A P P N O T E S SM

Tessent Diagnosis for First Silicon Scan Test Debug

By: Wu Yang
Last Modified: 28th December, 2013

Executive Summary

First silicon bring-up of big and complex designs is the critical first step for yield ramp. Scan pattern is widely used for first silicon testing because of its high test quality, low cost and its diagnosablity. The causes of first silicon scan failures are complicated and debugging is very challenging across the industry. This appnote addresses how to deal with common first silicon failures and how to use Tessent Diagnosis to facilitate first silicon scan test debug.

Table of Contents

1) Understanding First Silicon Scan Test Failures

Scan test is commonly used in first silicon test. Scan patterns are created in ATPG session and it may include chain test pattern, stuck-at test pattern, at-speed test pattern and other types of patterns such as deterministic bridge patterns. To save the test time and debug, it's recommended to apply scan test in the following order:

a) Chain test

b) Stuck-at patterns and other type of patterns such as bridge deterministic patterns

c) At-speed pattern at low frequency and high frequency

d) Other types of patterns if needed

When a device fails its first silicon scan test, a manufacturing defect may not be the only root cause. Other sources can contribute to the failures as well. The following are some common things:

- Power issues (IR drop, power management)
- Timing margin, crosstalk or design related problem
- ATE and its fixture (including load board and probe) setup
- Design and ATE pin mapping
- Test program development
- Poorly verified test patterns (test patterns did not pass simulation)
- Environment including temperature
- Packaging (if device are packaged)
- Mismatch between the netlist used for pattern generation and actual silicon
- Etc.

When a device fails, it might be a natural thought to use the Tessent Diagnosis scan diagnosis tool to find out what is wrong. Tessent Diagnosis is optimized to target manufacturing defects, not to find out failures caused by functional errors, wrong patterns, test hardware setup, etc. Therefore, prior to using Tessent Diagnosis for first silicon debug, it is highly recommended to go through the checklist shown in Table 1:

Item	Symptoms	Source Issue	How to Rule Out	Details
1	Massive failures	Wrong test pattern	Make sure test patterns are compatible with design	See AP-A 1-a)
2	Massive failures	Poorly verified test patterns (such as timing information was not used in the pattern simulation)	Perform test pattern simulation with full timing back-annotation	See *AP-A 1-b)
3	Massive failures	Higher power consumption during test	a) hierarchical DFT; b)make sure load board does not have IR drop between power and socket	See AP-A 1-c)

			c)Use less toggle rate chain patterns	
4	Massive failures	Unbonded pin	Perform pin test for packaged test and make sure all pins are bonded	See AP-A 1-d)
5	Many failures	Test program development related issue	Avoid post-processing test patterns for short chain padding	See AP-A 1-e)
6	Many failures for a certain clk domain, usually chain test fails	ATE and design pin mapping mismatches	Make sure ATE pin and design pin are mapped	
7	Chain pattern failed	Clock hold time issue	Perform timing simulation on test patterns and shmoo vdd during test	
8	Many failures during capture	Scan path defect	a) perform timing simulation b) run Tessent Diagnosis to find out which scan cell failed	
9	Many failures during capture	Slow signals stuck at failures	a) add dead cycle after scan-en goes to 0 b) add slow pad c) expand clk cycle	See AP-A 1-f)
10	Relatively stable failures	Clk edge placement too close to force_PI or measure PO	a) Shmoo clk edge during test and adjust time template accordingly b) slow down clk	
11	Passed on stuck-at test, failed at-speed test	Manufacturing defects	Running Tessent Diagnosis, find out suspect, further analysis by design team or feedback to fab	
12	Passed at-speed pattern at low speed but failed at high speed	Static or resistive defect	Run Tessent Diagnosis normal mode without turning on at-speed diagnosis	
13		At-speed defect	Perform time simulation on patterns at higher frequency; use Tessent Diagnosis to locate at-speed suspect	

*AP-A: Appendix A

Table 1: common issues in first silicon and their solutions

2) Running Tessent Diagnosis for First Silicon Scan Test Debug

After getting rid of test setup and other environment related issue as discussed above, Tessent Diagnosis can be used to facilitate first silicon debug and provide fast path to the root cause of the failures.

2.1 Data preparation for Tessent Diagnosis diagnosis

In order to run Tessent Diagnosis, the user needs to have the design image (the flat model), test patterns and failure files. The following discusses a couple of common considerations when archiving the flat model and test patterns. Please refer to the "Tessent Diagnosis User's Guide" chapter 2 for details.

