A Brief Tutorial of Test Pattern Generation Using Fastscan
v 0.2

Mahmut Yilmaz
Department of Electrical and Computer Engineering
Duke University, NC 27708

January 11, 2008
# Contents

## List of Figures

1. Introduction 4

2. Running ATPG with Mentor Graphics Fastscan 5

   2.1 Stuck-at Fault Test Generation 5
   2.1.1 Flow Steps 6

   2.2 Transition Fault Test Generation without Timing Information 7
   2.2.1 Flow Steps 7

   2.3 Transition Fault Test Generation with Timing Information 11
   2.3.1 Background 12
   2.3.2 Flow Steps 13

   2.4 Path Delay Fault Test Generation without Timing Information 16
   2.4.1 Flow Steps 17

   2.5 Bridging Fault Test Generation 21
   2.5.1 Bridging Fault Model Limitations 21
   2.5.2 Flow Steps 23

3. Examples 25

   3.1 A Simple Example without Scan Flip-flops: c3540 25
   3.1.1 Preparation 25
   3.1.2 Stuck-at Fault ATPG . 25
   3.1.3 Transition Fault ATPG . 26

   3.2 An Example with Scan Flip-flops: s38584 28
   3.2.1 Preparation 28
   3.2.2 Stuck-at Fault ATPG . 28
   3.2.3 Transition Fault ATPG: Launch on Capture (LOC) 31
   3.2.4 Transition Fault ATPG: Timing-Aware ATPG 33
   3.2.5 Fault Grading 35

A Stuck-at Fault Model 38

B Delay Fault Models 39

   B.1 Introduction 39
   B.2 Transition Fault Model 41
   B.3 Path Delay Fault Model 41

C Bridging Fault Model 43

D Transition Fault Test Generation 46
## Mentor Graphics File Formats

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.1</td>
<td>Path Description File</td>
<td>48</td>
</tr>
<tr>
<td>E.1.1</td>
<td>Syntax</td>
<td>48</td>
</tr>
<tr>
<td>E.1.2</td>
<td>Keyword Descriptions</td>
<td>48</td>
</tr>
<tr>
<td>E.1.3</td>
<td>Example</td>
<td>49</td>
</tr>
<tr>
<td>E.1.4</td>
<td>Path Definition Checking</td>
<td>50</td>
</tr>
<tr>
<td>E.2</td>
<td>Bridging Pair File</td>
<td>51</td>
</tr>
<tr>
<td>E.2.1</td>
<td>Syntax</td>
<td>51</td>
</tr>
<tr>
<td>E.2.2</td>
<td>Keyword Descriptions</td>
<td>51</td>
</tr>
<tr>
<td>E.2.3</td>
<td>Example</td>
<td>53</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Mentor Graphics stuck-at fault test flow ........................................ 5
2.2 Mentor Graphics transition fault test flow without timing information ........................................ 8
2.3 Basic Launch Off Shift Timing for Transition Test ........................................ 9
2.4 Broadside Timing Example for Transition Test ........................................ 10
2.5 Delay Defect Distribution ........................................ 12
2.6 Propagation delays and slack ........................................ 13
2.7 SDQM calculation ........................................ 13
2.8 Mentor Graphics timed transition test flow ........................................ 14
2.9 Delay defect distribution function ........................................ 17
2.10 Mentor Graphics path delay fault test flow without timing information ........................................ 18
2.11 Mentor Graphics bridging fault test flow ........................................ 22

A.1 Stuck-at fault example ........................................ 38

B.1 Resistive open defect ........................................ 39
B.2 Delay defect timing ........................................ 40
B.3 Generic hardware model and clock timing ........................................ 40

C.1 Intra-layer bridging fault ........................................ 43
C.2 Inter-layer bridging fault ........................................ 43
C.3 Bridging fault examples ........................................ 44
C.4 Capacitive coupling and bridging fault ........................................ 44

D.1 A scan flip-flop ........................................ 46
D.2 Transition test example ........................................ 46
D.3 Clock timing for transition test ........................................ 47

E.1 Example use of Transition condition statement ........................................ 49
E.2 Example of ambiguous path definition ........................................ 50
E.3 Example of ambiguous path edges ........................................ 50
Chapter 1

Introduction

The purpose of this document is to give an overview of the test pattern generation process using Mentor Graphics Fastscan ATPG tool. This tutorial uses text and examples from Mentor Graphics help documents such as Mentor Graphics DFT Guide and Fastscan Manuals. Complete help documents can be found in the installation directory of the Mentor Graphics DFT tools binaries. Currently, these files are located at the following path (for Duke University ECE users):

/home/software/mentor/2007/linux64/dft_2007_1_10/docs/pdfdocs

The following files may be useful while using FastScan:

atpg_gd.pdf, dfta_ref.pdf, dft_common.pdf, fsfttk_ref.pdf

Chapter 2 is a summary of Mentor Graphics documents and complete documents can be obtained from the path given above. Chapter 3 provides simple examples for a wide range ATPG applications. Appendices presents some of the useful background information.

This tutorial will be updated whenever it is necessary to include new information and whenever I have time. Please send your feedbacks or requests to my@ee.duke.edu.
## Chapter 2

### Running ATPG with Mentor Graphics Fastscan

#### 2.1 Stuck@ Fault Test Generation

An overview of the stuck-at fault processing methodology is shown in Figure 2.1. Each step in this flow is described below.

![Figure 2.1: Mentor Graphics stuck-at fault test flow](image)

**Required Inputs:**

2.1.1 FLOW STEPS

- Gate-Level Design Netlist
- ATPG Library: Library of all models used in the design netlist
- SDC File to define multi-cycle and false paths (Optional)

Outputs:

- Patterns in STIL or Verilog format.
- Fault sites file

2.1.1 Flow Steps

Step 1: Import Design Models Invoke FastScan (<mgcdft tree>/bin/fastscan) and load the design netlist.
The tool invokes in setup mode. Once the tool is invoked, a dialog box prompts you for the required arguments (design name, design format, and library). Browser buttons are provided for navigating to the appropriate files. Once the design and library are loaded, the tool is in Setup mode and ready for you to begin working on your design.

Using the second option requires you to enter all required arguments at the shell command line.

Listing 2.1: Fastscan Command Line Invocation

```
<mgcdft tree>/bin/fastscan
   {{{design_name [-VERILOG | -VHDL | -TDL | -GENIE | -EDIF | -FLAT]} |
    {-MODEL {cell_name | ALL} }
   }
   {-LIBRARY library_name}
   [-INCDIR include_directory...]
   [-INSENSITIVE | -SENSITIVE]
   [-LOGfile filename [-REPlace]]
   [-NOGui]
   [-TOP model_name]
   [-DOFile dofile_name [-History]]
   [-LICense retry_limit]
   [-DIAG]
   [-32 | -64]
   [-LOAD_warnings]
   }} |
   {{[-HELP | -USAGE | -MANUAL | -VERSION]}}
```

Step 2: Define clocks, scan chains, and pin constraints Define clocks, scan chains, and pin constraints using FastScan commands. It is also possible to use a Test Procedure File which includes all the set-up information. Please see the Mentor Graphics Scan and ATPG Process Guide for more information.

Basically, FastScan obtains the timing information from the test procedure file. This file describes the scan circuitry operation to the tool. You can create it manually, or let DFTAvisor create it for you after it inserts scan circuitry into the design. The test procedure file contains cycle-based procedures and timing definitions that tell the ATPG tool how to operate the scan structures within a design. For detailed information about the test procedure file, see the Design-for-Test Common Resources Manual.

For more information about this step, please check page 9.

---

2.2 Transition Fault Test Generation without Timing Information

An overview of the transition fault processing methodology without timing information is shown in Figure 2.2. Each step in this flow is described below.

Required Inputs:

- Gate-Level Design Netlist
- ATPG Library: Library of all models used in the design netlist
- SDC File to define multi-cycle and false paths

Outputs:

- Patterns in STIL or Verilog format.
- Fault sites file

2.2.1 Flow Steps

Step 1: Import Design Models Invoke FastScan (<mgcdft tree>/bin/fastscan) and load the design netlist.

The tool invokes in setup mode. Once the tool is invoked, a dialog box prompts you for the required arguments (design name, design format, and library). Browser buttons are provided for navigating to the appropriate files. Once the design and library are loaded, the tool is in Setup mode and ready for you to begin working on your design.

Using the second option requires you to enter all required arguments at the shell command line.

---

Step 3: Read in SDC File (Optional) STA tools typically provide a command to write out the false and multi-cycle path information identified during the STA process into a file or script in SDC format. If you can get the information into an SDC file, you can use the command `read sdc` command in FastScan or TestKompress to read in the false path definitions from the file. This command's usage is:

```
read sdc my_sdc_file
```

Step 4: Switch to ATPG mode Enter `set system mode atpg`. The design rule checks (DRCs) are run and the tool switches to ATPG mode.

Step 5: Enable stuck-at fault model Enter `set fault type stuck`. This sets the fault model the tool uses to develop or select ATPG patterns using the stuck-at fault model.

Step 6: Add faults to the fault list Enter `add fault -all` to add fault sites except the ones defined in SDC file.

Step 7: Create test patterns Enter `create patterns -auto`. Stuck-at test patterns are generated.

Step 8: Save the created patterns Enter `save patterns <patterns.format>`. The test patterns are saved to the specified file in the given format.

---


ATPG Using Fastsan
Figure 2.2: Mentor Graphics transition fault test flow without timing information
Step 2: Define clocks, scan chains, and pin constraints Define clocks, scan chains, and pin constraints using FastScan commands. It is also possible to use a Test Procedure File which includes all the set-up information. Please see the Mentor Graphics Scan and ATPG Process Guide\(^5\) for more information.

