Effective TARO Pattern Generation

Intaik Park[1]*, Ahmad Al-Yamani*[1,2] *and Edward J. McCluskey*[1]
[1]*Center for Reliable Computing* [2]*TNT, Advanced Development*
Stanford University *LSI Logic Corporation*
http://crc.stanford.edu *http://www.lsilogic.com*
{intaik | alyamani | ejm}@crc.stanford.edu

Abstract

TARO test patterns are transition fault test patterns that sensitize each transition fault to all of the outputs that can be reached from the fault location. We were not able to identify any ATPG tool that can generate TARO test patterns directly. This paper describes a technique to use an existing transition fault ATPG tool to efficiently generate TARO test patterns. This technique was used to generate TARO patterns for the ELF35 test chip. When these patterns were applied to the ELF 35 chips, all of the defective chips were discovered (no test escapes).

1. Introduction

When we were studying the test results for the ELF-Murphy chips [1, 2, 3], we noticed that some of the defective chips passed our transition fault test patterns. Further study showed that the test patterns sensitized the transition faults to some, but not all, of the reachable outputs. When the transition patterns were augmented so that all of the outputs were used, there were no longer any test escapes [4].

We wanted to have a technique for automatically generating TARO patterns without writing an ATPG tool from scratch. This paper describes a procedure for modifying an existing transition fault ATPG tool so as to obtain TARO patterns. This procedure was used to generate TARO patterns for the ELF35 chip [4]. No defective chip escaped these patterns. The results of applying other types of test sets are shown to permit comparison.

This paper is organized as follows. In Sec. 2, the background for TARO generation is presented and in Sec. 3, the generation procedure is described. The experimental results are discussed in Sec. 4, and Sec. 5 concludes the paper.

2. Preliminaries

TARO test propagates each transition fault to all of its reachable outputs and the required information for generation is a list of reachable outputs (outputs through which a fault can be propagated and detected) for each fault. In previous work, exhaustive simulations were used to obtain reachable outputs [4, 5]. An alternative approach is to use detectable faults. Detectable faults for an output are defined as a list of faults that can be observed through the output. Since they are reversible concepts, lists of reachable outputs for each fault can be constructed from lists of detectable faults for each output. One way is to perform logical cone analysis on netlist. All fault sites that are included in a logical cone of an output are candidates for detectable faults, but it requires additional simulations because there is no guarantee that all these faults can be activated and a path from a site to the output can be sensitized.

Another way to obtain reachable output information is to use an output mask, which is a constraint for an ATPG tool that prohibits propagation of faults through the masked output or force the ATPG tool to ignore a detection of a fault if the masked output was used to observe the fault. If a test is generated with all the outputs (primary outputs and scan flip-flops) masked except one output (unmasked output), then, detected faults of this test are detectable faults of the unmasked output for given effort level, which is the number of backtracks the ATPG tool tries before it gives up on a fault and categorizes it as an undetectable fault. An example of this operation is presented in Fig. 1. The square marks represent fault sites and the circles represent outputs (primary outputs and scan flip-flops). The arrows from a fault site to an output represent fault-output pairs or transition paths. If output masks are placed on outputs 2 and 3, a test pattern generated with this setup will only expose detectable faults of output 1.

Iteration of this procedure over all the outputs will produce lists of detectable faults for all the outputs. A matrix of outputs versus faults can be constructed from these lists and transposed to make a matrix of faults versus outputs, which represents reachable outputs of faults. The number of detectable faults obtained this way varies depending on the effort level of the ATPG tool. Hence, the effort level in detectable fault analysis can be used as a primary tuning knob to control the thoroughness of the TARO test.