Archiving Flat model and Test Patterns:

When the flat model is saved, it contains not only the verilog netlist but also the ATPG settings such as pin constraints, cell constraints and test procedures. If for example, you have different verilog netlists for generating test patterns in stuck-at and at-speed test modes respectively, different flat models need to be archived. Always save the flat model right after pattern generation of different atpg runs. **Table 2** shows an ATPG session command example.

```
// Example with different pin settings for different test modes
set_system_mode_atpg
create_patterns
write_flat_model netlist_stuck_at.v.flat –all
write_patterns pat_sta.v –verilog –replace
write_ patterns pat_sta.stil –stil –replace // includes both chain and scan test

set_system_mode_setup
//you make changes to some constrain values and add different test procedure files
add_pin_constrain test_mode 1
add_scan_group proc_for_at_speed
…
set_system_mode_atpg
set_fault_type_ transition –no_launch_shift
create_patterns
write_flat_model netlist_transition.v.flat –all
write_patterns pat_tra.v –verilog –replace
write_patterns pat_tra.stil –verilog –replace –scan
```

Table 2: atpg commands to archive test patterns and flat models

For details, please refer to section "Preparing the Design Netlist" of "Tessent Diagnosis User's Guide".

Collecting failure files

Please refer to the appnotes "Teradyne Tester Interface to Tessent Diagnosis Failure File format" and "Verigy 93K Interface to Tessent Diagnosis Failure File Format" for collecting Tessent Diagnosis failure files from different types of ATEs.

In some cases, the test patterns are saved in multiple files and tested as different bursts on ATE. Correspondently the data logs are collected separately per burst or per test pattern file. When this happens, use the "test_suite" keyword in the failure file. Each "test_suite" keyword tracks the failure bits per pattern file or burst. Please refer to "Understanding Multiple Test Suite Failure File" in the "Tessent Diagnosis User's Guide".

2.2 Chain failure diagnosis

To run an accurate diagnosis on chain failures, the following requirements need to be satisfied (to better understand the structure of chain test, please refer to "TK Logic and Chain Test" in Tessent TestKompress User's Manual):

- Include the chain test failures. With chain failure information, Tessent Diagnosis is able to determine the faulty chain(s) and fault type(s). This is the preferred method.
- Alternatively, if you have no chain test failures but know which chain(s) failed, then you must specify the faulty chain(s) and the fault type associated with each faulty chain.
- Include failures from at least 32 failing scan test patterns to achieve good chain diagnosis resolution.

If you do shmoo on chain test, the chain failure may happen at the pass/fail boundary of shmoo plot. Sometime, the device under test may not show enough failing bits needed to run a successful chain test. It is recommended to push the device a little further into the failing zone so that enough failing bits can be logged. Running Tessent Diagnosis chain diagnosis:

```
read_pattern c101pat.stil –stil
diagnose_failure new_failure_file_from_c101Pat
```

Table 3: Running diagnosis based on iterative patterns

To understand chain diagnosis results, please refer to "Understanding the Chain Diagnosis Report" in Tessent Diagnosis User's Guide.

In some cases, iterative diagnosis (**Figure 1**) in Tessent Diagnosis can be used to generate additional test patterns in order to improve the diagnostic resolution with Tessent Diagnosis. This can be used for both chain and logic diagnosis.

The following scenarios can benefit from using iterative chain diagnosis:

- For TestKompress designs, if a failing chain is one of the multiple failing chains at certain edt channel, you may not be able to get good resolution without having masking patterns. The masking patterns are used to mask some internal chains at an edt channel so that the remaining one or more internal chains can be observable at edt channel. The following shows a chain diagnosis only reports faulty chain suspects but not scan cells:

```
Tessent Diagnosis v8.2009_2.10   Fri May 29 04:15:47 GMT 2009
tracking_info_begin
tracking_info_end
#faulty_chains=0 #symptoms=0 #suspects=0 CPU_time=412.18sec fail_log=fail1.log
Faulty chain c101 is one of the multiple failed chains at edt_channel2.
The current version does not support diagnosing c101 without appropriate masking scan
patterns
//The above message shows that there are no masking scan patterns available in the
// pattern set used for diagnosis. In order for the internal chain c101 to be diagnosed
// successfully, additional diagnosis friendly patterns are needed. That's where iterative
// diagnosis can be used here.
```