Basically, Fastscan obtains the timing information from the test procedure file. This file describes the scan circuitry operation to the tool. You can create it manually, or let DFTAdvisor create it for you after it inserts scan circuitry into the design. The test procedure file contains cycle-based procedures and timing definitions that tell the ATPG tool how to operate the scan structures within a design. For detailed information about the test procedure file, see the Design-for-Test Common Resources Manual\(^6\).

Figure 2.3: Basic Launch Off Shift Timing for Transition Test

Within the test procedure file, timeplates are the mechanism used to define tester cycles and specify where all event edges are placed in each cycle. As shown conceptually in Figure 2.3 for broadside testing, slow cycles are used for shifting (load and unload cycles) and fast cycles for the launch and capture. Figure 2.4 shows the same diagram with example timing added.

Figure 2.4 now shows 400 nanosecond periods for the slow shift cycles defined in a timeplate called \(tp\_slow\) and 40 nanosecond periods for the fast launch and capture cycles defined in a timeplate called \(tp\_fast\).

The following are example timeplates and procedures that would provide the timing shown in Figure 2.4. For brevity, these excerpts do not comprise a complete test procedure. Normally, there would be other procedures as well, like setup procedures.

Listing 2.3: Fastscan Command Line Invocation

```
<mgcdft tree>/bin/fastscan
   {({design_name [-VERILOG | -VHDL | -TDL | -GENIE | -EDIF | -FLAT]}) |
   {MODEL {cell_name | ALL}} }
   {LIBrary library_name}
   [-INCDIR include_directory...]
   [-SENsitive | -SEnSitive]
   [-LOGfile filename [-REPlace]]
   [-NOGui]
   [-TOP model_name]
   [-DOFile dofile_name [-History]]
   [-LICENSE retry_limit]
   [-DIAG]
   [-32 | -64]
   [-LOAD_warnings]
   {[-HELP | -USAGE | -MANUAL | -VERSION]}
```

2.2.1. FLOW STEPS

Listing 2.4: Example timeplates and procedures for transition test

```
timeplate tp_slow =
  force_pi 0;
  measure_po 100;
  pulse clk 200 100;
  period 400;
end;

timeplate tp_fast =
  force_pi 0;
  measure_po 10;
  pulse clk 20 10;
  period 40;
end;

procedure load_unload =
  scan_group grp1;
  timeplate tp_slow;
  cycle =
    force clk 0;
    force scan_en 1;
  end;
  apply shift 127;
end;

procedure capture =
  timeplate tp_fast;
  cycle =
    force_pi;
    measure_po;
    pulse_capture_clock;
end;

procedure shift =
  timeplate tp_slow;
  cycle =
    force_sc;
    measure_sc;
    pulse clk;
end;

procedure clock_sequential =
  timeplate tp_fast;
```

Figure 2.4: Broadside Timing Example for Transition Test
2.3. TRANSITION FAULT TEST GENERATION WITH TIMING INFORMATION

Step 3: Set sequential depth (Optional) It is possible to instruct FastScan to use clock sequential handling by selecting the -sequential option to the Set Pattern Type command. During test generation, FastScan generates test patterns for target faults by first attempting combinational, and then RAM sequential techniques. If unsuccessful with these techniques, FastScan performs clock sequential test generation if you specify a non-zero sequential depth.

Setting the -Sequential switch to either 0 (the default) or 1 results in patterns with a maximum sequential depth of one, but FastScan creates clock sequential patterns only if the setting is 1 or higher.

Listing 2.5: Setting sequential depth

```plaintext
set pattern type -sequential 2 //sequential depth depends on design
```

A depth of 0 indicates combinational circuitry. A depth greater than 1 indicates limited sequential circuitry. You should, however, be careful of the depth you specify. You should start off using the lowest sequential depth and analyzing the run results. You can perform several runs, if necessary, increasing the sequential depth each time. Although the maximum allowable depth limit is 255, you should typically limit the value you specify to five or less, for performance reasons.

Step 4: Read in SDC File Static Timing Analysis (STA) tools typically provide a command to write out the false and multicycle path information identified during the STA process into a file or script in SDC format. If you can get the information into an SDC file, you can use the command read sdc command in FastScan or TestKompress to read in the false path definitions from the file. For basic ATPF process, you may want to skip this step. This command’s usage is:

Listing 2.6: Reading SDC File in Fastscan

```plaintext
read sdc my_sdc_file
```

Step 5: Switch to ATPG mode Enter set system mode atpg. The design rule checks (DRCs) are run and the tool switches to ATPG mode.

Step 6: Enable transition fault model Enter set fault type transition. This sets the fault model the tool uses to develop or select ATPG patterns using the transition fault model.

Step 7: Add faults to the fault list Enter add fault -all to add fault sites except the ones defined in SDC file.

Step 8: Create test patterns Enter create patterns -auto. Transition test patterns are generated.

Step 9: Save the created patterns Enter save patterns <patterns.format>. The test patterns are saved to the specified file in the given format.

2.3 Transition Fault Test Generation with Timing Information

Mentor Graphics has recently added timing-aware ATPG capability to its tools. Timing-aware ATPG reads timing information from SDF and generates a pattern to detect a transition fault with the longest detection path. This idea

---

7 Understanding How to Run Timing-Aware ATPG, Mentor Graphics AppNotes, April 25, 2006
is based on SDQM technology (Statistical Delay Quality Model) proposed by STARC (Semiconductor Technology Academic Research Center). Please refer to the ITC paper "Invisible Delay Quality - SDQM Model Lights Up What Could Not be Seen" (ITC2005 Paper 47.1). STARC and Mentor Graphics worked together to implement SDQM into FastScan and TestKompress.

2.3.1 Background

SDQM is a metric to measure the quality of transition pattern set. Because transition fault assumes a delay more than one clock cycle on the fault site, it can be detected regardless of the detection path. On the other hand, small delay fault which is the fault model for SDQM tries to propagate down the longest path for detection. According to STARC, the probability of a smaller delay occurring is more likely than a bigger delay. "Delay Defect Distribution Chart" in Figure 2.5 shows the probability of different delay. This chart comes from either ASIC vendor or foundry who owns the process. This chart varies depending on the process.

Another factor to be used to calculate SDQM is slack information. Please see the Figure 2.6. Lets assume there are three paths that can detect a fault. Each path has 9.5ns, 7.5ns and 7.0ns path delay respectively, and the clock speed is 10ns. That means 0.5ns, 2.5ns and 3ns are their respective slacks. Slack within Small Delay Fault Model is "smallest delay defect can be caught". For example, in the path with 0.5ns slack, the smallest delay defect can be caught is 0.5ns because 0.5ns + 9.5ns = 10ns which is the same as the clock speed so more than 0.5ns delay defect can be captured. Similarly for the path with 2.5ns, the smallest detectable delay defect is 2.5ns. That means you can detect smaller delay defects by using a path with a smaller slack.

The Figure 2.7 below shows a delay defect distribution chart with the slack information in Figure 2.6. If ATPG generates a pattern with the longest path (R1), then the only uncovered (undetected) area is the pink area. The pink area is called "timing redundant" because the delay defects on this area is never detected. The slack for the longest path (R1) is referred to as Testable minimum slack (Tms) and the slack used by ATPG is the actual test slack (Ta). In this example, SDQM will be calculated as follows.

1. If the fault is detected by R1, it will be 0.
2. If the fault is detected by R2, it will be the area of red.
3. If the fault is detected by R3, it will be the area of read and blue.
2.3.2 Flow Steps

Timed transition test has basically the same steps as untimed transition test except the usage of SDF information and clock settings. Now, we will discuss these steps in more detail. An overview of the transition fault processing methodology without timing information is shown in Figure 2.8. Each step in this flow is described below.

Required Inputs:

- Gate-Level Design Netlist
- ATPG Library: Library of all models used in the design netlist
- SDF File for timing information

Figure 2.6: Propagation delays and slack

Figure 2.7: SDQM calculation

4. If the fault is not detected, it will be the area of read, blue and green. An SDQM number is calculated for each fault and summed up for the chip-level SDQM.
Outputs:

- Patterns in STIL or Verilog format.

**Step 1: Import Design Models** Invoke FastScan (`<mgcdft tree>/bin/fastscan`) and load the design netlist.

**Figure 2.8:** Mentor Graphics timed transition test flow
The tool invokes in setup mode. Once the tool is invoked, a dialog box prompts you for the required arguments (design name, design format, and library). Browser buttons are provided for navigating to the appropriate files. Once the design and library are loaded, the tool is in Setup mode and ready for you to begin working on your design.

Using the second option requires you to enter all required arguments at the shell command line.

**Listing 2.7: Fastscan Command Line Invocation**

```
<mgcdft tree>/bin/fastscan
   {[[design_name [ -VERILOG | -VHDL | -TDL | -GENIE | -EDIF | -FLAT ] ] |
   { [ -MODEL {cell_name | ALL} ] |
   [ -LIBRARY library_name ] |
   [ -INCDIR include_directory... ] |
   [ -INSENSITIVE | -SENSITIVE ] |
   [ -LOGfile filename [ -REPLACE ] ] |
   [ -NODGui ] |
   [ -TOP model_name ] |
   [ -DOFile dofile_name [ -History ] ] |
   [ -LICENSE retry_limit ] |
   [ -DIAG ] |
   [ -32 | -64 ] |
   [ -LOAD_warnings ] ] | |
   { [ -HELP | -USAGE | -MANUAL | -VERSION ] }
```

**Step 2: Define clocks, scan chains, and pin constraints** Define clocks, scan chains, and pin constraints using FastScan commands. It is also possible to use a *Test Procedure File* which includes all the set-up information. Please see the Mentor Graphics Scan and ATPG Process Guide 8 for more information.

