PoDemFan N-detect ATPG Test Compression

Group 1
Yu-Hung Pan, Wei-Shen Wang, Hsin-Tzu Chang

Dept. of Electrical Engineering, National Taiwan University, Taipei, Taiwan

b08901050@ntu.edu.tw, b08901051@ntu.edu.tw, b08901053@ntu.edu.tw

***Abstract*—This work presents the implementation method of our N-detect Automatic Test Pattern Generation (ATPG) Test Compression - PoDemFan. We implemented two ATPG algorithms, PODEM and FAN for transition delay fault. We also implemented parallel processing to choose the better pattern set generated by the two algorithms we implemented.**

***Index Terms*—ATPG, FAN, PODEM, N-detect, Test Compression**

I. Problem Description

The problem formulation involves a circuit netlist and a positive integer N. The objective is to generate a test pattern set to detect transition delay faults with N patterns under the Launch-on-Shift (LOS) mode for this circuit.

In this context, a fault in the circuit is marked as $detected$ only if it is detected by at least N patterns in the generated test pattern set. Additionally, V1 and V2 must adhere to the relation specified by the LOS mode.

The primary goal of our program is to achieve high fault coverage while reducing test length and execution time.

II. Past Research

A. PODEM algorithm [1]

PODEM algorithm makes decision at primary inputs (PI). It only performs forward implication and does not need justification. It uses 3 heuristics while doing backtrace to assign PIs and fault propagation to POs :

1) Backtrace path:

When the gate is decision gate, that is, only one input can control the gate output to objective value, we choose the easiest gate input. When the gate is imply gate, that is, one input can't control the gate output to objective value, we choose the hardest gate input.

2) Backtrace value:

Assign the same value as objective if the path has even inversion parity. Assign the opposite value to objective if the path has odd inversion parity.

3) Propagate path:

Choose the shortest X-path to PO. Backtrack the assigned PIs if all X-path disappears. When backtracking PIs assignment, flip the value of the last assigned PI.

B. FAN algorithm [2]

FAN algorithm has several improvements based on PODEM algorithm, and its flow chart is shown in Fig. 1.

First, it makes decisions at headlines or fanout stems, which achieves better trade-off between performance and runtime compared to D-algorithm and PODEM algorithm. Second, in addition to Forward Implication used in PODEM algorithm, FAN algorithm also adopts Backward Implication, which helps to make correct decisions. Third, if there is only one gate in the D-frontier, FAN algorithm performs additional unique sensitization step.

Last but not least, in contrast to the single backtrace in PODEM algorithm, FAN adopted multiple backtraces. Multiple backtraces use breadth-first search (BFS), while single backtrace applies depth-first search (DFS), so FAN algorithm has higher efficiency when it comes to backtracing.

In summary, FAN algorithm applies branch-and-bound concept in order to detect inconsistency as early as possible, and it wisely chooses the headlines and fanout stems to assign values.

C. Benware [3]

For N-detect ATPG, this work proposes to do N rounds to generate the N-detect pattern. When we are in the K-th round, we choose the target fault that has been detected exactly $K-1$ times.

III. Proposed Techniques

A. PODEM algorithm

We modify the source code of PODEM provided by the VLSI Testing course in National Taiwan University and implemented the N-detect transition delay fault ATPG as the baseline for our project. The modified parts are specified in README.md. The typical approach for ATPG in detecting transition delay faults is to first generate V2 and then generate V1 based on the fixed V2. However, a drawback of this method is that if V1 cannot be successfully generated due to constraints from V2, the algorithm fails.

Nevertheless, it is possible that another V2 pattern can also detect the fault, and we can find a fault-activating V1 pattern that is compatible with this V2 pattern. Therefore, by searching back and forth on the decision trees of V1 and V2, we can achieve a more complete solution space and obtain the results with higher fault coverage.