Table 4: EDT design chain failure diagnosis needs masking patterns

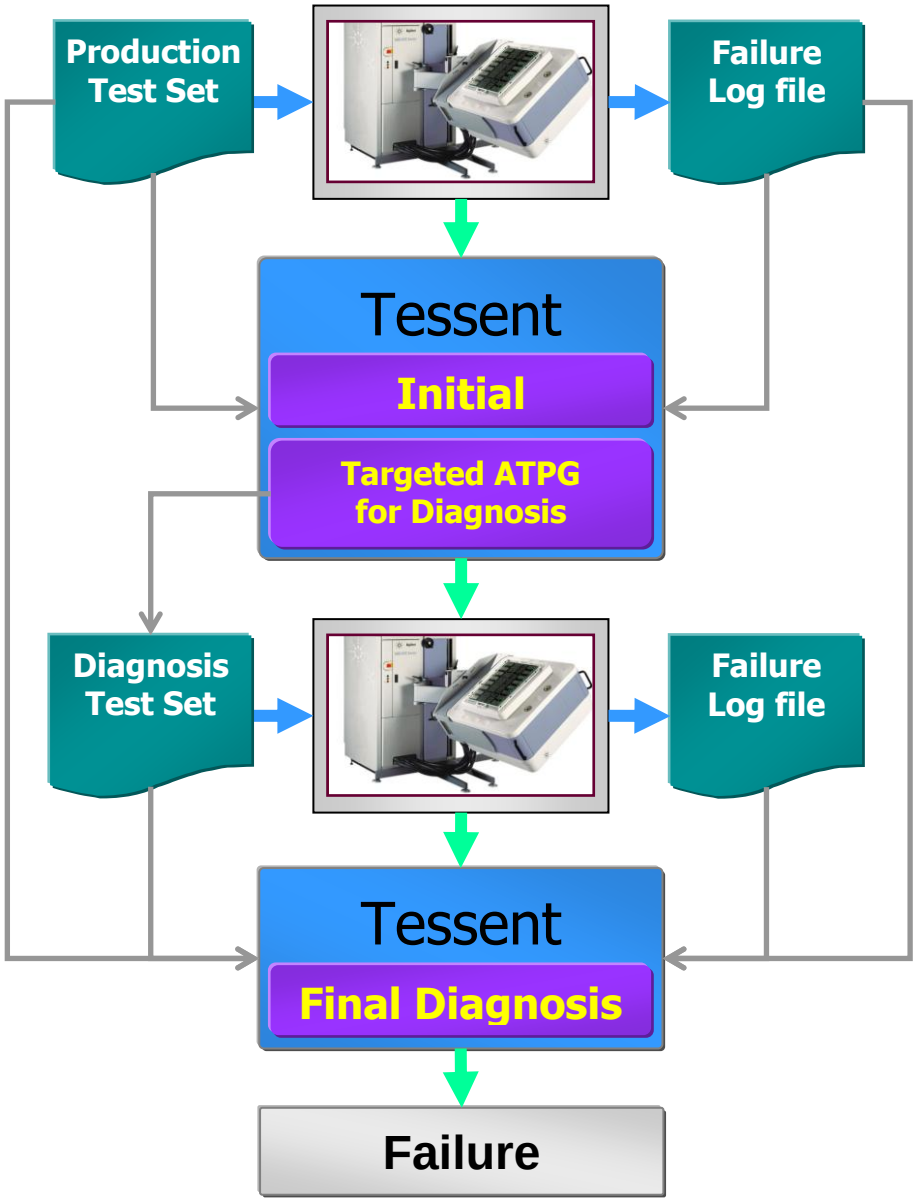

Figure 1: Iterative diagnosis flow in Tessent Diagnosis

Table 2 shows the example commands in Tessent Diagnosis to create masking patterns:

```
read_pattern pat.stil –stil
diagnose_failure faiure_file_1stRound
create_diagnosis_patterns -chain C101
write_pattern c101pat.stil –stil
```

Table 5: iterative diagnosis for chain failure

You need to re-test the patterns on the ATE, collect a new failure file and re-run Tessent Diagnosis for chain diagnosis. Remember to simulate test patterns before running test.

- To improve resolution. Sometime, Tessent Diagnosis chain diagnosis reports a range of scan cells as suspects and you can use iterative diagnosis to narrow down problematic cells.

```
read_pattern pat.stil –stil
diagnose_failure faiure_file_1stRound
create_diagnosis_patterns
write_pattern more_pattern.stil –stil
```

Table 6: chain iterative diagnosis example

Again, you need to do timing simulation of the test patterns, re-test them on the ATE, collect a new failure file and re-run Tessent Diagnosis for chain diagnosis.

a) Scan logic failures

Make sure the chain test passes before running diagnosis on scan logic failures. To understand the diagnosis result and how to link the suspects to the physical layout data, please refer to the "Viewing the Layout-Aware Results"section in the Tessent Diagnosis User's Guide.

If running diagnosis shows too many suspects, you can use iterative diagnosis (**Table 7**)

```
//collect the datalog on the ATE; assume you name the datalog "fail1.log"
//traditional diagnosis
read_pattern src/bridge_patterns.stil -stil
diagnose_failures results/fail1.flog
//view the results, and if diagnosis is needed, write the diagnosis
//result to a file
write_diagnosis -file results/fail1 -rep
//create diagnosis patterns based on the internally stored diagnosis result
create_diagnosis_patterns
//save the targeted diagnosis patterns
write_patterns results/fail1.stil -stil
//save Verilog patterns and run simulation to check for mismatches
//go back onto the ATE and collect the datalog with the new pattern;
//assume you name the datalog "fail1_iter_diag.log"
//ITERATIVE DIAGNOSIS
//load the pattern that the first datalog was collected with
read_pattern src/bridge_patterns.stil -stil
//append the diagnostic pattern
read_pattern results/fail1.stil -stil -append
//read in the original fail log
read_failures results/fail1.flog
//read in the diagnostic pattern generated fail log
read_failures results/fail1_iter_diag.flog -append
//diagnose both of the fail logs that are stored in internal memory
diagnose_failures -internal
```

Table 7: Iterative diagnosis for logic diagnosis

Experiments have shown that iterative diagnosis can significantly reduce suspect number by an average of 50%. We have seen some customers were able to reduce the suspect from 25 to 1.

