Parallel Order ATPG for Test Compaction

***Abstract*—Generating a compacted test set is very important to reduce the cost of testing. In this paper, we proposed a novel test compaction algorithm which achieve high compaction, called Parallel Order Dynamic Test Compaction (PO-DTC). Our results show that the order of secondary faults within a single test generation is very important for test compaction. We use GPU to launch many parallel ATPG with different orders of secondary faults. Then we choose the best test pattern, which detects the largest number of faults. Experimental results show that our test length is 48% shorter than that of a highly compacted commercial ATPG. Our test length is the smallest among all previous work published so far. Our results show that our technique is also useful to compact *N*-detect test sets. Our test length is at least 1/4 shorter than that of the commercial ATPG for *N*=3, *N*=5, *N*=8.**

Keywords—parallel ATPG, test compaction

I. INTRODUCTION

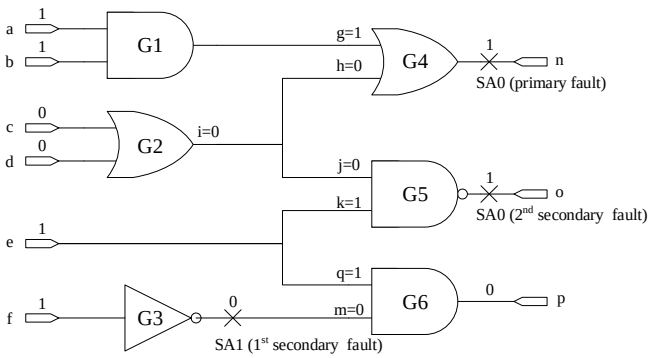
For modern VLSI designs, compact test sets are highly desirable to reduce test application time. To enhance test quality and test yield, recently high-quality test sets (such as N-detect test sets [Ma 95][Chang 98][Tseng 01][Benware 03], and timing-aware [Lin 06] , power-aware test sets [Girard 02] [Yilmaz 08]), are gaining more and more attention. However, these high-quality test sets are generally very long so they cannot be applied in practice. Recent roadmap predicts that 1,000 times compression is needed by 2020 [ITRS 13]. Therefore, test compaction techniques are very needed.

Although hardware DFT compaction techniques [Barnhart 01][Touba 06] have been applied, they can reduce scan chain length, hardware technique cannot reduce number of test patterns. Software ATPG techniques are still needed to reduce the number of test patterns.

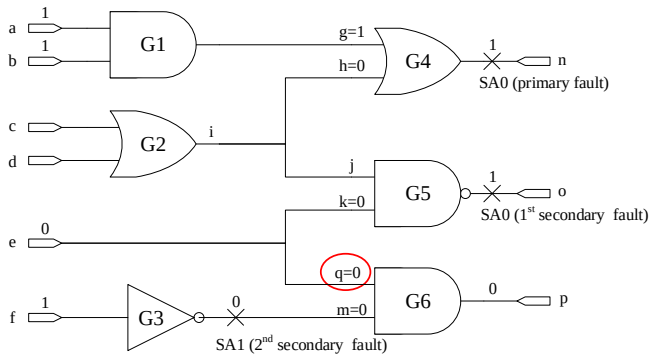
N-detect test sets have been shown to be more effective to achieve high defect coverage than single-detect test sets [Ma 95], [Chang 98], [Tseng 01], [Benware 03]. And the effectiveness of *N*-detect test sets based on transition faults in detecting defects that affect the timing behavior of a circuit is better than stuck-at faults [Pomeranz 00]. However, the number of *N*-detect patterns increases linearly with N [Benware 03], [Amyeen 04]. It leads to long test application time, and it may be impossible to store all patterns due to memory limit on ATE. Generating a compacted test set for high-quality test is very important.

Past research in software test compaction can be divided into dynamic and static test compaction. The former is more effective and timing consuming than the latter. Dynamic test compaction is performed during test pattern generation while static compaction is done after test pattern generation. Dynamic compaction consists of two parts: (1) generate a *partially specified test pattern* for a *primary fault*, and (2) fill in the unspecified inputs in the existing partially specified test pattern to detect more *secondary faults*. Past research in dynamic test compaction, such as [Pomeranz 91][Krauss 94], mainly focus on the order of primary faults, rather than the order of secondary faults. Other research, such as [Raghunathan 95][Remersaro 09], selects the most suitable secondary fault list, but they did not try different order of secondary faults.