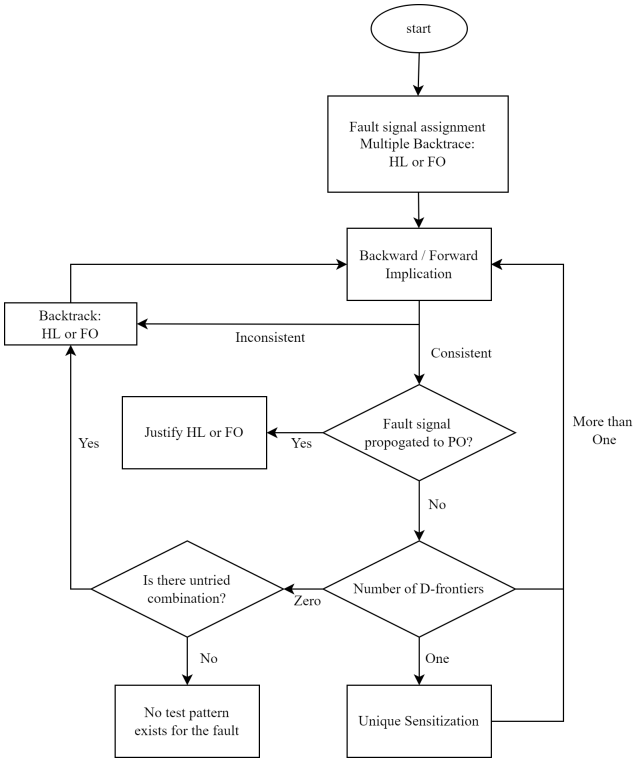

Fig. 1. Flow chart of FAN algorithm

The overall flow of the algorithm is shown in Fig. 2. The following paragraphs explain the four primary parts: Test Pattern Generation for Primary Faults, Primary Fault Selection, Dynamic Test Compression (DTC), and Static Test Compression (STC). The experimental results will be presented in the next section.

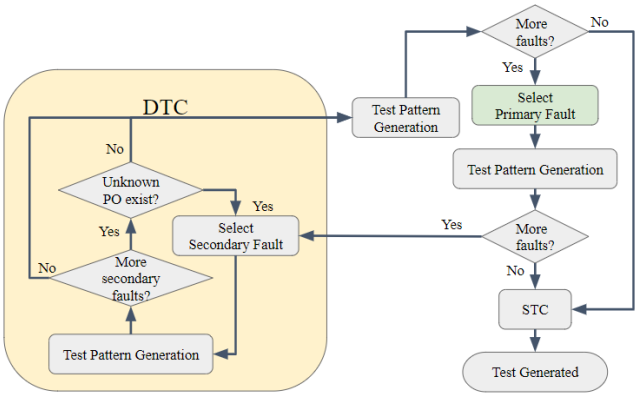

Fig. 2. Overall Flow of PODEM algorithm

1) Test Pattern Generation for Primary Faults:

We introduce two boolean flags, valid_v1 and valid_v2, to keep track of the states of V1 and V2 generation.

For a selected fault, we first backtrace one level for fault activation. If a conflict occurs, the test generation fails. If no conflict occurs, we set valid_v1 to true if the fault has been activated, and false if it hasn't. The flow chart is shown in Fig. 3.

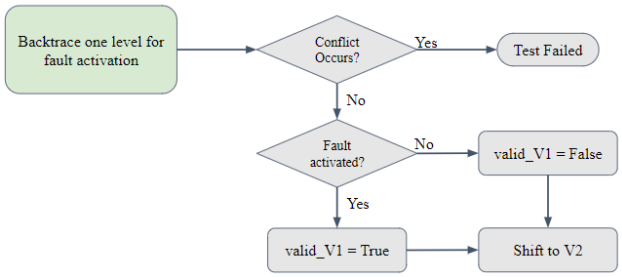

Fig. 3. Backtracing for Fault Activation

Next, we shift the V1 pattern and backtrace one level for fault detection. Similarly, we set the value of valid_v2 according to whether the fault is detected. Now, we have four possible combinations of the values of valid_v1 and valid_v2, and we perform concurrent backtracking according to the different states. Table I shows the actions of concurrent backtracking based on the values of the two flags.

TABLE I
CONCURRENT BACKTRACK: ACTIONS UNDER DIFFERENT STATES

	valid_V2 = True	valid_V2 = False
valid_V1 = True	Test generated	Backtrack V2
valid_V1 = False	Backtrack V1	Backtrack V1 then V2

If both valid_v1 and valid_v2 are true, then we have successfully found the test. If only one of the flags is false, for example, valid_v1 is false, we backtrack on V1 and then update the state. If both flags are false, backtracking on V1 will be performed first, and then we backtrack on V2. The procedure is repeated until a test is successfully generated or the program reaches the user defined backtrack limit.

