TetraMAX II ADV
ATPG Fault Coverage Report and Test Vectors Safety Manual

N-2017.09

Synopsys, Inc.

March 20167

Copyright Notice and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at

http://www.synopsys.com/Company/Pages/Trademarks.aspx.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Customer support is available through SolvNet online customer support and by contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services, which include downloading software, viewing Documentation on the Web, and entering a call to the Support Center.

To access SolvNet:

1. Go to the SolvNet Web page at http://solvnet.synopsys.com/.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click SolvNet Help in the Support Resources section.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com/ (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters.

1
Scope of this Document

This document describes the methodology for using the fault coverage report and test vector data in TetraMAX II ADV ATPG so that it does not negatively impact the safety of developed products.

The *TetraMAX II ADV ATPG Fault Coverage Report Safety Manual* describes the mandatory requirements for the use of the fault coverage report in safety-critical applications according to ISO 26262.

This document is intended to assure confidence in the use of the TetraMAX II ADV ATPG fault coverage report *and test vector data*. TetraMAX II ADV ATPG generates test vectors that are instrumental in the identification of properly functioning (safety-critical) components. The fault coverage report for these test vectors establishes a metric for the thoroughness of these tests and requires qualification according to the ISO 26262 standard.

Section 3 describes requirements and definitions related to the environment and scope of the use of the TetraMAX II ADV fault coverage report in safety-critical applications. Section 4 describes the ISO 26262 qualified use cases for the TetraMAX II ADV fault coverage report *and test vector data*.

2 Glossary

This section defines technical terms used within this document.

Terms and Definitions

Term	Definition
Component	A part of an electronic system that implements a function in the vehicle. See also Part 1 of the ISO 26262 standard for the definition. The Standard also refers to elements and items, but for the purpose of this *Safety Manual* this makes no difference.
Defect	Error
Error	In this document, refers to a *potential error*
Fault coverage	Refers to the percentage of a specific type of fault that can be detected during the test of any engineered system. High fault coverage is particularly valuable during manufacturing test, and techniques such as Design For Test (DFT) and automatic test pattern generation are used to increase fault coverage.
Fault summary report	The result of the compilation process. This report is a summary of results of the faults and fault coverage and/or ATPG effectiveness of the component based on the fault universe and patterns generated for that given component.
TetraMAX II ADV fault coverage report	The fault coverage report that is part of the TetraMAX II ADV ATPG tool, and which is qualified for use in safety-critical applications.
Test vector data	A set of inputs or test sequences automatically generated by TetraMAX II ADV ATPG that are applied to a digital circuit, The vectors target specific areas of a software design so that automated test equipment (ATE) can accurately distinguish between the correct circuit behavior and the faulty circuit behavior.
Tool	The TetraMAX II ADV ATPG tool – a development tool according to ISO 26262. The TetraMAX II ADV ATPG tool is comprised of TetraMAX II

	ADV ATPG and TetraMAX II DSMTest. DSMTest is a license that allows the running of fault simulation on advanced fault models and the output fault coverage report which is also certified as safe with or without DSMTest.
Software	The TetraMAX II ADV ATPG program that runs on a soft component in a safety-critical process.
STAR	Synopsys Technical Action Request. A STAR reports, documents and tracks a product Bug or Enhancement request (called a “B” or “E”). It is stored in a CRM database – well maintained, documented, tracked, and monitored comprehensive graphical interface. Complex reporting is possible using Web-based access and searches. The CRM stores all relevant information and the testcase data is stored in UNIX. The CRM is accessible by Synopsys employees only, however limited STAR informaiton is displayed on SolvNet for customers who are associated with a STAR's User Site. Customer contacts are automatically notified when STARs are filed or when a status changes.
The standard	In this document, refers to the *ISO 26262 Standard for Road Vehicles – Functional Safety* – (ISO26262)
Tool classification	Determination of the required tool confidence level (TCL)
TCL	Tool confidence level as defined by ISO 26262

3 TetraMAX II ADV Fault Coverage Report Product Description

This section provides a general description of safety-related requirements and definitions concerning the scope and environment of the TetraMAX II ADV fault coverage report.

Version

The *TetraMAX II ADV ATPG Fault Coverage Report Safety Manual* is intended to be used with the fault coverage report produced by TetraMAX II ADV ATPG version K-2014.06-SP2 (see section 4 for a more detailed description of the individual tools and qualification status).

Tool	Version	Safety Qualified
TetraMAX II ADV ATPG	N-2017.09	

ISO 26262, ASIL, TCL

According to ISO 26262, the TetraMAX II ADV ATPG test vector data and fault coverage report can be used for the development of safety critical components if it is used in compliance with this document, the *TetraMAX II ADV ATPG Fault Coverage Report Safety Manual*. The TetraMAX II ADV fault coverage report can be used for components with a maximum Automotive Safety Integrity Level ASIL D.

Independent qualification of the fault coverage report by SGS-TÜV has classified a Tool Confidence Level TCL1 (see the “SGS-TÜV certificate of ISO 26262 Compliance of the TetraMAX II ADV ATPG Tool”).

Product Documentation and Support

The documentation of the TetraMAX II ADV test vector data and fault coverage report is included as part of TetraMAX II ADV ATPG.

If TetraMAX II ADV ATPG is used for developing safety-critical components for road vehicles (as defined by ISO 26262), the development process must comply with the guidelines and constraints defined in the *TetraMAX II ADV Fault Coverage Report and Test Vector Data Safety Manual* (TMAX-SM).

Guidelines to assist with achieving the goals of the ISO 26262 standard for the safe use of TetraMAX II ADV can be found in the *TetraMAX II ADV ATPG Fault Coverage Safety Guide* (TMAX-TSG). This document includes a template of the use case-specific *Software Tool Criteria Evaluation Report*, which is intended specifically for the TetraMAX II ADV fault coverage report and test vector data.

The Synopsys Technical Support Center has on-line support and information for TetraMAX II ADV ATPG, including the TetraMAX II ADV fault coverage report. Among other items, it has the latest information on known defects and workarounds related to the TetraMAX II ADV fault coverage report. The Synopsys Technical Support Center is the channel by which TetraMAX users can get support and report defects. See the section on “Customer Support” at the start of this document for details of the Synopsys Technical Support Center.

Using the Reference Methodology Flow

The Synopsys reference methodologies provide a set of product- and release-specific reference scripts that are intended to show you the latest recommended methodology for a specific product and release. These scripts are used as a starting point that you can adapt and build upon for your design requirements. You should use these scripts as a template for developing your flows. The scripts are documented with embedded comments that are designed to be clear and self-explanatory.

Standard versions of the reference methodology scripts are available for the Design Compiler Tool. The scripts are available for download from SolvNet:

https://solvnet.synopsys.com/rmgen

RMgen and Product Reference Methodology Training Link via SolvNet

https://solvnet.synopsys.com/retrieve/025090.html

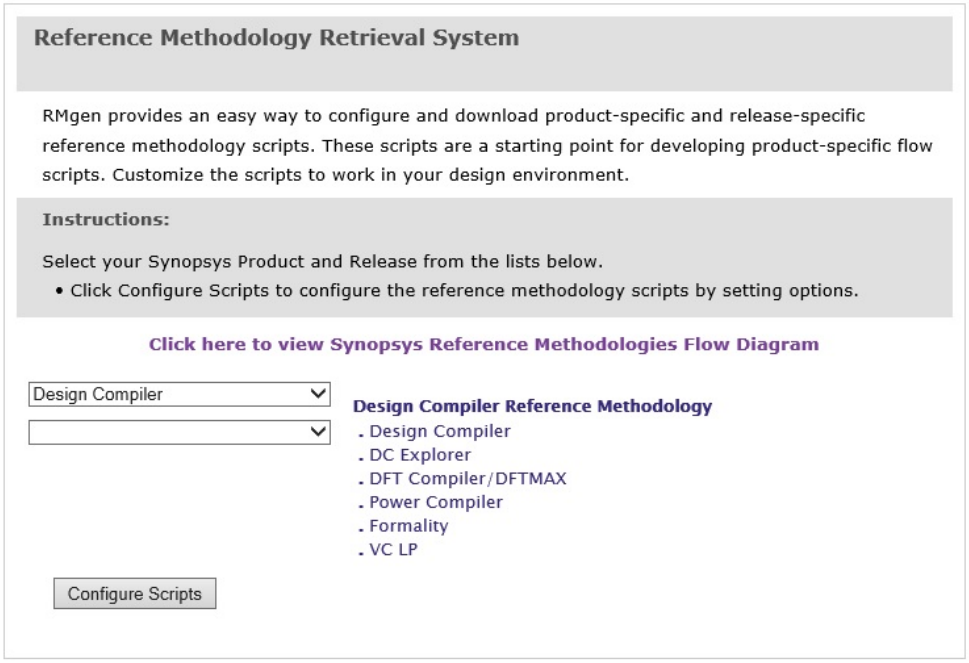
RMgen provides an easy-to-use Web-based environment for you to obtain default or customized reference methodology scripts for your design environment. The user interface has Web pages for selecting, configuring, and downloading the reference methodology scripts.

