EasyChair Preprint

№ 588

Online Scan Diagnosis - A Novel Approach to Volume Diagnosis

I-De Huang, Pallav Gupta, Loganathan Lingappan and Vijay Gangaram

October 24, 2018

Online Scan Diagnosis
A Novel Approach to Volume Diagnosis

I-De Huang, Pallav Gupta, Loganathan Lingappan, Vijay Gangaram
Intel Corporation
1900 Prairie City Road, Folsom, CA 95630, USA
{i-de.huang|pallav.gupta|loganathan.lingappan|vijay.gangaram}@intel.com

***Abstract*—In this paper, we present a novel approach to run scan diagnosis in a high volume manufacturing (HVM) environment. Our methodology enables two usage modes during scan diagnosis; we can perform diagnosis on *all* failing partitions on *all* units (a) on production testers directly or (b) on a *few* compute machines. Diagnosis or fault dictionary based methods are employed to pre-generate a suspect database (DB) for each design partition. Computationally efficient capture and chain diagnosis algorithms are developed to query the DB to form a suspect universe that explains the fail signature. Only certain pages of the DB are loaded in memory at any given time, requiring no more than a few megabytes of RAM. The technique has been successfully demonstrated on two Intel products with significant cost savings realized over the traditional approach. Diagnosis results have been validated by comparing against the output of a commercial diagnosis tool on thousands of wafers; more importantly, 97% of actual defects confirmed by physical failure analysis (FA) are successfully identified by the proposed approach.**

I. Introduction

Yield analysis (YA) is performed to (a) accelerate yield ramp, (b) debug excursions, and (c) improve mature yield. Scan diagnosis is a useful tool in the YA framework [1] [2] [3] [4] [5], and currently it can be summarized into two categories: hardware/tester based methods [6] [7] [8] [9] [10] and software based methods [11] [12] [13] [14]. However, selecting representative units, which highlight either existing or new defect modes, for failure analysis (FA) is difficult. Consequently, the electronic design automation (EDA) industry has developed software based volume scan diagnosis flows that statistically analyze diagnosis results from a large number of units to identify potential causes of yield loss at end-of-line. Timely resolution, whether it be in process, design, or test, is a boon to any product.

Although volume scan diagnosis flows are well defined, deploying them in an HVM environment is fraught with challenges. Our ability to scale to *multiple* products simultaneously is severely limited because we cannot afford to invest the necessary resources required to enable and maintain the flows. Based on our work at Intel, the main problems that plague the traditional approach can be summarized, in priority order, as follows:

1) It is a herculean task to coordinate with the various scan test content teams to collect, sanity-check, and manage diagnosis input collaterals (partition models, test patterns, layout DB, *etc*) in a timely fashion that are synchronized with the production test contest for every product stepping. Otherwise, significant delay is incurred in enabling diagnosis; and
2) A large number of *dedicated* compute machines are needed to process the diagnosis jobs with a reasonable turnaround time. Otherwise, heavy sampling is used which limits visibility for YA teams; and
3) Tester datalogs quickly grow big when there are lots of failures. Diagnosis throughput time is affected as we have to spend time splitting datalogs to generate individual failure files (one file per failing partition) for running diagnosis.

To address the aforementioned shortcomings, this paper proposes a novel, yet streamlined approach to enable volume scan diagnosis. The fundamental premise is to employ diagnosis or fault dictionary based methods to pre-generate a suspect DB for each design partition for all diagnosis scan test patterns only. This can be done during the test content generation phase. Using heuristic algorithms, to be described later, we perform fast lookup operations to form a suspect universe that explains the fail signature. The technical merit of this approach is that the algorithms are so simple and efficient that they can either run directly (hence, Online Scan Diagnosis (OSD)) on an automatic test equipment (ATE) (with minimal test time overhead) or on a few, undedicated computing machines. To be clear, we have demonstrated both use cases on two Intel products.

The reader may harbor suspicion regarding the feasibility of pre-generating diagnosis DB's for all design partitions in a reasonable amount of time. First, it should be mentioned that diagnosis scan test patterns generally are a subset of the full scan production test suite (10-50% per partition). Second, while we have used a brute force approach to generate the DB's, this is mainly due to the limitation imposed by our automatic test pattern generation (ATPG) tool. With some modifications, an ATPG tool can generate the diagnosis DB as a by-product during test generation. Later, we will present some experimental data to justify our claim.

The key contribution of this work is that, to the best of the authors' knowledge, this is the first practical demonstration of successfully running volume scan diagnosis on ATEs directly on a complex product with hundreds of partitions. In doing so, our methodology addresses the limitations of the traditional approach as follows:

1) Scan test content teams are now responsible for delivering diagnosis lookup DB's. Since these are generated during ATPG, they are easily synchronized with production test content.
2) Dedicated compute machines are no longer required. If running OSD on ATE's is not desired (this can

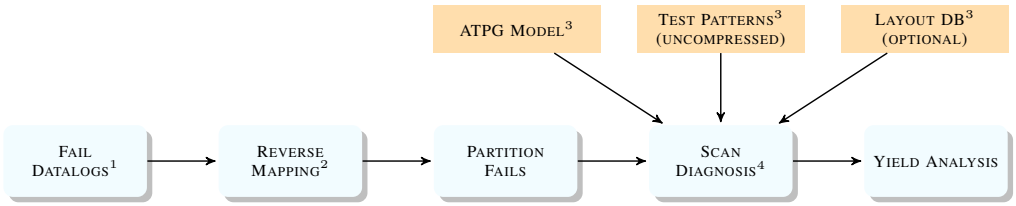

Fig. 1. A traditional scan diagnosis flow diagram.

happen on products with mature yields where test time is the primary concern), we can run diagnosis on a few, undedicated compute machines.

3) If OSD is run on an ATE, instead of capturing failures, we generate candidate suspects. Thus, the datalogs are significantly smaller in size.

The remainder of this paper is organized as follows: In Section II, we discuss some preliminaries that will be helpful in understanding this work. We describe our OSD methodology in detail in Section III. Capture and chain diagnosis algorithms based on this methodology are developed in Sections IV and V, respectively. The challenges and their solution in implementing OSD on automatic test equipment (ATEs) is discussed in Section VI. The encouraging results from deploying our work on two Intel products is presented in Section VII. We conclude this paper in Section VIII.

II. Preliminaries

In this section, we present some background material that will be helpful in understanding this work.

The traditional scan diagnosis flow diagram is shown in Fig. 1. Scan pattern failures are collected for every partition on every unit during wafer sort. The failures are represented using a 3-tuple: pattern index, cycle number, and I/O pin name. Since the I/O pins are at ATE-level (*i.e.*, full-chip), and diagnosis runs at partition-level, a reverse mapping procedure is applied to convert the cycle fails to scan cell fails. Failure files are then created by splitting the datalog to grouping fails by partition across different units. The diagnosis jobs are then submitted to a compute farm for processing using an EDA diagnosis tool. After the diagnosis runs are completed, results are transferred to a DB where they can be analyzed by YA engineers.

III. Overview of OSD Methodology

In this section, we describe our OSD methodology in details.