By performing concurrent backtrack, the algorithm has higher flexibility and is more likely to generate valid test patterns.

2) Primary Fault Selection:

We implement two flows for selecting the primary fault for each round. In flow 1, we repeatedly choose the same fault until it is N-detected or the ATPG fails to generate test patterns. In flow 2, we adopt the concept from the paper [3]. The process involves N rounds of test pattern generation, from $K = 1$ to $K = N$. For each round, we traverse the fault list and only select the fault with the current detected time exactly equals to $K-1$. Note that the detections in DTC are also taken into account.

3) Dynamic Test Compression:

We apply the PODEM-X algorithm [4] for DTC. In order to improve runtime, the test pattern generation for secondary faults is simplified. We generate V2 first and then generate V1 based on V2. If V1 fails, no backtracking is applied to V2.

For the secondary fault selection, we choose a primary output (PO) with an unknown value and apply a breadth-first

search (BFS) starting from the selected PO. Candidate faults are pushed into a queue and tried in order.

4) Static Test Compression:

The flow chart of STC is shown in Fig. 4. First we apply reverse order fault simulation to remove redundant test patterns. This greatly reduces the test length. For further improvement, we apply several times of random order fault simulation. For each round, we do fault simulation for each pattern with random order and remove the redundant faults. This procedure would be repeated until there is no further improvement for 5 consecutive rounds.

Fig. 4. Flow chart of Static Test Conpression

B. PODEM V1 + FAN V2

In this part, we implement FAN ATPG utilizing the open source code FAN ATPG [5]. The original open source code only generate pattern set for stuck-at faults. Therefore, we need to implement transition delay fault LOS ATPG by adding some functions to the open source code. The modified part are specified in README.md of the github repository. The overall ATPG flow is shown in Fig. 5.

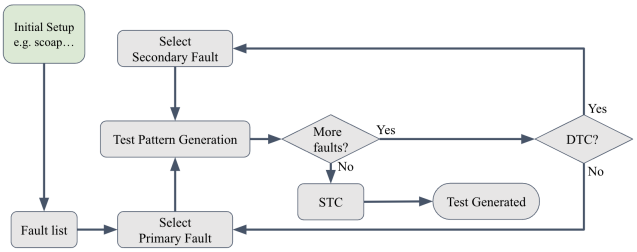

Fig. 5. Overall Flow of FAN+PODEM algorithm

1) Implemented Heuristic:

First, we introduce some of the heuristics we implemented. We first calculate the SCOAP [6] parameters including 0-controllability (CC0), 1-controllability (CC1) and observability (CO) of every gate output. Each fault in the fault list extracted from the circuit has corresponding gate output. In order to detect hard-to-detect faults first, we sort the fault list according to its CO from large to small. We also utilize CC0 and CC1 when choosing the easiest/hardest input during the backward implication in FAN and backtracing in PODEM. Furthermore, we reverse the remaining fault list temporary for the search of secondary fault during DTC stage. The reason is that after some bits of the pattern is already specified for detecting the primary fault, the pattern has lower possibility to detect hard-to-detect faults. Therefore, we search easy-to-detect faults for secondary fault.

2) Test Pattern Generation:

There are two flows we implemented. The first and default flow is V1 first test pattern generation. It involves using the PODEM algorithm to generate V1 first, and then we shift the pattern and do logic simulation. With some gates' values specified, we use the values as constraint and use the FAN algorithm to generate V2. The flow chart is shown in Fig. 6. The second and backup flow is V2 first test pattern generation. It involves using the FAN algorithm to generate V2 first, and then we shift the pattern back and do logic simulation. With some gates' values specified, we use the values as constraint and use the PODEM algorithm to generate V1. The flow chart is shown in Fig. 7.

In this implementation, we try flow1 (V1 first test pattern generation) first. If flow1 fails, we will try flow2 instead. This result in longer runtime but better fault coverage. The flow chart is shown in Fig. 8.