According to our experiment, *secondary fault order* is very important for dynamic test compaction. *Secondary fault order* is the order of secondary faults selected by the ATPG dynamic compaction. Given the same primary fault, different secondary fault order may have different result of compaction. Figure 1(a)(b) shows an example of the effect of secondary fault order. Given the primary fault *line n stuck-at 0*, we can generate a partially specified test pattern (*a*, *b*, *c*, *d*, *e*, *f*) = (1, 1, X, X, X, X). Now consider two secondary faults: *Line m stuck-at 1* and *line o stuck-at 0* that are compatible with the primary fault. If we target *line m stuck-at 1* first, we fill in the test pattern as (1, 1, X, X, 1, 1). Then, we target the other secondary fault, *line o stuck-at 0*, we fill in the test pattern as (1, 1, 0, 0, 1, 1), like Figure 1(a). In this way, we compact two secondary faults successfully into one test pattern. However, if we target *line o stuck-at 0* fault first, we may fill in the test pattern as (1, 1, X, X, 0, X). After that, we fail to generate a test pattern to detect *line m stuck-at 1* because *line q* = 0 is a controlling value of gate *G6*, as is shown in Figure 1(b).



(a) Choose fault *m stuck-at 1* first



(b) Choose fault *o stuck-at 0* first

Figure 1. Example of different *secondary fault orders*

To show the impact of *secondary fault order* to dynamic test compaction, we did the following experiment. Figure 2 shows a histogram of detected faults. In this figure, each dot represents the number of test patterns for a given number of detected faults. We selected 2,264 faults for test generation. They are randomly ordered in 256 ways, each of which generates 8 different test pattern. The horizontal axis is the number of detected faults; the vertical axis is the number of test patterns. We can see large variation in the horizontal axis. There are two best test patterns that detected more than 400 faults, while there is one worst test pattern that detected less than 250 faults. This experiment tells us that, good secondary fault order *during a single dynamic test generation* is very important.

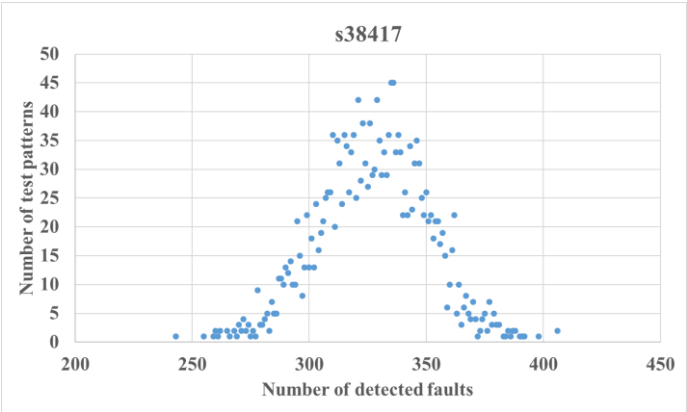


Figure 2. Impact of *secondary fault order*

In this paper, we propose a novel ATPG that exploits different secondary fault orders in parallel. Because there are too many fault orders, it is difficult to predict which order is the best. We use GPU to exploit many different secondary fault orders in parallel and add randomness to GPU-based ATPG so that the test patterns generated have variation. We permute the order of secondary faults to generate many different *secondary fault lists*. Each *secondary fault list (SFL)* is composed of the same group of secondary faults but they have different secondary fault orders to each other. We launch many different *secondary fault lists* in parallel ATPG and select the test pattern with the most number of detected faults.

The contribution of this paper is that we propose a novel test compaction algorithm by parallel ATPG, called *Parallel Order Dynamic Test Compaction (PO-DTC).* Unlike traditional parallel ATPG, which has test inflation problem, our parallel ATPG reduces the number of test patterns significantly. According to our experimental results on benchmark circuits, our number of test patterns is 48% shorter than that of highly compacted commercial ATPG on average. Our number of test patterns is the smallest among all previous works.

II. Previous Research

In order to reduce test length, several test set compaction algorithms have been proposed. Static compaction is performed after test pattern generation is complete. Static compaction handles X-filled test patterns [WWW book], while dynamic compaction is performed during test pattern generation. Generally speaking, dynamic compaction is more effective than static compaction. In dynamic compaction category, we can further divide papers into (1) primary fault order and (2) secondary fault selection.

It is well known that *primary fault ordering* in dynamic test compaction has big impact on test length. *Independent fault set* is a set of faults, where no two faults can be detected by the same test pattern [Akers 87]. *Maximum independent fault set* (MIFS) is an independent fault set of maximum size in a given combinational circuit. MIFS is used to select *primary faults* at the beginning of every test generation [Pomeranz 91]. They proposed to select primary faults in decreasing order of the size of independent fault sets. Krauss ordered the fault list according to the detectability probability [Krauss 94]. Pomeranze proposed another *primary fault ordering* method based on the accidental detection index [Pomeranze 05]. Ku ordered fault list based on the structure of fan-in cone [Ku 14]. Eggersglüß extracted hard-to-detect faults from an untestable identification phase and ordered hard-to-detect faults at the beginning [Eggersglüß 14]. They have the same problem that there are no clear methods to order *secondary faults*.

