Department of Electrical and Computer Engineering
Utah State University

ECE 5930/6930: VLSI Testing and Verification

Created By: Shayan (Sean) Taheri

Lab 04

Assigned Date: Month/Day/Year

Due Date: Month/Day/Year

In this lab, the goal is applying some set of test vectors and then getting the information of faults and test coverage for each of them.

Before starting the lab, you need to be familiarized with some related concepts and commands for this lab.

TetraMAX has three modes for generating patterns that are according to the following:

Basic Scan Only

1) Very fast

2) Excellent coverage for Full-Scan search

3) Combinational-Only

Important: Basic-Scan can be similar to D-Algorithm.

Command:

run_atpg basic_scan_only

Fast Sequential

1) Higher fault coverage (Specially for near-full scan designs).

2) Good for shadow logic around memories.

3) Limited non-scan (flip-flops, latches, bus keeper, RAMs)

4) Clocks controlled by PIs

5) Work on Primary Inputs (PIs)

Important: Fast Sequential can be similar to Sequential PODEM (S - PODEM) Algorithm.

Sequential PODEM Algorithm = It is the developed version of PODEM Algorithm that can be applied to both combinational and sequential circuits for detecting their faults. There are "11 Value" in this algorithm:

$$\mathrm{D}, \overline{\mathrm{D}}, 0, 1, \mathrm{X}, \mathrm{P}, \mathrm{P0}, \mathrm{P1}, 1\mathrm{P}, 0\mathrm{P}, \text{and } \mathrm{PP}$$

Note: Fast Sequential can help to detect AU (ATPG Untestable) remaining after Basic-Scan ATPG.

Command:

run_atpg fast_sequential_only

Full Sequential

1) Similar to Fast-Sequential

2) Powerful engine supporting more complex designs

Command:

run_atpg full_sequential_only

Important: You can combine these modes to get new mode (algorithm) for generating patterns.

Comparison of D-Algorithm and PODEM Algorithm

D-Algorithm

1) Emphasis on D-propagation.

2) Full-Scan search

3) Too much backtracking (in some circuits) makes it slow.

Reason: Undoing a previous decision and making a different choice. So, all possible choices may have to be tried in the worst case.

4) Some faults require multiple path sensitizations.

PODEM Algorithm

1) Work on Primary Inputs (PIs).

2) Search space is smaller than that of D-Algorithm.

3) Exponential complexity.

4) Faster than D-Algorithm.

	# of Cells	Run Time		Fault Coverage	
		PODEM	D-ALG	PODEM	D-ALG
ckt #1	828	1	34.5	100.0	99.7
ckt #2	935	1	12.8	100.0	93.1
ckt #3	951	1	2.2	99.5	99.5
ckt #4	1,566	1	3.1	97.4	97.4
ckt #5	1042	1	3.2	96.6	96.6

Comparison of Run-Time and Fault Coverage between D-Algorithm and PODEM Algorithm

Working with Fault Lists

TetraMAX maintains a list of potential faults for a design, along with the categorization of each fault. A list of faults is stored in a fault list file (An ASCII file which can be read and written using the read_faults and write_faults commands).

A fault list file contains one fault entry per line. Each entry consists of three items separated by one or more spaces. The first item indicates the stuck-at value (sa0 or sa1), the second item is the two-character fault class code, and the third item is the pin path name to the fault site. Any additional text on the line is treated as a comment.

If the fault list contains equivalent faults, then the equivalent faults must immediately follow the primary fault on subsequent lines. Instead of a class code, an equivalent fault is indicated by a fault class code of “--”.

Notice 1: TetraMAX ignores blank lines and lines that start with a double slash and a space (//).

Notice 2: You can control whether the fault list contains equivalent faults or primary faults by using the -report option of the set_faults command or the -collapsed or -uncollapsed option of the write_faults

command.

Important Commands:

In the following, some important commands have been presented that you should use some of them in this lab:

1) Writing all faults:

TEST-T> write_faults filename -all -replace

2) Using an existing fault list from another tool or from a previous run:

TEST-T> read_faults filename

3) Reporting collapsed fault list:

TEST-T> set_faults -report collapsed

TEST-T> report_faults -summary

4) Reporting uncollapsed fault list:

TEST-T> set_faults -report uncollapsed

TEST-T> report_faults -summary

5) Reporting detailed information of fault list:

TEST-T> set_faults -summary verbose

TEST-T> report_faults -summary

6) Reporting fault coverage:

TEST-T> set_faults -fault_coverage

TEST-T> report_faults -summary

7) Limit Generating Patterns:

set_atpg –patterns N // (default: 0 – to disable limit)

ATPG process stops when the N patterns are generated or fault is detected.

8) Generating random test vectors:

run_atpg -random

Important Equations:

Test coverage = Detected Faults / Detectable Faults

Fault coverage = Detected faults / All Faults

ATPG effectiveness = ATPG - Resolvable Faults / All Faults

Fault Class Hierarchy

1) DT - Detected

- DS = Detected by Simulation
- DI = Detected by Implication
- DR = Robustly Detected Delay Fault

2) PT - Possibly Detected

- AP = ATPG Untestable (Possibly Detected)
- NP = Not Analyzed (Possibly Detected)

3) UD – Undetectable

- UU = Undetectable Unused
- UT = Undetectable Tied
- UB = Undetectable Blocked
- UR = Undetectable Redundant

4) AU - ATPG Untestable

- AN = ATPG Untestable (Not Detected)

5) ND - Not Detected

- NC = Not Controlled
- NO = Not Observed

Starting the Lab

For completing the lab, you should do the following tasks:

1) Apply the given 10 test vectors to the given circuit and determine the test coverage.

2) Use Fast Sequential Only mode to generate patterns for all faults. Indicate how many test patterns

are generated, how many faults are found to be “Undetectable”, test coverage, and fault coverage.

3) Generate random 10 test vectors using Full Sequential Only mode and apply it to the circuit to get the test coverage and the fault coverage.

4) Write a small C/C++ program that generates 10 random test vectors. Then, apply the test vectors to the circuit to specify the test coverage.

Important: For task 4, you need to specify the outputs of the circuit in the STIL file based on the inputs (random test vectors) to get the test coverage; because the tool should know both the inputs and the outputs to detect the faults that can be happened in the circuit. So, please complete the following steps to find the outputs of each random test vector:

1. Consider **"L"** for each of the pins at outputs in the STIL file.

2. Enter the following commands:

- ✓ reset_state
- ✓ set_patterns -delete
- ✓ set_patterns -external file_name.stil
- ✓ run_simulation

3. Then, the tool would give error because of the incorrect values of outputs in the STIL file and guide you to correct them. Therefore, based on the guidance you can put the correct values of outputs for each test pattern in the STIL file.

Example: The format of the statements that you will get in the error is according to the following:

Pattern_Number Pin_Name (exp=0, got=1)

It means that the tool expects to see "0" at output pin “Pin_Name” for the specified pattern (Pattern_Number), because you considered **"L"** for it in the STIL file, but the correct value at the pin is "1" based on the simulation. Hence, you should change its value to **"H"** in the STIL file.

4. You can get the test coverage of the random test vectors using the following command:

- ❖ run_fault_sim -detected_pattern_storage

What to turn in for this lab:

Please submit the results of the tasks according to the following formats via Canvas:

1) A table that contains the test coverage of all tasks and the fault coverage of tasks 2 and 3.

2) Snapshots of the report for all four tasks.

3) STIL files of tasks 2, 3, and 4.

4) The C/C++ program that generates 10 random test vectors.

Good Luck