A. DB Generation

Some approaches use the fault simulation based dictionary for diagnosis existed before [1] [15] [16] [17], but the common challenges all come down to that it cannot be scaled to a larger partition as the dictionary size depends on the number of faults and patterns. Some researchers were able to apply dictionary approach on chain diagnosis since the number of faults are determined by the total number of scan cells which is usually $\sim$2 to 3 order of magnitudes smaller. However, in order to reduce the dictionary size, additional overheads to calculate relative signature (instead of direct lookup) were introduced [15].

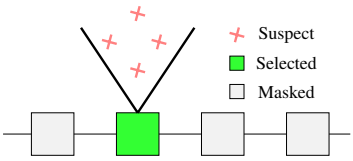

Fig. 2. An example of creating suspects for a single scan cell fail.

As mentioned earlier, the fundamental differences in our approach compared with other fault dictionary based approach are (a) to use diagnosis to pre-generate a suspect DB, and (b) two DB's are generated for every partition so direct lookups can be applied to avoid the overheads to perform forward and backward analysis. The first DB enumerates a set of pinpaths (*e.g.*, pins of standard cells) that will propagate a defect to an observation point (*i.e.*, a scan cell or a primary output). We call this the *Fail* DB. The second DB captures all the scan cells that will observe a defect on a pinpath. We call this the *Pass* DB. The reader will observe that once one of the DB is generated, we can directly compute the Pass (Fail) DB by reversing the data in the Fail (Pass) DB. Furthermore, the process of generating the Fail DB is equivalent to running diagnosis, while the process of generating the Pass DB is equivalent to running non-dropping fault simulation.

1) Diagnosis-based Fail DB: For generating the suspect DB (Fail DB) using diagnosis, we assume that there is a mismatch on a single scan cell while masking all other scan cells. We then ask a diagnosis tool to report a set of suspects that explain the failure on this scan cell for a scan pattern. The procedure is repeated for every scan cell in every scan chain using all the diagnostic patterns for a given partition. In essence, the tool is generating the fanin cone of the scan cell, and enumerating all the faults that can be propagated within the cone to that scan cell. Fig. 2 illustrates this concept.

2) Fault Simulation-based Pass DB: For generating the Pass DB using fault simulation, one suspect/fault is simulated in an ATPG tool to report a set of scan chain and cell failures for every scan pattern. The procedure is repeated for every suspect for a given partition. In essence, the tool is generating the fanout cone of a suspect, and enumerating all scan chains and cells where the fault can be observed. The brute force approach to run fault by fault simulation is used in order to generate Pass DB due to the limitation imposed by our automatic test pattern generation (ATPG) tool which will be discussed in Section VII-A

It is worth mentioning that the DB generation process described above can work on both compressed and uncompressed patterns. Furthermore, it is fault model agnostic as we do not put any limitation on the diagnosis tool on the types of faults

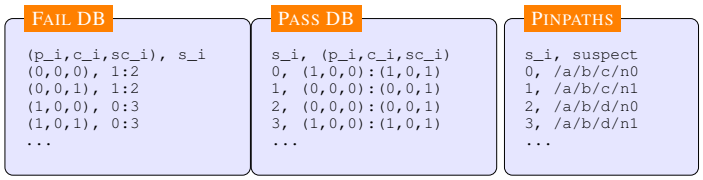

Fig. 3. An example of DB schemas.

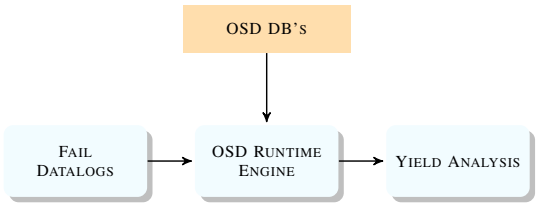

Fig. 4. A simplified OSD-based scan diagnosis flow diagram.

to seed.

B. DB Schema

Once the necessary information for the Pass and Fail DB's has been generated, it needs to be stored in an efficient manner to enable fast query operations. We investigated multiple options, and settled on Berkeley DB [18] [19] - a high-performance embedded DB for key/value data. Unlike traditional dictionary based algorithms, Berkeley DB requires only a small chunk of a dictionary is loaded into memory at any given time. This size of the chunk is configurable (e.g., 256KB) and pages of the chunk can be swapped with the dictionary at a very high speed (with the help of B+tree and extended linear hashing access methods) when a dictionary lookup results in a miss in the chunk. Even with a very large size of DB (e.g., several GB), it is confirmed that loading and accessing DB has negligible impact on both of the memory footprint and algorithm runtime.

Our DB schemas and an example are shown in Fig. 3. The Pass DB is indexed using a bit-packed 3-tuple which comprises of the pattern number, scan chain, and scan cell index (shown as p_i, c_i, and sc_i, respectively, in Fig. 3), and such order makes strong form of locality of reference to reduce the miss rate in DB cache. The return value is a set of integers delimited by ':'. Each integer represents a unique pinpath. A separate file is maintained that maps pinpaths to these integers. The Fail DB is indexed using a pinpath index which is assigned to the pinpath by the order of pattern number, scan chain, and scan cell index, so the strong form of locality of reference is also enforced. It also returns a set of integers that can be unpacked to obtain the pattern number, scan chain, and scan cell index. For our use case, 32-bit integers are sufficient. By this point, readers may already know how to estimate the size of Fail/Pass DB. For example, the size of Fail DB is determined by the total number of scan chains, the scan chain length, the total number of patterns, and the average number of suspects that can be observed at a scan cell.

C. OSD Flow Diagram

Now that we know how to generate the DB's, we can significantly simplify the traditional scan diagnosis flow shown in Fig. 1 to the new flow shown in Fig. 4. The reverse mapping of ATE I/O fails to partition level I/O fails is now a part of the runtime engine. We parse the datalog to group the fails for each partition on every unit. Then we process each partition in sequential order by opening a handle to the DB's to diagnose all the jobs associated with that partition. In this manner, we are able to obtain speedup that is over *two* orders of magnitude compared with the conventional diagnosis flow with the help of pre-generated DB's and OSD algorithms.

TABLE I
REMOVE SUSPECT(S) WITH A FAIL MISMATCH.

Suspects	Expected Fails	Actual	Result
1	Pat3, Cell500	PASS	Eliminate
	Pat5, Cell502	FAIL	
	Pat7, Cell500	FAIL	
2	Pat4, Cell500	FAIL	Keep
	Pat6, Cell502	FAIL	
	Pat8, Cell500	FAIL	

IV. OSD CAPTURE ALGORITHM

A Fault-based OSD capture algorithm is first proposed, and it demonstrates high diagnosis accuracy when the suspect defect type is Stuck-at (SA). (Here we assume the diagnosis results returned by the commercial diagnosis tool used in house are golden.) In order to obtain high diagnosis accuracy on the defect type other than SA (that includes Open/Short, Bridge, Cell, and Unknown defects), Cell-based OSD capture algorithm is introduced. In this section, we describe both OSD capture algorithms in details, and the experimental results are presented in Section VII.

