Transition Delay Fault Automatic Test Pattern Generation

Hsiang-Ting Wen[1], Fu-Yu Chuang[2], and Chen-Hao Hsu[1]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
[2]Graduate Institute of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
{r06943090, r06921041, r07943107}@ntu.edu.tw

1 INTRODUCTION

Testing of integrated circuits (ICs) is one of the most important process in very-large-scale integration (VLSI). It can not only guarantee IC quality, shorten time to market, but also enhance profit for foundries. A good fault modeling lies in the central part of a high-quality testing by quantifying test qualities and enabling *automatic test pattern generation* (ATPG). The single stuck-at fault (SSF) proposed by Galey *et al.* [5] is the simplest and most commonly adopted fault model to test the function of a circuit. However, to test the timing of a circuit, delay tests are required. In this project, we deal with the *transition delay fault* (TDF) model introduced in [1] with *launch-on-shift* (LOS) mode.

Moreover, to improve test quality, generating N-detect test pattern set has been shown to be an effective method [2, 12]. In N-detect, a fault will be dropped only when it is detected more than N times, and thus the quality and robustness of a test set would be improved. However, since N-detect test pattern set is much more larger than 1-detect test pattern set, the test cost significantly increases. To reduce test lengths as well as test costs, several test compression techniques should be adopted, including both *dynamic test compression* (DTC) and *static test compression* (STC).

In this project, we tackle with the N-detect TDF ATPG with test compression problem, which is formally defined as follows:

- **N-detect TDF ATPG with Test Compression Problem:** Given a circuit netlist, generate a N-detect test set to detect as many TDFs in LOS mode as possible while minimizing the test length.

The remainder of this report is organized as follows. Section 2 reviews previous works related to TDF ATPG and test compression. Section 3 details our proposed algorithm. Section 4 shows the experimental results and concluding remarks are given in Section 5.

2 PREVIOUS WORKS

In this section, we briefly review some previous works related to ATPG, testability measure, and test compression.

2.1 ATPG

Roth *et al.* first proposed the *D-algorithm* to generate test patterns for a combinational circuit by deciding values of not only primary inputs (PIs) but also internal nodes [10] to detect SSFs. Since D-algorithm allows assignments for internal signals, the search space of the algorithm is extremely large and thus makes it inefficient when dealing with complex combinational circuits [6]. As a result, Goel *et al.* proposed the *Path Oriented Decision Making* (PODEM) algorithm, which only makes decision at PIs [6]. Fujiwara *et al.* also proposed the *Fanout-orient test generation* (FAN) algorithm similar to the PODEM algorithm, except there were several speedup techniques including making decision at head lines and/or fanout stems [4]. In this project, our algorithm is based on the PODEM algorithm with modification to detect TDFs.

2.2 Testability Measure

One of the important components in the PODEM Algorithm is path selection during a backtrace to PIs. *Sandia Controllability Observability Analysis Program* (SCOAP) [8] serves as a good criterion for finding the easiest/hardest gate input. SCOAP computes the how difficult a node is to be controlled to a certain value and be observed at primary outputs (POs) by considering gate types and logic levels. It can also be used to evaluate how difficult a fault is to be detected.

2.3 Test Compression

As the scale and complexity of VLSI circuits increase, the amount of test patterns for testing a chip grows up rapidly. Thus, an efficient and effective test compression algorithm is desired to reduce test time and automatic test equipment (ATE) memory cost. DTC and STC are two common software techniques. For DTC, Goel *et al.* [7] proposed the PODEM-X algorithm which iteratively generates test patterns of secondary faults after a primary fault pattern is generated. Furthermore, for STC, the Quine-McClusckey method [9] can effectively reduce the test length but require a full fault dictionary which is quite memory-consuming. To do STC without a fault dictionary, Schulz *et al.* [11] suggests to perform *reverse-order fault simulation* to efficiently compact a set of X-filled test patterns.

3 PROPOSED ALGORITHMS

In this section, we first present our TDF ATPG algorithm overview, and then detail the methods used in each stage.

3.1 Algorithm Overview

Figure 1 shows the overview of our proposed algorithm. It is composed mainly of three stages: (1) fault list generation, (2) TDF ATPG, and (3) post-processing.

In the first stage, a TDF fault list is generated. It needs to be mentioned that, different from SSFs, fault collapsing should not be performed on any TDFs.