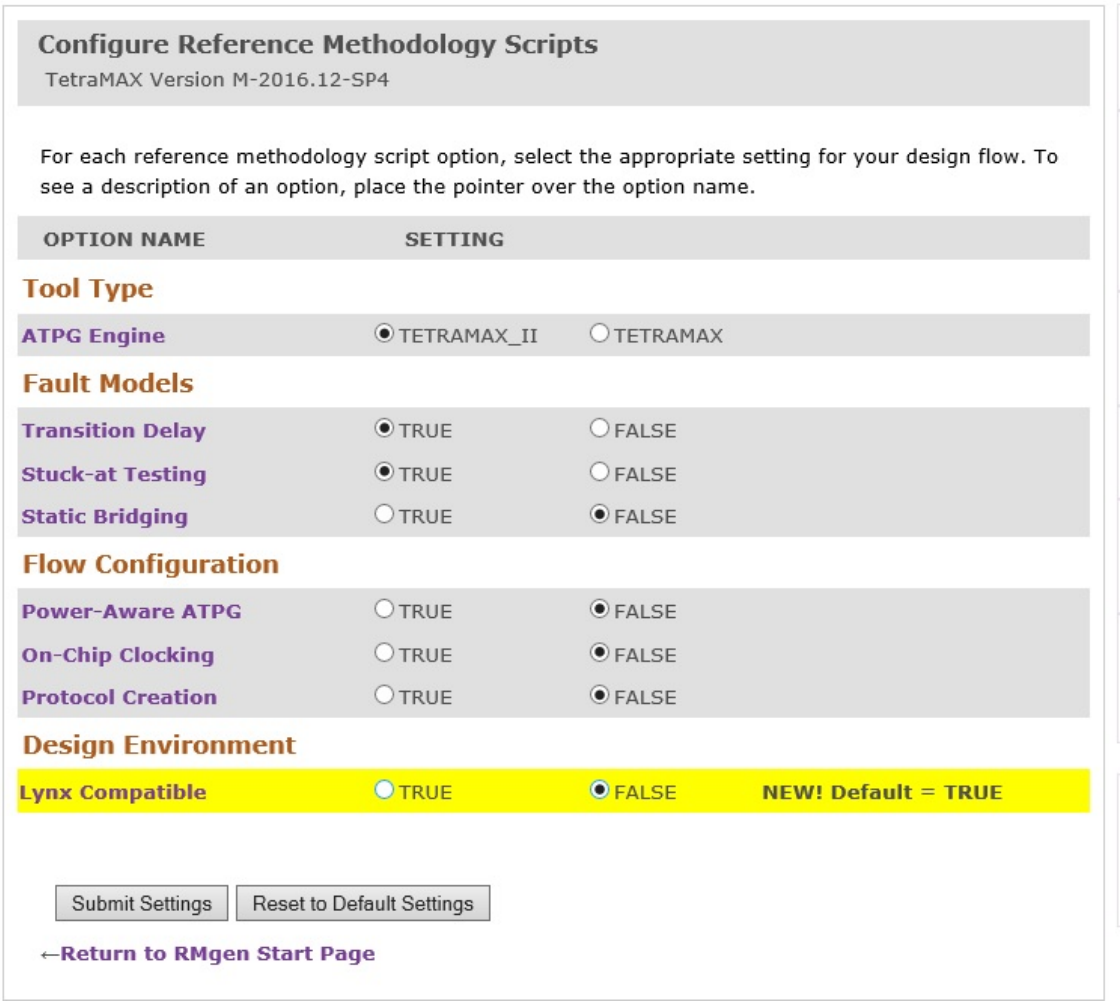
Select Product and Release

The Reference Methodology Retrieval System page shown in **Figure** 1 on the following page allows you to select the product and release for the intended script.

Select Product and Release -> Configure Scripts -> Download Scripts

Figure 1: RMgen Reference Methodology Retrieval System Page: Select Product and Release


Reference Methodology Retrieval System
RMgen provides an easy way to configure and download product-specific and release-specific reference methodology scripts. These scripts are a starting point for developing product-specific flow scripts. Customize the scripts to work in your design environment.
Instructions:
Select your Synopsys Product and Release from the lists below.
• Click Configure Scripts to configure the reference methodology scripts by setting options.
Click here to view Synopsys Reference Methodologies Flow Diagram
Design Compiler
Design Compiler Reference Methodology
. Design Compiler
. DC Explorer
. DFT Compiler/DFTMAX
. Power Compiler
. Formality
. VC LP
Configure Scripts


Configure Reference Methodology Scripts
TetraMAX Version M-2016.12-SP4
For each reference methodology script option, select the appropriate setting for your design flow. To see a description of an option, place the pointer over the option name.
OPTION NAME
SETTING
Tool Type
ATPG Engine
TETRAMAX_II
TETRAMAX
Fault Models
Transition Delay
TRUE
FALSE
Stuck-at Testing
TRUE
FALSE
Static Bridging
TRUE
FALSE
Flow Configuration
Power-Aware ATPG
TRUE
FALSE
On-Chip Clocking
TRUE
FALSE
Protocol Creation
TRUE
FALSE
Design Environment
Lynx Compatible
TRUE
FALSE
NEW! Default = TRUE
Submit Settings
Reset to Default Settings
←Return to RMgen Start Page

Select the product and release using the two selection boxes. Click the “Configure Scripts” button to select the product’s option settings in the Configure Scripts page.

The TetraMAX Reference Methodology includes options to generate tests for various types of delay faults and bridging faults (in addition to stuck-at and iddq faults), for power-aware ATPG, and for ATPG with scan compression. Note that using these options requires additional licenses. The scripts clearly identify the sections related to these features, so you can easily include or exclude them when you prepare your test scripts.

Using RMgen and Reference Methodology Scripts Application Note, Version 4.2

https://solvnet.synopsys.com/retrieve/025091.html

https://solvnet.synopsys.com/retrieve/customer/application_notes/attached_files/025091/RMgen_and_Reference_Methodology_Application_Note_4.2.pdf

User Competence

To properly use with the TetraMAX II ADV fault coverage report and test vector data, a user must have a good understanding and working knowledge of the following:

- Design For Test
- The flow to create the TetraMAX II ADV test vector data and fault coverage report as described in Section 4, *Use Cases for the TetraMAX II ADV Fault Coverage report*
- Access to the on-line Synopsys Technical Support Center and interpretation of the published list of safety-critical defects for the TetraMAX II ADV ATPG Tool

Installation and Supported platforms

The installation of TetraMAX II ADV ATPG must follow the guidelines described in the *Synopsys Installation Guide*.

The user is required to download three tar archive bundles from Synopsys SolvNET link:

https://solvnet.synopsys.com/DownloadCenter/dc/product.jsp

i. Synopsys Common Licensing

```
 ii. Synopsys Installer
iii. TetraMAX ATPG
```

Note: the GNU ‘tar’ command saves a collection of files into a single tape or disk archive, and can restore individual files from the archive.

The supported platforms for the safe use of the TetraMAX II ADV report are:

- Red Hat Enterprise Linux v6.6 and v6.7
- x86_64 SUSE Enterprise v11
- 64-bit x86 Intel and AMD processors (administrator rights are not required).

System requirements:

- System (OS and version): Refer to the TetraMAXIIATPG_Installation_guide_N-2017.09.pdf: www.synopsys.com/install
- Binary-compatible hardware platform or operating system. See http://www.synopsys.com/qsc for the latest on supported platforms, including required OS patches.
- The 64-bit (x86_64) Linux software is binary compatible with the Intel or AMD x86_64 processors.
- Available RAM: A minimum of 2 GB of physical RAM memory is recommended. For large designs, allow at least an additional 1 GB per 1 million gates in a design
- Free storage space: 740M is required for IDDQ, TetraMAX overly to synthesis (TX), and TetraMAX stand-alone (TXS). 366 MB is required for TetraMAX stand-alone (TXS).
- Free storage space: 740M is required for IDDQ, TetraMAX overly to synthesis (TX), and TetraMAX stand-alone (TXS). 366 MB is required for TetraMAX stand-alone (TXS).

Note: Users do not need to be concerned with inconsistency issues caused by mixing installations. Different versions of installations are automatically place in different directories during the installation procedure.

The following link provides users with guidance compute platform roadmaps

http://www.synopsys.com/Support/LI/SupportPlatform/Pages/PlatformsRoadmap.aspx

The following link provides platform notices:

http://www.synopsys.com/Support/Li/SupportPlatform/PlatformNotices/Pages/default.aspx

The following link explains the license key retrieval process:

https://solvnet.synopsys.com/smartkeys/smartkeys.cgi

Managing Known Safety-Critical Defects

Known defects are defects that are known to be present in the TetraMAX II ADV fault coverage report or identified by continued testing and user feedback. Defects are reported online in the Synopsys Technical Support Center. In an ideal world, such errors would be fixed immediately after detection.

However, in practice, fixing and testing errors, including qualification, takes place over an extended timeframe.

Users can request specials builds of the software during the development of safety-critical hardware design and test patterns used for the TetraMAX II ADV ATPG fault coverage report.