A. Fault-based OSD Capture Algorithm

We first developed Fault-based OSD capture algorithm to serve the purpose of running scan diagnosis on ATE. It is computationally efficient and achieves speedup over two orders of magnitude compared with the conventional diagnosis flow. The proposed Fault-based OSD algorithm starts with querying Fail DB for every failed scan cell to form a suspect universe. Pass DB is queried for each suspect from the universe to obtain expected fail patterns, scan chains, and scan cells. If the expected fails does not match the actual fail signature, such suspect is removed. Table I demonstrated a case where there are 2 suspects, Suspect1 and Suspect2. Suspect1 supposes to fail Pat3, Pat5, and Pat7. However, the actual fail signature shows Pat5 and Pat7 fail while Pat3 passes. Hence, Suspect1 is removed from the suspect list. We then start to iterate through retained suspects to obtain the patterns where the fail signature is explained by the suspect. Suspects that have the maximum explained patterns are first picked up, and suspects that explain the same patterns are lumped together as a symptom. The process is repeated until all failed patterns are covered.

B. Cell-based OSD Capture Algorithm

As described earlier, the Fault-based OSD capture algorithm only identified suspects that can perfectly match actual fail signature for explained patterns. When the defect type is SA, the proposed algorithm is able to match commercial diagnosis tool, and those are confirmed 100% to be the actual defects by physical failure analysis (FA). However, we observed the

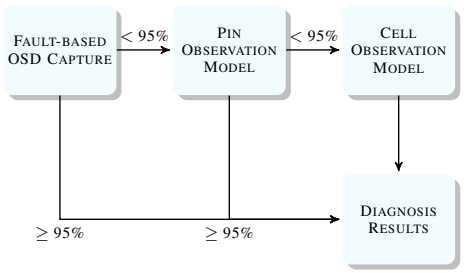

Fig. 5. The proposed Cell-based OSD capture algorithm flow diagram.

proposed algorithm matched poorly (match ∼15% of actual defects by FA) when the defects types are Open/Short, Cell, Bridge, and Unknown. After further investigation, the following root causes are identified:

1) The proposed algorithm forces restrict constraints by discarding suspects that do not match the fail signature perfectly for explained patterns even though they are the best candidates where the defect type other than SA may occur.
2) The Pass/Fail DB is generated pattern by pattern, and when a single pattern is diagnosed, the suspects in reports can only contain SA defects. However, for the defect type other than SA, it may require 2 patterns to detect it [20] [21]. It requires either additional DB to be generated for other types of defects specifically or a new heuristic algorithm to model other types of defects with exploiting the existing SA DB.

The Fault-based OSD capture algorithm is then enhanced with the following three key techniques to tackle the gaps identified above.

1) A scoring system is introduced to loosen the strict constraint which requires a suspect to match the fail signature perfectly. The score is calculated for every potential suspect on each pattern based on the percentage of fail signature match. At the end, the average score per pattern is recorded for every potential suspects and use it as a guideline to only select those suspects with the highest score in the diagnosis results.
2) A generic fault propagation and observation model at the pinpath level is proposed based on the following axiom.
   Axiom 1: Any type of defect that occurs at a pinpath can be the diagnosis suspect if both SA defects (SA0 and SA1) at a pinpath can collectively explain the failed pattern.
3) A generic fault propagation and observation model at the cell instance level is proposed based on the following axiom.
   Axiom 2: Any type of defect that occurs within a cell can be the diagnosis suspect if both SA defects (SA0 and SA1) at the cells output pinpaths can collectively explain the failed pattern.

The proposed cell-based OSD capture algorithm flow diagram is shown in Fig. 5, and details on each key idea are described next.

1) Scoring System: For a given potential suspect, the score is calculated for each pattern based on the percentage of fail signature (at scan cells) such suspect can explain (from the lookup in Pass DB). In the Fault-based OSD capture algorithm described earlier, the percentage of fail signature the heuristic is seeking to explain is always 100%, i. e., all failed scan cells need be explained by such suspect for a given pattern. If this requirement is not satisfied, the suspect will be dropped and not be included in the diagnosis results. Due to this 100% match constraint enforced in the algorithm, some cases end up reporting suspects that are able to explain much less number of failed patterns with 100% fail signature match (that leads to multiple symptoms with incorrect suspects) while dropping suspects that are able to explain all failed patterns with, lets say, 99% fail signature match. By having a scoring system with high threshold (e.g., 95% shown in Fig. 5) set in the heuristic helps to recover such suspects. As demonstrated in Fig. 5, if the proposed heuristic can identify a set of suspects that are able to explain all failed patterns with more than 95% match, the diagnosis is concluded. Otherwise, we will proceed to the next step.

Fig. 6. An example of fault propagation and observation model at pinpath level.

2) Fault Propagation and Observation Model at Pinpath Level: For any type of defect other than SA (e.g., Open/Short), the defect occurs at a cell pinpath may behave like a SA0, SA1, or no faulty value under different logic conditions by different patterns. In order for this pinpath to be a suspect, it need explain the failed pattern with a faulty value at the pinpath that can be propagated and observed at those failed scan cells. The faulty value can be either 0 or 1. Hence, the idea here is instead of analyzing and simulating a pattern to determine if the defect would behave like SA0 or SA1 at a pinpath, the expected fail signatures of SA0 and SA1 at the pinpath are both tested to check if either one matches the fail signature. If it matches, we can conclude that this failed pattern can be explained at this pinpath by a defect.

In the example shown in Fig. 6, there are total 6 failed patterns. P1 and P6 can be fully explained by the SA0 at the pinpath /a/b/c/out while P2, P5, P7, and P23 can be fully explained by SA1 at the same pinpath. Collectively SA0 and SA1 at the pinpath /a/b/c/out can explain all 6 failed patterns. The proposed heuristic will conclude there exist a defect at the pinpath /a/b/c/out. Readers may observe that the proposed heuristic will only identify that a defect occurs at the pinpath. However, it is not able to conclude the defect type. For yield analysis (YA) point of view, this is acceptable as the key information such as pinpath name, net that is connected to this pinpath, the cell template name, and even the layout information can still be obtained to perform YA without knowing the defect type. As demonstrated in Fig. 10, if the proposed heuristic can identify a set of suspects that are able to explain all failed patterns collectively with both

SA0 and SA1 with more than 95% match, the diagnosis is concluded. Otherwise, we will proceed to the next step. By this stage, only a small percentage of diagnosis runs (∼10%) found to proceed to the next step.

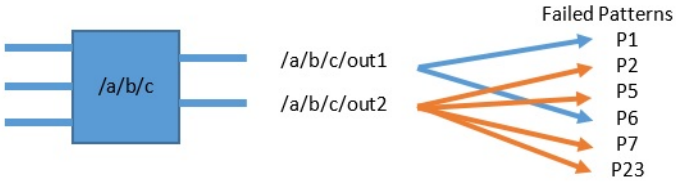

Fig. 7. An example of fault propagation and observation model at cell level.