Basically, Fastscan obtains the timing information from the *test procedure file*. This file describes the scan circuitry operation to the tool. You can create it manually, or let *DFTAdvisor* create it for you after it inserts scan circuitry into the design. The test procedure file contains cycle-based procedures and timing definitions that tell the ATPG tool how to operate the scan structures within a design. For detailed information about the *test procedure file*, see the *Design-for-Test Common Resources Manual* 9.

For more information about using *timeplates*, please check Untimed Transition Test Flow on page 10.

**Step 3: Enable transition fault model** Enter set fault type transition. This sets the fault model the tool uses to develop or select ATPG patterns using the transition fault model.

**Step 4: Switch to ATPG mode** Enter set system mode atpg. The design rule checks (DRCs) are run and the tool switches to ATPG mode.

**Step 5: Set sequential depth** It is possible to instruct FastScan to use clock sequential handling by selecting the -sequential option to the *Set Pattern Type* command. During test generation, FastScan generates test patterns for target faults by first attempting combinational, and then RAM sequential techniques. If unsuccessful with these techniques, FastScan performs clock sequential test generation if you specify a non-zero sequential depth.

Setting the -Sequential switch to either 0 (the default) or 1 results in patterns with a maximum sequential depth of one, but FastScan creates clock sequential patterns only if the setting is 1 or higher.

**Listing 2.8: Setting sequential depth**

```
set pattern type -sequential 2 //sequential depth depends on design
```

Transition test requires a sequential depth of minimum 2.

**Step 6: Read in SDF File** Timing information is stored in the SDF file.

Listing 2.9: Reading SDF File in Fastscan

```plaintext
READ SDF {sdf-filename} [-Minimum_delay| -Typical_delay| -Maximum_delay]
```

**Step 7: Provide clock speed information and enable SDQM** Please note that the timing unit is the one used in the SDF file.

Listing 2.10: Setting clock speed

```plaintext
SET ATPG Timing CLock <clock-pinname> <clock_cycle> <offset> <pulse_width>
// For instance:
// set atpg timing clock clk_in 36000 18000 18000
// NOTE: Time unit in SDF file is used

// Please use the following command to set a waveform other than ones you specified individually
SET ATPG Timing clock default <clock-pinname> <clock_cycle> <offset> <pulse_width>
// For instance:
// set atpg timing clock default 36000 18000 18000

// Enable SDQM -- required
SET ATPG Timing ON Source SDF slack_margin_for_fault_dropping [Off| slack_margin_percent%]
// For instance:
// set atpg timing on source sdf slack_margin_for_fault_dropping 50%

// slack_margin_for_fault_dropping specifies how you want to drop faults in
// Fault Simulation. In the tool default which is off, that means you want to
// drop faults regardless of its slack.
// slack_margin_percent% = (Ta-Tms)/Ta
// If you set it, Fault Simulation keeps faults for pattern generation until
// the threshold is met.
```

**Step 8: Define delay defect distribution function (Optional)** In this optional step, you can define Delay Defect Distribution Chart.

Listing 2.11: Defining delay defect distribution function

```plaintext
SET Delay_fault Distribution [Histogram <file_name>| Exponential <num1> <num2> <num3>] [-End <max>]
// For instance:
// set delay distribution Histogram hist1 end 11
```

The delay defect distribution for the above example given in Figure 2.9 below.

**Step 9: Add faults to the fault list** Enter `add fault -all` to add fault sites except the ones defined in SDC file.

**Step 10: Create test patterns** Enter `create patterns -auto`. Transition test patterns are generated.

**Step 11: Save the created patterns** Enter `save patterns <patterns.format>`. The test patterns are saved to the specified file in the given format.

### 2.4 Path Delay Fault Test Generation without Timing Information

An overview of the transition fault processing methodology without timing information is shown in Figure 2.10. Each step in this flow is described below.

---

2.4.1 FLOW STEPS

Figure 2.9: Delay defect distribution function

Required Inputs:

- Gate-Level Design Netlist
- ATPG Library: Library of all models used in the design netlist
- Test Procedure File (optional)
- Path description file

Outputs:

- Patterns in STIL or Verilog format.

2.4.1 Flow Steps

Step 0: Path description file creation This step is skipped in the ATPG flow because it is assumed that path description file is provided. The path description file can be used to provide the long path list to Fastscan in case you have such a list. An example path description file:

Listing 2.12: Mentor Graphics Path Description File

```
1  PATH "path0" =
2  PIN /I$6/Q + ;
3  PIN /I$35/B0 + ;
4  PIN /I$35/C0 + ;
5  PIN /I$1/I$650/IN + ;
6  PIN /I$1/I$650/OUT - ;
7  PIN /I$1/I$951/I$1/IN - ;
8  PIN /I$1/I$951/I$1/OUT + ;
9  PIN /A_EQ_B + ;
10  END ;
```
2.4.1. FLOW STEPS

START

Step 1
Import Design Models
fastscan [options]

Step 2
Define clocks, scan chains, and pin constraints
add pin constraint scan_en c0

Step 3
Set sequential depth
set pattern type s-sequential 2

Step 4
Switch to ATPG mode
set system mode atpg

Step 5
Enable path delay fault model
set fault type path_delay

Step 6
Load the path definition file
load path path_file_name

Step 7
Specify any ambiguous paths
add ambiguous paths -all -max_paths 4

Step 8
Add faults to the fault list
add faults -all

Step 9
Delete unsensitizable paths
delete paths -unsensitizable_paths

Step 10
Create path delay test patterns
create patterns

Step 11
Save patterns
save patterns <patterns.format>

Step 12
Write fault sites
write fault sites <file>

END

Figure 2.10: Mentor Graphics path delay fault test flow without timing information

ATPG Using Fastscan
2.4.1. FLOW STEPS

Step 1: Import Design Models
Invoke FastScan (<mgcdft tree>/bin/fastscan) and load the design netlist.

The tool invokes in setup mode. Once the tool is invoked, a dialog box prompts you for the required arguments (design name, design format, and library). Browser buttons are provided for navigating to the appropriate files. Once the design and library are loaded, the tool is in Setup mode and ready for you to begin working on your design.

Using the second option requires you to enter all required arguments at the shell command line.

Listing 2.13: Fastscan Command Line Invocation

```
<mgcdft tree>/bin/fastscan
   {{(design_name [-VERILOG | -VHDL | -TDL | -GENIE | -EDIF | -FLAT]) | 
      { -MODEL {cell_name | ALL } 
      
      [-LIBRARY library_name] 
      [-INCDIR include_directory...] 
      [-ENSENSITIVE | -SENSITIVE] 
      [-LOGfile filename [-REPLACE]] 
      [-NOGui] 
      [-TOP model_name] 
      [-DOFile dofile_name [-REPLACE]] 
      [-LCense retry_limit] 
      [-DIAG] 
      [-32 | -64] 
      [-LOAD_warnings] 
      } | 
   } [-HELP | -USAGE | -MANUAL | -VERSION]
```

Step 2: Define clocks, scan chains, and pin constraints
Define clocks, scan chains, and pin constraints using FastScan commands. It is also possible to use a Test Procedure File which includes all the set-up information. Please see the Mentor Graphics Scan and ATPG Process Guide for more information.

Basically, Fastscan obtains the timing information from the test procedure file. This file describes the scan circuitry operation to the tool. You can create it manually, or let DFTAdvisor create it for you after it inserts

---


---

ATPG Using Fastscan
scan circuitry into the design. The test procedure file contains cycle-based procedures and timing definitions that tell the ATPG tool how to operate the scan structures within a design. For detailed information about the test procedure file, check page[9] or see the Design-for-Test Common Resources Manual[12].

In this step,

- constrain the scan enable pin to its inactive state.
- turn on output masking (optional).
- add nofaults <x, y, z>.

Within your design, you may have instances that should not have internal faults included in the fault list. You can label these parts with a nofault setting. To add a nofault setting, you use the Add Nofaults command. This command's usage is as follows:

ADD NOfaults pathname... [-Instance] [-Stuck_at {01|0|1}]

Listing 2.14: Setup

```
1  add pin constraint scan_en c0
2  set output masks on
3  add nofaults <x, y, z>
```

Step 3: Set sequential depth It is possible to instruct FastScan to use clock sequential handling by selecting the -sequential option to the Set Pattern Type command. During test generation, FastScan generates test patterns for target faults by first attempting combinational, and then RAM sequential techniques. If unsuccessful with these techniques, FastScan performs clock sequential test generation if you specify a non-zero sequential depth.

Setting the -Sequential switch to either 0 (the default) or 1 results in patterns with a maximum sequential depth of one, but FastScan creates clock sequential patterns only if the setting is 1 or higher.

Listing 2.15: Setting sequential depth

```
1  set pattern type -sequential 2 //sequential depth depends on design
```

A depth of 0 indicates combinational circuitry. A depth greater than 1 indicates limited sequential circuitry. You should, however, be careful of the depth you specify. You should start off using the lowest sequential depth and analyzing the run results. You can perform several runs, if necessary, increasing the sequential depth each time. Although the maximum allowable depth limit is 255, you should typically limit the value you specify to five or less, for performance reasons.

Step 4: Switch to ATPG mode Enter set system mode atpg. This triggers the tool's automatic design flattening and rules checking processes. Then, the tool switches to ATPG mode.

Step 5: Enable path delay fault model Enter set fault type path_delay. This sets the fault model the tool uses to develop or select ATPG patterns using the path delay fault model.

Step 6: Load the Path Description File Enter load path path_file_name. Check Appendix E on page 48 for more information about Path Description File.