Not all defects are safety critical. The following are examples of issues that are *not* safety critical defects of the TetraMAX II ADV fault coverage report:

- Failure of the TetraMAX II ADV fault coverage report to compile a functionally correct program. In this case, a compilation fails, hangs, crashes, aborts or stops with an error and no output is produced. Although this may be a serious deficiency of TetraMAX II ADV, it cannot lead to safety-critical failures of the component.
- Failure of the TetraMAX II ADV fault coverage report to produce optimal coverage for a given program. Although this may make it harder for the user to achieve coverage-related goals, the fault coverage report still operates within its documented functionality.
- Creating a set of patterns with high fault coverage that fail in the simulation environment or on the tester is not safety critical. This is because the fault coverage report is for passing patterns. The fault coverage report is not applicable until the patterns are passing.
- Failure of the TetraMAX II ADV fault coverage report to give errors when features or functions are used that are not part of the documented functionality. Although this may lead to safety critical failures of the component, the defect is not caused by the tool.
- Failure to not mark a fault as detected or undetected, although a bug, is not safety critical as it is a pessimistic failure. Falsely marking a fault as detected when it is not detected is a safety critical failure.

The following is an example of a safety-critical defect:

- A fault is falsely detected. For example, a fault is dependent on the patterns and the patterns do not detect the fault.

The following are examples of safety usage issues, but are not tool safety-critical issues or defects.

- The user chooses to change the default value of the `-ax_credit` or `au_credit` options of the `set_faults` command that are used in the fault coverage calculation. This may be a case to consider safety-critical.
- The user should set the `-pt_credit` option of the `set_faults` command to “0“ (the default is 50). This option corresponds to potentially detected faults (PT).

The following actions are taken by Synopsys when a safety-critical defect in the TetraMAX II ADV ATPG tool is found:

- A STAR report is created on the Synopsys internal problem-tracking system; The STAR report marks the defect as *safety-critical.*
- The report contains a detailed description of the defect, such that users of the tool can identify the occurrence of the defect in their software.

- The report contains measures to be taken by the user of the tool to avoid the defect, for example by setting certain options in the compilation process.
- A summary listing of the safety-critical STAR reports can be accessed from the Synopsys customer relationship management (CRM) system
- Subscribed developers are notified of new defect listings by email notification (for fault coverage report problems the method described here is used for customer notification instead of the "Open and/or Closed STARs" Web mechanism that is used for processors).

Users of the TetraMAX II ADV fault coverage report are required to make sure that the developed software and the process of using TetraMAX II ADV avoid all known safety-critical defects.

TetraMAX II ADV Update Management

Synopsys may release new versions of TetraMAX II ADV ATPG at any time to extend functionality or fix defects in the tool. When a new version is available, a notification of a new version is posted in the Synopsys Technical Support Center.

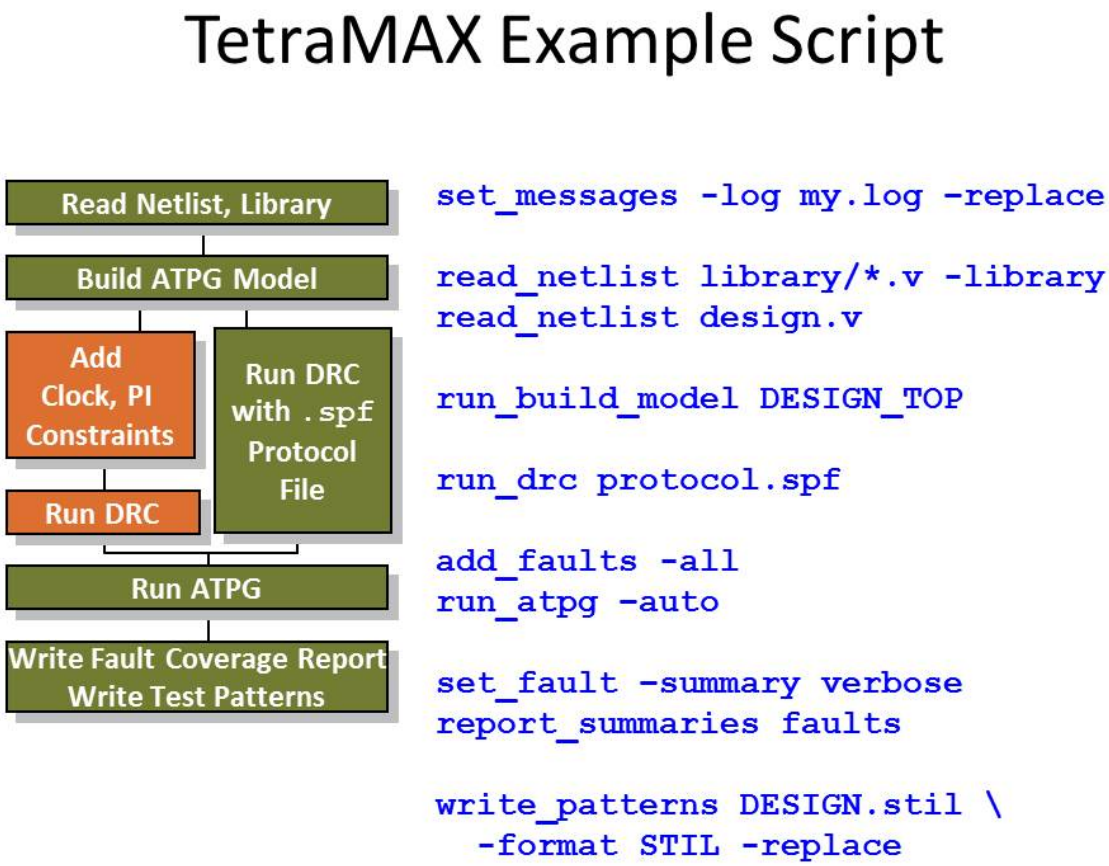
If a new version of the TetraMAX II ADV ATPG Tool is installed, users must check the following:

- The *Release Notes* of the software for any changes that are related to safety-critical aspects of using TetraMAX II ADV. It particularly important to check for changes in the Safety Manual.
- The Synopsys Technical Support Center for known safety-critical defects in the new version of TetraMAX.

4
Use Cases of the TetraMAX II ADV Fault Coverage Report and Test Vector Data

This section describes the safety-qualified use cases of the TetraMAX II ADV fault coverage report and test vector data. Only the use cases described in this section can be used for safety-critical applications. Other use cases require additional qualification.

Figure 1 shows an overview of the ATPG flow using the TetraMAX II ADV ATPG tool to generate a fault coverage report and test vector data.



Based on Figure 1 and the description above, the use cases described in the following sections are possible.

Use Case: Specifying Fault Lists

TetraMAX II ADV ATPG maintains a list of potential faults for a design. You can specify TetraMAX II ADV ATPG to use an existing fault list provided in a formatted ASCII file, create a fault list, or use only particular faults.

The following sections show several methods for specifying and creating fault lists:

- Selecting an Existing Fault List File
- Generating a Fault List Containing All Fault Sites
- Including Specific Faults in a Fault List
- Writing Faults to a File
- Example Fault Lists

Selecting an Existing Fault List File

To specify TetraMAX II ADV ATPG to use an existing fault list file, use the `read_faults` command, as shown in the following example:

```
TEST-T> read_faults my_faults.all
```

Generating a Fault List Containing All Fault Sites

To generate a fault list that includes all possible fault sites in the ATPG design model, specify the `add_faults` command, as shown in the following example:

```
TEST-T> add_faults –all
```

Including Specific Faults in a Fault List

These are examples of unsafe usage. While this may not produce the useful results in the fault coverage report, you must specify the `add_faults –all` command to ensure safety of the required faults.

You can exclude specific blocks, instances, gates, or pins from the fault list using any of the following methods:

- Specify objects to be excluded using the `add_nofaults` command and then execute the `add_faults -all` command, as shown in the following example:

  ```
  TEST-T> add_nofaults /sub_block_A/adder
  TEST-T> add_nofaults /io/demux/alu
  TEST-T> add_faults -all
  ```

- Remove faults based on fault locations in a fault list file specified by the `add_faults -all` command, as shown in the following example:

  ```
  TEST-T> add_faults -all
  TEST-T> read_faults fault_list_file -delete
  ```

- Remove faults using the remove_faults command after executing the add_faults -all command, as shown in the following example:

```
TEST-T> add_faults -all
TEST-T> remove_faults /sub_block_A/adder
TEST-T> remove_faults /io/demux/alu
```

- If you have a small number of faults, you can add them explicitly using the add_faults command:

```
TEST-T> remove_faults -all
TEST-T> add_faults /proc/io
TEST-T> add_faults /demux
TEST-T> add_faults /reg_bank/bank2/reg5/Q
```

Refer to page 19 for more details of options and usage on fault sites and fault format.

Writing Faults to a File

You can use the write_faults command to write a fault list to a file for analysis or to read back in for future ATPG sessions:

- Write a fault list containing only AU class faults, as shown in the following example:

```
TEST-T> write_faults faults.AU -class au -replace
```

- Write a fault list for all faults:

```
TEST-T> write_faults filename -all -replace
```

- Write only the undetectable blocked (UB) and undetectable redundant (UR) fault classes:

```
TEST-T> write_faults filename -class UB -class UR -replace
```

- Write only the faults down one hierarchical path:

```
TEST-T> write_faults filename /top/demux/core/mul8x8 -replace
```

- By default, the list of faults is either collapsed or uncollapsed as determined by the last set_faults -report command. The following command overrides the default by using the -collapsed option:

```
TEST-T> write_faults filename -all -replace -collapsed
```

Example Fault Lists

Example 1 shows a typical uncollapsed fault list. The equivalent faults always immediately follow the primary fault and are identified by two dashes (--) in the second column.