3) Fault Propagation and Observation Model at Cell Level: For any type of defect occurs within a cell may or may not cause a faulty value appears on at one of its output pinpath(s) under different logic conditions by different patterns. In order for this cell to be a suspect, it need explain the failed pattern with a faulty value at one of its output pinpath that can be propagated and observed at those failed scan cells. The faulty value can be either 0 or 1. Hence, the idea here is instead of analyzing and simulating a pattern to determine if the defect within a cell would behave like SA0 or SA1 and propagate to which of its output pinpath(s), the expected fail signatures of SA0 and SA1 at all its pinpath(s) are all tested to check if any one matches the fail signature. If it matches, we can conclude that this failed pattern can be explained by a defect within this cell.

In the example shown in Fig. 7, P1 and P6 can be fully explained by a defect occurs at the pinpath /a/b/c/out1 (It can be collectively explained by SA0 and SA1, or it can be explained only by either SA0 or SA1.) while P2, P5, P7, and P23 can be fully explained by a defect occurs at /a/b/c/out2. Collectively pinpath /a/b/c/out1 and /a/b/c/out2 can explain all 6 failed patterns. The proposed heuristic will conclude there exist a defect within the cell /a/b/c. As demonstrated in Fig. 10, if the proposed heuristic can identify a set of cells that are able to explain all failed patterns collectively with both SA0 and SA1 at cells output pinpaths with more than 95% match, the diagnosis is concluded. Otherwise, suspect(s) with highest score will be reported.

With these 3 key changes encapsulated in the Cell-based OSD capture algorithm, 97% of actual defects (improved from 15%) confirmed by FA are successfully identified as suspects in the diagnosis results by our proposed heuristic. More experimental results are presented in Section VII.

V. OSD Chain Algorithm

The normal practice of software-based chain diagnosis usually consists 2-step tests. A chain test that consists of shift-in and shift-out operations is first applied to detect any scan chain fail and to determine the defect type by simply analyzing the faulty shift-out sequence [22]. If a scan chain fails, scan/capture tests that are specifically generated by targeting the faults at scan cell pins are run to help allocate the defect location [23] [24].

Fig. 8 demonstrates a chain diagnosis case where a SA defect occurs at (chain_2, cell_5). Initially the lower bound

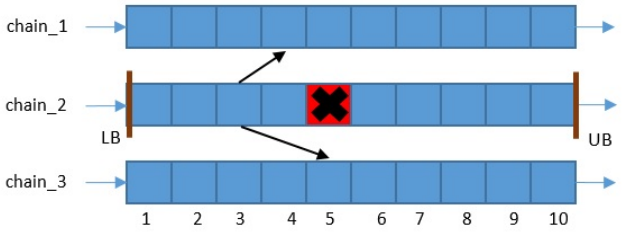

Fig. 8. An example of chain diagnosis case.

(LB) is set to (chain_2, cell_1) and the upper bound (UB) is set to (chain_2, cell_10). Lets assume it is learned from Pass DB that SA0 and SA1 at (chain_2, cell_3) would cause (chain_3, cell_5) and (chain_1, cell_4), respectively, to fail pattern_p. If (chain_3, cell_5) does not fail and the expected shift value is 1 at (chain_2, cell_3) for such pattern, it implies (chain_2, cell_3) will not have SA0 defect, i.e., it can be set to 1 by shift. Similarly, if (chain_1, cell_4) does not fail pattern_q and the expected value is 0 at (chain_2, cell_3) for that pattern, it implies (chain_2, cell_3) will not have SA1 defect, i.e., it can be set to 0 by shift. Since both value 0 and 1 can be set at (chain_2, cell_3) by shift, we can conclude that the defect at chain_2 must occur after cell_3. (If a SA occurs before the cell_3, cell_3 can only be set to one single value by shift.) Hence, LB can be refined to cell_4.

A. Implication Graph

We start with the definitions of controllable and observable which are used to refine LB and UB, respectively.

Definition 1: (chain_i, cell_j) is said to be controllable-1 and controllable-0 if it can be set to value 1 and value 0, respectively, with shift. (chain_i, cell_j) is said to be controllable if it is both controllable-1 and controllable-0.

Definition 2: (chain_i, cell_j) is said to be observable-1 and observable-0 if it passes with expected value 0 and 1, respectively, with shift. (chain_i, cell_j) is said to be observable if it is both observable-1 and observable-0.

Axiom 3: If (chain_i, cell_j) is controllable, (chain_i, cell_k) is controllable for all $k < j$. Hence LB can be improved to scan cell j.

Axiom 4: If (chain_i, cell_j) is observable, (chain_i, cell_k) is observable for all $k > j$. Hence UB can be improved to scan cell j.

The above exercise helps the idea of modeling Pass/Fail DB as an implication graph and use Boolean Constraint Propagation to improve the lower bound (LB) and upper bound (UB) of the potential defect location. Axiom 3 states if (chain_i, cell_j) is controllable, both value 0 and 1 can be set at (chain_i, cell_j) with shift. It indicates no defect can occur before (chain_i, cell_j). Hence, LB can be refined to cell_j. Axiom 4 states if (chain_i, cell_j) is observable, it indicates both expected value 0 and 1 can be observed and passed at this cell and no defect can occur at or after this cell. (If a SA defect occurs at or after this cell, cell_j would always tight to a single value.)

The implication graph can be derived from iterating through Fail DB. For example, given by the example Fail DB in Fig. 9, the implications are created in Fig. 10.

Pattern	Mismatch	Suspects
1	(chain_3, cell_4)(0)	(chain_2, cell_2)(SA0)
2	(chain_3, cell_1)(1)	(chain_2, cell_2)(SA1)

Implication from observable to controllable:
(chain_3, cell_4) = observable and Pass@Pattern1 => (chain_2, cell_2) = controllable-1
(chain_3, cell_1) = observable and Pass@Pattern2 => (chain_2, cell_2) = controllable-0

Implication from controllable to observable:
(chain_2, cell_2) = controllable => (chain_3, cell_4) = observable-0
(chain_2, cell_2) = controllable => (chain_3, cell_1) = observable-1

Fig. 9. An example Fail DB used for implications.

```
Observable[3,4] -> Controllable-1[2,2]
Observable[3,1] -> Controllable-0[2,2]
Controllable[2,2] -> Observable-0[3,4]
Controllable[2,2] -> Observable-1[3,1]
Controllable-1[2,2] & Controllable-0[2,2] -> Controllable[2,2]
Observable-0[3,4] & Observable-1[3,4]  -> Observable[3,4]
Observable-0[3,1] & Observable-1[3,1]  -> Observable[3,1]
```

Fig. 10. Example implications built from a Fail DB.

B. OSD Chain Algorithm

Given by a chain fail datalog for a partition, the proposed OSD chain algorithm starts with building implication graph from its FAIL DB. Then the mismatches on the faulty chain are processed to identify the defect type. Once the defect type is realized, we proceed with the following 2-step process. The first step is to quickly refine UB based on the shift-out sequences for scan tests at the faulty chain followed by feeding built implication graph to Boolean Constraint Propagation engine [25] to improve LB.

Lets assume a scan chain has length of m, and scan vector shift in starting at cell index equal to 1 and shift out at cell index equal to m. The defect type is determined to be SA0. The search to identify the first failing scan cell (that has an expected value 1) starts from the scan cell index equal to m. Another search to identify the first passing scan cell (that also has an expected value 1) starts from the scan cell index equal to 1. The minimum value of these two search results will be assigned to UB. Observable is set to TRUE for every scan cell index larger than UB in the faulty chain.

