N-DETECT TDF ATPG AND COMPRESSION

Min-Hsin Liu, Kai-Wei Lin, Kuan-Hung Yeh

Group 13

1. INTRODUCTION

Testing integrated circuits (IC) is an important process in very-large-scale integration (VLSI). It helps producers to guarantee IC quality, shorten time to market and increase profit. A high-quality testing procedure gives a good fault modeling especially by introducing automatic test pattern generation (ATPG). Until now, the single stuck-at fault (SSF) ATPG is still the most commonplace fault model to test the functionality of a circuit. However, some defects change circuit timing but not function. Therefore, delay test is required. In this final project, we will focus on transition delay fault (TDF) ATPG based on launch-on-shift (LOS) mode.

Besides, for better test quality, we generate N-detected test pattern set. Due to significantly increased test cost by N-detect, we also need to apply test compression to reduce test length. We adopt two kinds of compression techniques, including static test compression (STC) and dynamic test compression (DTC).

Section 2 reviews previous researches about TDF ATPG and test compression. Section 3 illustrates our proposed methods. Section 4 demonstrates the experimental results and section 5 concludes our works, observations and possible improvements.

2. PREVIOUS RESEARCH

In this section, we briefly review some previous research considering ATPG, testability measure and test compression technique.

2.1. Automatic Test Pattern Generation (ATPG)

Roth first proposed the D-algorithm in 1966 to generate test patterns for combinational circuits by driving D-frontier and justifying J-frontier. It assigns values at both primary inputs (PIs) and internal nodes to detect SSFs. However, since D-algorithm requires the large search space, it performs inefficiently while dealing with complicated combinational circuits. Accordingly, Goel proposed the Path Oriented Decision Making (PODEM) algorithm in 1981, which only decides values at PIs. And it only utilizes forward implication rather justification. Furthermore, Fujiwara also proposed the FANout-oriented test generation (FAN) algorithm in 1983, which is similar to PODEM but with several optimized techniques, including making decision at head lines and multiple parallel search.

As for delay fault testing, it can detect timing-related defects which might not be disclosed by SSF tests. In this project, we consider two types of TDF associated with each wire: a slow-to-rise (STR) fault and a slow-to-fall (STF) fault. Detecting a TDF requires a vector pair to model sequential test. In the next section, we have proposed the effective algorithm with modifications to the PODEM algorithm to generate delay tests and fault simulate them in order to detect TDFs.

The large test length generated for detecting TDFs is foreseeable. Thus, test compression and testability measure attempt to reduce the size of test data and lessen the times for backtracing.

2.2. Testability Measure

A critical part in PODEM algorithm is path selection during doing a backtrace to PIs. Some previous versions use logic level to determine easy or hard control. However, Sandia Controllability Observability Analysis Program (SCOAP) serves a good criteria to find the easiest or hardest input control. SCOAP helps us evaluate how difficult a node is to be controlled to a certain value and to be observed at the primary output (PO) via knowing gate types and logic levels.

2.3. Test Compression

Due to the large scale and huge complexity of VLSI structure, the amount of test patterns for testing a chip skyrockets, especially for sequential circuits. Hence, an efficient and effective test compression technique is vital to enhance test performance. STC and DTC are two common methods for test compression we apply in our project. For STC, we implement the reverse-order fault simulation to compact X-filled test patterns. In addition, for DTC, Goel proposed the PODEM-X algorithm to repetitively generate test pattern to cover a secondary fault after a primary fault test pattern is generated.

3. PROPOSED METHODS

We proposed a method that can generate patterns for multiple detection of transition delay faults given a input circuit. Fig.1 demonstrates our overall flow of ATPG. First, we compute SCOAP for the circuit. Then, we start our main ATPG procedure with dynamic test compression and static test compression. The details of dynamic test compression and static test compression will be illustrated more thoroughly in the following sections.

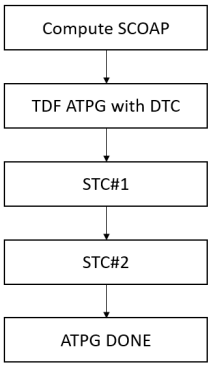

Fig. 1: Overall Flow

3.1. Dynamic Test Compression

Dynamic test compression is one of the most efficient method to reduce the test patterns. Since we want every possible fault in the circuit to be detected N times, the number of test patterns is bound to be large, especially when N is a large number. Thus, dynamic compression is quite important to this project. In our project, we apply a modified version of PODEM algorithm to realize PODEM-x dynamic test compression and transition delay faults.