To replace the (--) below with the actual fault status; use set_faults –noequiv_code. The default is to use a double dash "--". Use of -noequiv_code suppresses the use of this character string and uses instead the fault class of the fault to which this fault is equivalent. When specifying an alternative equivalence code, use a two-character string. Also note that the current setting of the equivalence code can affect fault lists read using the read_faults command.

Example 1 Uncollapsed Fault List

```
sa0 NP /moby/bus/Logic0206/N01
sa0 -- /moby/bus/Logic0206/H01
sa0 -- /xyz_nwr
sa0 NP /moby/i278/N01
sa0 -- /moby/i278/H01
sa0 -- /moby/i337/N01
sa0 -- /moby/i337/H02
sa1 -- /moby/i337/H01
sa0 -- /moby/i222/N01
sa0 -- /moby/i222/H01
sa0 -- /moby/i222/H02
sa0 NP /moby/core/PER/PRT_1/POUTMUX_1/i411/N01
sa0 -- /moby/core/PER/PRT_1/POUTMUX_1/i411/H03
sa0 -- /moby/core/PER/PRT_1/POUTMUX_1/i411/H04
sa1 -- /moby/core/PER/PRT_1/POUTMUX_1/i411/H01
sa1 -- /moby/core/PER/PRT_1/POUTMUX_1/i411/H02
```

For comparison, Example 2 shows the same fault list written with the `-collapsed` option specified.

Example 2 Collapsed Fault List

```
sa0 NP /moby/bus/Logic0206/N01
sa0 NP /moby/i278/N01
sa0 NP /moby/core/PER/PRT_1/POUTMUX_1/i411/N01
```

Use Case: Working with Faults

Faults are assigned to classes corresponding to their current fault detection or detectability status. A two-character code is used to specify a fault class. Fault classes are hierarchically defined: low-level fault classes can be grouped together to form a high- level fault classes. Faults are only assigned the low-fault classes but the high level fault classes may be used for reporting. The fault class hierarchy for all fault classes is described in the following sections.

Fault Class Hierarchy

DT - Detected

- **DR** - Detected Robustly
- **DS** - Detected by Simulation
- **DI** - Detected by Implication
- **D2** - Detected clock fault with loadable nonscan cell faulty value of 0 and 1
- **TP** - Transition partially detected

PT - Possibly Detected

- **AP** - ATPG Untestable Possibly Detected

NP - Not analyzed, Possibly Detected

P0 - Detected clock fault and loadable nonscan cell faulty value is 0

P1 - Detected clock fault and loadable nonscan cell faulty value is 1

UD - Undetectable

UU - Undetectable Unused

UO - Undetectable Unobservable

UT - Undetectable Tied

UB - Undetectable Blocked

UR - Undetectable Redundant

AU - ATPG Untestable

AN - ATPG Untestable Not-Detected

AX - ATPG Untestable Timing Exceptions

ND - Not Detected

NC - Not Controlled

NO - Not Observed

DT (Detected) = DR + DS + DI + D2 + TP

The "detected" fault class is comprised of faults which have been identified as "hard" detected. A hard detection guarantees a detectable difference between the expected value and the fault effect value. The detection identification can be performed by simulation or implication analysis.

- DR (Detected Robustly)

  DR faults are hard detected by the fault simulator using weak non-robust (WNR), robust (ROB), or hazard-free robust (HFR) testing criteria to mark path delay faults. During ATPG, at least one pattern that caused the fault to be placed in this class is retained. This classification applies only to Path Delay ATPG.

- DS (Detected by Simulation)

  DS faults are hard detected by explicit simulation of patterns. During ATPG, at least one pattern that caused the fault to be placed in this class is retained.

- DI (Detected by Implication)

  DI faults are detected by an implication analysis. Faults which reside on pins which are in the scan chain path are declared detected due to the application of a scan chain functional test. Faults on ungated circuitry that connect to the shift clock line of scan cells are also considered detected by implication. Faults on ungated circuitry that connect to the set/reset lines of scan cells and cause the set/reset to be active are also considered detected by implication. No credit is given when the scan chain path is multiply sensitized. Faults are immediately placed into this fault class when they are added to the fault list.

- D2 (Detected in 2 Passes)

  A fault is classified as D2 if a clock fault is detected and the loadable nonscan cell faulty value is set to both 0 and 1. Note that the loadable nonscan cells feature must be active. For more information, see "Using Loadable Nonscan Cells in TetraMAX."

- TP (Transition Partially-Detected)

  TP faults are detected with a slack that exceeds the minimum slack by more than value specified by the `-max_delta_per_fault` option of the `set_delay` command. A TP fault can continue to be simulated with the intention of getting a better test for the fault.

PT (Possibly Detected) = AP + NP + P0 + P1

- AP (ATPG Untestable, Possibly Detected)

  AP faults are possibly detected faults. A faulty machine response will simulate an "X" rather than a 1 or 0. Analysis has determined that the fault cannot be detected with the current ATPG constraints and restrictions so the fault is removed from the active fault list and no further patterns for detecting this fault is attempted. At least one pattern which demonstrates that the presence of the fault will simulate as X is retained unless the `-pt_credit` option of the `set_faults` command has been used to indicate no credit should be given for PT faults. The default PT credit is 50%.

- NP (Not Analyzed, Possibly Detected)

  NP faults are identical to AP faults except that either analysis was not completed or could not prove that the fault would always simulate as an X. It is still possible that a different pattern could detect the fault and it's classification could become DS, until then it's classification remains NP and it remains in the active fault list. At least one pattern which demonstrates that the presence of the fault will simulate as X is retained unless the `-pt_credit` option of the `set_faults` command has been used to indicate no credit should be given for PT faults. The default PT credit is 50%.

- P0 (Requires 2 Passes, detected at 0)

  A fault is classified as P0 fault if a clock fault is detected and the loadable nonscan cell faulty value is set to 0. This classification applies only if the loadable nonscan cells feature is active. For more information, see "Using Loadable Nonscan Cells in TetraMAX."

- P1 (Requires 2 Passes, detected at 1)

  A clock fault is classified as P1 if a clock fault is detected and the loadable nonscan cell faulty value is set to 1. Note that the loadable nonscan cells feature must be active. For more information, see "Using Loadable Nonscan Cells in TetraMAX."

UD (Undetectable) = UU + UO + UT + UB + UR

The "undetectable" fault classes include faults which cannot be detected (either hard or possible) under any conditions. When calculating test coverage, these faults are not considered because they have no logical effect on the circuit behavior and cannot cause failures.

- UU (Undetectable Unused)

  UU faults are located on circuitry with no connectivity to an externally observable point. During the creation of the simulation model, the default is to remove this unused circuitry which results in these faults not existing. To expose these faults, you need to select the `-nodelete_unused_gate` option of the `set_build` command. Faults are immediately placed into this fault class when they are added to the fault list.

- UO (Undetectable Unobservable)

  UO faults are similar to UU faults, except they are located on unused gates *with* fanout (i.e., gates connected to other unused gates). Faults on unused gates *without* fanout are identified as UU faults.

- UT (Undetectable Tied)

  A UT fault is located on a pin that is tied to a value that is the same as the fault value. Faults are immediately placed into this fault class when they are added to the fault list.

- UB (Undetectable Blocked)

  A UB fault is located on circuitry that is blocked from propagating to an observable point due to tied logic. Faults are immediately placed into this fault class when they are added to the fault list.

- UR (Undetectable Redundant)

  UR faults are undetectable (using both hard detection and possible detection). Test generation fault analysis is performed when adding faults, during pattern-by-pattern test generation (as a result of the `run_atpg` command), and as a dedicated analysis of local or global redundancies (also as a result of `run_atpg`). When adding faults (using the `add_faults` command), an analysis is performed to identify and remove from the active list those faults which can easily be shown to be AU or UR. A simple form of ATPG is used during this analysis. Fault grading can never place a fault in this class.

AU (ATPG Untestable) = AN

"ATPG Untestable" faults include faults which can neither be hard detected under the current ATPG conditions nor proved redundant. When calculating test coverage, these faults are considered the same as untested faults because they have the potential to cause failures.

- AN (ATPG Untestable, Not Detected)

  AN faults have not been possibly detected and an analysis was performed to prove it cannot be detected under current ATPG conditions. The analysis also failed the redundancy check. Faults can immediately be placed in this class if they are inconsistent with the pre-calculated

constrained value information. Others can require test generation analysis. After they are placed in this class, they are removed from the active fault list and not given any further opportunity to become possible detected. Primary reasons for faults in this classification include:

- Fault untestable due to a constraint which is in effect.
- Fault requires sequential patterns for detection.
- Fault can only be possible detected.
- Fault requires using an unresolvable Z state for detection.

- AX (ATPG Untestable, Timing Exceptions)

  For each fault affected by SDC (Synopsys Design Constraints) timing exceptions, if all the gates in both the backward and forward logic cones are part of the same timing exception simulation path, then the fault is marked AU and is assigned an AX subclass. This analysis finds the effects of setup exceptions, so it does not impact exceptions that are applied only to hold time.

  To enable this type of analysis, use the `set_atpg -timing_exceptions_au_analysis` command. To configure separate reporting of these faults, use the `set_faults -summary verbose` command.