Step 7: Specify any ambiguous paths Specify any ambiguous paths you want the tool to add to its internal path list. The following example specifies to add all ambiguous paths up to a maximum of 4.

Listing 2.16: Specifying ambiguous paths

```
1  add ambiguous paths -all -max_paths 4
```

---


ATPG Using Fastscan

---

Step 8: Add faults to the fault list Enter `add fault -all` to add fault sites.

Step 9: Delete unsensitizable paths Enter `delete paths -unsensitizable_paths`. This will perform an analysis on the specified paths and delete those the analysis proves are unsensitizable.

Step 10: Create test patterns Enter `create patterns`. Path delay test patterns are generated.

Step 11: Save the created patterns Enter `save patterns <patterns.format>`. The test patterns are saved to the specified file in the given format.

Step 12: Write fault sites Enter `write fault sites <fault_site.file>`. The fault sites are written to the specified file.

### 2.5 Bridging Fault Test Generation

An overview of the bridging defect pattern fault processing methodology is shown in Figure 2.11. Each step in this flow is described below.

**Required Inputs:**

- Gate-Level Design Netlist
- ATPG Library: Library of all models used in the design netlist
- Bridging pair list file in Mentor Graphics format

**Outputs:**

- Patterns in STIL or Verilog format.
- Fault sites file

#### 2.5.1 Bridging Fault Model Limitations

Currently, bridge fault model testing has the following limitations:

- All signal driving strengths are ignored.
- All properties (like resistance) between bridged net pairs are ignored.
- Net names are supported in the fault definition file. However, bridge faults may not map exactly to the bridge entries in the fault definition file for any of the following reasons:
  - Net names are mapped to their driving pin names when they are loaded into the tool. If more that one driving pin exists for a net, one is arbitrarily selected and the resulting pin names are used to report bridge faults.

---

14 Please refer to Appendix E for details
Figure 2.11: Mentor Graphics bridging fault test flow
2.5.2 FLOW STEPS

By default, the Write Faults and Write Fault Sites commands try to map the pin names back to the net names. However, the pins are mapped to the first nets they connect to, which may have names different than the net names specified in the fault definition file. Even though the names may differ, the mapped nets are still the same nets specified in the fault definition file.

- Due to the limitations of the net-to-pin mapping, different net name pairs may map to the same bridge fault site.

For reference purposes, you can assign names to bridge entries in the definition file, and use them for a cross-reference point.

2.5.2 Flow Steps

Step 0: Bridging pair file creation This step is skipped in the ATPG flow because it is assumed that Bridging pair list file is provided. An example bridging pair list file:

```
Listing 2.17: Mentor Bridging Pair File

```

Step 1: Import Design Models Invoke FastScan (<mgcdft tree>/bin/fastscan) and load the design netlist. The tool invokes in setup mode. Once the tool is invoked, a dialog box prompts you for the required arguments (design name, design format, and library). Browser buttons are provided for navigating to the appropriate files. Once the design and library are loaded, the tool is in Setup mode and ready for you to begin working on your design.

Using the second option requires you to enter all required arguments at the shell command line.

```
Listing 2.18: Fastscan Command Line Invocation

```

ATPG Using Fastscan
2.5.2. FLOW STEPS

7 [-INsensitive | -SENsitive]
8 [-LOGfile filename [-REPlace]]
9 [-NOGui]
10 [-TOP model_name]
11 [-DOFile dofile_name [-History]]
12 [-LICense retry_limit]
13 [-DIAG]
14 [-32 | -64]
15 [-LOAD_warnings]
16 }
17 {{[-HELP | -USAGE | -MANUAL | -VERSION]}

Step 2: Define clocks, scan chains, and pin constraints Define clocks, scan chains, and pin constraints using FastScan commands. It is also possible to use a Test Procedure File which includes all the set-up information. Please see the Mentor Graphics Scan and ATPG Process Guide for more information.

Step 3: Switch to ATPG mode Enter set system mode atpg. The design rule checks (DRCs) are run and the tool switches to ATPG mode.

Step 4: Enable bridge fault model Enter set fault type bridge. Sets the fault model the tool uses to develop or select ATPG patterns using the static bridge model.

Step 5: Load the bridging pair list file Enter load fault sites <fault_definition_file_or_calibre.sites_file>. Loads all net pairs for which to create test patterns as defined in the specified file.

If it is necessary, the user can use the following commands to display or delete information associated with bridge entries in the bridging pair list file:
- Report Fault Sites
- Delete Fault Sites

Step 6: Create test patterns Enter create patterns -auto. Static bridge test patterns that test each of the loaded net pairs are generated.

Step 7: Save the created patterns Enter save patterns <bridge_patterns.format>. The test patterns are saved to the specified file.

Step 8: Write fault sites Enter write fault sites <fault_site.file>. The fault sites are written to the specified file.

---


---

ATPG Using Fastscan
Chapter 3

Examples

3.1 A Simple Example without Scan Flip-flops: c3540

c3540 is one of the ISCAS85 benchmarks. ISCAS85 benchmarks are combinational circuits without any state element. Thus, c3540 is a benchmark without scan flip-flops. We will run stuck-at and transition-test ATPG on c3540.

3.1.1 Preparation

- Download and copy the example directory to somewhere in your Unix home directory:
  
  http://www.duke.edu/~my6/fastscan_tutorial/examples/c3540.zip

- Unzip the directory:
  
  unzip c3540.zip

- Make sure that FastScan binary directory is in your $PATH. You may add the following line in your .cshrc file:
  
  setenv PATH /home/software/mentor/2007/linux64/dft_2007_1_10/bin:$PATH

For a simple ATPG run, the user needs the netlist (Verilog, VHDL or any supported format) of the benchmark and a DOFILE which includes all the FastScan commands to be run. These commands can also be executed one by one in an interactive session. In the example directory, you will also find *.run files. These files can be used to start FastScan faster.

3.1.2 Stuck® Fault ATPG

- Create the DOFILE:


```
1 // All the command should be inserted in this file
2 // Check fastscan help documents to get more information
3 // about the commands
4
5 // Add date to log files
6 system date
7
8 // set floating nets to X value
9 set z hand ext x
10
11 // Put FastScan into ATPG mode which will
12 // execute the DRC checks
13 // and will allow test generation commands:
14 set system mode atpg
15
16 // General ATPG - mode settings:
17 set split capture_cycle on
18
19 // Generate patterns for stuck-at faults :
20 // Set fault type to stuck-at
21 // Use n-detect patterns where n=15.
```

25
3.1.3. TRANSITION FAULT ATPG

```
// For an ordinary stuck-at test, n=1.
set fault type stuck -DEtections 15

// Add all the faults
add fault -all

// Create patterns
create patterns -auto

// Compress patterns -- OPTIONAL
// Warning: This command cannot be run when multiple-detection is enabled (N>1).
// compress patterns -multiple_detection -reset_au

// order the patterns to get the most effective ones on top
// OPTIONAL
// Warning: This command cannot be run when multiple-detection is enabled (N>1).
// order patterns 3

// Report statistics
report statistics > c3540_stuckat_patterns_N15.stats

// Report the faults -- OPTIONAL
// This command may create large files for large benchmarks
// Do not use if you have disk space problems
report faults > c3540_stuckat_faults_N15.fault_list

// Save your patterns to a file
save patterns c3540_stuckat_patterns_N15.ap -Ascii -replace

// Exit Fastscan
exit -d
```

- Run FastScan using the command line or use the provided run script:

```
#!/usr/bin/perl -w
system "fastscan -nogui -64 -verilog -replace c3540.v -top c3540 -dof c3540_stuckat_N15.dof -log c3540_stuckat_N15.log ";
```

- Check the log file, statistics file, and the pattern file.

- Change the dofile to disable multiple detection. When you disable multiple detection, you can enable pattern compression and pattern ordering. Uncomment corresponding lines in the dofile. Then, rerun FastScan and compare the results with the previous run.

  Note: Before you rerun FastScan, you may want to rename the output files of the previous run. Otherwise FastScan will overwrite them.

### 3.1.3 Transition Fault ATPG

- Create the DOFILE:

```
Listing 3.3: DOFILE for running 15-detect transition test ATPG on c3540
```
3.1.3. TRANSITION FAULT ATPG

```plaintext
// All the command should be inserted in this file
// Check fastscan help documents to get more information
// about the commands
// Add date to log files
system date
// set floating nets to X value
set z hand ext x
// Put FastScan into ATPG mode which will
// execute the DRC checks
// and will allow test generation commands:
set system mode atpg
// General ATPG - mode settings:
set split capture_cycle on
// Generate patterns for transition faults :
// Set fault type to transition
// Use n-detect patterns where n=15.
// For an ordinary transition test, n=1.
set fault type transition -DEtections 15
// Add all the faults
add fault -all
// Create patterns
create patterns -auto
// Compress patterns -- OPTIONAL
// Warning: This command cannot be run when multiple-detection
// is enabled (N>1).
// compress patterns -multiple_detection -reset_au
// order the patterns to get the most effective ones on top
// OPTIONAL
// Warning: This command cannot be run when multiple-detection
// is enabled (N>1).
// order patterns 3
// Report statistics -- OPTIONAL
report statistics > c3540_transition_patterns_N15.stats
report statistics
// Report the faults -- OPTIONAL
// This command may create large files for large benchmarks
// Do not use if you have disk space problems
report faults > c3540_transition_faults_N15.fault_list
// Save your patterns to a file
save patterns c3540_transition_patterns_N15.ap -Ascii -replace
// Exit Fastscan
exit -d
```

- Run FastScan using the command line or use the provided run script:

```
#!/usr/bin/perl -w
system "fastscan -nogui -64 -verilog -replace c3540.v -top c3540 -dof c3540_stuckat_N15.dofile -log c3540_stuckat_N15.log ";
```

ATPG Using Fastscan
3.2 An Example with Scan Flip-flops: s38584

s38584 is the largest ISCAS89 benchmark. ISCAS89 benchmarks are sequential circuits. One can replace the flip-flops of s38584 with scan flip-flops to make it testable. We will run stuck-at and transition-test ATPG on s38584. You may realize that s38584 benchmark in the example directory also has wrapper scan flip-flops (flips-flops on the primary inputs and primary outputs of the benchmark).