In the dynamic test compression stage, we will choose a primary fault at first. For this fault, we will try to generate v2 (a shift of v1) to check whether this fault can be propagate to primary output. After successfully generating v2, we left shift v2 to v1 and check whether this fault can be activated by this pattern. If we cannot detect this fault by the pattern, we will backtrack to v2 generating phase, and try to generate another pattern to detect this fault. We can set a user defined backtrack limit based on balancing the performance and the running time. If we have tried this procedure more than the backtrack limit, we will give up generating patterns for this fault.

Otherwise, we can find a pattern with unknowns bits for primary fault. Next, we check the number of unknown bits. If unknown bits are sufficient, we pick a secondary fault randomly. We apply the same procedure on the secondary fault as primary fault. If the test pattern for secondary fault can be generated based on the pattern for primary fault, or it fails to generate, we check if there are still untried secondary fault or there are enough unknown bits. If there are still untried secondary fault and there are enough unknown bits, we choose another secondary fault and do the same procedure.

Otherwise, we start to fill in the redundant unknown bits. In this project, we randomly fill the remaining unknown bits for the generated pattern. For example, if there is a pattern with eight unknown bits and we need a pattern for 8-detect, we exhaustively fill the first three bits to generated eight patterns with five unknown bits. For the remaining five unknown bits, we randomly give them either 0 or 1. After filling unknown bits, we do transition fault simulation with fault dropping on these patterns. If all faults are dropped or tried, we finish our dynamic test compression part. Otherwise, we choose another primary fault and implement the same procedure.

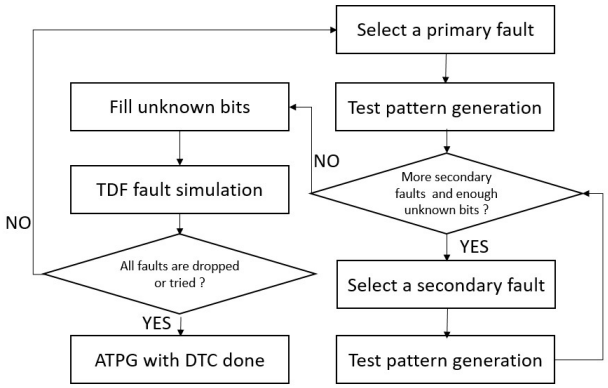

Fig. 2: Dynamic Test Compression

3.2. Static Test Compression

In the static test compression stage, we aim to remove redundant test patterns and keep those which are necessary to detect the listed faults. There are several ways to implement it. For example, random order fault simulation, reverse order fault simulation, and Tseng-Siewiorek algorithm.

In our project, we apply reverse order fault simulation to implement static test compression. The advantages of this method are that it is simple and it doesn't require any dictionary, which saves a lot of computation memory. In fact, it is one of the most popular methods. In the algorithm, we simulate x-filled patterns in reverse order of ATPG to check if the chosen pattern could detect any faults. If it couldn't, we remove the test pattern. Otherwise, we continue to simulate the next pattern if there is any. We continue this process until no test patterns remain. At this point, reverse order static test compression is done.

To compress more test patterns, we first sort patterns based on detected fault number. Then again, we repeat the previous process to simulate the selected test pattern. If there are no test patterns to simulate, then our whole static test

compression is officially completed. The following figure shows the experiment flow of static test compression.