Iterative diagnosis will not improve results for logic diagnosis if all suspects are equivalent to the first suspect, or if no suspect reported is at all.

You may use layout aware diagnosis to improve resolution (see "Performing Layout-Aware Diagnosis" in Tessent Diagnosis's User's Guide). Layout aware diagnosis helps the resolution if the root cause of the problem is a defect. It requires layout aware database in addition to test patterns, flat model and failure files. The example below (**Table 8**) illustrates using the layout-aware diagnosis flow and includes the following steps:

- Creating the layout file
- Running diagnosis
- Visualizing the results in the layout using Calibre DESIGNrev and Calibre DRC/RVE—see "Viewing the Layout-Aware Results." in Tessent Diagnosis User's Guide.

```
// The creation of the layout file is necessary only on
// the very first invocation. All later invocations can
// use the generated file directly.
create_layout ./src/design.dft.layout \
-lef ./src/design.lef -def ./src/design.def
Read_pattern ./data/patterns.stil -stil
open_layout ./src/design.dft.layout
diagnose_failure ./tester_files/file1.flog
```

Table 8: running layout aware diagnosis

b) At-speed diagnosis

The at-speed test patterns can be applied on ATE at low speed and at high or working speed. If test patterns failed at low speed, it's recommended to run diagnosis as for normal logic diagnosis. Once device passed the stuck-at patterns and at-speed pattern at low speed, it may still fail at working frequency. Tessent Diagnosis can be used to debug this speed related problem in first silicon [4]. Here are typical steps:

1. If possible, run test patterns on known good die in order to find the maximum passing speed.
2. If it is below the desired speed t_max, run transition fault tests at a speed t_max* > t_max and collect fail logs. Here t_max* is a little bit higher than t_max. This step is to make sure that device failed in a consistent way.
3. Run diagnosis (see example below) and get diagnosis report.

```
write_pattern pat_at_speed.stil –stil
set_diagnosis_option –at_speed on
diagnose_failure at_speed_failure_file
```

4. In Tessent Diagnosis, use the DFTVisualizer to highlight a symptom and pick a failing pattern to show the path. Please refer to Tessent Diagnosis User's Guide for details.
5. Check the paths to see whether they are real paths and if yes, please fix the design .e.g. running timing analaysis;
6. If the paths are false/multi-cycle paths, ATPG patterns need to be regenerated including false/multi-cycle paths. You can include false/multi cycle paths by reading SDC file or using the command "add false path".

For steps of adding false/multicycle paths and generating patterns, please refer to "Tessent Scan and ATPG User's Manual".
After patterns are generated, please re-run timing simulation and make sure there are no mismatches. Appling the new test patterns on ATE, if the test still fails at the desired speed, collect failure datalog from ATE and repeat steps 1) to 6)

Figure 2 shows an example where the device is able to run at the desired speed by fixing a timing problem reported by Tessent Diagnosis at-speed diagnosis.