According to our experiment in Figure 2, the choice of secondary faults within a single test generation is very important. Goel did a serial search through the fault list to select a secondary fault that was untested and whose associated block had its output at X [Goel 79]. Raghunathan selected secondary fault according to the *support set* [Raghunathan 95]. Remersaro selected secondary fault according to the *necessary assignments* of a fault [Remersaro 09]. Recently, Xiang selected secondary faults based on an *influence input* measure [Xiang 14]. Influence inputs are the subset of primary inputs (PIs) or pseudo primary inputs (PPIs) that need to be specified to detect a fault. The influence input measure is used to guide dynamic test compaction. Two faults can be compacted into a single test pattern if their structural influence inputs are disjoint. In our given example in Figure 1., we show that although every fault in the secondary fault set is compatible with the primary fault, different *secondary fault orders* may have different result of compaction.

There have been many test compaction techniques available for N-detect ATPG. Lee proposes a greedy static test compaction approach to select the most effective test patterns

among a pool of test patterns. This method needs fault dictionaries and many fault simulations [Lee 02]. This method is not scalable for large circuits due to its *pseudo-exhaustive* search. Another widely used method is integer linear programming (ILP) [Huang 06], [Kantipudi 06], [Kantipudi 07]. However, ILP needs complete fault dictionaries, which are not applicable to large circuits.

We propose a dynamic test compaction that tries many different orders of secondary faults and then chooses the best test pattern. We do not need fault dictionaries, nor many fault simulations. The test compaction results are better than previous static and dynamic techniques.

III. Parallel ATPG for Different Secondary Fault Orders

In this paper, we propose a novel technique, *Parallel Order Dynamic Test Compaction (PO-DTC).* This parallel ATPG technique permutes the order of *secondary fault list (SFL)* to compact test patterns. At a given time, we launch many different parallel ATPG of different orders of secondary fault list. Then we choose the best test pattern which detect the largest number of faults. In the following subsections, we are going to introduce the overall flow of our algorithm, the *Parallel Order Dynamic Test Compaction (PO-DTC).* In this paper, we use GPU-based parallel ATPG algorithm (SWK) [Liao 13] to implement our idea. Please note that our idea is also applicable to other parallel ATPG algorithms.

A. Overall flow

Figure 3 shows the overall flow of our algorithm. Given a netlist, we select a primary fault from the fault list. We randomly select a primary fault and a group of *secondary faults.* PO-DTC (Sec. B) then permutes the order of secondary faults to generate different secondary fault lists (SFLs). Then we use a GPU-based ATPG algorithm (Sec. C) to generate test patterns in parallel. After all faults in SFL have been tried, test generation returns the number of detections for each test pattern. We select a test pattern with the largest number of detection. Then we fault simulate the selected randomly filled test pattern and drop detected faults. If there are still undetected faults, we select a next primary fault and a group of secondary faults to perform again test generation.

B. Parallel Order Dynamic Test Compaction (PO-DTC)

The first step of PO-DTC is permutation. After permutation, many secondary fault lists are generated, each of which contains the same faults but their orders are different.

The second step in PO-DTC is to launch a parallel GPU-based ATPG. Suppose each GPU has b blocks, each of which runs a SWK test generation independently. Each word has W bits, each of which is an independent *clone* (see Sec. C) that generates a test pattern. Totally, we can generate W x b test patterns at a time. To increase the variety of test patterns, we allocate v clones (bits) to a SFL. So the total number of SFL (l) available in a single parallel ATPG simultaneously is

$$l = \frac{b \times W}{v}$$

Please note that v must be divisible by W so that we have integer number of SFLs.