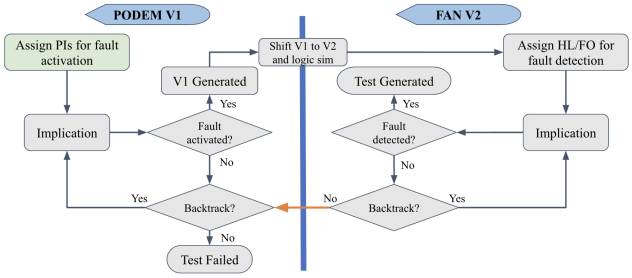

Fig. 6. V1 First Test Pattern Geneartion

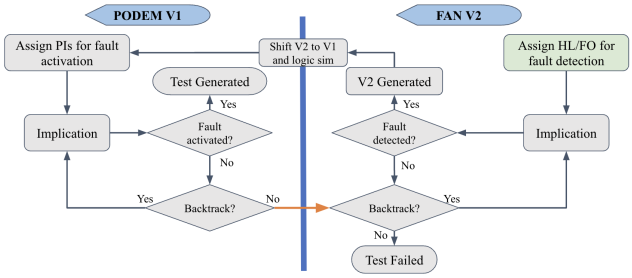

Fig. 7. V2 First Test Pattern Generation

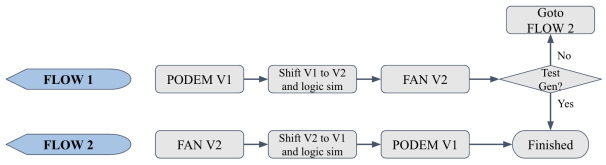

Fig. 8. Flow1 and Flow2 procedure

3) Test Compression:

In the STC phase, we first reverse the generated pattern set's order to do fault simulation and drop unnecessary patterns.

Then, we try multiple random fault simulation to keep dropping unnecessary patterns. The test pattern set will be decided at last if the number of patterns stops decreasing, as shown in Fig. 9. The DTC phase is similar to that of the original paper FAN [2] and a flow chart is provided in Fig. 9 .

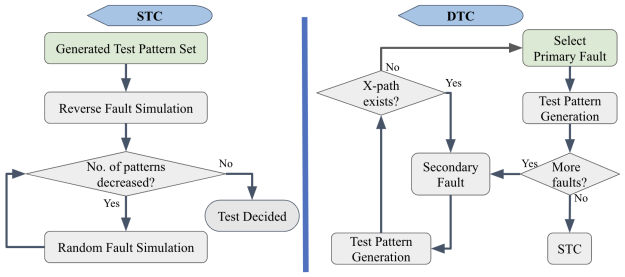

Fig. 9. FAN's STC and DTC

C. Parallel processing

We have proposed two ATPG algorithms in the previous subsections. From our observation, some benchmark circuits are able to have better test pattern sets with PODEM algorithm, while the others have better test pattern sets with PODEM V1 + FAN V2. If we run the two ATPG algorithms simultaneously, we can compare the two generated pattern set and choose the better one. Here, we choose the pattern set with higher fault coverage first, since fault coverage is the most important, then we choose the pattern set with smaller test length if the fault coverage are the same. With the described method, we improve the overall fault coverage and test length, with negligible trade off in runtime.