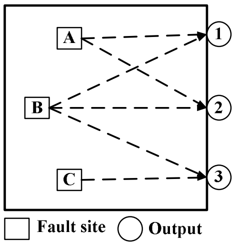

Figure 1: CUT with fault-output pairs

Another concept required for TARO generation is a *fault-output pair*. Unlike a transition fault test where a fault is detected if it is observed through any of its reachable output, in TARO test generation, there is a need to distinguish which output is used to observe a fault. *A fault-output pair is defined as a pair of a fault site and one of its reachable outputs*. It implies a transition path from the fault site to the output but it neglects various functional paths from the site to the output. By treating each fault-output pair as an independent fault, the TARO test generation can be simplified to a test generation that targets fault-output pairs.

3. Procedure of TARO generation

In this section, a detailed procedure of TARO generation is presented. First, a derivation of fault-output pairs is explained in Sec. 3.1 followed by a procedure for TARO patterns generation in Sec. 3.2. In Sec 3.3, a way of quantifying TARO test coverage is presented.

3.1. Derivation of Fault-Output Pairs

To find the fault-output pairs of a circuit, all the outputs except one are masked and a transition ATPG is performed as explained in Sec. 2, the detected faults in this setup form fault-output pairs with the associated output (unmasked output). If this procedure is iterated over all the outputs, the union of all the revealed fault-output pairs will form reachable output information. A flowchart depicting the overall procedure of derivation of fault-output pairs is presented in Fig. 2.

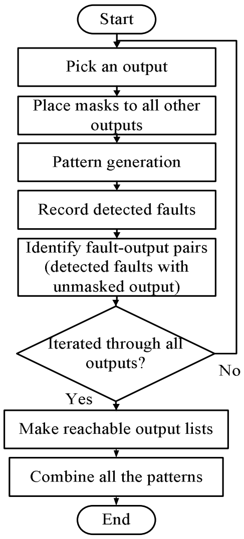
Start
Pick an output
Place masks to all other outputs
Pattern generation
Record detected faults
Identify fault-output pairs (detected faults with unmasked output)
Iterated through all outputs?
No
Yes
Make reachable output lists
Combine all the patterns
End

Figure 2: Derivation of fault-output pairs

The union of all the tests from individual ATPG iterations in this procedure is a TARO test set and we denote this generation as simplistic TARO test generation. However, it results in inefficient TARO test sets since each pattern exploits only one output at a time. A more efficient TARO generation algorithm is presented in Sec. 3.2.

3.2. TARO pattern generation

Simplistic TARO generation in previous results in an inefficient TARO test set because multiple outputs can be targeted simultaneously to reduce the test set size. However, the existing ATPG tools don't provide the option of fault-output pair recognition. This section introduces an algorithm to be combined with the existing ATPG tools [6, 7] capabilities to provide an efficient TARO generation algorithm. The algorithm obtains the reachable output information from the procedure explained in the previous section.

If all reachable outputs except one are masked, it is guaranteed that the test will use only one output (unmasked output) to detect faults. Hence, if two faults do not share any reachable outputs, the ATPG tool can be invoked to generate patterns for these faults simultaneously while the corresponding outputs are all unmasked. In Fig. 1, faults A and C do not share any reachable outputs, so, generating patterns for both faults together does not compromise the TARO coverage for the test set. However, since some logical cones starting from outputs overlap with each other, generating patterns for these faults simultaneously can lead the tool to compromise the TARO coverage of the fault-output pairs. In Fig. 1, to propagate fault B to output 2, masks should be placed on output 1 and 3 and output 2 should be unmasked. On the other hand, to propagate fault A to output 1, output 2 should be masked and output 1 has to be unmasked. If we unmask output 1 and 2, the ATPG tool will not generate TARO patterns for faults A and B because the generated patterns will not propagate each fault to all reachable outputs. This is an example of conflicting assignments. Fault B cannot be directed to output 2 when fault A is simultaneously assigned to output 1. To avoid conflicting assignments, two lists of outputs are maintained. One is a 'mask list' and the other is an 'unmask list'. These lists can be referred to when fault-output pairs are being assigned.

