposed procedure in identifying redundant faults in combinational logic circuits.

1. Introduction

Efficient methods to identify untestable faults in logic circuits are important for reducing test generation time and for logic optimization. Several methods have been proposed to identify untestable stuck-at faults in combinational logic circuits [5,6,9,10,11]. The procedures can be divided into two main categories, fault oriented and fault independent. Fault independent methods typically require smaller run times compared to fault oriented procedures. On the other hand, fault oriented procedures identify higher percentages and often all of the untestable faults in a circuit. Automatic test generation procedures that belong to the class of fault-oriented procedures and use the branch-and-bound approach are guaranteed to identify all the untestable faults given enough time. Efficient procedures for identification of untestable faults attempt to identify untestable faults without using branch-and-bound procedures used in test generators and hence are faster than test generation procedures, but may not identify all the untestable faults in a circuit. The procedure REDI proposed in this work is fault-oriented, yet it requires much smaller time than a complete ATPG algorithm to identify a large percentage of redundant faults.

The proposed procedure addresses the following problem: Given a set ***F*** of single stuck-at faults in a combinational logic circuit, identify the redundant faults in ***F***.

The paper is organized in the following way. In Section 2, the basic algorithm is introduced and basic concepts used in this work are defined and illustrated. In Section 3, several new techniques are described in detail. Experimental results on benchmark circuits are given in Section 4. Section 5 contains conclusions.

2. Overview

The basic algorithm of this work is similar to the propagation procedure of the Single-Path-Oriented Fault-Effect Propagation (SPOP) test generation procedure [8]. SPOP is an ATPG strategy that propagates fault effects to primary outputs through a single path prior to performing line justification. A similar strategy is used in SPIRIT[12]. It was shown in [8] that many redundant faults could be identified during the fault effect propagation phase.

A combinational logic circuit can be uniquely partitioned into internal fan-out free regions. An example of such a partition is given in Figure 1. The internal fan-out free regions (abbreviated as FFRs) are shown enclosed in triangles. From any input of a gate in a FFR there is a unique path to the output of the FFR. Hence there is a unique partial path to propagate the effect of a fault from an input of a gate in a FFR to the output of the FFR. Examples of partial paths in FFRs are shown by bold lines in Figure 1.

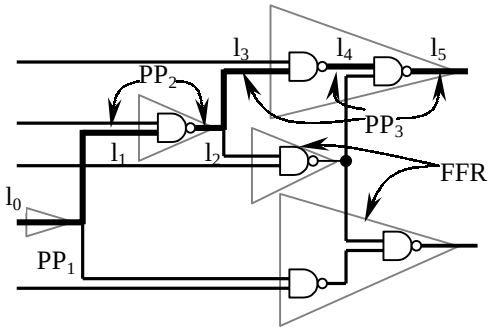

Figure 1. FFR Partitioning

* Research supported in part by NSF Grant No. MIP-9725053, and in part by SRC Grant No. 98-TJ-645

All the paths in a combinational logic circuit can be obtained by connecting partial paths in the FFRs of the circuit. We levelize the circuit after it is partitioned into FFRs starting from the FFR containing the fault, which is at level 1. This is illustrated in Figure 2. The partial path which starts from the fault site and ends at the output of the level 1 FFR is called the first level partial path. For any fault in a FFR, there is a unique first level partial path. The partial paths in FFRs that are directly driven by the first level partial path are called second level partial paths. Each second level partial path drives several 3rd level partial paths, and so on.

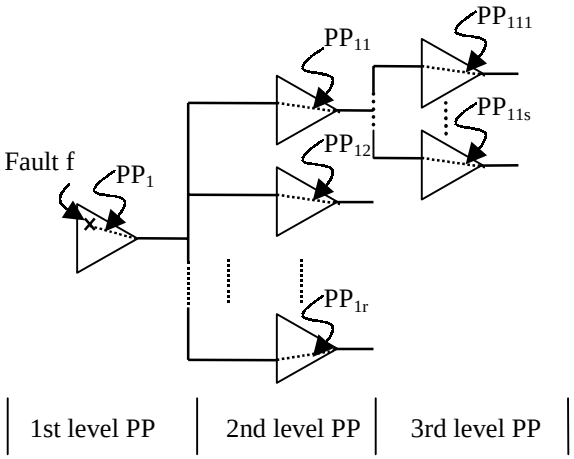

Figure. 2. Illustration of partial path levelization

To propagate the effect of a fault to the primary outputs, a sensitizable path from the fault site to a primary output must exist. Thus, if it can be proved that none of the paths from the fault site are sensitizable, then the fault is proved to be redundant. However, since the number of paths from a fault site to the primary outputs can be very large, efficient methods to analyze sensitizability of a large set of paths are necessary.