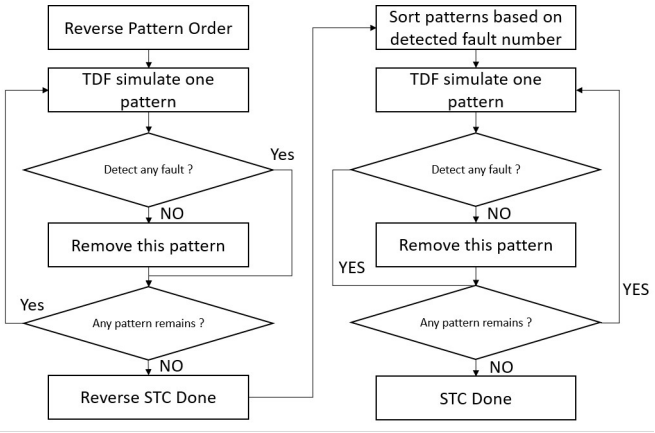

Fig. 3: Static Test Compression

4. EXPERIMENTAL RESULTS

In this paragraph, we will compare the results generated by our proposed method under different conditions. In Fig.(4) and Fig.(5), we can see that the test length is linear to the number of N for each circuit. However, the slopes after compression is obviously smaller than the ones before compression. In other words, when N is larger, we can remove more redundant patterns by means of dynamic and static test compression because the test length becomes larger.

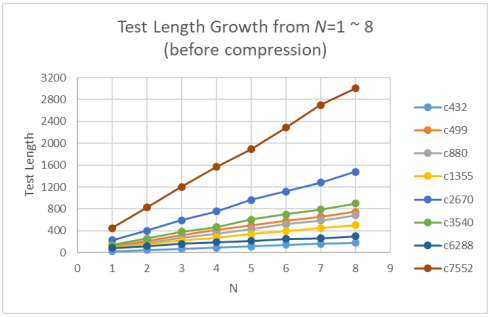

Fig. 4: Test Length before Compression

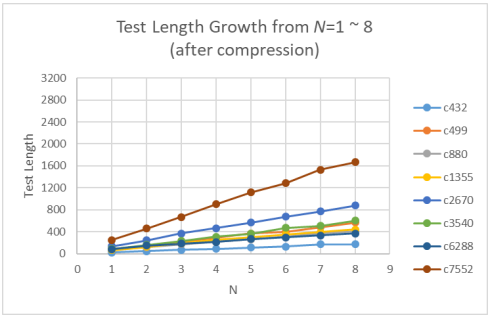

Fig. 5: Test Length after Compression

From Table (1), we compare the test length, fault coverage, run time for every circuits between cases with and without compression when N is equal to one. We can see that there are significant difference of test length. For C7552, the test length reduction rate can reach about forty-four percent, which means our method is quite useful for this circuit. While reducing the test length, we also maintain the fault coverage. The main cost of compression is that the run time grows by about sixty percent in average. However, it is still worth doing.

From Table (2), we can also see that the test length with compression is obviously lower and reduction rate can reach about twenty-three percent while maintaining the overall fault coverage. However, for circuit C6288, the test length with compression is larger. This situation may be caused by procedure of dynamic test compression. When we choose secondary fault, we randomly pick one from fault list without considering the relations between primary fault and secondary fault. Thus, the bad secondary fault may lead to a worse generated pattern and result in the negative reduction rate. Nevertheless, the overall reduction rate is still good enough, which proves that our method is applicable for most cases.

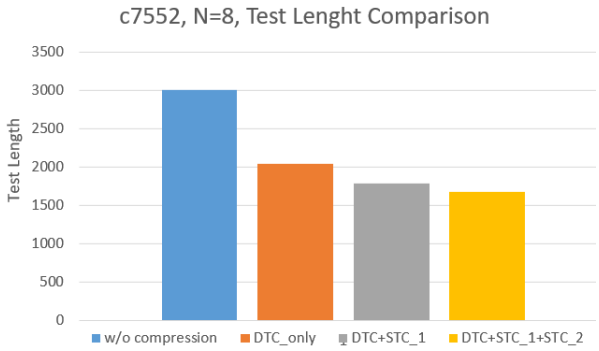

Fig. 6: Test Length after Each Approach

To show our proposed compression methods are all worth doing, we take C7552 for example to see the test length after each steps. From Fig (6), the test length significantly reduce after dynamic test compression. As for static test compression, the reverse order STC removes about 250 patterns and the second reordered STC according to detected fault numbers removes about 120 patterns.Although STC removed less patterns, but it is much faster. As for the reason why we don't keep applying STC is that the run time will keep accumulate, but test length is barely changed. Thus, the overall flow would become much more inefficient.