Similarly, a fault-output pair-based fault simulation can be performed with output masks. If output masks are assigned to all the outputs except one and fault simulation is performed, the detected faults in this setting are faults that are detected only through the unmasked output and detections of fault-output pairs are verified. This procedure can be iterated over all the outputs to obtain a complete fault-output pair grading.

The compacted TARO test generation procedure, based on these ideas, is described next.

Step 1: Pre-run
(obtaining reachable outputs lists for all the fault)

Tests are generated with one unmask setup and iterated over all outputs to make lists of detected faults which, in turn, are used to build a matrix of lists of detectable faults of all outputs. This matrix is transposed to obtain lists of reachable outputs for all faults. This step is explained in Sec. 3.1.

Step 2: Initial run
(transition fault ATPG without constraints)

The purpose of this step is to utilize the ATPG transition test generation to target as much as possible from the fault-output pairs that we identified in step one. So, we just run the ATPG tool without any output masking.

Step 3: Fault Simulations
(Fault simulation with output mask constraints)

The generated test is fault simulated with output masks to check which outputs are used for each fault. The detected fault-output pairs are dropped from the list of undetected pairs.

Step 4: Iteration
(ATPG with constraints and fault simulation)

This step consists of iterations of test generations and fault simulations to detect the rest of the undetected fault-output pairs. Masks are assigned such that tests are generated for only non-conflicting fault-output pairs simultaneously. Fault simulations to verify which output is used for each fault follow and all the detected fault-output pairs are marked after simulations. This procedure iterates with the rest of the undetected pairs and the resulting tests are combined with the test set from the initial run (step 2) to make a TARO test set. The overall procedure is presented as a flowchart in Fig. 3.

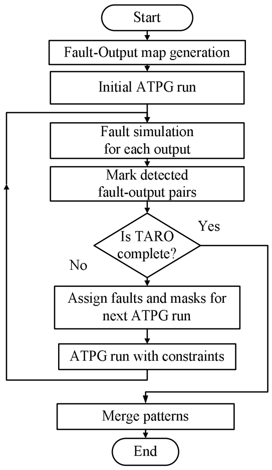
Start
Fault-Output map generation
Initial ATPG run
Fault simulation for each output
Mark detected fault-output pairs
Is TARO complete?
Yes
No
Assign faults and masks for next ATPG run
ATPG run with constraints
Merge patterns
End

Figure 3: Overall procedure of TARO generation

In step 3, the selection of fault-output pairs affect the degree of compaction achieved. Pessimistically, if each assigned fault is assumed to be propagated to only one of its outputs, the number of detected pairs of the test will be low. Optimistically, a smaller number of masked outputs would increase the chances to detect additional fault-output pairs through unmasked outputs. As a result, the goal is to assign as many faults as possible with the least possible number of output masks. The assignment procedure is similar to the widely studied graph coloring problem and a greedy algorithm is known to work well [8]. Hence, a greedy algorithm is used as shown in Fig. 4.

As stated earlier, a mask list and an unmask list are maintained and both lists are empty at the beginning of each iteration. The assignment starts by picking a fault with unused outputs (undetected fault-output pair) with predetermined selection criteria discussed later in the section. There are two conditions that must be satisfied for a fault to be targeted in the current iteration: 1) At least one of its unused reachable outputs is not in the mask list. Unused reachable outputs of the fault site are compared with the mask list and the fault is dropped if all the unused reachable outputs are masked. This fault cannot be assigned in this iteration since there is no output to detect the fault. 2) All of its used reachable outputs must be in the mask list. Used outputs are compared with an unmask list and if any used output is unmasked, the fault is dropped because if a used output is not masked, the ATPG can generate a test that propagates a fault to an already used output. If a fault satisfies these two conditions, the fault is assigned for the iteration. Then, all the used outputs of this fault are added to the mask list and any unused outputs that are not in mask list are added to the unmask list. The iteration continues until there are no more faults left.