The second step is to set Observable to TRUE for every passing scan cell in a non-faulty chain by iterating every scan pattern. SMT solver is then launched to perform Boolean Constraint Propagation, and after the run is completed, Axiom1 is applied to refine LB to conclude the defect location. More experimental results are presented in Section VII.

VI. HVM Tester Implementation

The low memory requirements and high computational efficiency of the online scan diagnosis algorithm makes it suitable to be deployed in a sort/class test program. During test program load, the diagnosis dictionary is copied to each tester site controller machine. Site controllers have several 100 GBs of disk space available and the dictionaries can be added without any impact on other disk related needs of the test program. The load time overhead associated with copying both PASS DB and FAIL DB to site tester controller is ∼5-10%. Please note that this is a one-time activity as a test program once loaded is used for several wafers and the overhead gets amortized across these wafers.

After the test program completed loading, test program starts initialization, so a chunk of the dictionaries is loaded into the tester memory. This size of the chunk is configurable and pages of the chunk can be swapped with the dictionary at a very high speed when a dictionary lookup does not find the needed information in the chunk. For each dictionary, the memory overhead at any given time is only 256 KB (size of the chunk). Several tester runs confirmed that this overhead has negligible impact on the memory footprint of the entire test program.

The test method used to run scan content and collect the failure datalogs is enhanced in the following two ways:

1) After the scan content is run on the unit/device under test, the failures or mismatches between expected and actual scan captures are fed directly to the online scan diagnosis algorithm.
2) Instead of storing mismatches, the scan datalog now stores the results of diagnosis algorithm in terms of suspects and symptoms. An added benefit of this approach is to drastically reduce the size of the scan datalog file. While scan datalog file sizes are not of much concern in a mature process node, it is actively managed during the ramp of new process nodes. Given the defect densities, the datalog sizes could get large enough to force either sampling or truncation. However, this is a non-issue with online scan diagnosis.

The test time overhead of adding the additional diagnosis step per wafer is ∼1% of the total wafer runtime. One critic here could be that the algorithm was run only on a few units per wafer. This experiment was done during the initial stages of a process ramp wherein potentially many units on any wafer would fail some portion of the sort content. Hence, the overhead per wafer also translates to a similar overhead per unit. The reduction in scan datalog size is ∼131X as the diagnosis candidates are captured instead of the mismatches. Fig. 11 demonstrates the overall compute time trends (on one CPU thread) for OSD and the commercial diagnosis tool on one of the actual partition. OSD compute time takes one big hit upfront to generate both PASS and FAIL DB's (about 54 hours), and then average 0.076 second per diagnosis case to remain almost flat due to its very low average runtime. The commercial diagnosis tool start at 0 overhead with averages 23 seconds per diagnosis case, and the compute time trend of the commercial diagnosis tool crosses that of OSD and continue to increase at a much faster pace when the total number of diagnosis cases reaches 8,453. (Please note that we give benefit of discounting the time to load model and patterns for the commercial diagnosis tool even though the time may not be always negligible.) This figure perfectly demonstrates OSD approach is more suited to scale up for volume diagnosis while the commercial diagnosis tool is well suited for unit level diagnosis.

The following two approaches aid in eliminating the overhead of the additional diagnosis step and further help in

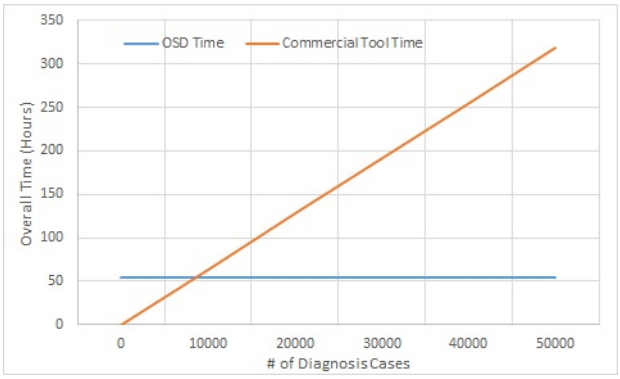

Fig. 11. Compute time trend for OSD and the commercial diagnosis tool.

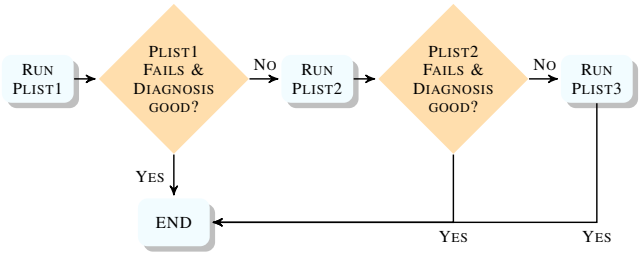

Fig. 12. Test time reduction by splitting pattern lists.

reducing the test time associated with running the scan content.

1) An approach is used to eliminate the test time overhead of diagnosis by launching it as a separate thread once the mismatches are available in the tester memory. An additional checkpoint is needed further down in the test program to wait for the thread to complete and write the diagnosis results to the datalog. Based on multiple wafer experiments, as long as the scan content execution and checkpoint stages are separated by 4-5 tests, the overhead associated with diagnosis can be completely eliminated by running diagnosis in parallel to these tests.
2) A diagnosis is considered to be of high quality if the number of diagnosis candidates (suspect) per symptom (defect) is small (ideally 1). Diagnosis algorithms need a combination of passing and failing pattern information to reason about the candidates. In general, depending on each defect, high quality diagnosis can be attained by using a subset of the test content as not all of the content is relevant for each defect. Given the high computational efficiency of the proposed algorithm, this observation can be exploited to reduce the overall test time per unit. Fig. 12 illustrates this process. The first step is to divide the scan content into smaller pattern lists (plist), said 3 plists. Plist1 is executed first and diagnosis is run. If the resulting diagnosis is of high quality, then the execution of remaining plists is skipped. This saves both the execution and diagnosis times associated with plists 2 and 3. However, if the diagnosis quality is not sufficient or if plist1 passes, then the next plist is executed and diagnosis is run. The process is repeated until a good quality diagnosis is obtained or all plists are executed. Implementation of this approach in a production test program not only eliminated the initial diagnosis overhead but also reduced the test time associated with scan content by 23%. One shortcoming of the approach could happen when plists 1 and 3 detect separate defects and the execution end after plist2 execution, i.e., plist3 is skipped. At a wafer level, we did not see any degradation in diagnosis quality due to this shortcoming as such scenario is extremely rare since it requires different pattern lists to target mutually exclusive faults and failing units to have exactly the same defects (mode and location) associated with the mutually exclusive faults.

TABLE II
ESTIMATED IDEAL RUNTIME FOR DB GENERATION.