The second stage is the main algorithm we proposed to do TDF ATPG. It consists of three components, including fault ranking, modularized edition of PODEM (MEDOP), and static test compression. During the fault ranking, the initial fault list is sorted according to the quality of the associated test pattern generated for a fault. The resulting ranked fault list is then used in MEDOP to select faults one-by-one. Then MEDOP, which is modified from the original PODEM-X algorithm [7], is adopted to generate test patterns for TDFs. Also, during MEDOP, DTC is performed to reduce the test length. Finally, some STC techniques are adopted to further compress the test set to obtain a shorter test length.

In the last stage, we perform post-processing on the generated test patterns. Since there may exist some faults which have been detected, but less than *N* times, the test patterns which detect these faults will be duplicated until they are successfully detected *N* times. This stage makes sure that all faults which can be detected by the generated patterns are detected *N* times.

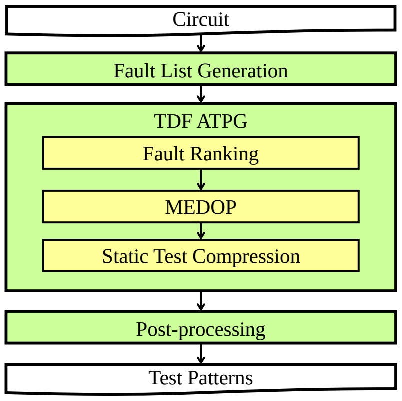


Figure 1: Our algorithm overview.

3.2 Fault Ranking

In MEDOP, a fault is selected as a target fault at a time and then a test pattern will be generated to detect it. The generated test pattern will then be applied to the circuits to check if other faults are also detected by it. A pattern which can detect more faults at a time is considered as a better pattern. Thus, a fault which is able to generate a better pattern is preferred to be selected first to drop more faults in the early stage of the whole algorithm.

To achieve this goal, we will first iterate through each fault, and try to generate a test pattern to detect it. The generated test pattern is then applied to the circuit to do a fault simulation to evaluate how "good" the pattern is. Several criteria can be considered during the evaluation:

- **Number of unknown bits in the pattern**: The more unknown bits in a pattern, the more possible it can be filled to generate a pattern to detect more faults. Thus, the more unknown bits a pattern is, the earlier the corresponding fault should be selected.
- **Number of faults detected by the pattern**: The more faults a pattern detect, the better it is. Thus, the corresponding fault should also be selected earlier.
- **Summation of SCOAP of faults detected by the pattern**: As mentioned in the previous section, the SCOAP can serve as an evaluation of how difficult a fault is to be detected. By summing up the SCOAP of faults detected by a pattern, the number can be a helpful criterion to evaluate the pattern.

Then, the fault list can be sorted according to the criteria mentioned above, from the one with a better generated pattern to the one with a worse one. The whole fault ranking flow is shown in Figure 2.

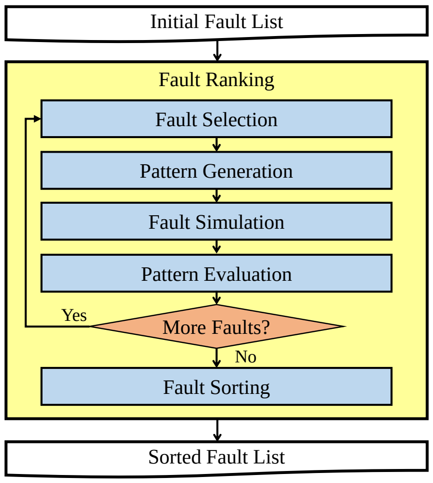


Figure 2: Fault ranking flow.

3.3 MEDOP

The *modularized edition of PODEM* (MEDOP) is our main algorithm for test pattern generation and DTC. Figure 3 shows the MEDOP flow. In each pass of MEDOP, one fault will be selected as a primary fault and we would try to generate a test pattern for it. If a test pattern is found, we can perform DTC based on it. We would iteratively select secondary faults and try to generate a test pattern which can detect that secondary fault and all previous primary/secondary faults, until the current pattern no longer has enough unknown values to be filled.

In our MEDOP algorithm, a fault will not be dropped from the fault list unless it is detected N times. Therefore, faults which are difficult to detect are very likely to be selected as a primary fault for at least once. This guarantees that we always search a pattern for a difficult fault under an unconstrained condition, which may improve the fault coverage.