N = 1, backtrack limit = 20

circuit number	Test length w/o compression	fault coverage(%)	run time(s)	Test length w/ compression	fault coverage(%)	run time(s)	Test length reduction(%)
C432	27	11.62	0.1	23	11.62	0.1	14.81
C499	118	94.31	0.4	84	94.73	0.6	28.81
C880	111	49.24	0.7	68	49.76	1.0	38.74
C1355	88	38.41	2.1	64	38.41	2.5	27.27
C2670	228	92.47	1.5	135	92.81	3.6	40.79
C3540	137	22.96	10.3	89	22.92	12.0	35.04
C6288	83	97.23	6.0	75	97.19	9.5	9.64
C7552	446	97.73	5.6	249	97.93	12.3	44.17
Average	154.75	62.99	3.34	98.38	63.19	5.21	29.91

Table 1: Experimental Result (N = 1)

N = 8, backtrack limit = 20

circuit number	Test length w/o compression	fault coverage(%)	run time(s)	Test length w/ compression	fault coverage(%)	run time(s)	Test length reduction(%)
C432	179	11.62	0.1	173	11.62	0.2	3.35
C499	750	91.17	0.7	566	91.34	1.0	24.53
C880	686	49.19	0.9	412	49.24	1.4	39.94
C1355	501	32.94	2.4	439	32.94	2.7	12.38
C2670	1481	91.73	2.8	875	92.38	6.3	40.92
C3540	896	22.88	11.5	600	22.91	14.8	33.04
C6288	303	96.82	9.8	337	97.06	14.2	-11.22
C7552	3007	97.58	12.9	1670	97.82	26.9	44.46
Average	975.38	61.74	5.14	634.00	61.91	8.44	23.42

Table 2: Experimental Result (N = 8)

5. DISCUSSION

After analyzing our results and the brainstorming between our group members, we come out some possible improvements that can be applied to our working system.
1) Before the procedure of the TDF ATPG, we can sort the fault list based on how difficult the fault can be detected. We should generate patterns earlier for the faults which is harder to be detected.
2) We can choose secondary fault more wisely. One possible approach is that we simulate the pattern generated for primary fault before choosing secondary fault. Only the faults not detected by this pattern are the candidates of secondary fault.
3) Instead of randomly filling unknown bits, we can apply x-unfilled STC, like Tseng-Siewiorek algorithm. Besides, we can try to randomly fill more unknown bits to get more patterns. Then we simulate each generated pattern and choose the top eight (for 8-detect) pattern to be our final patterns.

6. CONCLUSION

We proposed a test model which is able to perform TDF ATPG with N-detect and two types of test compression. We also try new ideas in an attempt to improve fault coverage, reduce test length and run time. First, we implement a modified PODEM-based algorithm which can support N-detect TDF ATPG and DTC at the same time. More importantly, it provides a good fault coverage result under an quite low run time. Lastly, we apply two rounds of STC, reverse order and reordered by detected fault numbers. From our experimental results, we can clearly see that our method provides a solution with good fault coverage and very low run time. Besides, our compression methods helps us to reduce test length by about twenty-three percent in average for N=8. All in all, our method is a efficient and effective way to give a solution to test pattern generator for multiple-detect transition delay fault.

7. CONTRIBUTION

Below are our work distributions for this project:
1) Min-Hsin Liu : DTC ,N-detect ATPG, and report
2) Kai-Wei Lin : SCOAP and Report
3) Kuan-Hung Yeh : STC and Report

8. REFERENCES

[1] Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," in IEEE Transactions on Computers, vol. C-30, no. 3, pp. 215-222, March 1981, doi: 10.1109/TC.1981.1675757.
[2] B. Benware et al., "Impact of multiple-detect test patterns on product quality," International Test Conference, 2003. Proceedings. ITC 2003., Charlotte, NC, USA, 2003, pp. 1031-1040, doi: 10.1109/TEST.2003.1271091.
[3] D. Xiang, W. Sui, B. Yin and K. -T. Cheng, "Compact Test Generation With an Influence Input Measure for Launch-On-Capture Transition Fault Testing," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 9, pp. 1968-1979, Sept. 2014, doi: 10.1109/TVLSI.2013.2280170.
[4] Kun-Wei Lin, "A parallel ATPG aiming at improving N-detect pattern quality". master's thesis, NTU GIEE, 2017, https://hdl.handle.net/11296/ghkwx3