  Note that AX analysis is applied only for transition delay faults. The commands used for AX analysis are accepted for other fault models, but the results will not show any AX faults.

ND (Not Detected) = NC + NO

An ND fault indicates that test generation has not yet been able to create a pattern that controls or observes the fault. For these faults, it is possible that increasing the ATPG effort with the `set_atpg -abort_limit` command will result in these faults becoming some other classification.

- NC (Not Controlled)

  The NC fault class indicates that no pattern was yet found that would control the fault site to the state necessary for fault detection. This is the initial default class for all faults.

- NO (Not Observed)

  The NO fault class indicates that, although the fault site is controllable, that no pattern has yet been found to observe the fault so that credit can be given for detection.

Fault Sites

Internal Pins:

- Input and output pins of gates
- Pin name must be defined
- Must be at design cell level

External Pins:

PIs, POs, PIOs

Exceptions:

- Nofaulted pins
- Bidirectional pins on internalgates
- Removed circuitry
    - Removing unused gates - An unused gate is one which has no output connections to other gates, including black box or empty box gates. When this optimization method is enabled, unused gates are removed during the flattening process. It is possible for fault sites to be dropped as these gates are removed.
        - To enable (default): `set_build -delete_unused_gates`
        - To disable: `set_build -nodelete_unused_gates`
    - Remodeled circuitry (DLAT/DFF equivalent)
        - Equivalent DLAT/DFF optimizations. This optimization method identifies equivalent DLAT and DFF devices, and merges the equivalent functions into a single device. The equivalent_initialized_dlat_dff setting, if enabled, will assume the devices are initialized to their steady state values before determining if they can be merged into a single device. Two DLAT or two DFF devices are equivalent if they share common input connections, including all clock, set, reset, and data inputs. The outputs of the two devices may be identical or complementary to each other. This optimization method replaces one equivalent device with a BUF or INV connected to the output of the other equivalent device. During this process, fault sites might be dropped, in which case they are reported as B22 violations.
        - To enable (default): `set_build -merge equivalent_dlat_dff`
        - To enable: `set_build -merge equivalent_initialized_dlat_dff`
        - To disable: `set_build -merge noequivalent_dlat_dff`
    - Remodeled circuitry (DLAT-DLAT equivalent to DFF)
        - DLAT pairs as DFF optimizations. This optimization method finds each pair of D-latches that operate together as a D flip-flop, and replaces them with a DFF primitive. This occurs when two DLAT devices are connected serially from the Q output of one to the D input of the other, share common set, reset, and complementary clocks from the same source. When this optimization occurs, the two DLAT devices are replaced with a DFF primitive, possibly causing fault sites to be dropped. If any fault sites are dropped, they are reported as B22 warning violations.
        - B22 warnings for most users, no action is required. For the test purist you can count up the number of pins dropped due to B22 violations and use this information to manually adjust the final test coverage or fault coverage. For most users this number is negligible.

        - flipflop_cell_from_dlat – When enabled, merging master-slave latches into a flip-flop is limited to those latch pairs that are part of the same design-level cell. This option should be used by DFT Compiler and when necessary to avoid pin loss due to merging latches to flipflops.
        - To enable (default): `set_build -merge flipflop_from_dlat`
        - To enable: `set_build -merge flipflop_cell_from_dlat`
        - To disable: `set_build -merge noflipflop_from_dlat`
    - Switches(SW) as BUFs or BUFZs optimizations. These are always enabled; no user control.
        - During the flattening process, each SW primitive found that has its control gate held constantly on is replaced with a BUFZ device; or if the propagation of a Z value is not needed, it is replaced with a BUF device. These BUF/BUFZ device may be removed later by the BUF elimination method which also has no user control. This optimization can cause fault sites to be dropped. If this happens, the dropped faults are reported as B22 warning violations.
    - BUF elimination optimization is always enabled with no user control. During the flattening process, buffers are eliminated wherever this is possible without eliminating any fault sites.
    - DLATs as BUFs optimizations are always enabled with no user control. During the flattening process, for each DLAT primitive found that has its gate/clock input held on and its set and reset lines held off so that the latch is always transparent, the DLAT is replaced with a BUF device. This optimization can cause fault sites to be dropped. If this happens, the dropped faults are reported as B22 warning violations.

Fault Format

Fields:

- Type (sa0, sa1 for stuck & iddq) (str, stf for transition)
- Two-character fault class code
- pin pathname

Equivalence Code:

- "--" for fault class code indicates equivalence with preceding fault.
- Code user selectable (must be 2 chars).
- May be turned off. The fault status is given to the equivalent.

Command Usage:

`set_faults [-Equiv_code <name> | -NOEquiv_code]`

See pages 14 and 15: "Example 1 Uncollapsed Fault List"

Creating Fault Lists

Fault List Creation Options

- Add all faults
- Add/Delete individual faults
- Add/Delete faults on a selected instance
- Add/Delete faults on every instance of a module
- External fault list (Retain/Noretain fault status)
- Sample option

Initial Fault Status

- DI, UU, UT, UB, some AN, otherwise NC.

Fault List Restrictions

- Cannot be modified once ATPG has created patterns
- Can only be created in TEST command mode and are deleted when exiting TEST mode.
- Faults of a fanout-free network are placed adjacently; therefore if you do not like the order of the fault list you can write out faults and reorder them.

Command Usages

```
add_faults <instance_name | -Module <name> | -All | pin_pathname [-Stuck
 <0|1|01> | -Slow <R|F|RF>]>
remove_faults <instance... | -Module <name> | -All | pin_pathname [-Stuck
 <0|1|01> | -Slow <R|F|RF>] | -Retain_sample <d>>
read_faults <file_name> [-Retain_code] [-Add | -Delete]
```

Note that the fault list that is created by TetraMAX II ADV is in the order that TetraMAX II ADV decided to use. Do not expect the fault list to be in any particular order.

Nofaulting

Source of Nofaulting

- User definitions
- Library cells of complexity or bisting
- For safety reasons, not recommended to nofault but at the user discretion
- It is recommended that you use `add_faults –all` with no faults removed

Command Usages

```
add_nofaults <instance | pin [-Stuck <0|1|01>] | -Module <name>
   [-Stuck <0|1|01>] [pin_names]... >
remove_nofaults <inst | pin [-Stuck <0|1|01> | -All> | -Module <name>
   [-Stuck <0|1|01>] [pin_names]... >
report_faults <inst | pin [-Stuck <0|1|01>] | -All> [-Max <d>]
```

Fault Reporting

Summary Report

- Gives the number of faults in fault classes with test coverage
- Optional fault coverage and/or ATPG effectiveness
- Verbose/Noverbose and collapsed summary option

Selected Pin, Instance, Class, or All

- Faults displayed in standard format
- Max, verbose, and collapsed option

Profile

- Displays test coverage and fault counts for hierarchal instances to a selected depth with a min fault count.
- Verbose and collapsed option.

Command Usages

```
set_faults [-Report <Collapsed | Uncollapsed>][-ATpg_effectiveness |
   -NOAtpg_effectiveness][-Fault_coverage | -NOFault_coverage][-Summary
   <Verbose | NOVerbose>]
report_faults <instance | <-CLass fault_class>... | pin_pathname [-Stuck
   <0|1|01> | -Slow <R|F|RF>] |-SUmmary | -Profile | -All | -Unsuccessful>
   [-COllapsed | -UNCollapsed] [-Max <d>] [-Verbose][-Level <depth>
   <min_count>]
```

ATPG Untestable vs. Undetectable

ATPG Untestable

- Under some conditions, fault in circuit may behave different than fault free circuit for some pattern.
- Fault may only be possible detected.
- Fault may be detectable by removing constraints on ATPG.

- Counted the same as undetected faults for test coverage calculation.

Undetectable Faults

- Logically undetectable (not even X/Z detectable).
- Ignored for test coverage calculation.
- May cause differences in timing.

Redundant Fault Identification

- Is a subclass of undetectable faults
- If test generator exhausts search space without finding a pattern, an additional test generation is performed which ignores constraints bounded by X/Z producing circuitry. Redundancy is a function solely of circuit topology and is not affected by constraints.

  For redundancy identification, the test generator first attempts to produce a test for the fault. If it finds that a test is impossible, then the fault is at least AU and possibly UD (which includes redundancy as a sub-category). Next, a second attempt is made to generate a test with the constraints being ignored. If this also finds that a test is impossible, then the fault is considered to be undetectable. The fault could show up as ND rather than UR if the test generator aborted on the first try. Or the fault could be AU rather than UR if the test generator aborted on the second try. This is not a safety issue because redundant faults are faults that could have been claimed undetectable but are instead left as detected status. This can be a safety issue if the faults are left as AU.

Detected by Implication (DI) Classification

Scan Chain Detection

- Automatically set when fault is created.
- Includes fault in scan chain path and ungated faults in shift clock port.
- Does not include faults in multiple sensitized circuitry.
- Credit for the select line of the MUXes – take credit for stuck@0 or stuck@1 depending on the inactive state of the mux select line.
- The `set_faults –mult_fanout_di_faults` command preserves DI faults that can be better detected by transition ATPG.