Figure 4 shows the detailed flow of the function "pattern generation" in Figure 3. To detect TDFs in LOS mode, test patterns of two consequent time frames (V1 and V2) should be generated. For each test pattern, we must ensure that V1 and V2 can activate a fault, and V2 can propagate it. Also, conflicting PI assignments between V1 and V2 must not be made under LOS mode.

To generate a pattern for a fault, we first perform a backward implication for V1 and check whether the V1 constraint is violated due to some structural restriction. This step helps prune undetectable faults in an early stage and accelerates the whole pattern generation process.

Secondly, we apply the PODEM algorithm for generating a V2 pattern for a fault. If a V2 pattern cannot be found, we would drop this fault immediately. Otherwise, the V2 pattern would be used as an initial condition for V1 pattern generation. The current state of the V2 pattern decision tree is also recorded for potential backtracks from V1 pattern generation.

Then, we shift the V2 pattern one bit backward on the scan chain and use the PI assignments as an initial condition (and also a constraint) for V1 pattern generation. The V1 pattern is generated by the PODEM algorithm without the propagation part. If a V1 pattern is not found, we would perform a backtrack to V2 pattern

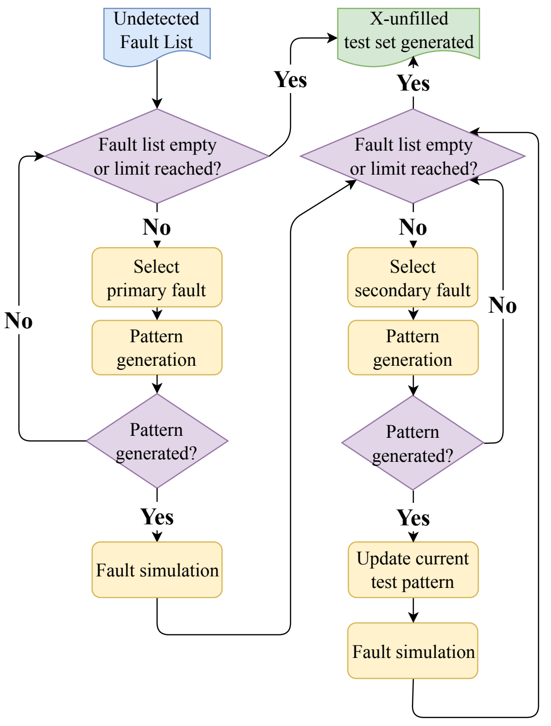
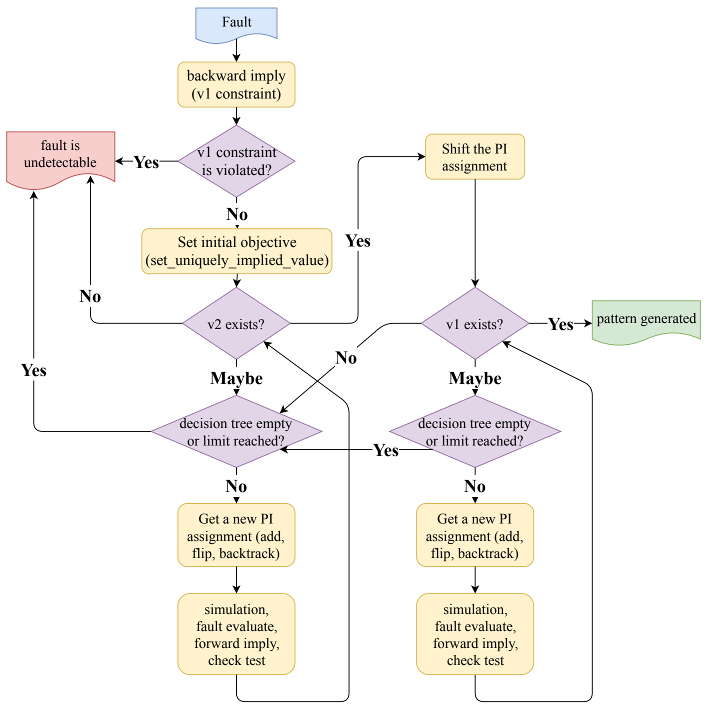


Figure 3: MEDOP flow.