In Figure 3, the pseudo code for the basic procedure to determine if the effect of a fault can be propagated to a primary output is given. If the procedure fails to propagate the effect of a fault, the fault is determined to be redundant. Several techniques described in the next section are used to reduce the run time of the procedure.

In the procedure given in Figure 3, sensitization of a partial path is determined by using direct and indirect implications learned through static learning. The X-path check [2] is also used to make sure that further propagation of fault effects is possible.

In a typical application of a procedure to identify the subset of redundant faults in a given set of faults ***F***, there may be several testable faults in ***F*** in addition to redundant faults. For this reason, we included in REDI techniques to speed up identification of redundant faults as well as a technique to speed up the identification of testable faults. These are described in the next section.

```
Redundancy_Identify (fault line/v)
{
   if (imply (line, v') == FAIL)
     return (REDUNDANT);
   set SENS=Φ;
   if (find_sensitizable_path(line,SENS) == FAIL)
     return (REDUNDANT);
   else return (SUCCESS);
}

find_sensitizable_path(line,SENS)
{
   find the set SENS' of values necessary to sensitize the partial
   path from line to the output of its FFR;
   if (imply(SENS ∪ SENS') == FAIL)
     return (FAIL);
   else if (the end line of the partial path is a PO)
     return (SUCCESS);
   else
     set SENS = SENS ∪ SENS';
     for (every fan out branch fob_i of the end line
          of the partial path)
          {
            if(find_sensitizable_path(fobi,SENS)
             == SUCCESS) return (SUCCESS);
          }
   return (FAIL);
}
```

Figure 3. Pseudo code for path sensitization

3. New Techniques

In this section, we describe the new techniques used to reduce the run time of the SPOP based procedure to identify redundant faults in combinational logic circuits.

3.1 Blockage Learning

Blockage learning is used to determine necessary conditions to propagate fault effects through partial paths in FFRs. These necessary conditions are in addition to those learned using static learning. We use these additional necessary conditions and those found using static learning in the procedure find_sensitizable_path () given in Figure 3. The example given in Figure 4 illustrates blockage learning. In Figure 4, it can be observed that to propagate the effect of a fault through FFR1 to any output requires S to be set to 1. Requirements of this nature can be learned by implying logic value 0 and 1 on each fan-out stem of the circuit, and backtracing from every unspecified input of a gate that has at least one input that was set to the controlling value. An FFR is blocked if every one of its fan-out branches is reached during backtracing. The procedure for learning the necessary conditions to avoid blocking of fault propagation is similar to the ones in FIRE [9] and MUST [11]. The important difference between the goals of the blockage learning in REDI and FIRE or MUST is that the blocked signal lines of interest in REDI are fan-out stems whereas in FIRE or MUST the targets are the faults in a given set of faults. Additionally, the necessary conditions found using blockage learning are used to eliminate partial paths through which effects of faults cannot be propagated during the execution of the procedure find_sensitizable_path () given

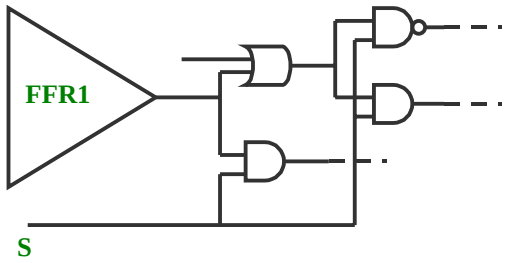

Figure 4. Example of Blockage

```
Blockage_Learning( )
{
  For (every stem S in circuit)
    {
      imply(S, 0);
      For (all blocked FFR_i)
        Learn: Fault through FFRi requires S =1;
      imply(S, 1);
      For (all blocked FFR_i)
        Learn: Fault through FFRi requires S =0;
    }
  Return;
}
```

Figure 5. Pseudo code for blockage learning

in Figure 3. The pseudo code of the procedure used for blockage learning is given in Figure 5.