Pattern	# of Patterns	Fault Simulation (Seconds)	Fault Simulation without Fault Dropping (Seconds)
P1	32	24	71
P2	64	64	66
P3	10	56	112
P4	636	118	1793
P5	27	149	162

VII. Industrial Results

The section starts with the fault simulation results to demonstrate the speedup OSD DB generation potentially can have with a fault simulation enhancement in the commercial ATPG tool. By assuming the diagnosis results produced by the commercial diagnosis tool are golden, the comparison in terms of diagnosis accuracy and resolution are demonstrated for Fault-based OSD capture algorithm, Cell-based OSD capture algorithm, and OSD chain algorithm for results on many wafers.

A. Ideal DB Generation Runtime

As described in the earlier section, a brute force approach to run fault by fault simulation (or scan cell by scan cell diagnosis) is used in order to generate Pass DB (Fail DB) due to the limitation imposed by the commercial ATPG (Diagnosis) tool. It is believed that with some modifications to report all observable points (at scan cell) for all detectable faults for every pattern, an ATPG tool can generate the Pass DB by running fault simulation without fault-dropping. Table. II illustrates the fault simulation time with and without dropping faults. It demonstrates even with a large number of 636 patterns for P4 and after adding some overheads to the forward simulation engine to track faults with all their observable points (at scan cells) plus I/O time to write out the dictionary table, Pass DB should still be able to be created within a reasonable time (within 24 hours on a single CPU thread). However, for the partition with the same pattern set P4, using a brute force way like what we are doing is this paper would take tens or even hundreds of days on a single CPU thread. This expensive overhead make it difficult to scale up and fully utilize the goodness of proposed method. The expected turnaround time to generate Pass/Fail DB may speed up by 2 to 4 orders of magnitudes if such capability exists.

B. Experimental Results for Fault-based OSD Capture Algorithm

The experiment is setup to randomly inject a number of 1 to 5 SA defects in a partition for totally 10k cases with two types of patterns (bypass patterns and compressed patterns). Failure is reported by performing fault simulation on those injected random SA defect(s) and then it is sent to both the commercial diagnosis tool and our proposed OSD algorithm to diagnose. The accuracy is determined by verifying if diagnosis results return with the injected defect(s) as suspects. We used the following accuracy definitions to help demonstrating the benefits of the proposed OSD algorithms given by k_i is the number of defects correctly reported in diagnosis results, m_i is the number of actual defects injected to the partition, and N is the total number of diagnosis cases.

$$Accuracy_w = \frac{\sum_{i=1}^{N} \frac{k_i}{m_i}}{N}$$

$$Accuracy_p = \frac{\sum_{i=1}^{N} \begin{cases} 1, k_i = m_i \\ 0, k_i \neq m_i \end{cases}}{N}$$

$$Accuracy_a = \frac{\sum_{i=1}^{N} \begin{cases} 1, k_i > 0 \\ 0, k_i = 0 \end{cases}}{N}$$

$Accuracy_w$ is the sum of all weighted accuracy calculated by the percentage of the injected defects identified in diagnosis results for a diagnosis case. If a test case injected by 4 SA defects is successfully diagnosed with 3 suspects, $Accuracy_w$ is equal to 0.75 for such case. $Accuracy_p$ is the sum of all diagnosis cases where all injected defects are identified in the diagnosis results. This indicates all injected SA defects need be in the diagnose report in order for $Accuracy_p$ to be 1. If any injected SA defect is missed in the diagnosis report, it will be 0. $Accuracy_a$ is the sum of all diagnosis cases where at least one injected defects is identified in the diagnosis results.

Fig. 13 demonstrates $Accuracy_w$ for compressed patterns with 8 EDT channels and bypass patterns. We noticed that the commercial diagnosis tool and OSD can both obtain 100% accuracy when there is only one injected SA defect. However, when it comes to multiple defects within a partition, OSD starts to have a slightly higher accuracy from 1% at 2-defect cases and gets bigger (3%) at 5-defect cases.

Fig. 14 illustrated the trend of $Accuracy_p$ by looking for perfect diagnosis (e.g., all injected defects need be reported). Both approaches suffer when the number of injected defects is larger, but still OSD demonstrates 1.5% - 8% and 3% - 5% higher accurate for bypass patterns and compressed patterns, respectively, when it is compared with the commercial diagnosis tool. Fig. 15 shows the resolution (the number of suspects per symptom) and overall OSD produces 50% - 60% less suspects due to more restrict constraints (always look for 100% fail signature matched) are used in our proposed approach.

As illustrated above for ideal SA defect cases, Fault-based OSD capture algorithm matches or perform slightly better than the commercial diagnosis tool in terms of accuracy. It also yields much less suspects as well as 2 to 3 orders of magnitudes speedup against the commercial diagnosis tool. For those real world diagnosis cases that further are been confirmed by physical failure analysis (FA), OSD is able to identify the defect location 100% accurately if the defect type is SA. However, for other defect types, OSD can only identify the defect location 15% accurately. Hence, we proposed Cell-based OSD capture algorithm to overcome the limitation and results are presented next.

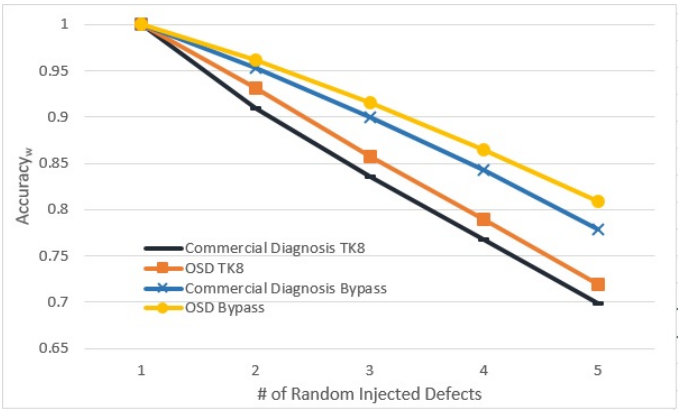

Fig. 13. Weighted diagnosis accuracy comparison for running diagnosis with compressed patterns and bypass patterns.

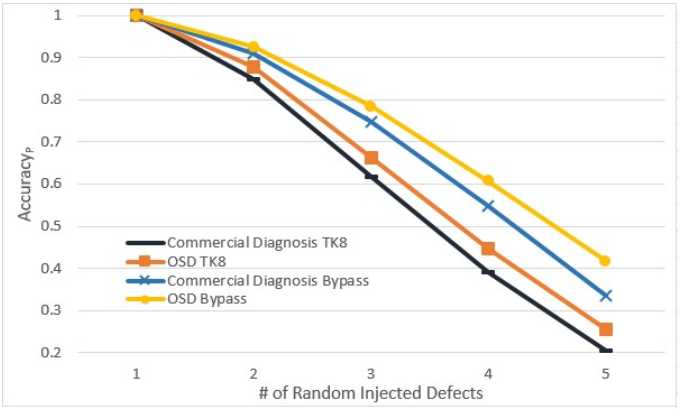

Fig. 14. Perfect diagnosis accuracy comparison for running diagnosis with compressed patterns and bypass patterns.