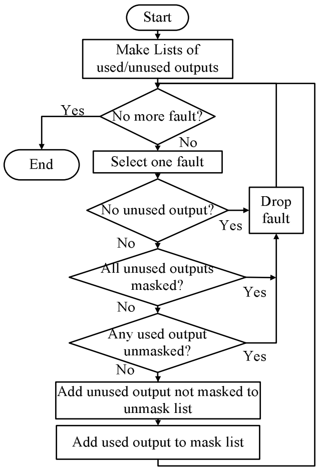
Start
Make Lists of used/unused outputs
No more fault?
Yes
No
End
Select one fault
No unused output?
Yes
Drop fault
No
All unused outputs masked?
Yes
No
Any used output unmasked?
Yes
No
Add unused output not masked to unmask list
Add used output to mask list

Figure 4: Assignment procedure

When picking a fault to assign, five different heuristics are used. One chooses faults with most number of reachable outputs while second selects faults with most number of unused outputs. The third way picks faults with least number of used outputs. These three heuristics are based on the reasoning that the faults with more widely spanning reachable outputs should be completed earlier than others so that there would be no masks of these faults that blocks assignments of others. Another heuristic is to choose faults with least number of unused outputs to complete faults that can be finished easier with fewer numbers of iterations. The last is to randomly pick faults. The resulting compaction ratios of five different heuristics are presented in experimental result section.

3.3. Evaluation of TARO test pattern set

Traditional transition fault coverage cannot be applied to TARO patterns because it does not distinguish fault-output pairs that share a fault site. However, in evaluating a TARO test, each fault site can have multiple reachable outputs and is detected fully only if all of its reachable outputs are used. Hence, in evaluating TARO test, weights are assigned to partially detected faults in order to reflect the incompleteness. For example, if a fault site has 10 reachable outputs and 7 of them are used to detect this fault, the fault has 7/10 weights on fault coverage. In case of fully detected faults, a full weight of 1 is given. The TARO coverage is the average of all the weights of faults sites calculated this way.

$$Weight\ of\ \ fault = \frac{\#output\ used}{\#reachable\ output}$$

$$TARO\ \ FC = \frac{\sum weights\ of\ \ faults}{\#all\ \ faults}$$

TARO fault coverage requires lists of reachable outputs for all faults as a reference. The most accurate way to obtain this is to use a detectable fault list obtained by logical cone analysis. We derive the fault-output pair information by running the ATPG tool as explained in Sec. 3.1. Due to the ATPG effort level, there maybe some pairs that are not identified. So for better accuracy, in this work, the fault-output pair information is obtained from the union of all fault-output pairs in TARO and transition fault test.

4. Experimental Result

The TARO test patterns were generated and applied to the ELF35 test chips. The ELF35 chip [3] was designed and manufactured with LSI's G10p technology. The nominal supply voltage for this chip is 3.3V and the effective channel length is 0.35um. It contains multiple copies of four combinational cores and two sequential cores. The sequential cores are: 1) LSI2901, which is an arithmetic processor from the LSI library and 2) TOPS2901, the same design synthesized using the TOPS synthesis-for-test tool developed by Stanford CRC. M12, SQR, MA and PB are combinational cores. PB is a pseudo-random-to-binary translator that contains many random-pattern-resistant faults and the other three combinational cores are data-path designs.

Table 1: ELF35 cores

core	#gate	#PO	#FF	#output	#faults
m12	1,309	12	NA	12	1,629
sqr	538	6	NA	6	625
ma	4,499	33	NA	33	9,973
pb	17,468	12	NA	12	57,520
lsi2901	12,338	64	544	608	11,819
tops2901	18,090	48	961	1,009	22,131

The characteristics of ELF35 cores are shown in Table 1. In the table, the number of gates, primary outputs and scan flip-flops are shown in the first three columns. Column 4 contains the number of outputs that are considered in TARO generation, which is sum of number of primary outputs and scan flip-flops. The last column represents the number of collapsed transition faults for each core to be contrasted with the number of fault-output pairs presented also in this section.