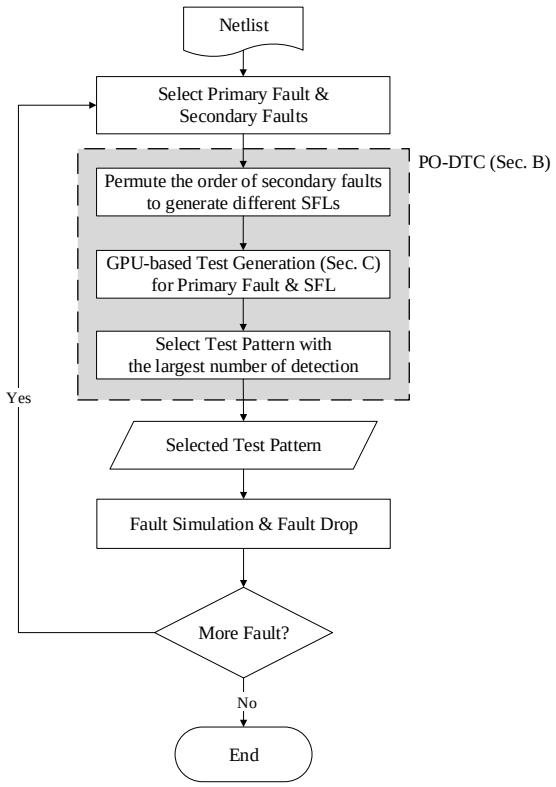


Figure 3. Overall Flow

Figure 4 gives a simple example. Given a primary fault f_1 and four secondary faults f_2, f_3, f_4, f_5. Suppose the GPU has two blocks (b=2). Each word has four bits (W=4) and the number of variation is two (v=2). Therefore, we can execute four (l=4) different SFLs. We randomly generate four different secondary fault lists: {f_2, f_3, f_4, f_5}, {f_4, f_5, f_3, f_2}, {f_5, f_2, f_4, f_3}, {f_3, f_4, f_2, f_5}. Figure 4 illustrates how we assign different secondary fault lists to two blocks during parallel dynamic compaction. In each square, f_x represents a target fault. At the beginning, every bit in each word targets the same primary fault f_1. After generating a partially specified test pattern, each bit targets the secondary fault according to the given secondary fault list. We assign bit 0 and bit 1 of block 0 to process SFL {f_2, f_3, f_4, f_5}, bit 2 and bit 3 of block 0 to process SFL {f_4, f_5, f_3, f_2}. Similarly, bit 0 and bit 1 of block 1 processes SFL {f_5, f_2, f_4, f_3}, bit 2 and bit 3 of block 1 processes SFL {f_3, f_4, f_2, f_5}. Although it is impossible to exhaustively try all orders of secondary fault list, we can achieve very good test compaction within hundreds of SFLs.

In our experiment, b=64, W=32 and v=8, then 2,048 test patterns and 256 different SFLs can be generated simultaneously. With so many test patterns, we can achieve very good test compaction out of 256 different SFL orders. In order to achieve such a massive parallelism, we need a GPU-based massively parallel ATPG algorithm, introduced in Sec. C.

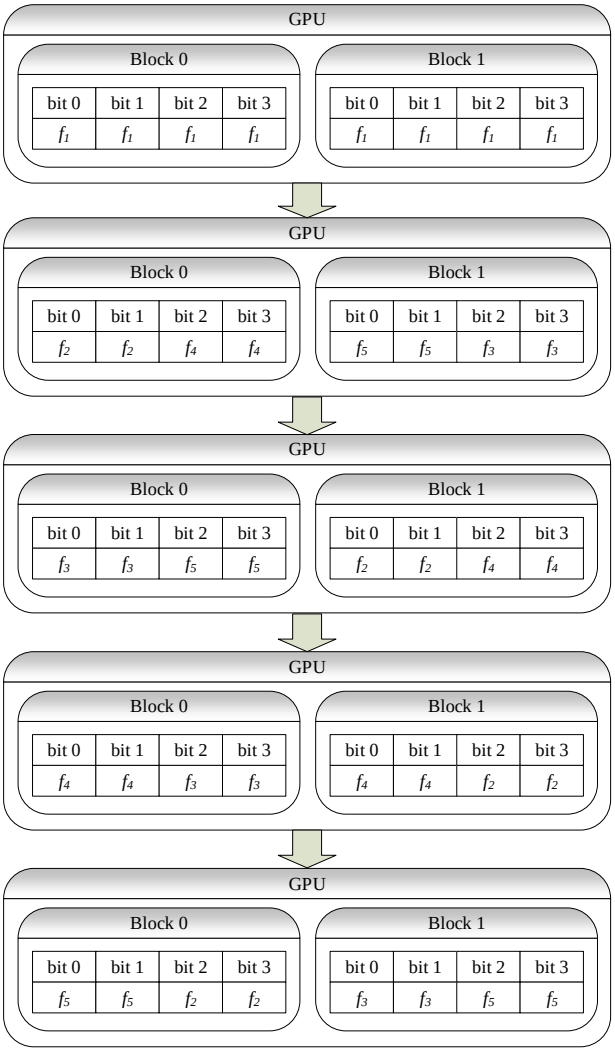


Figure 4. Example of PODTC
(b=2, W=4, v=2, l=4)

C. GPU-based Test Generation (SWK algorithm)

SWK algorithm is based on PODEM algorithm [Goel 81] which generates test patterns by repeatedly backtracing objectives to inputs and propagating fault effects to outputs. The test generation consists of two levels of parallelism: word-level parallelism for a block, block-level parallelism for a GPU. Each block executes a word-level parallel test generation algorithm and the GPU handles a statically partitioned fault list. The main concept of SWK algorithm is converting backtrace and propagation into bitwise logic operation so different branches of the decision tree can be explored at the same time.

In this algorithm, a signal Y is represented by seven words of W bits: $Y^0, Y^1, Y^d, Y^{\bar{d}}, Y^{b_0}, Y^{b_1}$, and Y^p. Each bit in a word represents an individual *clone*, which performs an independent search. The meaning of the first four words is as follows.

Y^0 =1: signal Y is zero (good 0/faulty 0).