Data-In Detection Credit

- Credit given to sa0 faults for single port nonscan DLATs without set/reset if data-in sao and sa1 detected.
- Credit given to sa0 and sa1 faults for single port nonscan DFFs without set/reset if data-in sao and sa1 detected.
- Analysis performed at end of Run ATPG and Run Fault_simulation.
- May be selected to not be performed.

Command Usage

```
set_atpg [-DI_analysis | -NODI_analysis]
```

Coverage Calculations

Test Coverage

TC = (DT+(NP+AP)*PT_credit)/(all_faults-UD-(AN*AU_credit)-(AX*AX_credit))

This calculation is used for official coverage.

Multiply by 100 to get percentage

Fault Coverage

FC = (DT+(NP+AP)*PT_credit)/all_faults

This calculation displays in the fault summary report when the `–pt_credit` option selected.

Multiply by 100 to get percentage

Coverage Options

- Collapsed/Uncollapsed
- Credit for possible detect faults
- Credit for ATPG untestable faults
- Avoid deleting unused circuitry during model build.

Command Usage

`set_faults` [-Report <Collapsed | Uncollapsed>] [-Pt_credit <d>]
[-AU_credit <d>] [-ATpg_effectiveness | -NOAtpg_effectiveness]
`[-Fault_coverage | -NOFault_coverage]`

ATPG effectiveness information is not related or useful to safety and therefore is not discussed here.

Use Case: Working with Fault Lists

TetraMAX II ADV ATPG maintains a list of potential faults for a design, along with the categorization of each fault. A fault list is contained in an ASCII file that can be read and written using the `read_faults` and `write_faults` commands.

The following topics describe how to work with fault lists.

Using Fault Lists

You can use fault list files to manipulate your fault list in the following ways:

- Add faults from a file, while ignoring any fault classes specified
- Add faults from a file, while retaining any fault classes specified
- Delete faults specified by a fault list file
- Add nofaults (sites where no faults are to be placed) specified by a fault list file

To access fault list files, you use the `read_faults` and `read_nofaults` commands, which have the following syntax:

```
read_faults file_name [-retain_code] [-add | -delete]
read_nofaults <file_name>
```

The `-retain_code` option retains the fault class code but behaves differently depending on whether the faults in the file are new or replacements for existing faults.

New Faults

For any new fault locations encountered in the input file, if the fault code is DS or DI, the new fault is added to the fault list as DS or DI, respectively. For all other fault codes, TetraMAX II ADV ATPG determines whether the fault location can be classified as UU, UT, UB, DI, or AN. If the fault location is determined to be one of these fault classes, the new fault is added to the fault list and the fault code is changed to the determined fault class. If the fault location was not found to be one of these special classes, the new fault is added with the fault code as specified in the input file.

Existing Faults

For any fault locations provided in the input file that are already in the internal fault list, the fault code from the input file replaces the fault code in the internal fault list. TetraMAX II ADV ATPG does not perform any additional analysis.

Collapsed and Uncollapsed Fault Lists

To improve performance, most ATPG tools collapse all equivalent faults and process only the collapsed set. For example, the stuck-at faults on the input pin of a BUF device are considered equivalent to the stuck-at faults on the output pin of the same device. The collapsed fault list contains only the faults at one of these pins, called the primary fault site. The other pin is then considered the equivalent fault site. For a given list of equivalent fault sites, the one chosen to be the primary fault site is purely random and not predictable.

You can generate a fault summary report using either the collapsed or uncollapsed list using the `-report` option of the `set_faults` command.

Example: Collapsed and Uncollapsed Fault Summary Reports

```
TEST-T> set_faults -report collapsed
TEST-T> report_faults -summary
```

```
       Collapsed Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT     120665
Possibly detected                PT       3749
Undetectable                     UD       1374
ATPG untestable                  AU       6957
Not detected                     ND       6452
-----------------------------------------------
total faults                             139197
test coverage                             88.91%
-----------------------------------------------

TEST-T> set_faults -report uncollapsed
TEST-T> report_faults -summary

       Uncollapsed Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT     144415
Possibly detected                PT       4003
Undetectable                     UD       1516
ATPG untestable                  AU       8961
Not detected                     ND       7607
-----------------------------------------------
total faults                             166502
test coverage                             88.74%
-----------------------------------------------
```

Random Fault Sampling

Random fault sampling can be a safety issue. Using a sample of faults rather than all possible faults can reduce the total runtime for large designs. You can create a random sample of faults using the `-retain_sample` *`percentage`* option of the `remove_faults` command.

The *`percentage`* argument of the `-retain_sample` option indicates a probability of retaining each individual fault and does not indicate an exact percentage of all faults to be retained. For example, if percentage = 40, for a fault population of 10,000, TetraMAX II ADV ATPG does not retain exactly 4,000 faults. Instead, it processes each fault in the fault list and retains or discards each fault according to the specified probability. For large fault populations, the exact percentage of faults kept is close to 40 percent, but for smaller fault populations, the actual percentage might be a little bit more or less than what is requested, because of the granularity of the sample.

For example, the following sequence requests retaining a 25 percent sample of faults in `block_A` and `block_B` and a 50 percent sample of faults in `block_C`.

```
TEST-T> add_faults /my_asic/block_A
TEST-T> add_faults /my_asic/block_B
TEST-T> remove_faults -retain_sample 50
TEST-T> add_faults /my_asic/block_C
TEST-T> remove_faults -retain_sample 50
```

You can combine the `-retain_sample` option with the capabilities of defining faults and nofaults from a fault list file for flexibility in selecting fault placement.

Example 1 is a fault list 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 "`--`".

Example: Typical Fault List Showing Equivalent Faults

```
// entire lines can be commented
sa0   DI   /CLK ; comments here
sa1   DI   /CLK
sa1   DI   /RSTB
sa0   DS   /RSTB
sa1   AN   /i22/mux2/A
sa1   UT   /i22/reg2/lat1/SB
sa0   UR   /i22/mux0/MUX2_UDP_1/A
sa0   --   /i22/mux0/A # equivalent to UR fault above it
sa0   DS   /i22/reg1/MX1/D
sa0   --   /i22/mux1/X
sa0   --   /i22/mux1/MUX2_UDP_1/Q
sa1   DI   /i22/reg2/r/CK
sa0   DI   /i22/reg2/r/CK
sa1   DI   /i22/reg2/r/RB
sa0   AP   /i22/out0/EN
sa1   AP   /i22/out0/EN
```

TetraMAX II ADV ATPG ignores blank lines and lines that start with a double slash and a space (`// `) contained within the fault list.

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.

Use Case: Analyzing ATPG Output.

You can analyze ATPG pattern generation output from the run_atpg command. This output includes the following formats using the Verbose Format with Merge with and without -auto_compression

The `set_atpg –verbose` option enabled gives extra messages to be displayed during the pattern merge operation. The default is `-noverbose`.

This message is in the form “N faults were unsuccessfully detected. (M136)”. During the ATPG process, test generation was successfully performed for a fault, but fault simulation failed to identify the fault as detected. This might be due to a user interrupt, pattern rejection due to a failure to satisfy constraints, pattern rejection due to contention, sequential effects, or a program error. You can display these faults using the `-unsuccessful` option of the `report_faults` command.

Use Case: Reviewing Test Coverage

You can view the results of the test coverage and the number of patterns generated using the `report_summaries` command.

The following example shows how to generate a fault summary report using the `report_summaries` command:

```
TEST-T> report_summaries
```

An example output report showing the fault counts and the test coverage obtained by using the uncollapsed fault list is shown in the following example:

Example: Uncollapsed Fault Summary Report

```
TEST-T> report_summaries
Uncollapsed Fault Summary Report
-----------------------------------------------
fault class                    code #faults
------------------------------ ---- ---------
Detected                        DT    83348
Possibly detected               PT      324
Undetectable                    UD     1071
ATPG untestable                 AU     3453
Not detected                    ND      212
-----------------------------------------------
total faults                           88408
test coverage                          95.62%
-----------------------------------------------
          Pattern Summary Report
-----------------------------------------------
#internal patterns                      1636
-----------------------------------------------
```

The following example shows the same report as the previous example with collapsed fault reporting. Notice that there are fewer total faults, and fewer individual fault categories.

Example: Collapsed Fault Summary Report

```
TEST-T> set_faults -report collapsed
```

```
TEST-T> report_summaries
Collapsed Fault Summary Report
-----------------------------------------------
fault class                    code #faults
------------------------------ ---- ---------
Detected                        DT    50993
Possibly detected               PT      214
Undetectable                    UD     1035
ATPG untestable                 AU     2370
Not detected                    ND      122
-----------------------------------------------
total faults                           54734
test coverage                          95.16%
-----------------------------------------------
          Pattern Summary Report
-----------------------------------------------
#internal patterns                      1636
-----------------------------------------------
```

Use Case: Fault Summary Reports

The following sections describe the various types of summary reports:

- Fault Summary Report Examples
- Test Coverage
- Fault Coverage

Fault Summary Report Examples

By default, TetraMAX II ADV ATPG displays fault summary reports using the five categories of fault classes, as shown in the following example.

Example: Fault Summary Report: Test Coverage

```
-----------------------------------------------
Uncollapsed Fault Summary Report
-----------------------------------------------
fault class                    code   #faults
------------------------------  ----  ---------
Detected                        DT     144361
Possibly detected               PT       4102
Undetectable                    UD       1516
ATPG untestable                 AU       8828
Not detected                    ND       7695
-----------------------------------------------
total faults                           166502
test coverage                           88.74%
-----------------------------------------------
```

Note that possibly detected faults should not be reported in your report if you consider these to be a safety implication and turn of `pt_credit`.