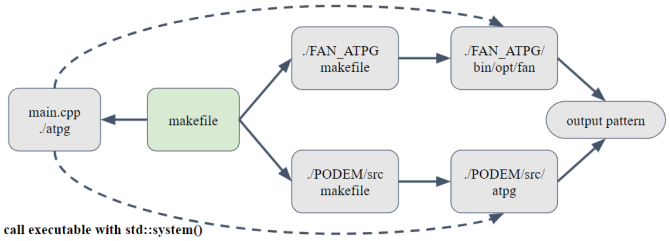

Fig. 10. Makefile

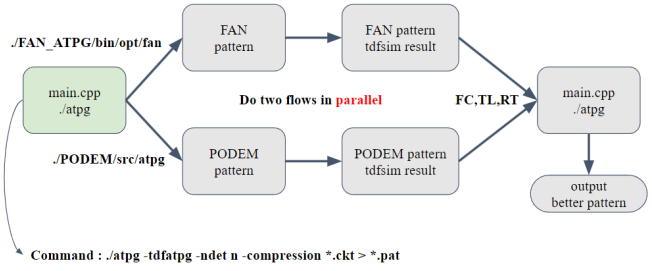

Fig. 11. Parallel processing flow

The flow of parallel processing is shown in Fig. 10 and Fig. 11. Fig. 10 shows how we generate the executable. Both ATPGs have their own makefiles. After typing $make$, the two separate makefile will be executed and compile their ATPG executable in specific directories. There is also a $main.cpp$, which will produce the main executable $./atpg$. When we execute the main executable, it will read the input arguments and generate the proper commands for the two ATPG executables. Finally, it will run the commands using $std :: system()$, and the ATPG pattern will be generated.

Fig. 11 shows the parallel processing flow starting from the main executable. After we type the command shown in Fig. 11, it will call the two ATPG executables and perform ATPG algorithms on the circuit, generateing the pattern sets. Then the patterns will be evaluated by fault simulation. The corresponding fault coverage, test length, and runtime will be sent back to the main executable. Finally, we output the better pattern set based on the fault coverage and test length.

The advantage of this technique is that the fault coverage has the potential to be higher than using only one ATPG, since we choose the pattern set with higher fault coverage between the two pattern sets. In addition, the test length might also be reduced after doing parallel processing. However, the drawback is that the runtime will be dominated by the larger runtime between two ATPGs. It become worse when the difference of two runtimes is too large. One way to alleviate this is to rerun the faster ATPG algorithm while waiting the slower ATPG to finish. By doing so, we may find potentially better pattern set when we rerun other ATPG algorithms.

IV. Experiment Results

A. PODEM algorithm

We expand the table given in the instruction document provided by VLSI Testing course in National Taiwan University and add the data when only STC is applied in the PODEM algorithm. Table II shows the results under different test compression techniques for normal transition delay fault ATPG ($N = 1$). Table III and Table IV show the results of the two flows under different test compression techniques for 8-detect ATPG.

For the case of $N = 1$, STC greatly reduces the test length while maintaining the same fault coverage. By adding DTC, the fault coverage for some cases slightly dropped, but improved on average by 0.01%. Moreover, the test length reduction rate increased from 30.25% to 44.17% with DTC.

For the case of $N = 8$, the effect of STC is more noticeable in Flow 1. However, we believe that this is because the initial test length in Flow 2 is much shorter than in Flow 1. By adding DTC techniques, the fault coverage improved by 0.05% for both flows, and the test length reduction rate grew to 45.11% and 33.69% respectively. The average runtime of the two flows with both STC and DTC is 21.96 and 18.9 seconds respectively, which is highly satisfactory.

Lastly, we compare the results of the two flows with $N = 8$ and compression in Table V. The fault coverage is slightly higher for Flow 1, but the test length and runtime of Flow 2 are approximately 10% and 14% less than those of Flow 1. In summary, both flows achieve a fault coverage of 63.55%,

TABLE II
PODEM - COMPARISON BETWEEN DIFFERENT TEST COMPRESSION TECHNIQUES (NDET = 1)

Circuit	w/o compression			STC only				DTC + STC			
	TL	FC	RT(s)	TL	FC	RT(s)	TL reduction	TL	FC	RT(s)	TL reduction
C432	24	11.62%	0.1	23	11.62%	0.1	4.17%	23	11.62%	0.1	4.17%
C499	140	94.69%	1.6	89	94.69%	1.8	36.43%	79	94.85%	1.8	43.57%
C880	113	50.38%	0.5	74	50.38%	0.5	34.51%	56	50.38%	0.8	50.44%
C1355	75	38.41%	0.1	59	38.41%	0.1	21.33%	60	38.41%	0.3	20.00%
C2670	248	94.13%	3.1	187	94.13%	3.6	24.60%	149	94.00%	4.4	39.92%
C3540	126	23.26%	9.7	84	23.26%	9.4	33.33%	75	23.26%	11	40.48%
C6288	109	97.66%	0.6	73	97.66%	1.6	33.03%	73	97.65%	2.6	33.03%
C7552	494	98.28%	3.5	338	98.28%	7.2	31.58%	227	98.32%	12.5	54.05%
Average	166.13	63.55%	2.40	115.88	63.55%	3.04	30.25%	92.75	63.56%	4.19	44.17%