In Figure 6, we show the total number of redundant faults identified in six ITC99 benchmark circuits B14s, B15s, B17s, B20s, B21s and B22s, with and without using blockage learning. These benchmark circuits are used in this experiment and in the one reported in the next subsection, since they contain faults that require relatively larger run times to classify. The horizontal axis in Figure 6 gives the number of FFR levels analyzed starting from the fault site to determine the number of redundant faults indicated by the height of the corresponding bar. The following observations can be made from Figure 6:

1. The total number of redundant faults identified with and without blockage learning is the same if all the paths from the fault sites to the circuit outputs are analyzed. This implies that blockage learning does not help to increase the total number of redundant faults identified if the number of levels is not restricted.
2. The number of redundant faults identified by analyzing partial paths over a restricted number of levels is higher with blockage learning than without blockage learning. This implies that, in this case, blockage learning reduces the number of partial paths analyzed to determine redundant faults, and hence helps to reduce run times.

For example, by analyzing partial paths with blockage learning through six levels of FFRs from the sites of faults, 3743 out of a total of 3896 redundant faults are identified in the benchmark circuits above. This means that 3743 redundant faults are identified because their effects could not be propagated through any of the partial paths in the first six levels from the fault sites. Without blockage learning, we could only find 3116 redundant faults by analyzing partial paths through six levels of FFRs.

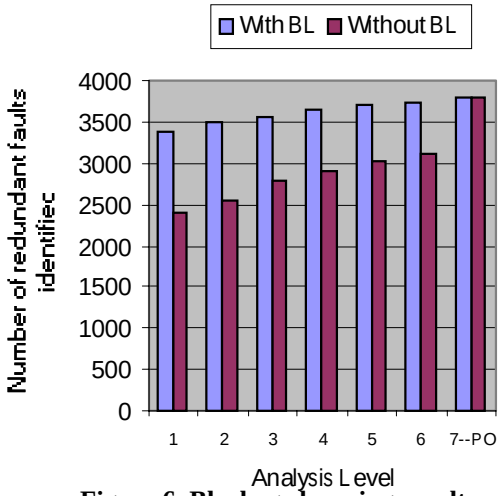

Figure 6. Blockage learning results

Blockage learning is also "inexpensive". What it requires is stem implications and backtracing to find blocked FFRs. Since static learning procedures also perform stem implications, one can combine these two procedures and then the increase in compute time for blockage learning over the time for static learning will be small.

3.2 Dynamic branch ordering

If a fault is testable, there must be at least one sensitizable path from the site of the fault. Quickly determining that such a sensitizable path exists reduces the time to determine that the fault cannot be proved to be redundant using conditions for fault propagation only.

Decisions on which partial path through a FFR is to be used to propagate the effect of a fault is made at a fan-out stem. This is done by selecting a branch of the stem. Previous strategies for selecting paths to propagate fault effects used static measures such as observalility[1] to select partial paths. We use the following dynamic procedure to order the branches of a fan-out stem.

1. The initial order of all the branches is based on the observability provided by SCOAP[1]. This ordering is only performed once in a preprocessing stage.
2. Consider the case where a redundant fault is determined on a gate in an arbitrary FFR, FFR0. Let g1, g2, …, gk be fan-out branches driving FFR0, and let h1,h2, …, hk be the fan-out stems connected to g1, g2, …, gk, respectively. In the list of fan-out branches of hj, $1 \le j \le k$, gj is moved to the end of the list. This ensures that gj is not considered again unless all the other branches of hj also contribute to redundant faults.
3. If the effect of a fault is successfully propagated along a cascade of partial paths to a circuit output, the fan-out branch driving each partial path is moved up to the first place.

The branch ordering proposed above does not aid in identifying redundant faults. Redundant faults are identified only if their effects cannot be propagated along

any path to the outputs. However, it helps reduce the number of paths tried to determine that the effect of a testable fault can be propagated to a circuit output.

In Table I, we give the average number of paths tried to determine that the effect of a testable fault is propagatable to an output. In the first row, we give the circuit name, in the second and third rows, we give the average number of paths analyzed without branch ordering and with branch ordering, respectively. It can be seen that branch ordering reduces the average number of paths analyzed to determine that the effects of a fault can be propagated to a circuit output.

CUT	B14s	B15s	B17s	B20s	B21s	B22s
No ord.	6.38	1.66	1.43	2.45	2.82	3.61
With ord.	3.83	1.11	1.22	1.25	1.32	2.23

Table 1. Dynamic branch ordering

3.3 Fault grouping

In order to reduce the run time of the proposed procedure, we analyze faults in a manner that allows maximal reuse of analyses done for faults targeted earlier. This is achieved by considering faults from circuit outputs to circuit inputs and considering faults in FFRs in a breadth first manner. In the following we discuss why this order of considering faults allows maximal reuse of analysis data needed to identify redundant faults.