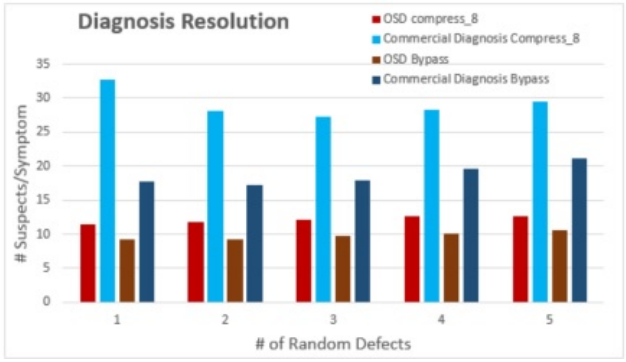

Fig. 15. Diagnosis resolution comparison.

TABLE III
CELL OVERLAP RESULTS FOR 10 WAFERS.

Wafer	Avg. Overlap	Perfect Match	Correlation Coff.
W1	68%	37%	0.97
W2	71%	41%	0.98
W3	68%	36%	0.93
W4	74%	45%	0.98
W5	69%	38%	0.96
W6	70%	41%	0.98
W7	71%	41%	0.98
W8	72%	42%	0.99
W9	67%	36%	0.96
W10	69%	39%	0.99

C. Experimental Results for Cell-based OSD Capture Algorithm

Fig. 16 illustrated the improvement Cell-based OSD capture algorithm over Fault-based OSD capture algorithm for actual diagnosis cases across 10 wafers. For every diagnosis symptom, the overlap is determined by the cell instances (discarded the cell pin in the pinpath) that are identified by both OSD and the commercial diagnosis tool. The average cell overlap is then calculated by the average values of all diagnosis symptoms. As seen in Fig. 16, the cell overlap is improved significantly from 21% (by Fault-based OSD algorithm) to 70% (by Cell-based OSD algorithm) on real world diagnosis cases (diagnosis results shown on all (100+) partitions for 10 wafers). Please note that "OSD Only" percentage is a good indication to determine how close the OSD resolution number is to that of the commercial diagnosis tool, and "Commercial Only" percentage is a good indication to identify the number of "missing" suspects that lower the overlap number. (Again, the results reported by the commercial diagnosis tool are used as the "golden" results for our comparison.)

Table. III shows the average overlap number (ranged from 67% to 74%) across 10 wafers. One interesting observation to be noted is the column of "Perfect Match %" whose values are ranged from 36% to 45%. The symptom is said to be "Perfect Match" if the reported cell suspects are identical between OSD and the commercial diagnosis tool. Overall, it is ~40% of symptoms have the identical diagnosis results between OSD and the commercial diagnosis tool. As the sole purpose of HVM diagnosis is to help YA, Correlation Coefficient on cell template analysis results obtained by OSD and the commercial diagnosis tool is used. It is calculated by the normalized fail rates (based on the instantiated number of a cell template) by having the same order of all cell templates for both approaches. It shows a good correlation (ranged from 0.93 to 0.99) between OSD and the commercial diagnosis tool. Fig. 17 demonstrated a good correlation for top 20 cell templates with Correlation Coefficient equal to 0.99 on 10 wafers. That indicates our proposed algorithm would yield a similar results as the commercial diagnosis tool for serving YA. In terms of accuracy and resolution, Cell-based OSD capture algorithm demonstrated high overlap with the commercial diagnosis tool while still maintaining 2 orders of magnitudes speedup over the traditional diagnosis. More importantly, 97% of actual defects (that include all defect types) confirmed by FA are successfully identified by Cell-based OSD capture algorithm.

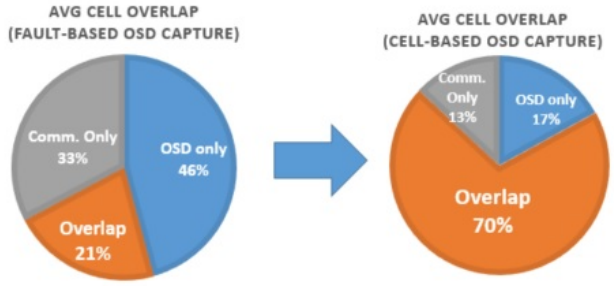

Fig. 16. Overlap improvement from Fault-based to Cell-based OSD capture algorithm.

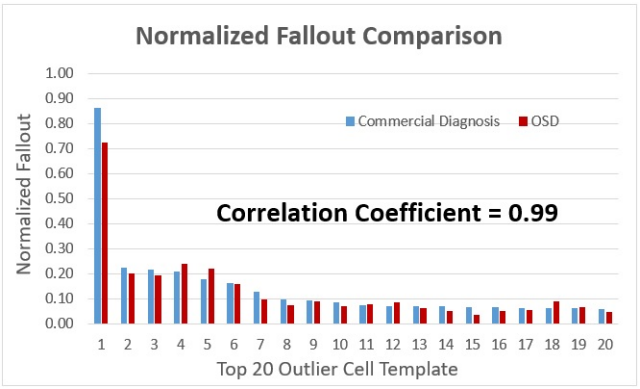

Fig. 17. Correlation Coefficient of commercial diagnosis tool and OSD for YA.

D. Experimental Results for OSD Chain Algorithm

Unlike the previous experiment to inject SA defect randomly in a partition, the fault simulation here is performed based on injecting random SA defects targeting at one single chain for bypass patterns to create 10K failed datalog for diagnosis. Fig. 18 shows the trends of $Accuracy_a$, $Accuracy_w$, and $Accuracy_p$ for both the commercial diagnosis tool and OSD chain algorithm. As seen in the figure, both of the commercial diagnosis tool and OSD yields close to 100% accuracy when there is only single SA defect injected to the chain. For more than one defect in the chain, OSD obtains a better $Accuracy_w$ ranged from 6% to 17%. Both OSD and the commercial diagnosis tool fail to identify all multiple injected SA defects in the chain perfectly. However, if we only consider those cases where we call the diagnosis success as long as one of multiple injected defects can be detected by the chain diagnosis, it is close to 100% accuracy for OSD chain algorithm while it is around 70% for the commercial diagnosis tool. That indicates OSD can at least identify one of the injected defects for close to 100% of cases which makes OSD more effective in the high defect regime. Fig. 19 shows that OSD reports ~55% more suspects for single defect but it is able to obtain a slightly better resolution when it comes to 2 or more SA defects occurring in the chain. Please note that the resolution number is reported only for those cases the commercial diagnosis tool is able to at least identify one defect accurately. The proposed OSD chain algorithm has been successfully demonstrated on two Intel products with significant speedup (~100-400X) realized over

the traditional approach. Diagnosis results have been validated by comparing against the output of a commercial diagnosis tool; more importantly, they have been confirmed by FA.

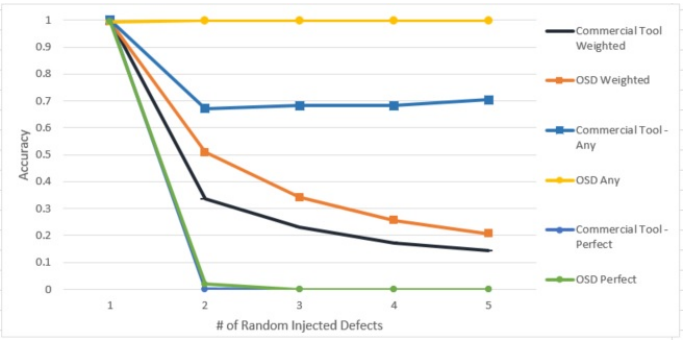

Fig. 18. Diagnosis accuracy comparison for injecting SA defect(s) at single chain.