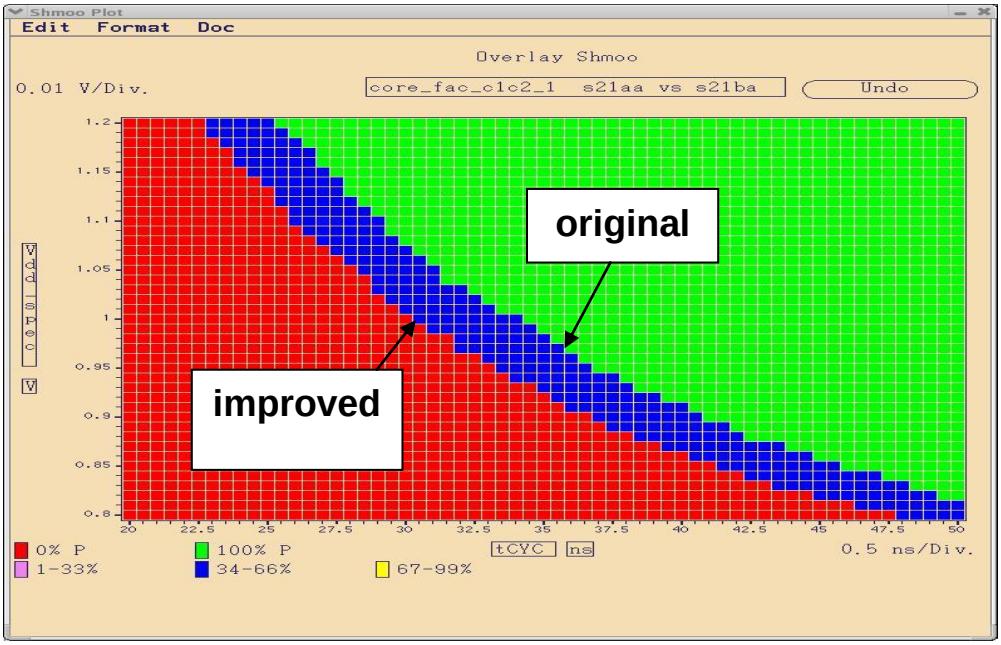

Figure 2: shmoo plot for at-speed test

c) Iddq Diagnosis

Tessent Diagnosis now allows you to perform IDDQ diagnosis on patterns that have failed IDDQ testing. IDDQ diagnosis can help you pinpoint the locations on defective devices that are causing the failures. The failure files you use as input to Tessent Diagnosis should include IDDQ measurements for passing and failing IDDQ patterns. To enable IDDQ diagnosis, specify the following command (for details, please refer to "IDDQ Diagnosis" section of "Tessent Diagnosis User's Guide"):

```
set_diagnosis_options -mode iddq
```

d) Find out internal scan cell path for failed test patterns

During 1st silicon bring up, some users want to be able to identify and mask the flops and quickly produce a stable pattern set. In addition, it may need to do identify/mask observe cells for certain patterns, or all pattern depending on how severe the fails are.

The methods described below are intended to achieve multiple goals

- Quickly get a set of patterns running on the ATE that can be used to screen parts even if we must mask out failures at the channel outputs.
- Identify the internal scan cell that observed the incorrect data and use that information to help identify root case of the mismatching patterns.
- Until root case can be corrected, minimize any coverage losses and create a robust set of compressed mode patterns by masking at the scan cell level, rather than pattern / channel position level.

The following discussed a couple of ways of reporting internal scan cell from failed patterns:

1. "read failures" in Tessent Diagnosis

If the user has a cycle-based failure file, 'read failures fail_file_cycle –pattern > fail_file_pattern' can convert it into pattern based failure file. By default, it also reports the correspondent scan cell ID and the pattern number for each failure bit if it's uncompressed pattern or it is a one hot masking edt pattern.

For compressed patterns, the report will show the internal scan cells for the failure bit of its edt_channel. For example, for an 8:1 compactor, there could be multiple, i.e. 8 internal scan cell paths reported for the failure bit at the edt_channel.

```
format pattern
tracking_info_begin
  injected_fault 0 /cpu_i/uPORT/ix1362/A0
tracking_info_end

//pat_id  channel/PO  channel_cycle  expect_value  sim_value  chain---cell_id---cell_path

4   edt_channel3    14          H           L           //  chain17---14---cpu_i/uPORT/ix1644
                                                        //  chain18---14---cpu_i/uPORT/ix1364
                                                        //  chain19---14---cpu_i/uPORT/ix1414
                                                        //  chain20---14---cpu_i/uPORT/ix757
                                                        //  chain21---14---cpu_i/uINTR/ix136
                                                        //  chain22---14---cpu_i/uMUL/ix928
                                                        //  chain23---14---cpu_i/uINTR/ix216
                                                        //  chain24---14---cpu_i/uPDR/ix520
11 edt_channel3    14          H           L           //  chain17---14---cpu_i/uPORT/ix1644
                                                        //  chain18---14---cpu_i/uPORT/ix1364
                                                        //  chain19---14---cpu_i/uPORT/ix1414
                                                        //  chain20---14---cpu_i/uPORT/ix757
                                                        //  chain21---14---cpu_i/uINTR/ix136
                                                        //  chain22---14---cpu_i/uMUL/ix928
                                                        //  chain23---14---cpu_i/uINTR/ix216
                                                       //  chain24---14---cpu_i/uPDR/ix520
…
```

Table 9: pattern based failure file with internal scan cell IDs and paths for compressed patterns

If the failure file is pattern based, use the command twice:

.

First convert the file into cycle based failure file:

```
read_failures failFile_pattern –cycle > failFile_cycle
```

Then convert it into pattern based failure file with scan cell information:

```
read_failures failFile_cycle –pattern > failFile_pattern2
```

2. Identify internal scan cell by generating one-hot pattern

If you have a failing EDT pattern and want to know which specific internal scan cell failed, it involves creating new debug-only patterns (**Figure 3**). This method is only guaranteed to work with the Xpress compactor.

One can tweak the masking information such that one can force the Xpress compactor to go one-hot, which then can be tested on ATE and identify the internal flops.