For combinational cores, TARO test patterns were generated and the results are presented in Tables 2. For simplistic TARO from Sec. 3.1, one set is compacted using dynamic compaction and the other is generated without it. However, both sets are generated with effort level of 100. Although the dynamic compaction resulted in a smaller pattern size, it could not be applied in the efficient TARO due to some ATPG tool limitations. Hence, for more accurate comparison, a fault-output pair list from simplistic TARO without dynamic compaction is used as a reference in TARO generation. All the patterns except the MA core pattern are statically merged and the size after compaction is also presented. In the tables, pattern sizes before static merge and after the merge are shown in columns 3, 4. The numbers of detected fault-output pairs are shown in column 5. The first row is for simplistic TARO with dynamic compaction and the second row shows simplistic TARO without dynamic compaction. The rest of the rows represent TARO with 5 different heuristics.

Table 2: TARO patterns for combinational cores

core	pattern	# pat	# cmp pat	# f-o pairs
m12	dyn. cmp	1,044	948	9,845
	no cmp	2,973	1,464	10,771
	heuristic1	2,051	1,236	11,391
	heuristic2	1,885	1,176	12,960
	heuristic3	1,899	1,176	13,739
	heuristic4	2,257	1,271	12,458
	heuristic5	1,918	1,188	12,561
sqr	dyn. cmp	117	102	1,357
	no cmp	395	187	1,668
	heuristic1	285	179	1,932
	heuristic2	252	173	2,066
	heuristic3	228	158	2,229
	heuristic4	274	171	2,089
	heuristic5	281	173	2,153
pb	dyn. cmp	21,063	10,286	139,416
	no cmp	57,204	10,707	140,225
	heuristic1	30,222	9,069	139,439
	heuristic2	28,068	8,969	140,908
	heuristic3	28,727	9,038	140,077
	heuristic4	29,129	9,059	139,466
	heuristic5	28,960	9,044	139,554
ma	dyn. cmp	6984	6,686	94,016
	no cmp	16,586	10,851	85,329
	heuristic1	7,915	-	138,482
	heuristic2	11,015	-	165,111
	heuristic3	13,047	-	155,660
	heuristic4	9,073	-	140,894
	heuristic5	9,400	-	148,763

Table 4: Comparisons of SSF, transition, N-detect and TARO tests

cut	pat	# pat.	cpu time	ssf cov	tr cov	taro cov
m12	ssf	61	3.68	99.01%	-	-
	tr	70	12.29	99.16%	97.92%	43.22%
	15ndet	327	9.83	99.47%	-	-
	taro	1,044	189.86	100%	99.88%	97.34%
sqr	ssf	30	2.55	97.00%	-	-
	tr	112	5.41	96.90%	78.56%	73.02%
	15ndet	137	3.94	97.00%	-	-
	taro	117	22.09	96.60%	94.12%	92.22%
ma	ssf	69	29.77	97.87%	-	-
	tr	103	101.62	97.84%	97.47%	60.37%
	15ndet	416	71.19	98.16%	-	-
	taro	6,984	8503.28	98.25%	96.84%	89.20%
pb	ssf	4,389	33.38	100%	-	-
	tr	21,440	732.26	100%	99.98%	59.15%
	15ndet	64,197	572.39	100%	-	-
	taro	21,063	1509.30	100%	100%	40.85%
lsi2901	ssf	6,357	31.11	99.93%	-	-
	tr	10,821	1443.03	99.51%	92.87%	46.53%
	15ndet	94,910	1339.50	99.93%	-	-
	taro	307,048	19219.13	99.87%	94.95%	97.57%
tops2901	ssf	432	5.03	99.96%	-	-
	tr	1,001	73.11	99.95%	83.66%	96.42%
	15ndet	3,791	19.04	99.96%	-	-
	taro	18,938	5372.30	97.64%	82.10%	97.37%

For sequential cores, simplistic TARO with dynamic compaction was applied and the resulting test patterns are presented in Table 3. Since each scan flip-flops are also counted as outputs in sequential cores, the total number of outputs a fault can propagate increased and, as a result, the number of fault-output pairs is relatively higher than combinational cores.