Y^1 =1: signal Y is one (good 1/faulty 1).

Y^d =1: signal Y is d (good 1/faulty 0).

$Y^{\bar{d}}$ =1: signal Y is $\bar{d}$ (good 0 /faulty 1).

For a given clone, the above four values are mutually exclusive so at most one of them equals one at a time. If all of them are zero, Y is unknown. When Y is unknown, the other three words indicate whether Y is on the *propagation path* or the *objective path*. The former is a path that faulty effect (d or $\bar{d}$) will potentially propagate to reach an output. The latter is a path that the objective backtrace follows to reach an input. For each clone,

Y^p =1: signal Y is on a propagation path.

Y^{b_0} =1: signal Y is on an objective 0 backtrace path.

Y^{b_1} =1: signal Y is on an objective 1 backtrace path.

Again, these three values are mutually exclusive so at most one of them equals one at a time.

Figure 5 shows the test generation flow of each block in our GPU-based test generation for transition delay faults. Given a set of faults F, word size W, and the number of variation v, each fault in F is handled by v clones. For example, if W = 32, F = {f_1, f_2, f_3, f_4}, and v = 8, then the first eight clones generate patterns for f_1, and the following eight clones generate patterns for f_2, and so on. For transition faults test generation, the two initial objectives are fault-free values at the fault site in two time frames. A backtrace is then performed from the fault site to the inputs. After inputs are assigned, fault effect propagation is performed from inputs to the outputs. The test generation of a clone is *successful* if a d or $\bar{d}$ has reached any output. This process ends if test generations for all w clones are successful or time limit has reached.

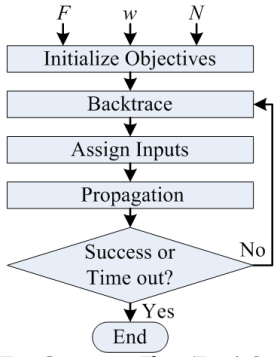


Figure 5. Test Generation Flow (For A Single Block)

Figure 6 to Figure 9 illustrate an example to generate a test pattern for a transition fault. This is a two-time frame circuit, each of which consists of three AND gates, three inputs (E, F, G), and one flip-flip (H). LV stands for logic level.

Suppose that the target fault is $G2$ slow-to-fall. The size of word is two bits (W=2) and the number of variation is two (v=2). We use two bits to represent two clones: clone #1 and clone #2. Since the fault is slow-to-fall, the initial objectives in time frame one and two are one and zero, respectively. They are denoted as {b_1, b_1} and {b_0, b_0} in Figure 6

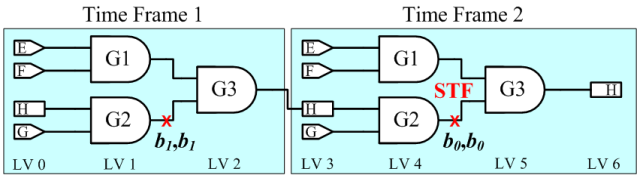


Figure 6. Test Generation Example (Initialize Objectives)

Figure 7 shows the first backtrace after Figure 6. Backtracing *G2* in time frame one is an *implication backtrace* because both gate inputs must be one to justify the output objective b_1. Backtracing *G2* in time frame two requires a decision because either *H*=0 or *G*=0 justifies the output objective b_0. An *objective split* is performed to assign two clones with different objectives. In this example, clone #1 backtraces b_0 on input *H* and clone #2 backtraces b_0 on input *G*.

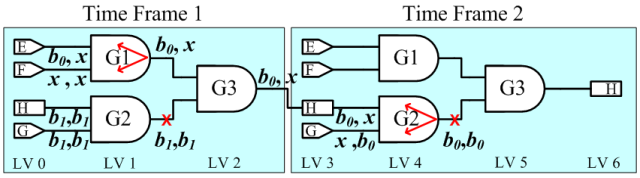


Figure 7. Test Generation Example (First Backtrace)

Figure 8 shows the propagation after Figure 7. The fault is excited for both clones so *G2* in time frame two is $\{\bar{d}, \bar{d}\}$ in the figure. Output *H* is {*p*, *p*} in the figure. The symbol '*p*' indicates that output *H* is on the propagation path.

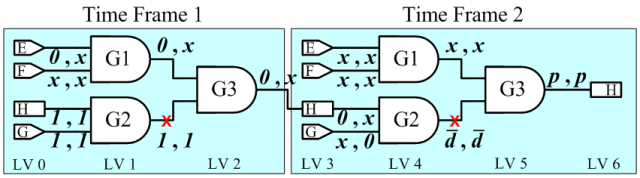


Figure 8. Test Generation Example (First Propagation)

Figure 9 shows the second backtrace followed by another propagation. To propagate the faulty effect to output, both clones of *E* and *F* in time frame two are set to one. Two test patterns *EFG*={(0x1, 11x), (xx1, 110)} and *H*={1, 1} are successfully generated. In this way, we generate test patterns of variation.