3.2.1 Preparation

- Download and copy the example directory to somewhere in your Unix home directory:
  http://www.duke.edu/~my6/fastscan_tutorial/examples/s38584.zip
- Unzip the directory:
  unzip s38584.zip
- Make sure that FastScan binary directory is in your $PATH. You may add the following line in your .cshrc file:
  setenv PATH /home/software/mentor/2007/linux64/dft_2007_1_10/bin:$PATH

For the following ATPG run, the user needs the netlist (Verilog, VHDL or any supported format) of the benchmark, the ATPG library file, a DOFILE that includes all the FastScan commands to be run, and a PROC file that includes the clock timings to control scan operation. In the example directory, you will also find *.run files. These files can be used to start FastScan faster.

3.2.2 Stuck-at Fault ATPG

- Create the PROC file. This file tells FastScan how the patterns are scanned in and out. You may use this basic template for other benchmarks after updating the scan chain length related lines:

```plaintext
Listing 3.5: s38584_tran_loc.proc1

1 timeplate tp_slov =
2     force_pi 0;
3     measure_po 100;
4     pulse clk 200 100;
5     period 400;
6     end ;
7
8 timeplate tp_fast =
9     force_pi 0;
10    measure_po 10;
11    pulse clk 20 10;
```

ATPG Using Fastscan
3.2.2. STUCK@ FAULT ATPG

```plaintext
period 40;

procedure load_unload =
  scan_group grp1;
  timeplate tp_slow;
  cycle =
    force clk 0;
    force scan_enable 1;
  end;
  // Length of the scan chain
  apply shift 1370;
end;

procedure shift =
  timeplate tp_slow;
  cycle =
    force_sci;
    measure_sco;
    pulse clk;
end;

procedure capture =
  timeplate tp_fast;
  cycle =
    force_pi;
    measure_po;
    pulse_capture_clock;
end;

procedure clock_sequential =
  timeplate tp_fast;
  cycle =
    force_pi;
    pulse_capture_clock;
    pulse_read_clock;
    pulse_write_clock;
end;
```

- Create the DOFILE.

Listing 3.6: DOFILE for running 1-detect stuck-at test ATPG on s38584

```plaintext
// All the commands should be inserted in this file
// Check fastscan help documents to get more information
// about the commands
// Add date to log files
system date

// clock and scan information
add clocks 0 clk
add scan group grp1 s38584_tran_loc.procl
add scan chains chain10 grp1 scan_data_in scan_data_out

// Treat floating nets as X
set z hand ext x

// Do not consider faults for the following module (wrapper)
addnofaults SDFFNSR -module
```
// Add pin constraints to disable testing of scan structures
add pin constraint scan_data_in C0
add pin constraint scan_enable C0

// Prevent transitions from Primary inputs during capture cycles.
// Create realistic test environment
set transition holdpi on

// Prevent measures on Primary outputs.
// Create realistic test environment
// If you want to see the PO output values
// in the pattern files, uncomment this line
set output masks on

// To prevent transition paths from passing though a bidir pin.
flattened model
add slow pad -all

// If you need multiple detections, set it here
set fault type stuck -DEtection 1
set pattern type -seq 1 -clock_po off

// If you want to add more processors
// This is for parallel runs, should not be
// necessary for small projects
// Add Processors dsl2:1
// Add Processors dsl3:1

// Put FastScan into ATPG mode which will execute the DRC checks
// and will allow test generation commands:
set system mode atpg

// These command are to create log files
// Optional ...
report scan groups > s38584_scaninfo
report scan cells -all >> s38584_scaninfo
report drc rules -verbose -summary > s38584_drcinfo
// Add report commands if you want

// General ATPG - mode settings:
set split capture_cycle on

// Add all the faults
add fault -all

// Create patterns
create patterns -auto
system date

// Report ATPG untestable faults
//Report Faults -class AU
// Report Unstable faults (UU+TI+BL+RE)
// UU UNUSED
// TI TIed
// BL Blocked
// RE Redundant
//Report Faults -class UT

// Report statistics
report statistics > stat/s38584_stuckat_n1.stat
report statistics
// Save your patterns to a file
save patterns pattern/s38584_stuckat_n1.pattern -Ascii -replace

// Exit Fastscan
3.2.3 TRANSITION FAULT ATPG: LAUNCH ON CAPTURE (LOC)

- Run FastScan using the command line or use the provided run script:

```
#!/usr/bin/perl -w
system("fastscan -nogui -verilog -64 -replace s38584.v -top s38584 -lib s38584_atpg.lib \ 
 -dof s38584_stuckat_n1.dofile -log log/s38584_stuckat_n1.log ");
```

```
./s38584_stuckat_n1.run
```

- Check the log file, statistics file, and the pattern file.
- Change the dofile to enable pattern compression and pattern ordering. Check the previous example to find the corresponding commands. Then, rerun FastScan and compare the results with the previous run.
- Change the dofile to enable multiple detection. Then, rerun FastScan and compare the results with the previous runs.

*Note: Before you rerun FastScan, you may want to rename the output files of the previous run. Otherwise FastScan will overwrite them.*

### 3.2.3 Transition Fault ATPG: Launch on Capture (LOC)

- Create the PROC file. Use the same PROC file given for stuck@ ATPG.
- Create the DOFILE.

```
// All the commands should be inserted in this file
// Check fastscan help documents to get more information
// about the commands
// Add date to log files
system date

// clock and scan information
add clocks 0 clk
add scan group grp1 s38584_tran_loc.proc1
add scan chains chain10 grp1 scan_data_in scan_data_out

// Treat floating nets as X
set z hand ext x

// Do not consider faults for the following module (wrapper)
add nofaults SDFPNSR -module

// Add pin constraints to disable testing of scan structures
add pin constraint scan_data_in C0
add pin constraint scan_enable C0

// Prevent transitions from Primary inputs during capture cycles.
// Create realistic test environment
set transition holdpi on

// Prevent measures on Primary outputs.
```
3.2.3. TRANSITION FAULT ATPG: LAUNCH ON CAPTURE (LOC)

// Create realistic test environment
// If you want to see the PO output values
// in the pattern files, uncomment this line
set output masks on
// To prevent transition paths from passing though a bidir pin.
flatten model
add slow pad -all

// If you need multiple detections, set it here
set fault type transition -no_shift_launch -DEtections 1
set pattern type -seq 2 -clock_po off

// *** IMPORTANT ***
// The commands for Launch on Shift (LOS) is exactly
// the same as given in this file, except the above 2 lines:
// If you want do run LOS, use the following commands instead:
//set fault type transition -DEtections 1
//set pattern type -seq 1 -clock_po off

// If you want to add more processors
// This is for parallel runs, should not be
// necessary for small projects
//Add Processors dsil2:1
//Add Processors dsil3:1

// Put FastScan into ATPG mode which will execute the DRC checks
// and will allow test generation commands:
set system mode atpg

// These command are to create log files
// Optional ...
report scan groups > s38584_scaninfo
report scan cells -all >> s38584_scaninfo
report drc rules -verbose -summary > s38584_drcinfo
// Add report commands if you want

// General ATPG - mode settings:
set split capture_cycle on

// Add all the faults
add fault -all

// Create patterns
create patterns -auto
system date

// Report ATPG untestable faults
//Report Faults -class AU
// Report Untestable faults (UU+TI+BL+RE)
// UU UNUsed
// TI Tied
// BL Blocked
// RE Redundant
//Report Faults -class UT

// Report statistics
report statistics > stat/s38584_tran_loc_n1.stat
report statistics

// Save your patterns to a file
save patterns pattern/s38584_tran_loc_n1.pattern -Ascii -replace

// Exit Fastscan
3.2.4 Transition Fault ATPG: Timing-Aware ATPG

Timing-aware ATPG is a special type of transition-test ATPG. Ordinary transition-test ATPG does not target long activation and propagation paths for testing faults. Most of the time, the faults are tested through short paths.

Very deep sub-micron (VDSM) process technologies are leading to increasing densities and higher clock frequencies for integrated circuits (ICs). However, VDSM technologies are especially susceptible to process variations, crosstalk noise, power-supply noise, and defects such as resistive shorts and opens, which induce small delay variations in the circuit components. Such delay variations are referred to as small-delay defects (SDDs) in the literature.

Although the delay introduced by each SDD is small, the overall impact can be significant if that path is critical, has low slack, or includes many SDDs. The overall delay of the path may become larger than the clock period, causing circuit failure or temporarily incorrect results. As a result, the detection of SDDs typically requires fault excitation through shortest-slack paths. The longest paths in the circuit, except false paths and multi-cycle paths, are referred to as the least-slack paths.

As a result of growing industry concerns regarding SDDs, commercial timing-aware ATPG tools have become available. Timing-aware Mentor Graphics FastScan is one of them. The following example shows how to run a simple timing-aware ATPG using FastScan. FastScan gets the timing information from a Standard Delay Format (SDF) file and uses this information to excite longest paths while testing for faults. Currently, FastScan runs timing-aware ATPG only in LOC setting.

Note: Before you rerun FastScan, you may want to rename the output files of the previous run. Otherwise FastScan will overwrite them.