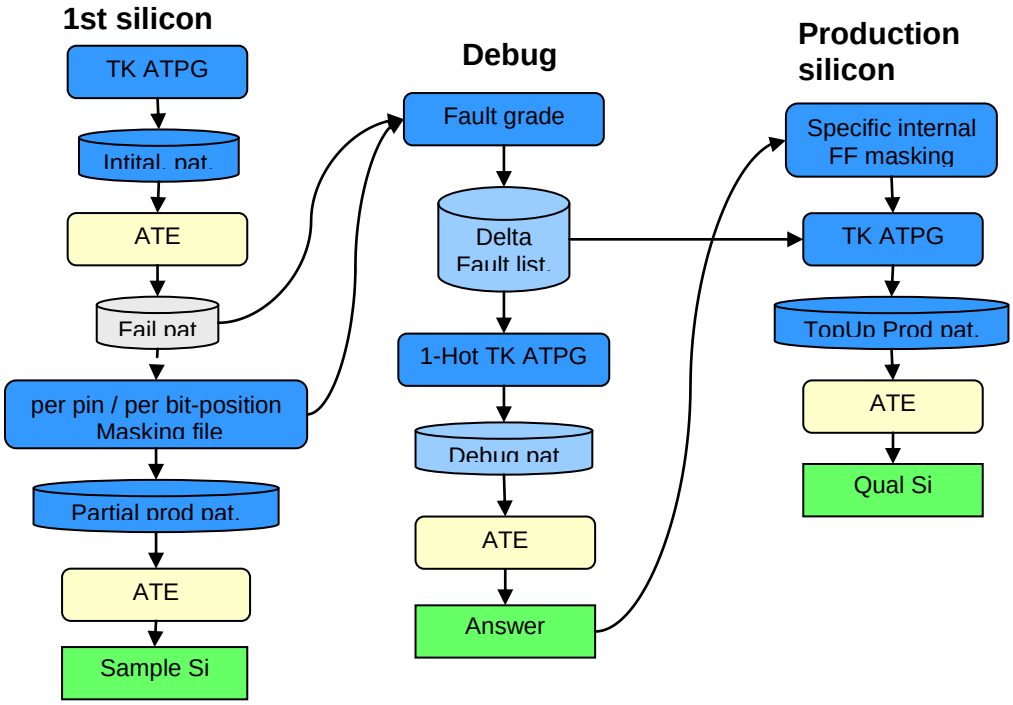

Figure 3: locate internal scan cell for compressed patterns

The following are the steps:

i. Archiving production pattern, flat model and fault list from ATPG process

```
create_pattern
write_pattern edtPat_0_1000.stil –stil
write_flat_model design.v.flat –all
write_fault fault_full -replace
```

ii. Run Pat_0_1000.stil ATE and log failures in Tessent Diagnosis failure file format. By knowing which pattern, pins and bit failed, a script is needed to mask the related failed pins and generate a masking file, i.e. mask_file. For a cycle-based failure file, it's recommended to convert the failure file into pattern based format and create the mask file based on it. The mask file looks like the following:

```
13 edt_06_scan_channel1 57
13 edt_06_scan_channel1 59
13 edt_06_scan_channel1 61
13 edt_06_scan_channel1 64
…
14 edt_06_scan_channel1 59
```

iii. Invoke TestKompress with design.v.flat

```
read_pattern edtPat_0_1000_mask.stil –stil –mask mask_file
set_system_mode_fault
load_fault fault_full
```

```
reset_state
run
write_fault fault_partial –class UC –class UD –class UO
```

iv. Generate one-hot masking patterns

```
set_edt –force_1hot_masking ON
delete_fault -all
load_fault fault_partial
create_pattern
write_flat_model design.v.flat_1hot -all
write_pattern edtPat_1hot.stil –stil
```

v. Apply edtPat_1hot.stil to ATE, collect failure file in Tessent Diagnosis format

vi. Use method e-1) above to report specific internal failing scan cell ID and path for any particular failing bit.

There is some special case for identifying internal scan cell by generating one-hot pattern

It's possible that with the mask file, all the faults are detected in the fault simulation so that there are no undetected faults left to generate one hot edt pattern. In this case, using the following flow:

i. From failure file, get the failing pattern IDs, if you have cycle based failure file, convert into pattern based failure file and get the pattern IDs.

ii. Save the failed patterns out:

```
read_pattern edtPat_0_1000.stil –stil
set_pattern_filter –list 5 10 22 –range 50 55
saave_pattern_ filtered.pat.stil –stil
read_pattern filterer.pat.stil –stil
delete_fault –all
load_fault fault_full
set_system_mode_fault
run
write_fault fault_partial –class DS –class DI –class PD –class PU –class PT
set_edt –force_1hot_masking ON
delete_fault –all
load_fault fault_partial
create_pattern
write_pattern pat_1hot.stil –stil
```

iii. Test the pattern pat_1hot.stil on ATE and collect the failure file and convert it into pattern based failure file to get the internal scan cell path.

3. Identify internal scan cell by using internal scan cell profiling

This feature is available in release 2013.3. Internal scan cell profiling, like 1hot pattern expansion, allows you to report the internal scan cells for compressed failing patterns. This troubleshooting method can help you debug failures by tracing the failures back to the scan cells that have the highest probability of observing the failed device. To enable internal scan cell profiling, set the following command:

```
set_diagnosis_options -internal_failing_cells on
```

For usage details and how to understand the report, please refer to section "Internal Scan Cell Profiling for Compressed Patterns" of "Tessent Diagnosis User's Guide".

e) Common Things to Check When No Suspect Reported

i. Diagnosis message shows actual values in failure files are simulated as Xs. Usually it indicates wrong flat model vs test patterns. Check the ATPG logfile to make sure the flat model was saved after pattern generation.
ii. Massive failures. It usually indicates something is wrong with settings. Please check "Prepare for scan diagnosis" for details.
iii. For edt chain test, it might need masking chain patterns. You can use iterative diagnosis to create masking chain patterns. Then re-run test and diagnosis.
iv. Make sure that the chain pattern passes before running logic diagnosis.