TABLE III
PODEM - COMPARISON BETWEEN DIFFERENT TEST COMPRESSION TECHNIQUES (NDET = 8, FLOW 1)

Circuit	w/o compression			STC only				DTC + STC			
	TL	FC	RT(s)	TL	FC	RT(s)	TL reduction	TL	FC	RT(s)	TL reduction
C432	24	11.62%	0.1	23	11.62%	0.1	4.17%	23	11.62%	0.1	4.17%
C499	1072	94.60%	2.1	615	94.60%	2.8	42.63%	600	94.85%	2.8	44.03%
C880	722	50.38%	0.6	427	50.38%	1.2	40.86%	345	50.38%	3	52.22%
C1355	600	38.41%	0.2	474	38.41%	1.1	21.00%	469	38.41%	1.1	21.83%
C2670	1785	94.06%	3.9	1291	94.06%	9.4	27.68%	983	94.06%	17.4	44.93%
C3540	126	23.26%	9.7	608	23.26%	15.5	36.27%	518	23.26%	28.4	45.70%
C6288	461	97.59%	1.3	358	97.59%	17.1	22.34%	410	97.63%	26.1	11.06%
C7552	3246	98.26%	6.5	2105	98.26%	80.7	35.15%	1454	98.33%	96.8	55.21%
Average	1127.25	63.52%	3.05	756.13	63.52%	15.99	32.92%	618.75	63.57%	21.96	45.11%

TABLE IV
PODEM - COMPARISON BETWEEN DIFFERENT TEST COMPRESSION TECHNIQUES (NDET = 8, FLOW 2)

Circuit	w/o compression			STC only				DTC + STC			
	TL	FC	RT(s)	TL	FC	RT(s)	TL reduction	TL	FC	RT(s)	TL reduction
C432	179	11.62%	0.1	170	11.62%	0.2	5.03%	170	11.62%	0.2	5.03%
C499	684	94.35%	2.1	633	94.35%	2.6	7.46%	582	94.69%	2.7	14.91%
C880	528	50.38%	0.5	461	50.38%	1	12.69%	299	50.38%	1.5	43.37%
C1355	477	38.41%	0.2	456	38.41%	0.6	4.40%	463	38.41%	0.8	2.94%
C2670	1418	94.06%	4.2	1321	94.06%	9.6	6.84%	874	94.06%	12.6	38.36%
C3540	665	23.26%	9.5	607	23.26%	15.2	8.72%	486	23.26%	14.6	26.92%
C6288	424	97.63%	1.2	370	97.63%	15.7	12.74%	387	97.63%	19.4	8.73%
C7552	2351	98.25%	5.9	2097	98.25%	53	10.80%	1199	98.31%	99.4	49.00%
Average	840.75	63.50%	2.96	764.38	63.50%	12.24	9.08%	557.5	63.55%	18.9	33.69%

a test length below 619, and a runtime under 22 seconds on average. Even for the largest benchmark circuit, the runtime of both flows is under 100 seconds, which are reasonable values.

B. PODEM V1 + FAN V2

See TABLE VI for the result of comparison between with and without compression when $N = 8$, TABLE VII for the result of comparison between with and without compression when $N = 1$ and TABLE VIII for the result of comparison between withour compression, STC only and STC+DTC when $N = 8$.

C. Parallel processing

See TABLE IX. The yellow cells indicate the pattern we choose. Overall, the fault coverage increases, the test length decreases, and the runtime increases a little.