Listing 3.9: s38584_tran_loc_n1.run

```perl
#!/usr/bin/perl -w
system ("fastscan -nogui -verilog -64 -replace s38584.v -top s38584 -lib s38584_atpg.lib \ 
-dof s38584_tran_loc_n1.dofile -log log/s38584_tran_loc_n1.log");
```

- Create the PROC file. Use the same PROC file given for stuck@ ATPG.
- Create the DOFILE.

Listing 3.10: DOFILE for running timing-aware transition-test ATPG on s38584

```plaintext
// timing-aware transition test atpg
system date

// clock and scan information
add clocks 0 clk
add scan group grp1 s38584_tran_loc.proc1
add scan chains chain10 grp1 scan_data_in scan_data_out

set z hand ext x
add nofaults SDFFNSR -module

// Add pin constraints to disable testing of scan structures
add pin constraint scan_data_in C0
add pin constraint scan_enable C0

// Prevent transitions from Primary inputs during capture cycles.
set transition holdpi on

// Prevent measures on Primary outputs.
set output masks on

// To prevent transition paths from passing through a bidir pin.
flattened model
add slow pad -all

set fault type transition -no_shift_launch
set pattern type -seq 2 -clock_po off

// If you want to add more processors
// This is for parallel runs, should not be
// necessary for small projects
Add Processors dsil2:1
Add Processors dsil3:1

set system mode atpg
set split capture_cycle on

// Read the timing information from the SDF file
read sdf s38584.sdf -typical_delay

// Set clock waveforms
set atpg timing ON -source SDF -clock_waveform clk 20 10 10
set atpg timing ON -source SDF -clock_waveform default 20 10 10

// Required for SDQM statistics report
set delay distribution exponential 0.00158 0.0021 0.00000494 -end 50000

add fault -all
create patterns -auto
system date

// Report ATPG untestable faults
//Report Faults -class AU
// Report Untestable faults (UU+TI+BL+RE)
// UU UNUsed
// TI TIed
// BL Blocked
// RE Redundant
```

ATPG Using Fastscan
3.2.5 Fault Grading

It is a common practice to generate transition-test patterns, fault grade these patterns against stuck-at ATPG, and then run top-off stuck-at ATPG. The idea is that once we apply transition-test patterns to the IC, these patterns will detect most of the stuck-at patterns anyway. As a result, we don’t need to apply all the stuck-at ATPG patterns afterwards. We only need additional stuck-at ATPG patterns for undetected faults.

- Create the PROC file. Use the same PROC file given for stuck-at ATPG.
- Create the DOFILE.

Listing 3.12: DOFILE for running fault grading ATPG on s38584

```plaintext
// All the commands should be inserted in this file
// Check fastscan help documents to get more information
// about the commands
// Add date to log files
system date

// clock and scan information
add clocks 0 clk
add scan group grp1 s38584_tran_loc.proc1
add scan chains chain10 grp1 scan_data_in scan_data_out

// Treat floating nets as X
set z hand ext x

// Do not consider faults for the following module (wrapper)
add nofaults SDFFNSR -module

// Add pin constraints to disable testing of scan structures
add pin constraint scan_data_in C0
add pin constraint scan_enable C0
```
24 // Prevent transitions from Primary inputs during capture cycles.
25 // Create realistic test environment
26 set transition holdpi on
27 // Prevent measures on Primary outputs.
28 // Create realistic test environment
29 // If you want to see the PO output values
30 // in the pattern files, uncomment this line
31 set output masks on
32 // To prevent transition paths from passing though a bidir pin.
33 flatten model
34 add slow pad -all
35 // If you need multiple detections, set it here
36 set fault type transition -no_shift_launch -DEtections 1
37 set pattern type -seq 2 -clock_po off
38 // **** IMPORTANT ****
39 // The commands for Launch on Shift (LOS) is exactly
40 // the same as given in this file, except the above 2 lines:
41 // If you want to run LOS, use the following commands instead:
42 // set fault type transition -DEtections 1
43 // set pattern type -seq 1 -clock_po off
44 // If you want to add more processors
45 // This is for parallel runs, should not be
46 // necessary for small projects
47 //Add Processors dsl2:1
48 //Add Processors dsl3:1
49 // Put FastScan into ATPG mode which will execute the DRC checks
50 // and will allow test generation commands:
51 set system mode atpg
52 // General ATPG - mode settings:
53 set split capture_cycle on
54 // Add all the faults
55 add fault -all
56 // Create patterns
57 create patterns -auto
58 system date
59 // Report statistics
60 report statistics > stat/s38584_faultgrade_tr.stat
61 report statistics
62 // Save your patterns to a file
63 save patterns pattern/s38584_faultgrade_tr.pattern -Ascii -replace
64 //--------- Grade for stuck-at fault coverage -------------------
65 set fault type stuck
66 add faults -all
67 // Read in previously saved path delay and transition patterns and
68 // add them to the internal pattern set when they are simulated.
69 set pattern source external pattern/s38584_faultgrade_tr.pattern -all_patterns
70 run
71 report statistics > stat/s38584_faultgrade_tr_fg_stuck.stat

ATPG Using Fastscan
3.2.5. FAULT GRADING

- Run FastScan using the command line or use the provided run script:

```bash
#!/usr/bin/perl -w
# Auto generated
system ('"fastscan -nogui -verilog -64 -replace s38584.v -top s38584 -lib s38584_tpg.lib \\
        -dof s38584_faultgrade.dofile -log log/s38584_faultgrade.log "');
./s38584_faultgrade.run
```

- Check the log file, statistics file, and the pattern file. Compare the results to the previous ordinary transition-test ATPG and stuck-at ATPG. Compare the total pattern counts and CPU time.
Appendix A

Stuck-at Fault Model

The single stuck-at fault model is the most commonly used fault model due to its simplicity. The assumptions of the single stuck-at fault model are listed below.

- Only one line in the circuit is faulty at a time.
- The fault is permanent.
- The effect of the fault is as if the faulty node is tied to either $V_{DD}$ (stuck-at-1) or $GND$ (stuck-at-0).
- The function of the gate in the circuit is unaffected by the fault.

![Figure A.1: Stuck-at fault example: A stuck-at-1 on net x](image)

The numerous advantages of the single stuck-at model are the reason it is most often used. It is simple to use during test generation and fault simulation; research has shown that it covers a large percentage of physical defects; and other fault models, that can increase defect coverage, can be mapped into sequences of single stuck-at faults. There are however, some defects that cannot be covered by the single stuck-at model. The advantages and disadvantages of this fault model is given below.

- **Advantages**
  - Can be applied at the logic level or the module level.
  - The number of faults is reasonable ($2^n$ for a circuit with $n$ nets).
  - Algorithms for ATPG are well-developed and efficient.
  - Research indicates that single-stuck at fault model covers about 90% of the all possible manufacturing defects in CMOS circuits.
  - Other fault models can be mapped into a series of stuck-at faults.

- **Disadvantages**
  - It does not cover all defects of CMOS circuits.
Appendix B

Delay Fault Models

B.1 Introduction

With the advent of nanometer technologies (130 nm and below), new process elements such as copper wiring and low-K dielectric changed the frequency of different defect mechanisms. More defects now tend to alter the timing of the circuit instead of changing the function, and are located in the interconnection wiring. Resistive opens and, to a lesser degree, resistive shorts alter the time constants inherent in CMOS logic and dominate the defect distribution.

In addition, more and more of the chip manufacturing process steps and the resulting circuit defects are due to the wiring. Each generation adds wiring levels and complexity. (It is estimated that there are 7 kilometers of wire in each square centimeter of a typical 130 nm chip.) As feature sizes are scaled down, the metal pitch and height are reduced to increase density. Stacked vias have become very tall and narrow compared to older structures. This increased density creates many opportunities for resistive shorts and opens, resulting in more timing failures that are not detected with traditional stuck-at fault testing. Wiring, rather than the actual transistors, has become the focus of manufacturing tests.

For a typical example of a resistive open defect, see Figure B.1. On the left, a photo of the chip surface shows a break in a net on the third metal level, an apparent open. On the right, when we slice through this defect and look at the cross section, we can see a fine layer of copper bridging the gap. This layer acts as a series resistor, changing the time constant of the net. In this case, it slowed the net transitions by 200 ps.

Failures due to the above mentioned facts may cause logic circuits to malfunction at the desired clock rate and thus violate timing specifications. Such failures are modeled as delay faults, and they facilitate delay testing. These faults may not show themselves at the speeds lower than the desired clock speed.

In Figure B.2, the upper curve shows the defect free output response, and the bottom curve shows the response typical of a traditional stuck-at fault (stuck-at-0 in this case). Threshold voltages are marked, and the vertical dashed line shows the required arrival time at the next logic stage for the transition to be captured in a downstream flop, per static timing analysis. For a delay defect, the middle curve shows that the output goes to logic 1 too late for proper operation, resulting in a timing failure.

Since delay faults do not alter the logic function realized by a circuit and since the tests for stuck-at faults are normally applied at a slow clock rate (due to the ATE limitation or other reasons), they are inadequate for detecting delay faults. Special two-pattern test vectors are required for detecting delay faults.

1Delay Test Methods in Encounter Test Application Note

Figure B.1: Resistive open defect
Figure B.2: Delay defect timing

Figure B.3: Generic hardware model and clock timing
A generic hardware model used in delay fault testing is shown in Figure B.3. Here the vector pair \( <V_1, V_2> \) constitutes a delay test and signals \( \text{Clock}_1 \) and \( \text{Clock}_2 \) are used to clock the input and output latches, respectively. At time \( t_0 \), an initializing input vector \( V_1 \) is applied, and the circuit is allowed to stabilize under input \( V_1 \). At time \( t_1 \), the propagation vector \( V_2 \) is applied, and the outputs are sampled at time \( t_2 \), where \( (t_2 - t_1) \) is the intended time interval between the input and output clocks, called the rated clock interval \( T_C \).