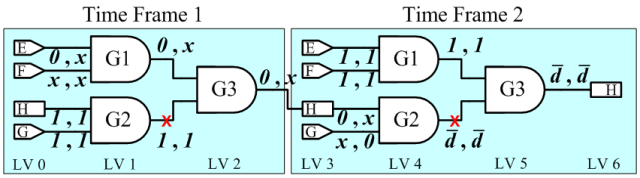


Figure 9. Test Generation Example (Second Backtrace and then Propagation)

IV. Experimental Results

To validate the proposed technique, experiments were performed on ISCAS'89 and ITC'99 benchmark circuits. We generate test patterns for launch-on-capture transition delay fault to compare our results with the previous best result and a commercial ATPG tool. The specification of CPU platform we used in the experiment is Intel Core i7-3930K 3.2GHz 8-core processors. The specification GPU platform is GeForce GTX TITAN X.

Table I shows the number of total faults, the size of the SFL, the test length (TL) and the runtime (RT) of the baseline ATPG and the ATPG with the proposed PO-DTC. The size of SFL is represented as percentage of the total faults for each test generation. In our implementation, we randomly select a group of secondary faults. The size of SFL is decided based on the number of secondary faults that can be fit into GPU memory. We use a simple GPU-based ATPG without PO-DTC as a baseline ATPG. This baseline ATPG has no dynamic test compaction; it has only a simple reserve order static test compaction. In our experiment, we use 64 blocks (*b*=64) to launch our parallel ATPG. Each word has 32 bits (*W*=32) and the number of variation is 8 (*v*=8). We execute 256 different SFLs (*l*=256) simultaneously. The last row shows the normalized test length and run time with respect to the baseline ATPG. Results show that the PO-DTC achieves over 85% test compaction with respect to the baseline ATPG. The run time of PO-DTC is about 32 times more than the baseline ATPG.

TABLE I. COMPARE WITH BASELINE ATPG

Circuits	Total Faults	SFL size(%)	baseline ATPG		ATPG w/ PO-DTC	
			TL	RT(min)	TL	RT(min)
s5378	15,868	5	525	2	87	70.4
s9234	28,676	3	925	8.4	182	299.8
s13207	41,226	2.5	1,169	18.5	164	430.9
s15850	48,990	5	914	26.5	98	705.2
s35932	89,378	5	120	29.3	23	511.8
s38417	113,244	2	2,332	32.5	134	1,714.3
Normalize	-	-	1.00	1.00	0.11	31.8

Table II shows the results of our proposed PO-DTC compared with previous methods and a commercial ATPG tool. Xiang uses influence input measure to implement dynamic test compaction [Xiang 14]. This is the best compaction result we can find so far in literature. The parenthesis numbers in the test length column are the reduction percentage with respect to those of the commercial tool. A negative percentage means that the test length is shorter than that of commercial tool. The parenthesis numbers in the fault coverage column are the increment in fault coverage with respect to those of the commercial tool. A negative percentage means that the fault coverage is lower than that of commercial tool. Results show that our test length is shortest among all the CUT. Please note that in order to achieve minimal test length, the commercial tool tries to generate launch-on-shift patterns first and then generate launch-on-capture patterns for the remaining faults that are not detected by the launch-on-shift patterns. Thus, the fault coverage of commercial tool is better than our work and previous work, which only use launch-on-capture method to generate test patterns. Given the same fault coverage, we still have better compaction result than the commercial tool. For the example of s38417, our test length is 29% shorter than the commercial tool at the same fault coverage. Overall, compared with the commercial tool, our proposed PO-DTC has 48% test length improvement and 1.48% fault coverage degradation on the average. Our test length is the smallest among all previous works.

Table III shows that our algorithm is also useful to compact *N*-detect test sets. Table IV shows the test length (TL) and the

test length reduction (Reduction) compared with the commercial tool (Com.) and the previous work [Huang 06]. The last row shows the averaged test length and test length reduction. We generate test patterns for *N*-detect transition delay fault in different *N* at the same fault coverage as the commercial tool. Experiments were performed on s38417, B02, B04 and B07 benchmark circuits. In this experiment, we use 8-block implement (*b*=8) to launch PO-DTC. Overall, our test length is 25.2%, 28.5%, 25.8% short than those of the commercial tool in *N*=3, *N*=5, *N*=8 on average. We compare our test length reduction with the previous work [Huang 06]. Huang provides good results of *N*-detect test sets, which are generated for single stuck at fault by *Integer Linear Programming* (*ILP*). Although the result of Huang is generated for single stuck-at faults, our proposed technique still has good reduction of test length as good as Huang. Figure 10 shows the growth trend of *N*-detect test length for s38417. The X-axis is number of detection (*N*), the Y-axis is test length. We can see that our test length grows slower than the commercial ATPG tool as the number of *N* increases.