1. The blockage information collected for fan-out stems of FFRs is common to all the faults in a FFR as well as faults that drive the FFR when their effects are propagated through it. Similarly, the dynamic branch ordering uses the knowledge of partial paths in FFRs containing redundant faults and sensitizable partial paths to dynamically order the branches of the fan-out stems in the preceding levels.
2. By processing faults from the output to the inputs of a FFR, once a redundant fault is found, appropriate faults in the fan-in cone of the fault site can be immediately marked as redundant. This observation was also used in [4]. In addition, implications made for a fault in a FFR can be reused for succeeding faults by incrementally adding to the earlier implications.

3.4 Additional techniques

In order to further reduce the run time of the procedure we used five valued logic (0, 1, D, D' and X) for fault effect propagation. To compensate for the reduction in the number of implications due to the use of five valued logic, we used the parity of the number of inversions on a path driven by the fault site to determine cases where X values must be replaced by 0 or 1 values to prevent fault effects of the fault under consideration from blocking sensitization.

Since we analyze partial paths through FFRs, the number of levels of FFRs analyzed can be easily limited to reduce the run time. We restricted the number of FFR levels analyzed from a fault site to six in the experiments reported in the next section.

4. Experimental Results

The procedure described in Figure 3 together with the techniques proposed in Section 3 was implemented in one package in the C programming language. We call this procedure REDI. Results of applying REDI to ISCAS-85 and the combinational logic of ISCAS-89 and ITC-99 benchmark circuits are reported in this section.

In addition to the new techniques described in the last section, we also used some well-known techniques in REDI. We used fault simulation of random patterns to identify easily detectable faults and reduce the number of candidate faults targeted. Fault simulation was stopped when a set of 32 consecutive random patterns did not detect any yet undetected faults. We used only simple implications and static learning. After we imply all the necessary values for a partial path, we perform the x-path check[2].

The set of faults ***F*** given to the procedure was the set of all the collapsed single stuck-at faults. For the results given in this section we limited the number of levels of FFRs analyzed to six. We also set a limit of 100 on the number of partial paths considered per fault.

In Table II, we report the results of REDI for ISCAS-85, ISCAS-89 and ITC-99 benchmark circuits. In Table II after the circuit name we give the number of redundant faults identified by [3] and [12], followed by the results obtained by REDI. The run times reported are for a SUN Blade-1000 workstation using Unix OS. The run times reported are the total times including time for fault simulation.

From Table II it can be seen that REDI finds all the redundant faults in 20 out of 28 benchmark circuits. Out of a total of 11239 redundant faults in the 28 circuits, 11047 redundant faults are identified; i.e., approximately 98.3% of all the redundant faults in the benchmark circuits are found by REDI.

In Table III we give run times to identify the redundant faults by REDI and two deterministic test generators, ATALANTA [6] and a test generator from Kyushu Institute of Technology (KIT). All the procedures were run only on the redundant faults identified by REDI. The recorded run times are only to process these faults and do not include time for fault simulation, static learning and processing other faults. All programs were run on the same workstation described above. The ATPG programs were allowed 100 backtracks per fault. After the circuit name we give the number of redundant faults identified by REDI that were given to all three procedures to process, followed by the run times for REDI, ATALANTA and the ATPG from KIT. Since the ATPGs aborted on some faults we give the number of aborted faults in parentheses, when applicable. From Table III it can be seen that the run time for REDI to identify the redundant faults is much smaller than that for the two ATPGs. Furthermore, some faults are aborted by the ATPGs with the selected backtrack limit.

5. Conclusions

In this work, a new procedure called REDI to efficiently identify redundant faults in combinational circuits was

presented. Redundant faults were identified by using implications to determine that fault effects cannot be propagated to circuit outputs. New techniques used to reduce the run times to identify redundant faults are blockage learning, dynamic branch ordering and fault grouping. Experimental results showed that REDI identifies most of the redundant faults in benchmark circuits.

References

[1] L. M. Goldstein and E. L. Thigen, "SCOAP: Sandia Controllability /Observability Analysis Program," Proc. DAC, June 1980, pp. 190-196.

[2] P. Goel, "An implicit Enumberation Algorithm to Generate Tests for Combianational Logic Circuits," IEEE Trans. on Computers, vol. C-30, No. 3, March 1981, pp. 215-222.

[3] J. A. Waicaukauski, P. A. Shupe, D. J. Giramma and A. Matin, "ATPG for ultra-large structured designs," Proc. ITC, Oct. 1990, pp. 44-51.