Fault
backward imply (v1 constraint)
v1 constraint is violated?
fault is undetectable
Set initial objective (set_uniquely_implied_value)
Shift the PI assignment
v2 exists?
v1 exists?
pattern generated
Yes
No
Maybe
decision tree empty or limit reached?
Get a new PI assignment (add, flip, backtrack)
simulation, fault evaluate, forward imply, check test

Figure 4: pattern generation flow.

generation since no test under one specific V2 pattern does not mean that this fault is undetectable. Note that the decision tree of V1 generation and that of V2 generation are different trees. Therefore, the recorded decision tree of V2 should be restored and used as an initial state to generate another V2 pattern.

These two steps would be repeated until one legal V2-V1 pattern is generated or the backtrack limit is reached.

3.4 Static Test Compression

To reduce the size of the X-unfilled test pattern set generated in the previous stages, we propose a STC framework that integrates compatibility graph which is applied for X-unfilled patterns and random-order fault simulation which is applied for both X-unfilled and X-filled patterns. Figure 5 shows our proposed STC flow.

First of all, we do random-order fault simulation on the X-unfilled test set to drop the useless patterns. In our implementation, the number of iterations varies from 20 to 50, which depends on the size of circuits. Since simulation on larger circuits takes more CPU runtime, we set less iteration number to larger circuits.

To compact X-unfilled patterns, we build a compatibility graph $G(V, E)$, where each $v \in V$ represents a test pattern and an edge $e_{v_i,v_j} \in E$ indicates that the corresponding test patterns of v_i and v_j are compatible.

Definition 1. *Two test vectors are* compatible *if and only if there is no conflict in their specified bits.*

The test compression problem thus becomes the *minimum clique partition (MCP) problem* on a compatibility graph, which is known to be an NP-hard problem. To obtain a solution in reasonable runtime, we adopt *Tseng-Siewiorek algorithm* which iteratively merges two adjacent vertices of maximum common neighbors into a *supervertex* until no more edges. The test patterns whose corresponding vertices belong to the same supervertex will be merged into a single pattern. A merged pattern is considered to become *powerful* since it can detect more faults than before. However, the reduction of the test size may cause fault coverage to decrease due to the N-detect constraint. Therefore, we duplicate the whole test set n times, where n is the maximum size of all supervertices.

After the test set expansion, we randomly fill all unknown values to make all X-unfilled patterns become X-filled patterns since an X-unfilled pattern usually detects less faults than it is X-filled. Finally, we perform random-order fault simulation again to further shorten the X-filled test set.

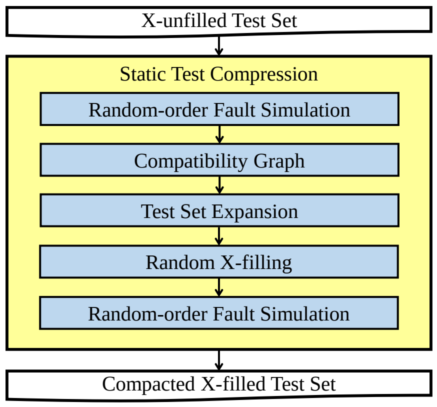


Figure 5: Static test compression flow.

4 EXPERIMENTAL RESULTS

We implemented our algorithm in the C++ programming language. All experiments were performed on a Linux workstation with Xeon 3.5 GHz CPUs and 72GB memory. The experiments were conducted on ISCAS'85 benchmark circuits [3], where the benchmark statistics is summarized in Table 1.

Table 1: Benchmark statistics. #PI, #PO, #Gate, #Wire, and #TDF denote the numbers of primary inputs, primary outputs, gates, wires, and transition delay faults, respectively.

Circuit	#PI	#PO	#Gate	#Wire	#TDF
c432	36	7	245	281	1110
c499	41	32	554	595	2390
c880	60	26	545	605	2104
c1355	41	32	554	595	2726
c2670	233	140	1785	2018	6520
c3540	50	22	2082	2132	7910
c6288	32	32	4800	4832	17376
c7552	207	108	5679	5886	19456

To evaluate the solution quality of our TDF ATPG algorithm, we applied our proposed TDF ATPG algorithm with DTC and STC techniques to generate 8-detect test sets. Table 2 gives our 8-detect results.

Table 2: 8-detect results of our TDF ATPG algorithm with test compression techniques. TL, #DF, FC, and CPU denote test length, number of detected faults, fault coverage, and CPU runtime, respectively.