For a detailed breakdown of fault classes, use the `-summary verbose` option of the `set_faults` command:

```
TEST-T> set_faults -summary verbose
```

The following example shows a verbose fault summary report, which includes the fault classes in addition to the fault categories.

Example: Verbose Fault Summary Report

```
-----------------------------------------------
Uncollapsed Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT     144415
  detected_by_simulation         DS    (117083)
  detected_by_implication        DI     (27332)
Possibly detected                PT       4003
  atpg_untestable-pos_detected   AP        (403)
  not_analyzed-pos_detected      NP       (3600)
Undetectable                     UD       1516
  undetectable-unused            UU          (4)
  undetectable-tied              UT        (565)
  undetectable-blocked           UB        (469)
  undetectable-redundant         UR        (478)
ATPG untestable                  AU       8961
  atpg_untestable-not_detected   AN      (8961)
Not detected                     ND       7607
  not-controlled                 NC        (503)
  not-observed                   NO      (7104)
-----------------------------------------------
total faults                              166502
test coverage                              88.74%
-----------------------------------------------
```

Note that possibly detected faults should not be reported in your report if you consider these to be a safety implication and turn of `pt_credit` made up of `au_credit` and `ax_credit`.

The test coverage figure at the bottom of the report provides a quantitative measure of the test pattern quality. You can optionally choose to see a report of the fault coverage or ATPG effectiveness instead.

The possible quality measures are defined as follows:

- Test coverage = detected faults / detectable faults
- Fault coverage = detected faults / all faults

Test Coverage

Test coverage gives the most meaningful measure of test pattern quality and is the default coverage reported in the fault summary report. Test coverage is defined as the percentage of detected faults out of detectable faults, as follows:

$$\text{Test Coverage} = \frac{(DT+(NP+AP)*PT\_credit)}{(all\_faults-UD-(AN*AU\_credit)-(AX*AX\_credit))} X\ 100$$

PT_credit is initially 50 percent, AX_credit is initially 0, and AU_credit is initially 0. You can change the settings for PT_credit or AU_credit and AX_credit using the `set_faults` command.

PT fault credit should be set to “0” to be safety critical. When this value is set to zero then the equation above is

$$\text{Test Coverage} = \frac{DT}{All\_Faults - UD} X\ 100$$

By default, the fault summary report shows the test coverage, as in Example 1 and Example 2.

Use Case: Fault Coverage

Fault coverage is defined as the percentage of detected faults out of all faults, as follows:

$$\text{Fault Coverage} = \frac{DT + (PT \times PT\_credit)}{All\_Faults} \times 100$$

$$\text{Fault Coverage} = \frac{DT}{All\_Faults} X\ 100$$

Fault coverage gives no credit for undetectable faults. By default, PT_credit is 50 percent but is “0” for safety critical mode.

To display fault coverage in addition to test coverage with the fault summary report, use the `-fault_coverage` option of the `set_faults` command.

The following example shows a fault summary report that includes the fault coverage.

Example: Fault Summary Report: Fault Coverage

```
TEST-T> set_faults -fault_coverage
TEST-T> report_faults -summary
-----------------------------------------------
Uncollapsed Fault Summary Report
```

```
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT     144361
Possibly detected                PT       4102
Undetectable                     UD       1516
ATPG untestable                  AU       8828
Not detected                     ND       7695
-----------------------------------------------
total faults                             166502
test coverage                             88.74%
fault coverage                            87.93%
-----------------------------------------------
```

Note that possibly detected faults should not be reported in your report if you consider these to be a safety implication and turn of `pt_credit` made up of `au_credit` and `ax_credit`.

Use Case: Fault Simulation

Fault simulation is an inherently slow standalone methodology. TetraMAX II ADV fault simulation is optimized for patterns generated by the TetraMAX II ADV ATPG capability.

Fault simulation coverage results depend on the input functional vectors to detect faults. Fault simulation of functional vectors requires sequential simulation which is much slower than the patterns generated by TetraMAX II ADV ATPG.

You should perform a good machine simulation using the functional patterns before running a fault simulation, to compare the TetraMAX II ADV simulation responses to the expected responses found in the patterns. If the good machine simulation reports mismatches; then any fault simulation faults will be invalid.

After performing a good machine simulation to verify that the functional patterns and expected data agree, you can perform a fault grading or fault simulation of those patterns. Performing fault simulation includes setting up the fault simulator, running the fault simulator, and reviewing the results.

```
run_build top_mod_name
run_drc atpg_related.spf
add_faults -all
run_atpg
set_fault -summary verbose
report_summaries faults
write_patterns <pat.stil> -format stil –replace   // save patterns
write_fault <pass1.flt> -all -uncollapsed –replace   // save fault list
drc –force   //Return to DRC Mode and Remove Settings
remove_pi_constraints -all
remove_clocks -all
set drc -nofile
```

```
test  //Enter Test mode
set_pattern –external -strobe ... <functional_patterns.evcd>  //set the patterns
read_faults <pass1.flt> -force_retain   // set the fault list
run_sim –sequential                //run simulation –> good machine
run_fault_sim –sequential          //run fault grading_simulation
set_fault -summary verbose
report_summaries faults   //review test coverage
```

There are several `run_fault_sim` and `run_simulation` command options that are not supported by multicore simulation (refer to the TetraMAX II ADV User Guide for details).

Use Case: Stuck-At Coverage From Transition Patterns

This is a more time consuming flow. The preferred way is to use the persistent fault model flow. This flow may be needed as a workaround for cases only where we find bugs. This is a safe flow to use.

```
set_faults -model transition       // TetraMAX Delay ATPG
set_delay -launch system_clock
add_faults -all
run_atpg -auto
write_faults ...
write_patterns -format binary ...  //Transition Test Patterns

remove_faults -all
set_faults -model stuck
add_faults -all

set_patterns -external ...         //TetraMAX Stuck-at Faultsim
run_fault_sim

set_patterns –internal             //TetraMAX Incremental Stuck-at ATPG
run_atpg -auto
write_patterns ...                 //Stuck-at Test patterns
```

Use Case: Transition Coverage From Stuck-at Patterns

This is a time consuming flow. The “transition coverage from stuck-at patterns” flow is not a potential workaround for persistent fault model bugs, because persistent fault models would give zero transition fault credit in this case. This is an independent flow, though it has some drawbacks of its own:

1. The transition coverage from fault simulation is likely to be very low, since transition detection requirements are more stringent than stuck-at detection requirements.

2. The transition detections from LOS using the stuck-at timing will be at relaxed timing, and this will also apply to ATPG detections in this flow. Basically, this flow uses the incorrect order of ATPG runs. It should only be used if stuck-at patterns have already been generated and must be re-used. This is a safe flow to use.

```
set_faults -model stuck     //stuck-at ATPG
add_faults -all
run_atpg -auto
write_faults ...
write_patterns -format binary ... //setup for SSA Test Patterns

remove_faults -all
set_faults -model transition
set_delay -launch last_shift
add_faults -all

set_patterns -external ...
run_fault_sim        //Transition Faultsim

set_patterns -internal
run_atpg –auto              //Incremental Transition ATPG
write_patterns ...   //Transition Test patterns
```

Use Case: Persistent Fault Model Flow

The persistent fault model flow automatically maintains the persistent fault lists for all supported fault models at the same time. This flow automatically preserves the internal pattern set for fault grading. This flow also automatically does fault crediting to stuck-at fault from transition fault list without fault simulation.

When moving to a different fault model, the fault list for the active fault model is automatically saved in a cache as an inactive fault model. When going back to an inactive fault model, the saved fault list is automatically restored and when path_delay faults are preserved, the delay paths are also preserved. When going back to DRC mode, you cannot interact with the fault list in DRC mode, but

the fault list will still be available when you return to TEST mode. AU faults are automatically reset for any fault list that was in a cache during DRC process.