[4] M. Abramovici, D. T. Miller, and R. K. Roy, "Dynamic Redundancy Identification in Automatic Test Generation", IEEE Trans. On CAD, Vol. 11, No. 3, March 1992, pp. 404-407.

[5] P.R. Menon and H. Ahuja, "Redundancy Removal and Simplification of Combinational Circuits", Proc. VTS, April 1992, pp. 268-273.

[6] S. Kajihara, H. Shiba, and K. Kinoshita, "Removal of Redundancy in Logic Circuits under Classification of Undetectable Faults", Proc. FTCS, July 1992, pp.263-270.

[7] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Technical Report No. 12_93, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University

[8] M.Henftling, H.Wittmann and K. Antreich, "A Single-Path-Oriented Fault-Effect Propagation in Digital Circuits Considering Multiple-Path Sensitization," Proc. of ICCAD, Nov. 1995, pp. 304-309.

[9] M. Iyer and M. Abramovici, "FIRE: A Fault Independent Combinational Redundancy Identification Algorithm," IEEE Trans. on VLSI Systems, Vol. 4, No. 2, June 1996, pp. 295-301.

[10] J-.K Zhao, E. M. Rudnick, and J. H. Patel, "Static Logic Implication with Applications to Redundancy Identification," Proc. VTS, April 1997, pp. 288 – 293.

[11] Q. Peng, M. Abramovici, and J. Savir, "MUST: Multiple-Stem Analysis for Identifying Sequentially Untestable Faults," Proc. ITC, Oct. 2000, pp. 839-846.

[12] E. Gizdarski and H. Fujiwara, "SPIRIT: A Highly Robust Combinational Test Generation Algorithm," Proc. VTS, Apr. 2001, pp. 346-351.

		REDI				REDI	
		red.	run time (s)			red.	run time (s)
C432	4	1	0.01	S349	2	2	0.01
C499	8	8	0.01	S444	14	14	0.01
C1355	8	8	0.04	S713	38	38	0.02
C1908	9	9	0.08	S1238	69	66	0.10
C2670	117	98	0.08	S1423	14	14	0.03
C3540	137	137	0.38	S1494	12	12	0.13
C5315	59	59	0.15	S5378	40	40	0.20
C6288	34	34	0.25	S9234	452	440	0.90
C7552	131	131	0.40	S13207	151	150	1.62
B14s	274	264	6.57	S15850	389	388	1.71
B15s	508	500	44.24	S35932	3984	3984	7.47
B17s	1356	1323	165.72	S38417	165	165	2.68
B20s	486	451	11.12	S38584	1506	1506	11.27
B21s	496	466	10.94	B22s	776	739	21.15

Table II. Results for ISCAS-85, ISCAS-89 and ITC-99 Benchmark circuits

		Run time in sec. (#abort)					Run time in sec. (#abort)		
		REDI	ATA. [6]	ATPG*			REDI	ATA. [6]	ATPG*
C432	1	0.00	0.00	0.00	S349	2	0.00	0.00	0.00
C499	8	0.00	0.00	0.00	S444	14	0.00	0.00	0.00
C1355	8	0.00	0.00	0.00	S713	38	0.00	0.01	0.00
C1908	9	0.00	0.01	0.00	S1238	66	0.01	0.02	0.03
C2670	98	0.00	0.51 (20)	0.14	S1423	14	0.00	0.00	0.00
C3540	137	0.01	0.02	0.02	S1494	12	0.00	0.00	0.01
C5315	59	0.01	0.02	0.01	S5378	40	0.00	0.01	0.00
C6288	34	0.01	0.01	0.01	S9234	440	0.07	4.29 (44)	1.80 (4)
C7552	131	0.02	2.82 (54)	3.54	S13207	150	0.05	2.40 (9)	2.58
B14s	264	0.48	12.99 (93)	12.45	S15850	388	0.08	1.59 (1)	0.51
B15s	500	1.52	51.74 (245)	113.84(9)	S35932	3984	1.36	10.82	4.95
B17s	1323	4.40	233.23 (648)	317.98(27)	S38417	165	0.02	2.44 (4)	0.08
B20s	451	0.48	24.90 (148)	20.79	S38584	1506	0.54	8.29 (14)	2.72
B21s	466	0.50	21.65 (116)	11.69	B22s	739	1.03	58.13 (252)	30.54

*This ATPG was given to us by Professor Seiji Kajihara of Kiyushu Institute of Technology.

Table III. Comparison of REDI with ATPGs