Circuit	TL	#DF	FC (%)	CPU (s)
c432	152	129	11.62	1.2
c499	499	2285	95.61	7.8
c880	301	1060	50.38	8.6
c1355	407	1047	38.41	6.7
c2670	797	6109	93.70	119.2
c3540	461	1840	23.26	40.5
c6288	361	16966	97.64	341.4
c7552	1560	19102	98.18	176.8
Average	567	6067	63.60	87.8

To see the effects on test length and fault coverage for different N, we generated N-detect results for each case, where $1 \leq N \leq 8$. From Figure 6(a), we can see that the test lengths grow up *linearly* as N increases, while the fault coverage for each case remains almost the same. To maintain the same fault coverage with higher N, the test length should increase which shows the results are reasonable.

In order to evaluate the effectiveness of test compression, we conducted experiments to generate 1-detect results and 8-detect results with and without our proposed test compression techniques. Table 3 shows the results. On average, the test length reductions (TLR) for 1-detect and 8-detect results are very high. In particular, the highest TLRs are 94.46% and 94.75% for 1-detect and 8-detect results respectively. The results illustrate that our test compression techniques are effective to reduce test lengths and also efficient to compact test sets.

To show the effectiveness of DTC and STC individually, we further conducted experiments to generate 8-detect results by using DTC only, STC only, and both DTC and STC. Table 4 gives the results which show that for most cases, our test compression techniques worked the best when both DTC and STC were applied. However, for `c3540` and `c6288`, our algorithm obtained the best results when only STC was applied than when both DTC and STC were applied. It might result from the higher number of X-unfilled patterns when only STC was applied.

5 CONCLUSIONS

We proposed a framework which is able to perform TDF ATPG with N-detect and test compression. We also proposed some novel ideas for improving fault coverage, reducing test length and reducing runtime. First, we proposed a smarter evaluation method for fault ranking, which could help improve the efficiency and effectiveness of fault selection. We implemented MEDOP, a modularized PODEM-based process which can support N-detect TDF ATPG and DTC with good fault coverage in acceptable runtime. We proposed a series of STC techniques, which are proved to be effective and efficient for reducing the test length.

REFERENCES

[1] Z. Barzilai. Comparison of ac self-testing procedures. In *International Test Conf., 1983*, 1983.

[2] B. Benware, C. Schuermyer, S. Ranganathan, R. Madge, P. Krishnamurthy, N. Tamarapalli, K.-H. Tsai, and J. Rajski. Impact of multiple-detect test patterns on product quality. In *null*, page 1031. IEEE, 2003.

[3] F. Brglez. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in fortran. In *Int. Symposium on Circuits and Systems, Special Session on ATPG and Fault Simulation, June 1985*, pages 663–698, 1985.

[4] H. Fujiwara. On the acceleration of test generation algorithm. *IEEE transactions on Computers*, 30(3):215–222, 1978.

[5] J. M. Galey, R. E. Norby, and J. P. Roth. Techniques for the diagnosis of switching circuit failures. *IEEE Transactions on Communication and Electronics*, 83(74):509–514, 1964.

[6] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE transactions on Computers*, (3):215–222, 1981.

[7] P. Goel and B. C. Rosales. Podem-x: An automatic test generation system for vlsi logic structures. In *Proceedings of the 18th Design Automation Conference*, pages 260–268. IEEE Press, 1981.

[8] L. H. Goldstein and E. L. Thigpen. Scoap: Sandia controllability/observability analysis program. In *Design Automation, 1980. 17th Conference on*, pages 190–196. IEEE, 1980.

[9] E. J. McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[10] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM journal of Research and Development*, 10(4):278–291, 1966.

[11] M. H. Schulz, E. Trischler, and T. M. Sarfert. Socrates: a highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):126–137, 1988.

[12] C.-W. Tseng and E. J. McCluskey. Multiple-output propagation transition fault test. In *Test Conference, 2001. Proceedings. International*, pages 358–366. IEEE, 2001.