V. DISCUSSION

A. PODEM vs FAN+PODEM

In this work, we implement two ATPGs. The first one is the PODEM algorithm with concurrent backtrack. The second one is FAN algorithm for V2 and PODEM algorithm for V1 with heuristics regarding SCOAP implemented. The result shows that in average, the FAN+PODEM algorithm we implemented produce higher fault coverage where as the PODEM algorithm we implemented produce lower test length. However, different algorithm tends to perform better on different circuits as shown in the experimental results. We conclude that this is the due to the different circuits' logic structure. As for the runtime, we can see that the runtime of FAN+PODEM is longer than that of PODEM's. We believe the reason is not related to the algorithm itself, but rather the C++ code implementation. We

TABLE V
RESULT OF THE TWO FLOWS WITH PODEM ALGORITHM (NDET = 8, COMPRESSION ON)

Circuit	Flow 1			Flow 2		
	FC	TL	RT(s)	FC	TL	RT(s)
C432	11.62%	171	0.1	11.62%	170	0.2
C499	94.85%	600	2.8	94.69%	582	2.7
C880	50.38%	345	3	50.38%	299	1.5
C1355	38.41%	469	1.1	38.41%	463	0.8
C2670	94.06%	983	17.4	94.06%	874	12.6
C3540	23.26%	518	28.4	23.26%	486	14.6
C6288	97.63%	410	26.1	97.63%	387	19.4
C7552	98.33%	1454	96.8	98.31%	1199	99.4
Average	63.57%	618.75	21.96	63.55%	557.5	18.9

TABLE VI
PODEM V1 + FAN V2 - NDET = 8

ndet=8	w/o compression			w/ compression			
Circuit	TL	FC	RT(s)	TL	FC	RT(s)	TL↓
C432	203	11.62%	0.2	170	11.62%	0.5	16.26%
C499	884	95.61%	3.0	394	95.56%	7.3	55.43%
C880	725	50.38%	8.4	368	50.33%	21.9	49.24%
C1355	639	38.37%	1.4	469	38.41%	7.3	26.60%
C2670	1885	93.94%	320.7	883	94.13%	422.9	53.16%
C3540	857	23.26%	92.6	495	23.26%	182.7	42.24%
C6288	893	97.66%	81.3	375	97.66%	328.2	58.01%
C7552	3144	98.01%	35.3	1945	98.21%	1176.0	38.14%
Average	1153.8	63.61%	67.9	637.4	63.65%	268.4	42.38%

TABLE VII
PODEM V1 + FAN V2 - NDET = 1

ndet=1	w/o compression			w/ compression			
Circuit	TL	FC	RT(s)	TL	FC	RT(s)	TL↓
C432	25	11.62%	0.2	22	11.62%	0.2	12.00%
C499	116	95.61%	2.0	67	95.61%	2.4	42.24%
C880	101	50.38%	8.4	55	50.38%	9.6	45.54%
C1355	87	38.41%	1.2	61	38.41%	1.8	29.89%
C2670	265	94.13%	301.2	130	94.06%	338.7	50.94%
C3540	116	23.26%	91.6	72	23.26%	104.9	37.93%
C6288	137	97.65%	52.6	68	97.66%	88.7	50.36%
C7552	507	98.19%	28.7	288	98.25%	229.3	43.20%
Average	169.3	63.66%	60.7	95.4	63.66%	97.0	39.01%

come to this conclusion because our FAN+PODEM algorithm is implemented based on a large legacy code base, which we don't have enough time to optimize for runtime. Whereas the PODEM algorithm is implemented based on a rather smaller and newer code base, in which we have the ability and time to do more optimization for runtime.

B. PODEM and FAN+PODEM parallel processing

As mentioned in the preceding discussion subsections, PODEM and FAN+PODEM perform better on different circuit. Therefore, we decided to run both executable in parallel and choose the better generated pattern set. In return, we can obtain the result with both better fault coverage and test length for different circuit with little increase in runtime compare to the FAN+PODEM runtime. However, we notice that we need to wait for FAN+PODEM to finish executing even when PODEM produce better result. Therefore, we came to the conclusion that our parallel processing technique still has space for runtime improvement. For example, we can divide the whole fault list into multiple subset and run them all in parallel.

C. Fault Ordering

We tried to obtain better fault coverage and test length by choosing better fault ordering. We have already tried detecting hard-to-detect fault first. However, we don't know for sure if this is the optimal way to order a fault list. We believe part of the key to optimizing fault coverage and test length is highly correlated to the ordering of the fault in the fault list.