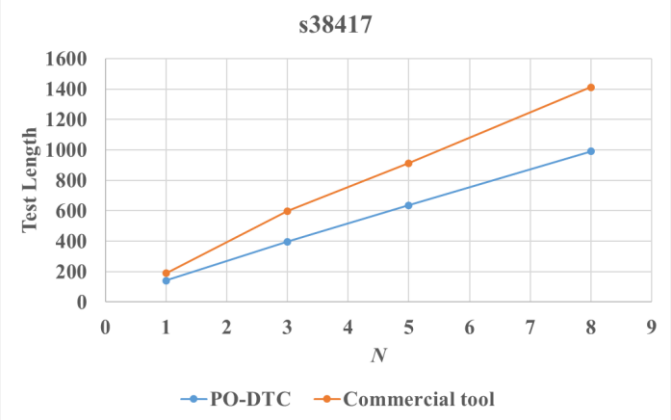


Figure 10. Growth trend of *N*-detect test length for s38417

V. DISCUSSION

The runtime of our proposed PO-DTC is very long. It was assumed that GPU-based massively parallel ATPG is good for this application. After the implementation, it was found that GPU-based is actually slower than CPU-based ATPG. The purpose of this work, however, is to demonstrate that secondary fault order does make a large difference. Our idea can still be implemented on CPU-based ATPG.

VI. CONCLUSIONS

In this paper, we propose a novel test compaction algorithm, which achieves very significant compaction, called *Parallel Order Dynamic Test Compaction (PO-DTC)*. We show that *secondary fault order* is important to test compaction. We use GPU to launch many different orders of *secondary fault list* in parallel ATPG and choose the best test pattern, which detects the largest number of faults. Experimental results show that our number of test patterns is 48% shorter than that of highly compacted commercial ATPG. For the benchmark circuits, our test length is shortest among all previous works. Our results show that our technique is also useful to compact *N*-detect test sets. Our test length is at least 25% shorter than that of the commercial tool.

TABLE II. COMPARISON WITH PREVIOUS METHODS AND COMMERCIAL TOOL

Circuit	Commercial tool		[Xiang 14]		PO-DTC	
	TL	FC(%)	TL	FC(%)	TL	FC(%)
s5378	192	91.66	90 (-53.1%)	90.13 (-1.53%)	87 (-54.7%)	91.20 (-0.46%)
s9234	289	90.01	213 (-26.3%)	81.22 (-8.79%)	182 (-37.0%)	89.29 (-0.72%)
s13207	408	91.01	175 (-57.1%)	82.26 (-8.75%)	164 (-59.8%)	89.23 (-1.78%)
s15850	259	89.49	109 (-58.0%)	74.34 (-15.15%)	98 (-62.2%)	86.86 (-2.63%)
s35932	42	89.50	32 (-23.8%)	84.26 (-5.24%)	23 (-45.2%)	86.19 (-3.31%)
s38417	189	98.59	162 (-14.3%)	96.92 (-1.67%)	134 (-29.1%)	98.59 (-0.00%)
Average	-	-	-38.8%	-6.86%	-48%	-1.48%

TABLE III. COMPARISON OF OUR N-DETECT TEST LENGTH AND REDUCTION WITH COMMERCIAL TOOL AND PREVIOUS WORK

Circuits	*N*=3				*N*=5				*N*=8			
	TL		Reduction		TL		Reduction		TL		Reduction	
	Com.	PO-DTC	PO-DTC	[Huang]	Com.	PO-DTC	PO-DTC	[Huang]	Com.	PO-DTC	PO-DTC	[Huang]
s38417	598	459	23.2%	-	913	725	20.6%	-	1,411	1,131	19.8%	-
B02	36	33	8.6%	5.6%	63	43	31.7%	0%	96	53	44.8%	-
B04	256	130	49.2%	19.3%	371	214	42.3%	11.5%	597	353	40.9%	-
B07	178	143	19.7%	13.1%	294	237	19.4%	11.7%	465	337	27.5%	-
Average	267	191	25.2%	12.7%	410	305	28.5%	7.7%	642	469	25.8%	-

REFERENCES

[Akers 87] Akers, Sheldon B. "On the role of independent fault sets in the generation of minimal test sets." *Proc. of International Test Conference, October 1987*. 1987.

[Amyeen 04] Amyeen, M. Enamul, et al. "Evaluation of the quality of N-detect scan ATPG patterns on a processor." *Test Conference, 2004. Proceedings. ITC 2004. International*. IEEE, 2004.

[Barnhart 01] Barnhart, Carl, et al. "OPMISR: the foundation for compressed ATPG vectors."*Test Conference, 2001. Proceedings. International*. IEEE, 2001.

[Benware 03] Benware, Brady, et al. "Impact of multiple-detect test patterns on product quality." *null*. IEEE, 2003.