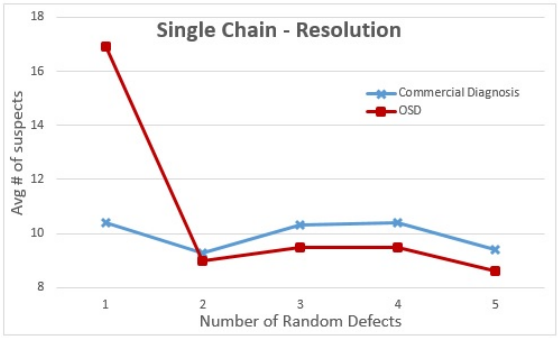

Fig. 19. Diagnosis resolution comparison for injecting SA defect(s) at single chain.

VIII. Conclusion

In this paper, we present a novel approach to run scan diagnosis in a HVM environment. Our methodology enables diagnosis runs either on ATE directly with negligible test time and memory footprint overhead or on a few compute machines. Computationally efficient capture and chain diagnosis algorithms are developed to query the DB to form a suspect universe that explains the fail signature. The technique has been successfully demonstrated on two Intel products with significant cost savings realized over the traditional approach. OSD capture and chain algorithms are observed to obtain 2 to 3 and 1 to 2, respectively, orders of magnitude speedup compared with the commercial diagnosis tool. Diagnosis results have been demonstrated to perform better especially on the high defect regime than a commercial diagnosis tool in terms of accuracy and resolution, and validated on thousands of wafers; more importantly, 97% of actual defects confirmed by physical failure analysis (FA) are successfully identified by the proposed approach.

Our on-going works include to extend OSD chain algorithms to support other than SA defects. With the modifications to report all observable points (at scan cell) for all detectable faults in the commercial ATPG tool, we can also extend OSD algorithms for User Defined Fault Models (UDFM) which enable YA in finer granularity. Fail/Pass OSD DB can also be used to obtain the test coverage to help normalize the fail rate for YA and to help select best diagnosis patterns to improve diagnosis resolution.

References

[1] C. Hora, R. Segers, S. Eichenberger, and M. Lousberg, "an effective diagnosis method to support yield improvement," in *Proc. of International Test Conference*, 2004, pp. 260–269.

[2] B. Seshadri, I. Pomeranz, S. Venkataraman, M. E. Amyeen, and S. M. Reddy, "Dominance based analysis for large volume production fail diagnosis," in *Proc. of VLSI Test Symposium*, 2006, pp. 394–399.

[3] M. Sharma *et al.*, "Efficiently performing yield enhancements by identifying dominant physical root cause from test fail data," in *Proc. of International Test Conference*, 2008, pp. 1–9.

[4] Y. Huang, W. Yang, and W.-T. Cheng, "Advancements in diagnosis driven yield analysis (ddya): A survey of state-of-the-art scan diagnosis and yield analysis technologies," in *Proc. of European Test Symposium*, May 2015.

[5] R. Turakhia, M. Ward, S. Goel, and B. Benware, "Bridging dfm analysis and volume diagnostics for yield learning-a case study," in *Proc. of VLSI Test Symposium*, 2009, pp. 167–172.

[6] K. De and A. Gunda, "Failure analysis for full-scan circuits," in *Proc. of International Test Conference*, 1995, pp. 636–645.

[7] S. Narayanan and A. Das, "An efficient scheme to diagnose scan chains," in *Proc. of International Test Conference*, 1997, pp. 704–713.

[8] F. Motika, P. J. Nigh, and P. T. Tran, "Diagnostic method for structural scan chain designs," in *US Patent 6961886*, November 2005.

[9] S. Kundu and S. Chattopadhyay, "An ate assisted dfd technique for volume diagnosis of scan chains," in *Proc. of Design Automation Conference*, 2013, pp. 1–6.

[10] H. Chen, Z. Qi, L. Wang, and C. Xu, "A scan chain optimization method for diagnosis," in *Proc. of International Conference on Computer Design*, October 2015.

[11] K. Stanley, "Stuck and transient fault diagnostic system," in *US Patent 6694454*, February 2004.

[12] I. Pomeranz, S. Venkataraman, S. M. Reddy, and E. Amyeen, "Defect diagnosis based on pattern-dependent stuck-at faults," in *Proc. of VLSI Design Conference*, 2004, pp. 475–480.

[13] R. Guo, Y. Huang, and W. Cheng, "A complete test set to diagnose scan chain failures," in *Proc. of International Test Conference*, 2007, pp. 1–10.

[14] I. Pomeranz, "Improving the accuracy of defect diagnosis by considering reduced diagnostic information," in *Proc. of VLSI Test Symposium*, 2015, pp. 1–6.

[15] R. Guo, Y. Huang, and W.-T. Cheng, "Fault dictionary based scan chain failure diagnosis," in *Proc. of Asian Test Symposium*, 2007, pp. 45–50.

[16] C. Liu *et al.*, "Hyperactive faults dictionary to increase diagnosis throughput," in *Proc. of Asian Test Symposium*, 2008, pp. 173–178.

[17] I. Pomeranz and S. M. Reddy, "On the generation of small dictionaries for fault location," in *Proc. of International Conference on Computer-Aided Design*, 1992, pp. 272–279.

[18] M. Olson, K. Bostic, and M. Seltzer, "Berkeley db," in *Proc. of the 1999 Summer Usenix Technical Conference*, June 1999.

[19] M. Seltzer, "Berkeley db: A retrospective," in *IEEE Data Eng. Bull.*, 2007, pp. 21–28.

[20] J. C.-M. Li and E. J. McCluskey, "Diagnosis of resistive-open and stuckopen defects in digital cmos ics," in *Trans. on Computer-Aided Design*, 2005, pp. 1748–1759.

[21] H. Wunderlich and S. Holst, "generalized fault modeling for logic diagnosis," in *Models in Hardware Testing*, 2009, pp. 133–155.

[22] W.-S. Chung, W.-C. Liu, and J. C.-M. Li, "Diagnosis of multiple scan chain timing faults," in *IEEE Trans. on CAD of Integrated Circuits and Systems*, June 2008, pp. 1104–1116.

[23] R. Guo and S. Venkataraman, "A technique for fault diagnosis of defects in scan chains," in *Proc. of International Test Conference*, 2001, pp. 268–277.

[24] Y. Huang, W.-T. Cheng, S. Reddy, C.-J. Hsieh, and Y.-T. Hung, "Statistical diagnosis for intermittent scan chain hold-time fault," in *Proc. of International Test Conference*, 2003, pp. 319–328.

[25] A. ErezEmail and A. Nadel, "Finding bounded path in graph using smt for automatic clock routing," in *International Conference on Computer Aided Verification*, 2015, pp. 20–36.