Table 3: TARO patterns for sequential cores

pattern	# patterns	# cmp pat	# f-o pairs
tops2901	18,938	8,611	84,925
lsi2901	307,048	298,018	2,666,322

The generated TARO test patterns were applied to all the combinational and sequential cores. All the defective chips from previously applied tests failed the test and the TARO had no escapes. Table 4 shows comparisons of SSF tests, transition tests, N-detect tests with N = 15 and TARO tests for ELF35 cores. Column 3 represents the test lengths, which are numbers of vectors for SSF and N-detect test and numbers of vector pairs for TR and TARO test. Column 4 shows CPU time in seconds. Next 3 columns represent SSF coverage, transition fault coverage, and TARO coverage, which is defined in Sec. 3.3. Lists of all fault-output pairs detected by TR and TARO test were referenced when calculating TARO coverage.

Finally, Table 5 shows test escapes of all four tests applied. While all other tests have at least 3 escapes, TARO test detected all the bad chips (no escapes).

Table 5: Test escapes of ELF35

core	m12	sqr	ma	pb	lsi2901	tops2901
ssf	2	2	0	0	2	0
tran	1	1	0	0	2	0
15ndet	1	1	0	0	2	0
taro	0	0	0	0	0	0

5. Conclusion

This paper shows how to generate TARO test patterns efficiently using currently available commercial ATPG tools. It also demonstrates the effectiveness of the patterns generated using this technique; TARO test patterns were generated for the ELF35 chips and used to test these chips. The results of applying both the TARO patterns as well as standard patterns (single stuck-at, N-detect, transition) are presented. The TARO patterns detected all of the defective ELF35 chips including those that escaped SSF, N-detect and transition tests. This demonstrates the importance of considering TO which outputs the faults are sensitized when doing ATPG.

Acknowledgement

This work was supported by LSI Logic under contract number 16517 and by NSF under contract number CSL-FY00-28.

The authors are, as usual, indebted to Dr. Subhasish Mitra of Intel and Stanford CRC for his very valuable comments and suggestions

References

[1] S. C. Ma, P. Franco, E. J. McCluskey, “An experimental chip to evaluate test techniques experiment results,” in Proc. 1995 Intl. Test Conf., pp.663-672, 1995

[2] E. J. McCluskey, C.W. Tseng, "Stuck-at Tests vs. Real Defects," in Proc. 2000 Intl. Test Conf., pp.336-343, 2000.

[3] E. J. McCluskey, A. Al-Yamani, C. W. Tseng, E. Volkerink, F. F. Ferhani, E. Li, S. Mitra, "ELF-Murphy data on defects and test sets," in Proc. 2004 VLSI Test Symp., pp.16-22, 2004

[4] C.W. Tseng, E. J. McCluskey, "Multiple-output Propagation transition fault test," in Proc. 2001 Intl. Test Conf., PP.358-366, 2001.

[5] I. Pomeranz, and S. M. Reddy, "On N-detection Test Sets and Variable N-detection Test Sets for Transition Faults," in Proc. 1999 VLSI Test Symp., pp.173-179, 1999.

[6] J. D. Lesser, J. J. Shedletsky, "An Experimental Delay Test Generator for LSI Logic," IEEE trans. on Computers, Vol. C-29, No.3, pp.235-248, 1980.

[7] J. Waicukauski, et. al., "Transition Fault Simulation, "IEEE Design and Test, pp. 32-38, April 1987.

[8] T. Etzion, P. R. J. Ostergard, "Greedy and heuristic algorithms for codes and colorings," IEEE trans. on Information Theory, Vol 44, Issue 1, pp.382-388, January 1998.