Exhaustive testing is quite impractical for delay faults since the total number of pattern-pairs required will be \( (2^n)(2^n - 1) \), which is of the order \( 2^{2n} \), for a circuit having \( n \) inputs. One must derive suitable and reasonable delay fault models and devise algorithms that can generate tests for the modeled faults. Various fault models used in delay fault testing. The most common ones, transition and path delay fault models, are discussed in the following sections.

### B.2 Transition Fault Model

The transition fault model is considered as a logical model for a defect that delays a rising or falling transition at inputs and outputs of logic gates. There are two kinds of transition faults, i.e., slow-to-rise and slow-to-fall. The slow-to-rise (fall) transition fault temporarily behaves like a DC stuck-at-0 (1) fault.

A test for a transition fault is a pair of input patterns, one (initialization pattern) to set up the initial state for the transition and another (propagation pattern) to cause the appropriate transition and observe its effect at a primary output. The propagation pattern is identical to a pattern that detects the corresponding DC stuck-at fault.

The transition fault coverage is a measure of the effectiveness of the delay test in detecting large delay variations. Transition fault model defects for which the delay is large enough to cause a logical failure when the signal propagates along any path through the site of the fault. The main drawback of this model is the assumption of a large gate delay defect. Also, it is difficult to tell how small a delay fault can be, before it is not detectable. In practice, delay variations tend to be distributed over many circuit elements. Thus, many small gate delay faults, each undetectable as a transition fault, can give rise to a large path delay fault.

### B.3 Path Delay Fault Model

The path delay fault model was first proposed by Smith. This model has received greater attention than the transition fault model, and has been quite extensively studied. A considerable amount of research has already been reported on various aspects of test generation and fault simulation of path delay faults.

In path delay fault model, any path with a total delay exceeding the system clock interval is said to have a path delay fault. This models distributed defects that affect an entire path. For each physical path \( P \), connecting a primary input to a primary output of the circuit, there are two corresponding delay paths. The rising path (falling path) is the path traversed by a transition that is initiated as a rising (falling) transition at the input of path \( P \) and changes the direction of transition whenever it passes through an inverting gate.

The following definitions that are frequently used in path delay fault testing:

**Definition 1:** Let \( G \) be a gate on path \( P \) in a logic circuit, and let \( r \) be an input to gate \( G \); \( r \) is called an off-path sensitizing input if \( r \) is not on path \( P \).

**Definition 2:** A two-pattern test \( <V_1, V_2> \) is called a robust test for a delay fault on path \( P \), if the test detects that fault independently of all other delays in the circuit.

---

Definition 3: A two-pattern test \(< V_1, V_2 >\) is called a *non-robust test* for a delay fault on path \( P \) if it detects the fault under the assumption that no other path in the circuit involving the off-path inputs of gates on \( P \) has a delay fault.
Appendix C

Bridging Fault Model

The bridging fault model tests for resistive shorts between two normally unconnected instance pins or net names, which could be caused by a piece of metal from the sputtering process. Studies showed that bridging faults are the closest fault models to real defects\(^1\)\(^2\). Bridging defects cannot be caught by capacitive coupling and noise analysis simulations, because they are caused by fabrication defects.

![Intra-layer bridging fault](image1)

**Figure C.1: Intra-layer bridging fault**

Bridging faults can be inter-layer and intra-layer. Figure C.1 shows an intra-layer bridging fault, and Figure C.2 shows an inter-layer bridging fault.

![Inter-layer bridging fault](image2)

**Figure C.2: Inter-layer bridging fault**

By examining a physical layout of a design and by observing the patterns of bridging defects, one can arrive at the following truisms relating to bridging defects. The probability of bridged net pairs\(^3\):

- Increases as the distance the wires run in parallel on the same layer increases
- Increases as the distance between parallel wires decreases
- Increases as the size of the contaminant producing bridging increases
- Decreases for wires that do not run in parallel

\(^2\)http://www.reed-electronics.com/tmworld/article/CA489463.html
\(^3\)Source: Cadence Encounter Test - Identifying, Modeling, and Testing Bridging Defects in Encounter Test
• Decreases for wires that are on different layers

These truisms form the criteria for identifying bridging net pair candidates. Note that bridging defects can also occur on wires running on two different layers due to oxide layer breakdown, but the probability for this happening is very low as the area of overlap is extremely small.

The bridge between nets can affect each other in different ways. For instance, one of the nets may force the other one to get the opposite value, or vice versa. Bridging faults are generally modeled as wired-or or wired-and. However, the test tool may allow more complicated modeling of bridging fault. Most of the time, it is required to tell the ATPG tool how the interaction between the bridging net pairs occur.

![Bridging fault examples](image)

Figure C.3: Bridging fault examples

![Capacitive coupling and bridging fault](image)

Figure C.4: Capacitive coupling and bridging fault

Directly examining the physical database for parallel or close proximity wires is the best way to determine bridging pair nets. As a not-so-good alternative, measurement of mutual capacitances and crosstalk noise can be a method of obtaining bridging net pairs. Since bridging net pairs tend to run in parallel, RC extraction of mutual capacitances would indicate the physical proximity of nets. By performing RC extraction and examining the magnitude of mutual capacitances one could obtain bridging fault candidates. However, as noted by Mattiuzzo et al., this method is not as accurate as directly examining the physical database for parallel wires. This is because a high coupling capacitance

---

does not necessarily mean a high bridging probability. Two wires facing each other for a long path and at a long
distance are less likely to bridge than two nets facing each other for a short path and at a short distance as shown in
Figure [C.4] Similarly, for the case of Figure [C.3]b, potential corner bridges has low coupling capacitance.
Appendix D

Transition Fault Test Generation

In this section, we will explain how the LOC transition fault test works in double clock scan flop designs (widely used in industrial circuits). Figure D.1 shows how a scan flop is constructed and Figure D.2 shows a circuit using these flip-flops. Each flop accepts 3 clock signals:

- **CLK** System clock
- **SC1** Scan clock 1
- **SC2** Scan clock 2

The second latch accepts two clocks: **CLK** and **SC1**. This latch either passes D value to **Q** when **CLK** is ON, or **SDI** (Scan Data In) to **Q** when **SC1** is ON.

![Figure D.1: A scan flip-flop](image)

During scan-in and scan-out, the system clock is kept at **OFF** state. Using the **SDI** port, the scan data is scanned-in by first switching **SC1**, thus pushing the data to the input of third latch, and then switching **SC2**, thus pushing the data to the **SDO** (Scan Data Out) port.

When **SC1** and **SC2** are kept at the **OFF** state, the flop behaves like a regular D flip-flop.

![Figure D.2: Transition test example](image)

Now, consider the example given in Figure D.2. Assume that we want to have a transition from 1 to 0 on the net connected to the **Q** port of the second flip-flop (FF2). For this to happen, we should first set the value of this net to 1. We can do it by pushing a 1 to FF2. Then, we need to provide a transition to 0 on the same net. For that, we need to use the combinational logic connected to the **D** port of FF2. As it can be seen in Figure D.2, we need to use FF1 to force one of the inputs of the AND gate to 1. So, we need to push 1 to the first scan flop.

As a result, the bit values we need to scan in is 1-1-X for this case (*X: don’t care*). After scan-in, the **Q** port of FF1 and the **Q** port of FF2 will have the value 1.
The clock timing of this test is shown in Figure D.3. As it can be seen in this figure, after the scan-in, the system clock is toggled, first to launch and then to capture.

![Figure D.3: Clock timing for transition test](image)

When CLK is toggled for the first time (launch), the values on the D ports of all flip-flops are sent to the Q ports. The 0 value on the D port of FF2 to sent to the Q port at this first clock. Doing this, we obtain the required transition. After this, we need to capture the result of this transition. This is done when the system clock is toggled for the second time (capture). The result of the transition is seen at the D port of FF3 and this result is sent to the Q port of FF3 by the capture clock.

The captured value is then sent to SDO port of FF3 by switching SC2 (master-observe). After this, a scan-out is done to scan the captured values out.
APPENDIX E
MENTOR GRAPHICS FILE FORMATS

E.1 Path Description File

Path description file is an ASCII file in which all paths to be tested in the test set are defined.

E.1.1 Syntax

```
1 PATH <pathname> =
2 CONDition <pin_pathname> <0|1|Z>;
3 TRANSition_condition <pin_pathname> <Rising|Falling|Same|Opposite>;
4 PIN <pin_pathname> [+|-];
5 PIN <pin_pathname> [+|-];
6 ...
7 PIN <pin_pathname> [+|-];
8 END [pathname];
```

E.1.2 Keyword Descriptions

The ASCII path definition file has several syntax requirements. The tools ignore as a comment any line that begins
with a double slash (//) or pound sign (#). Each statement must be on its own line. The four types of statements
include:

**Path** A required statement that specifies the unique pathname of a path.

**Condition** An optional statement that specifies any conditions necessary for the launch and capture events. Each
condition statement contains two arguments:

- A full pin pathname for either an internal or external pin
- A value for that pin

Valid pin values for condition statements are 0, 1, or Z. Condition statements must occur between the path
statement and the first pin statement for the path.

**Transition_condition** An optional statement that specifies additional transitions required in the test pattern. Each
transition_condition statement contains two arguments:

- A full pin pathname for either an internal or external pin
- A direction

Transition_condition statements must occur between the path statement and the first pin statement for the path. The direction can be one of the following: rising, falling, same, or opposite. Rising and falling specify that
a rising edge and falling edge, respectively, are required on the specified pin at the same time as launching a
transition into the first pin of the path. Same specifies for the tool to create a transition in the same direction
as the one on the first pin in the path definition. Opposite creates a transition in the opposite direction.