TABLE VIII
PODEM V1 + FAN V2 - W/O COMPRESSION VS STC ONLY VS STC+DTC

ndet=8	w/o compression			STC only				STC+DTC			
Circuit	**TL**	**FC**	**RT(s)**	**TL**	**FC**	**RT(s)**	**TL ↓**	**TL**	**FC**	**RT(s)**	**TL ↓**
C432	203	11.62%	0.2	173	11.62%	0.4	14.78%	170	11.62%	0.5	16.26%
C499	884	95.61%	3.0	418	95.61%	3.6	52.71%	394	95.56%	7.3	55.43%
C880	725	50.38%	8.4	455	50.38%	9.9	37.24%	368	50.33%	21.9	49.24%
C1355	639	38.37%	1.4	469	38.37%	2.7	26.60%	469	38.41%	7.3	26.60%
C2670	1885	93.94%	320.7	1196	93.94%	342.0	36.55%	883	94.13%	422.9	53.16%
C3540	857	23.26%	92.6	572	23.26%	103.1	33.26%	495	23.26%	182.7	42.24%
C6288	893	97.66%	81.3	317	97.66%	99.5	64.50%	375	97.66%	328.2	58.01%
C7552	3144	98.01%	35.3	2044	98.00%	105.6	34.99%	1945	98.21%	1176.0	38.14%
Average	1153.8	63.61%	67.9	705.5	63.61%	83.4	37.58%	637.4	63.65%	268.4	42.38%

TABLE IX
PARALLEL PROCESSING RESULT (NDET = 8, COMPRESSION ON)

Circuit	PODEM			PODEM_V1+FAN_V2			FINAL			Total
	FC	**TL**	**RT(s)**	**FC**	**TL**	**RT(s)**	**CHOOSE**	**FC**	**TL**	**RT(s)**
c432	11.62%	170	0.1	11.62%	170	0.5	**PODEM**	**11.62%**	**170**	**0.6**
c499	94.69%	582	2.7	95.56%	394	7.3	**PODEM_V1+FAN_V2**	**95.56%**	**394**	**7.4**
c880	50.38%	299	1.6	50.33%	368	21.9	**PODEM**	**50.38%**	**299**	**22.0**
c1355	38.41%	463	1.0	38.41%	469	7.3	**PODEM**	**38.41%**	**463**	**7.4**
c2670	94.06%	874	13.8	94.13%	883	422.9	**PODEM_V1+FAN_V2**	**94.13%**	**883**	**423.1**
c3540	23.26%	486	16.3	23.26%	495	182.7	**PODEM**	**23.26%**	**486**	**182.0**
c6288	97.63%	387	20.3	97.66%	375	328.2	**PODEM_V1+FAN_V2**	**97.66%**	**375**	**328.8**
c7552	98.31%	1199	103.0	98.21%	1945	1139	**PODEM**	**98.31%**	**1199**	**1139.8**
Average	63.55%	557.5	19.9	63.65%	637.4	263.7		**63.67%(+0.03%)**	**533.6(-4.3%)**	**264.0(+0.11%)**

VI. CONTRIBUTION

- Yu-Hung Pan
  - Parallel processing
  - TDF simulation and command line modification in FAN
- Wei-Shen Wang
  - ATPG and test compression in PODEM V1 + FAN V2
- Hsin-Tzu Chang
  - ATPG and test compression in PODEM algorithm
  - ckt-to-verilog conversion for FAN

REFERENCES

[1] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, Vol. 30, No.3, pp215-222, 1981.

[2] Fujiwara and Shimono, "On the Acceleration of Test Generation Algorithms," in IEEE Transactions on Computers, vol. C-32, no. 12, pp. 1137-1144, Dec. 1983

[3] B. Benware , C. Schuermyer , S. Ranganathan , R. Madge , P. Krishnamurthy, " Impact of multiple detect test patterns on product quality, "IEEE Int'l Test Conference, 2003.

[4] P. Goel and B. C. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures," 18th Design Automation Conference, Nashville, TN, USA, 1981, pp. 260-268.

[5] FAN ATPG, Laboratory of Dependable Systems, National Taiwan University

[6] L. Goldstein, "Controllability/observability analysis of digital circuits," in IEEE Transactions on Circuits and Systems, vol. 26, no. 9, pp. 685-693, September 1979.