[Chang 98] Chang, Jonathan TY, et al. "Analysis of pattern-dependent and timing-dependent failures in an experimental test chip." *Test Conference, 1998. Proceedings., International*. IEEE, 1998.

[Eggersglüß 14] Eggersglüß, Stephan, and Rolf Drechsler. "An effective fault ordering heuristic for SAT-based dynamic test compaction techniques." *it-Information Technology*56.4 (2014): 157-164.

[Goel 79] Goel, Prabhakar. "Test generation and dynamic compaction of tests." *Digest of Papers 1979 Test Conf., Oct.*. 1979.

[Girard 02] Girard, Patrick. "Survey of low-power testing of VLSI circuits." *IEEE Design & test of computers* 19.3 (2002): 82-92.

[Hamzaoglu 98] Hamzaoglu, Ilker, and Janak H. Patel. "Compact two-pattern test set generation for combinational and full scan circuits." *Test Conference, 1998. Proceedings., International*. IEEE, 1998.

[Huang 06] Huang, Yu. "On N-detect pattern set optimization." *Proceedings of the 7th International Symposium on Quality Electronic Design*. IEEE Computer Society, 2006.

[ITRS 13] Wilson, Linda. "International technology roadmap for semiconductors (ITRS)."*Semiconductor Industry Association* (2013).

[Krauss 94] Krauss, P. A., and M. Henftling. "Efficient fault ordering for automatic test pattern generation for sequential circuits." *Test Symposium, 1994., Proceedings of the Third Asian*. IEEE, 1994.

[Kantipudi 06] Kantipudi, Kalyana R., and Vishwani D. Agrawal. "On the size and generation of minimal N-detection tests." *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*. IEEE, 2006.

[Kantipudi 07] Kantipudi, Kalyana R., and Vishwani D. Agrawal. "A reduced complexity algorithm for minimizing N-detect tests." *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*. IEEE, 2007.

[Ku 14] Ku, Jerry CY, et al. "Suppressing test inflation in shared-memory parallel Automatic Test Pattern Generation." *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2014.

[Lee 02] Lee, Sooryong, et al. "A new ATPG algorithm to limit test set size and achieve multiple detections of all faults." *Proceedings of the conference on Design, automation and test in Europe*. IEEE Computer Society, 2002.

[Lin 06] X Lin, K. Tsai, C. Wang, M. Kassab, J. Rajaski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada, and T. Aikyo, "Timing-aware ATPG for high quality at-speed testing of small delay defects," *Proc. Asian Test Symp.*, Nov. 2006, pp.139-146.

[Liao 13] K.-Y. Liao, S.-C. Hsu, and J. C.-M. Li, "GPU-based N-detect transition fault ATPG," *Proc. IEEE/ACM Design Automation Conf.*, Jun. 2013.

[Ma 95] Ma, Siyad C., Piero Franco, and Edward J. McCluskey. "An experimental chip to evaluate test techniques: Experiment results." *ITC*. 1995.

[Pomeranz 91] Pomeranz, Irith, Lakshmi N. Reddy, and Sudhakar M. Reddy. "COMPACTEST: A Method to Generate Compact Test Sets for Combinatorial Circuits."*Proceedings of the IEEE International Test Conference on Test: Faster, Better, Sooner*. IEEE Computer Society, 1991.

[Pomeranz 00] Pomeranz, Irith, and Sudhakar M. Reddy. "On n-detection test sets and variable n-detection test sets for transition faults." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.3 (2000): 372-383.

[Pomeranz 05] Pomeranz, Irith, and Sudhakar M. Reddy. "The accidental detection index as a fault ordering heuristic for full-scan circuits." *Design, Automation and Test in Europe*. IEEE, 2005.

[Raghunathan 95] Raghunathan, Anand, and Srimat T. Chakradhar. "Acceleration techniques for dynamic vector compaction." *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society, 1995.

[Remersaro 09] Remersaro, Santiago, et al. "A scalable method for the generation of small test sets." *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009.

[Tseng 01] C.-W. Tseng and E. J. McCluskey, "Multiple-Output Propagation Transition Fault Test," *Proc. Int'l Test Conf.*, pp. 358-366, 2001.

[Touba 06] Touba, Nur A. "Survey of test vector compression techniques." *IEEE Design & Test of Computers* 23.4 (2006): 294-303.

[WWW Book] Wang, Laung-Terng, Cheng-Wen Wu, and Xiaoqing Wen. *VLSI test principles and architectures: design for testability*. Academic Press, 2006.

[Xiang 14] Xiang, Dong, et al. "Compact test generation with an Influence input measure for launch-on-capture transition fault testing." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.9 (2014): 1968-1979.

[Yilmaz 08] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-pattern grading and pattern selection for small-delay defects," *Proc. IEEE VLSI Test Symp.*, Apr. 2008, pp. 233-239.