3) Conclusion

Tessent Diagnosis can be used in first silicon scan test debug for chain, scan logic and at-speed related. Examples are shown in recently published articles [4][5][6]. Reducing massive failures and getting stable failures are the keys for successful first silicon diagnosis. Iterative diagnosis is a very important feature of Tessent Diagnosis to achieve refined diagnosis results.

Appendix A: Preparing Tessent Diagnosis for First Silicon Debug

- **1) Reducing Massive Failures to Limited Failures**

When devices fail in first silicon test, it's not uncommon to see massive failures. Before jumping into using Tessent Diagnosis immediately, there are things to check which can help you achieve better diagnosis resolution.

a. *Wrong data*

This sounds very simple but it is the first thing to check. Make sure the test patterns map the device being tested. It's not uncommon to apply a test program for a new device revision

b. Pre-silicon simulation

Formal verification, physical verification, static timing analysis and signal integrity analysis need to be performed per your design specifications. Common considerations were summarized by TI [1].

Scan pattern simulation is another important step to assure high quality test. Poor quality scan patterns can cause first silicon failures and it's hard to diagnose. In case there are simulation mismatches, debug them first. If you run across any mismatch debug issue, please call Mentor support line or open up a service request through supportnet at www.mentor.com/supportnet.

c. Power consumption

There are some preventive DFT methods to make sure power consumption under control. Hierarchical DFT technique and other low power related methods can be considered during scan pattern generation. Please refer to appnotes "Controlling Power Consumption during Scan Test" and "Hierarchical DFT Methodology in a FastScan or TestKompress Flow" for details.

IR drop caused by a load board on the ATE can cause device to fail during test as well: Voltage levels of power at source and at sockets have to be the same. For example, if the metal strip between load board power source and socket driving the device is very thin, it's very likely to cause IR drop. Widening the strip can resolve this issue.

Power-related chain test failure. In ATPG, by default, uncompressed chain patterns are generated with a background of 001100110011. Other background such as 000100000111111010010001100 appears more random and may be good at testing chain. However, if the chain test runs at a higher frequency, the high toggle rate of random chain patterns could potentially cause IR drop and fail chain test. Lowering the chain test frequency can get rid of this kind of chain failures. It's also not a bad idea to generate test patterns with background e.g. 0000111100001111 at high rate chain tests. This is going to reduce the toggle rate.
Sometime, if a pattern failed but fault simulation of this pattern shows it only detects faults not detected by other patterns, it can rule out possibility that the failure is due to power issues.

d. Unbonded pins

If devices are packaged, make sure the pins are bonded. We did see a case where massive failures were caused by this type of problem. Since all the pins were floating, scan diagnosis may not give any meaningful results.

e. Test program development

If the tester supports WGL and STIL pattern format directly, then it is recommended to save the test patterns in the same format from ATPG process.

Padding issues: Some ATE requests short chain padding during test program development. Traditionally, it posts no problem to Fastscan patterns. However it does not work for TestKompress patterns. The padding patterns are calculated by ATPG with relation to next pattern loading values. It is very important not to change any TestKompress pattern padding values.

WGL vs STIL patterns: For uncompressed patterns, the STIL pattern shows the values at the pins while WGL patterns the values at scan cells. In case there are inversions, the vector value in WGL pattern file will be different from what STIL has. For compressed patterns, there are no differences regarding vector value locations for either WGL or STIL.

f. Add dead cycle after last load

This approach will give scan_enable enough time to settle. If passed test, it may imply slow scan enable problem. Re-do timing simulation. At the same, it proves that it's not power related issue.

g. Shmooing

Shmoo is a good technique to help locate a stable device operation zone with minimized failures and even , identify power related issue[2] and locate a stable device operation zone in order for running Tessent Diagnosis.

Shmooing Vdd vs clock frequency is widely used and it's not a bad idea to shmoo force_pi, measure_po and clock phase as well. The goal, again, is to find out a table failure state so that you can run analysis or diagnosis.

Another benefit of shmoo is to help find out right point for chain diagnosis. It may not be a good idea to pick the shmoo junction point where the failure bits are the least. What you want to focus is the shmoo zone where it passes the junction point but towards more failures. The point to pick should see stable failure bits and have decent amount of failing bits in both chain failures and scan failures. Remember successful chain diagnosis requests not only chain failure information but also the scan failure information.

After all the checks and appropriate correction methods, some chips may pass. Even no chip passed, the failures can be minimized and be stable.

- **2) Get stable and repetitive failures**

After the efforts of minimizing massive failures, next important step is to make sure device fail in a stable and repetitive manner. Stability means, under certain test setups, re-run testing a couple of times, the failing datalogs recorded by ATE from run to run do not show fluctuations. Unstable failures usually indicate of interim failures of device and unstable test settings or environment. When it happens, it may not achieve a good diagnosis resolution.