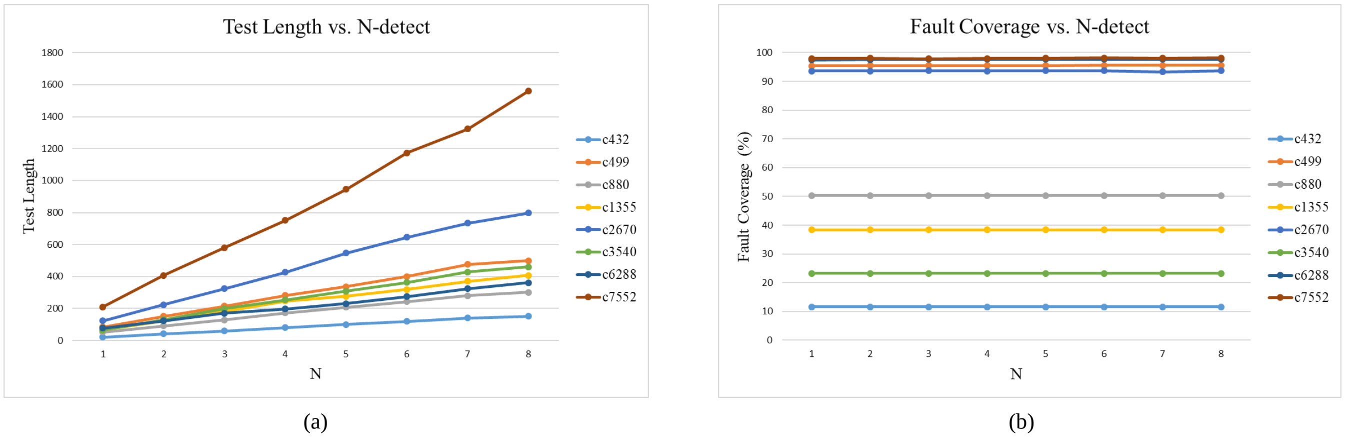


Figure 6: (a) The line charts of test length against N for each benchmark. (b) The line charts of fault coverage against N for each benchmark.

Table 3: 1-detect results and 8-detect results with and without test compression. TLR denotes test length reduction.

	1-detect							8-detect						
	w/o compression			w/ compression				w/o compression			w/ compression			
Circuit	TL	FC (%)	CPU (s)	TL	FC (%)	CPU (s)	TLR (%)	TL	FC (%)	CPU (s)	TL	FC (%)	CPU (s)	TLR (%)
c432	41	11.62	0.1	21	11.62	0.2	48.78	320	11.62	0.1	152	11.62	1.2	52.50
c499	255	95.36	2.4	85	95.36	3.2	66.67	1837	95.61	2.9	499	95.61	7.8	72.84
c880	425	50.38	3.6	52	50.38	4.4	87.76	3427	50.38	4.0	301	50.38	8.6	91.22
c1355	94	38.41	0.9	64	38.41	1.6	31.91	648	38.41	1.0	407	38.41	6.7	37.19
c2670	1666	93.71	60.5	123	93.63	72.8	92.62	13823	93.94	70.4	797	93.70	119.2	94.23
c3540	284	23.26	6.9	64	23.26	12.6	77.46	2149	23.26	8.5	461	23.26	40.2	78.55
c6288	157	97.56	286.6	78	97.43	294.2	50.32	1727	97.66	291.2	361	97.64	341.4	79.10
c7552	3791	98.10	65.83	210	97.92	77.5	94.46	29742	98.14	122.9	1560	98.18	176.8	62.1
Average	839	63.55	53.41	87	63.50	58.3	68.75	6709	63.63	62.6	567	63.60	87.8	52.9

Table 4: 8-detect results by using DTC only, STC only, and both DTC and STC.

	DTC only				STC only				DTC + STC			
Circuit	TL	#DF	FC (%)	CPU (s)	TL	#DF	FC (%)	CPU (s)	TL	#DF	FC (%)	CPU (s)
c432	304	129	11.62	0.1	152	129	11.62	1.2	152	129	11.62	1.2
c499	932	2285	95.61	2.6	566	2283	95.52	8.6	499	2285	95.61	7.8
c880	723	1060	50.38	4.5	303	1060	50.38	18.5	301	1060	50.38	8.6
c1355	648	1047	38.41	1.0	407	1047	38.41	6.7	407	1047	38.41	6.7
c2670	1664	6107	93.67	68.6	999	6127	93.97	289.2	797	6109	93.70	119.2
c3540	859	1840	23.26	9.0	420	1840	23.26	50.7	461	1840	23.26	40.2
c6288	1700	16965	97.63	295.2	344	16969	97.66	337.9	361	16966	97.64	341.4
c7552	3635	19091	98.12	78.7	2011	19104	98.19	360.5	1560	19102	98.18	176.8
Average	1308	6066	63.59	57.5	650	6070	63.63	134.1	1560	19102	98.18	176.8