Figure E.1 shows an example where a transition_condition statement could be advantageous.

A defined path includes a 2-input AND gate with one input on the path, the other connected to the output of a scan cell. For a robust test, the AND gates off-path or gating input needs a constant 1. The tool, in exercising its preference for a robust test, would try to create a pattern that achieved this. Suppose however that you wanted the circuit elements fed by the scan cell to receive a 0-1 transition. You could add a transition condition statement to the path definition, specifying a rising transition for the scan cell. The path capture point maintains a 0-1 transition, so remains testable with a non-robust test, and you also get the desired transition for the other circuit elements.

Pin A required statement that identifies a pin in the path by its full pin pathname. Pin statements in a path must be ordered from launch point to capture point. A "+" or "-" after the pin pathname indicates the inversion of the pin with respect to the launch point. A "+" indicates no inversion (you want a transition identical to the launch transition on that pin), while a "-" indicates inversion (you want a transition opposite the launch transition).

If you use "+" or "-" in any pin statement, you must include a "+" for the launch point. The polarity of the launch transition must always be "+".

You must specify a minimum of two pin statements, the first being a valid launch point (primary input or data output of a state element or RAM) and the last being a valid capture point (primary output, data or clk input of a state element, or data input of a RAM). The current pin must have a combinational connectivity path to the previous pin and the edge parity must be consistent with the path circuitry. If a statement violates either of these conditions, the tool issues an error. If the path has edge or path ambiguity, it issues a warning.

Paths can include state elements (through data or clock inputs), but you must explicitly name the data or clock pins in the path. If you do not, FastScan does not recognize the path and issues a corresponding message.

End A required statement that signals the completion of data for the current path. Optionally, following the end statement, you can specify the name of the path. However, if the name does not match the pathname specified with the path statement, the tool issues an error.

E.1.3 Example

Use the Load Paths command to read in the path definition file. The tool loads the paths from this file into an internal path list. You can add to this list by adding paths to a new file and re-issuing the Load Paths command with the new filename.

```
1 PATH "path0" =
2 PIN /I$6/Q + ;
3 PIN /I$35/B0 + ;
4 PIN /I$35/CO + ;
5 PIN /I$1/I$650/IN + ;
6 PIN /I$1/I$650/DUT - ;
7 PIN /I$1/I$951/I$1/IN - ;
```
E.1.4  Path Definition Checking

FastScan checks the points along the defined path for proper connectivity and to determine if the path is ambiguous.
Path ambiguity indicates there are several different paths from one defined point to the next. Figure E.2 indicates a path definition that creates ambiguity.

In this example, the defined points are an input of Gate2 and an input of Gate7. Two paths exist between these points, thus creating path ambiguity. When FastScan encounters this situation, it issues a warning message and selects a path, typically the first fanout of the ambiguity. If you want FastScan to select more than one path, you can specify this with the Add Ambiguous Paths command.

During path checking, FastScan can also encounter edge ambiguity. Edge ambiguity occurs when a gate along the path has the ability to either keep or invert the path edge, depending on the value of another input of the gate. Figure E.3 shows a path with edge ambiguity due to the XOR gate in the path.

E.2 Bridging Pair File

E.2.1 Syntax

- Precede each line of comment text with a pair of slashes (/). 
- Do not modify keywords. They can be in upper-or lowercase.
- Use an equal sign to define a value for a keyword.
- Enclose all string values in double quotation marks (" ").
- Bridge declarations must be enclosed in braces ({}).
- A semicolon (;) must separate each entry within the bridge declaration.

```
// Comment
VERSION <VERSION_NUMBER>
FAULT_TYPE = "<FAULT_TYPE>"

BRIDGE {
  NET1 = "<NET1_DESC>";
  NET2 = "<NET2_DESC>";
  <FAULTS = {<FAULT_LIST>;>
  <NAME = "<FAULT_NAME>";
  <WEIGHT = <NUMBER>;>
  <LENGTH = <NUMBER>;>
  <WIDTH = <NUMBER>;>
  <LAYER = "<LAYER_NAME>";>
  <TYPE = "<TYPE>";>
  ...}

BRIDGE {
  ...
}
```

E.2.2 Keyword Descriptions

VERSION  Required. Used to specify the version of the bridge definition file and must be declared before any bridge entries. Must be a real number or integer written in non-scientific format starting with 1.1.

E.2.2. KEYWORD DESCRIPTIONS

FAULT_TYPE Optional. Used to indicate what type of fault is declared by the FAULTS keyword. Must be a string value enclosed in quotation marks (" ").

BRIDGE Required. Used to start a bridge entry.

NET1 / NET2 Required. Identifies the net/pin pair for the bridge entry. Specifies the regular or hierarchical net/pin pathname to gate(s). Use the following guidelines when using these keywords:

- Declare either net or pin value pairs.
- Net/Pin pairs are the first two items declared in the bridge entry.
- Net/Pin pathnames are string values and should be enclosed in quotation marks (" ").
- A net can be used in multiple bridge entries.
- Nets can be defined in any order.

FAULTS Optional. Provides a fault classification for each of the four components of the 4-way bridge fault model. Four classifications must be specified in a comma-separated list enclosed in braces (). You can use any of the following 2-digit codes for each of the four components:

- UC (uncontrolled) Undetected faults, which during pattern simulation, never achieve the value at the point of the fault required for fault detection—that is, they are uncontrollable.
- UO (unobserved) Faults whose effects do not propagate to an observable point.
- DS (det_simulation) Faults detected when the tool performs fault simulation.
- PU (posdet_untestable) The posdet, or possible-detected, fault class includes all faults that fault simulation identifies as possible-detected but not hard detected. A possible-detected fault results from a 0-X or 1-X difference at an observation point. PUs are proven ATPG untestable and hard undetectable posdet faults.
- PT (posdet_testable) The posdet, or possible-detected, fault class includes all faults that fault simulation identifies as possible-detected but not hard detected. A possible-detected fault results from a 0-X or 1-X difference at an observation point. PT means potentially detectable posdet faults. PT faults result when the tool cannot prove the 0-X or 1-X difference is the only possible outcome. A higher abort limit may reduce the number of these faults.
- TI (tied) The tied fault class includes faults on gates where the point of the fault is tied to a value identical to the fault stuck value. The tied circuitry could be due to:
  - Tied signals
  - AND and OR gates with complementary inputs
  - Exclusive-OR gates with common inputs
  - Line holds due to primary input pins held at a constant logic value during test by CT0 or CT1 pin constraints you applied with the the FastScan or FlexTest Add Pin Constraint command
- BL (blocked) The blocked fault class includes faults on circuitry for which tied logic blocks all paths to an observable point. The tied circuitry could be due to:
  - Tied signals
  - AND and OR gates with complementary inputs
  - Exclusive-OR gates with common inputs
  - Line holds due to primary input pins held at a constant logic value during test by CT0 or CT1 pin constraints
- AU (ATPG_untestable) The ATPG untestable fault class includes all faults for which the test generator is unable to find a pattern to create a test, and yet cannot prove the fault redundant. Testable faults become ATPG untestable faults because of constraints, or limitations, placed on the ATPG tool (such as a...
pin constraint or an insufficient sequential depth). These faults may be possible-detectable, or detectable, if you remove some constraint, or change some limitation, on the test generator (such as removing a pin constraint or changing the sequential depth). You cannot detect them by increasing the test generator abort limit.

- **NF** (directs the tool to no fault this component of the bridge)

Using fault classifications, you can filter and display the desired fault types. Please refer to Mentor Graphics Scan and ATPG Process Guide for more information on fault classes.

**NAME** Optional. A unique string, enclosed in quotation marks (" "), that specifies a name for the bridge.

**LENGTH** Optional. Real number that specifies the length attribute for the bridge entry.

**WIDTH** Optional. Real number that specifies the width attribute for the bridge entry.

**LAYER** Optional. String that specifies the layer attribute for the bridge entry. Must be enclosed in quotation marks (" ").

**WEIGHT** Optional. Real number that specifies the weight attribute for the bridge entry.

**TYPE** Optional. 3-, 4-, or 5-character code that specifies a type identification for the bridge entry. Must be enclosed in quotation marks (" "). Use one of the following codes:

- **S2S** → Side-to-side
- **SW2S** → Side-Wide-to-Side: same as S2S but at least one of the two signal lines is a wide metal line
- **S2SOW** → Side-to-Side-Over-Wide: same as S2S, but the bridge is located over a wide piece of metal in a layer below
- **C2C** → Corner-to-Corner
- **V2V** → Via-to-Via: an S2S for vias
- **VC2VC** → Via-Corner-to-Via-Corner
- **EOL** → End-of-Line: the end head of a line faces another metal line

### E.2.3 Example

```plaintext
//fault definition file begins
VERSION 1.1
FAULT_TYPE = "BRIDGE_STATIC_4WAY_DOM"
BRIDGE {
  NET1 = "/top/u1/a";
  NET2 = "/top/u2/b";
}
BRIDGE {
  NET1 = "/top/u3/a";
  NET2 = "/top/u4/b";
  FAULTS = {UC, UD, DS, NF};
}
BRIDGE {
  NET1 = "/top/u5/a";
  NET2 = "/top/u6/b";
  FAULTS = {AU, UD, DS, PT};
  NAME = "bridge_flt1";
  WEIGHT = 0.8;
  LENGTH = 3.56e-6;
  WIDTH = 4.32e-7;

```
E.2.3. EXAMPLE

21 LAYER = "metal4";
22 TYPE = "S2S";
23 }
24 //Fault definition file ends

ATPG Using Fastscan