Shmoo is good tool to get stable failure in first silicon scan test. Usually it will take several steps:

- Shmooing the device (vdd vs clk frequency) to the least failing state; or
- Masking the scan channels failed the most and shmooing the device to the least failing or even no failing state for the remaining scan channels.

You may be able to see a shmoo plot similar to **Figure 4**. It may be a case where the bad channels were masked and testing shows a clear stable fail/passing zone. Next is to set the device under conditions in the middle of grey area (passing zone) and retest the device without masking. Collect failure files and run diagnosis. If it's the result of shmooing without any masking and you want to find why device failed under certain conditions, a spot in the dark area is the best candidate to test and collect failure files.

The setting along the pass/fail border area is not good to test the chip and collect failures for running diagnosis.

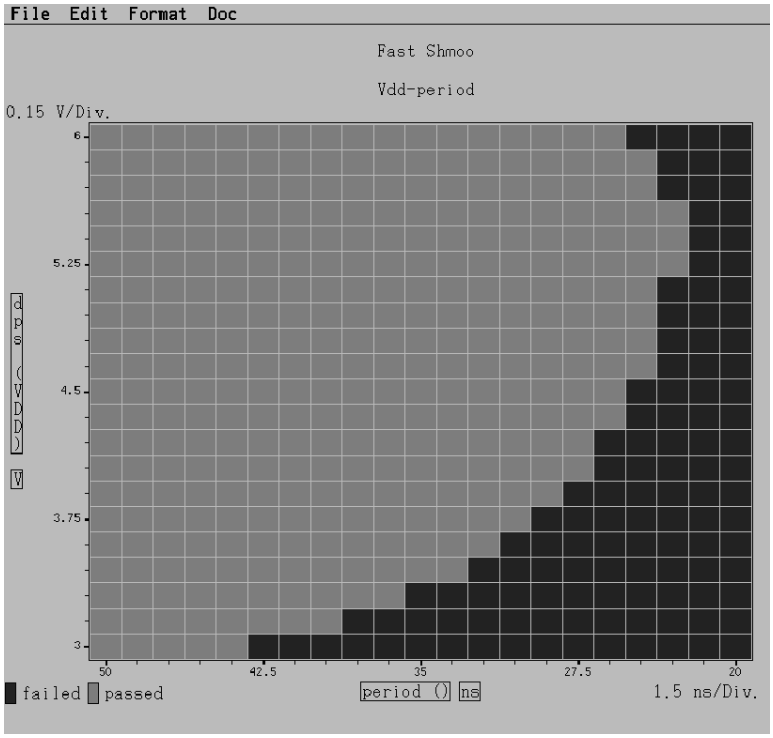

Figure 4: shmoo plot of Vdd versus period (courtesy of [3])

Other parameters such as force_pi, measure_po and clock phases can be shmooed as you need.

After locating the stable zone, rerun the device a couple of times to make sure the failure bits are stable.

- **3) Differentiate problems from EDT logic and scan logic**

If the edt chain test failed but bypass chain test passed, some people may immediately assume that edt logic has a problem. It's not necessarily true. EDT chain test patterns are created by

TestKompress and are used to test internal chains as well as edt logic. Usually its chain test background is not regular 001100110011 and appears more random.

In order to make sure the chain failure is not part of edt, a bypass chain pattern equivalent to the edt chain patterns needs to be created and tested. Or you can run bypass scan test tests, and if all pass, it might be a good indication that edt logic is wrong.

We have seen some cases where edt logic failed test due to a timing margin problem. Re-run static timing analysis helped to resolve the issue.

Appendix B: References

[1] Balachandran H; Butler, K.M; Simpson, N; “Facilitating rapid first silicon debug”; ITC 2002

[2] K. Baker, J.V. Beers, 1997 “Shmoo Plotting: The Black Art of IC Testing”, Design& Test of Computers,

[3] Y. Huang, W.T. Cheng, N. Tamarapalli, J. Rajski, R. Klingenberg, W. Hsu and Y.S. Chen, “Diagnosis with Limited Failure Information”, ITC, 2006

[4] N. Tendolkar, D. Belete, B. Schwarz, B. Podnar, A. Gupta, S. Karako, W.T. Cheng, A. Babin, K.H. Tsai, N. Tamarapalli, G. Aldrich, “Improving Transition Fault Test Pattern Quality through At-Speed Diagnosis”, ITC 2006

[5] I. Ahmed, J. Bartsch, S. Sharma, Y Huang, W.T. Cheng, W. Yang “Yield Improvement with Compressed Pattern Diagnosis”, *SDD 06*,

[6] J. Mekkoth, M. Krishna, J. Qian,W.Hsu, C.H. Chen YS Chen,N. Tamarapalli, W.T. Cheng, J. Tofte, and M. Keim “Yield Learning with Layout-aware Advanced Scan Diagnosis” , *ISTFA 2006*