You can set a different fault model even if you have faults in active fault model.

```
set_faults -persistent_fault_models
set_faults -model transition
add_faults -all
run_atpg -auto
# (Transition faults exist)
set_faults -model stuck

# Without  -persistent_fault_models flow, you would see an error.
# Error: Fault model may not be changed when faults are in current fault list.
(M106)
```

Internal pattern set (generated patterns in general) is preserved even if the fault model is made changed. You can run fault simulation with an alternate fault model

```
set_faults -persistent_fault_models
set_faults -model stuck
add_faults -all
set_faults -model transition
add_faults -all
run_atpg -auto
set_faults -model stuck
# (Transition fault pattern set is still preserved as internal patterns)
rum_fault_sim
```

The fault list is automatically saved in a cache as an inactive fault model when the fault model is changed or when you go back to DRC mode.

```
set_faults -persistent_fault_models
add_pi_constraint 1 mem_bypass
run_drc compression.spf
set_faults -model stuck
add_faults -all
set_faults -model transition
# (Stuck-at fault list is saved)
add_faults -all
run_atpg -auto #for transition
drc -f
# (Transition fault list is saved)
remove_pi_constraints -all
add_pi_constraint 0 mem_bypass
run_drc compression.spf
```

The fault list is restored automatically from a cache as an active fault model when exiting DRC mode or when fault model is changed back

```
set_faults -persistent_fault_models
add_pi_const 1 mem_bypass
run_drc compression.spf
```

```
set_faults -model stuck
add_faults -all
set_faults -model transition
# (Stuck-at fault list is saved)
add_faults -all
run_atpg -auto
drc -f
# (Transition fault list is saved)
remove_pi_constraints -all
add_pi_const 0 mem_bypass
run_drc compression.spf
# (Transition fault list is restored)
run_atpg -auto
set_faults -model stuck
# (Transition fault list is saved)
# (Stuck-at fault list is restored)
update_faults -direct_credit
run_fault_sim
```

You must be careful when adding faults. Even when the `-persistent_fault_models` option is enabled, you can't add faults when internal pattern set is present, which is an unchanged behavior. You should create necessary fault list files before using the `run_atpg` command.

```
set_faults -persistent_fault_models
set_drc compression.spf
run_drc
set_faults -model stuck
add_faults -all
set_faults -model transition
add_faults -all
run_atpg -auto
set_faults -model stuck
# (You can’t add faults here because internal pattern set is present)
update_faults -direct_credit
run_fault_sim
```

If the `-persistent_fault_models` option is not enabled, you can use the `-external` option do direct crediting to stuck-at faults if you have a transition fault list

```
run_drc compression.spf
set_faults -model stuck
add_faults -all
update_faults -direct_credit -external <tran_flt.list>
run_atpg –auto
```

The following example shows the commands used in a typical persistent fault model flow:

```
read_netlist des_unit
run_build_model des_unit
set_delay -launch system_clock
#
# Activate persistent fault model feature
```

```
#
set_faults -persistent_fault_models
#
# Model=Transition memory bypass=No OCC=Yes
#
add_pi constraints 0 memory_bypass
run_drc des_unit.spf -patternexec comp
set_faults -model stuck
add_faults -all
set_fault -model transition
add_faults -all
run_atpg -auto
write_patterns trans_bp0_occ1.bin -format binary
set_faults -model stuck
#
# Credit transition detections to stuck-at faults.
#
update_faults -direct_credit
#
# Optional step to increase fault coverage
# run_fault_sim
#
drc -force
remove_pi_constraints -all
remove_clocks -all
#
# Model=Stuck-at memory bypass=Yes OCC=No
#
add_pi_constraints 1 memory_bypass
run_drc des_unit.spf -patternexec comp_occ_bypass
set_fault -model stuck
run_atpg -auto
write_patters stuck_bp1_occ0.bin -format binary
drc -force
remove_pi_constraints -all
remove_clocks -all
#
# Model=Stuck-at memory bypass=No OCC=No
#
add_pi_constraints 0 memory_bypass_mode
run_drc des_unit.spf -patternexec comp_occ_bypass
set_faults -model stuck
run_atpg -auto
write_patterns stuck_bp0_occ0.bin -format binary
```

5 Limitations

This section describes all known limitations and possible workarounds in TetraMAX II ADV ATPG.

LIM-01: Read New Behavior Of Synchronous RAMS Supported For ATPG Only

TetraMAX II ADV ATPG now supports read/write contention behavior of `read new` for RAMS with edge triggered read and write ports. These allows the modeling of a RAM with clocked read and write ports to have the write data appear on the read port outputs during the write cycle (write-through). However, this behavior is supported for ATPG pattern generation only and not for sequential fault simulation (`run_simulation -sequential`, `run_fault_sim -sequential`). This means that attempting to fault simulate functional patterns which rely on the read new (write through) behavior will give misleading results.

LIM-02: Not All Verilog Structural Syntax Supported

TetraMAX II ADV ATPG does not support every variation and nuance of the Verilog hardware description language. Rather, the tool supports the syntax most commonly produced by the Synopsys synthesis product flows. If your design methodology includes hand-generated Verilog, you can encounter syntax that gets rejected as a "parse error". Contact your Synopsys support center if you encounter unsupported Verilog variations.

LIM-03: No VHDL Library Support

TetraMAX II ADV ATPG does not support reading libraries in VHDL format using either generics or VITAL syntax. However, it does support reading designs in VHDL format as long as the description is structural and simple signal types of either *std_logic* or *std_ulogic* are used.

LIM-04: Reading Encrypted Verilog Not Supported

TetraMAX II ADV ATPG cannot read encrypted Verilog modules. So when it encounters the protected directive it issues an N1 violation.

Appendix A
Qualified Options and Option Combinations

Qualified and Non-Qualified Options

All documented commands in TetraMAX Online Help that are not listed below are qualified for safety usage. Undocumented commands or non-production fault classes are not qualified and are not listed below, and they may have safety implications.

Table 1: Options Qualified for Safety-Critical Use with or without implications

Option	Description
`set_faults –au_credit`	Leave this at default value “0”
`set_faults –ax_credit`	Leave this at default value “0”
`set_faults –pt_credit`	Change the default from “50” to “0”
`set_faults –model stuck`	
`set_faults -model iddq`	
`set_faults -model transition`	
`set_faults -model path_delay`	
`set_faults -model iddq_bridging`	
`set_faults -model hold_time`	
`set_faults -model bridging`	
`set_faults -model dynamic_bridging`	
`add_faults –all`	This is preferred and recommended command
`set_faults [-Report <Collapsed | Uncollapsed>] [-Fault_coverage | -NOFault_coverage][-Summary <Verbose | NOVerbose>]`	Use with caution
`add_nofaults`	Use with caution
`read_faults`	Use with caution
`read_nofaults`	Use with caution
`remove_faults`	Use with caution
`report_faults`	The `report_faults –summary –all` command is the preferred and recommend usage

report_summaries faults	This is preferred and recommend command
report_summaries [-launch <launch_clock>] [-capture <capture_clock>]	This is preferred and recommend command usage to report transition faults
run_build_model –remove_pio_build	
run_simulation –sequential_update	Must run fault simulation afterwards
run_simulation -update	Must run fault simulation afterwards
run_simulation –resolve_differences	Must run fault simulation afterwards
run_simulation –override_differences	Must run fault simulation afterwards
run_simulation –disable_masking	Must run fault simulation afterwards
set_atpg [–di_analysis | -nodi_analysis]	Affects how the faults are reported
set_atpg [-optimize_bridge_strengths | -nooptimize_bridge_strengths]	Must specify the run_fault_simulation –strong_bridge command afterwards
set_atpg [-timing_exceptions_au_analysis | -notiming_exceptions_au_analysis]	Quality implications: When ax_credit is equal to 1 or greater you get wrong results. When ax_credit equals zero it is safety okay.
set_build [-delete_unused_gates | -nodelete_unused_gates]	These affect the fault sites
set_build [-merge <equivalent_dlat_dff | equivalent_initialized_dlat_dff | noequivalent_dlat_dff | flipflop_from_dlat | flipflop_cell_from_dlat | noflipflop_from_dlat | dlat_from_flipflop | odlat_from_flipflop| wire_to_buffer | nowire_to_buffer | mux_from_gates | muxpins_from_gates | muxx_from_gates | nomux_from_gates | xor_from_gates | xorpins_from_gates | noxor_from_gates| cascaded_gates_with_pin_loss | nocascaded_gates_with_pin_loss | tied_gates_with_pin_loss | notied_gates_with_pin_loss | global_tie_propagate | noglobal_tie_propagate | bus_keepers | nobus_keepers | feedback_paths | nofeedback_paths >]	These affect the fault sites
set_build [-black_box <module_name> | -empty_box <module_name> | -design_box <module_name> | -portfault_box <module_name> | -nobox <module_name> | -reset_boxes]	These affect the fault sites

set_build [-fault_boundary <lowest | hierarchical>]	Strongly recommend to fix library cell rather than override default
set_build [-instance_modify <<instance_name> <gate_type>> | -noinstance_modify <instance_name> -noinstance_modify_all]	Used in OCC flow to set to tiex ; has implications if this does not match the circuit Verilog simulation behavior
set_build [-net_connections_change_netlist | -nonet_connections_change_netlist]	
set_delay [-max_delta_per_fault <value>]	
set_delay [-slackdata_for_faultsim | -noslackdata_for_faultsim]	
set_delay [-sdql_coefficients <{a b c d e}>] [-sdql_histogram <file name>] [-sdql_power_function] [-sdql_exponential_function]	
set_delay [-max_tmgn <value>] [-multicycle_length <d>]	
set_faults [-ATpg_effectiveness | -NOAtpg_effectiveness]	
set_faults –fault_coverage	Recommend using
set_faults [-report <collapsed | uncollapsed>]	Recommend using
set_faults [-pt_credit <d>] [-au_credit <d>] [-ax_credit <d>]	
set_faults [-equiv_code <code_name> | -noequiv_code]	
set_faults [-bridge_inputs | -nobridge_inputs]	
set_faults [-multi_fanout_di_faults | -nomulti_fanout_di_faults]	
set_faults [-test_coverage_include <undetectable_fault_classes>] [-test_coverage_exclude <undetectable_fault_classes>]	
set_iddq [-toggle | -notoggle]	
set_iddq[-bridge_equivalence | -nobridge_equivalence]	
set_netlist [-enable_portfaults | -noenable_portfaults]	
set_netlist [-suppress_faults | -nosuppress_faults]	

`set_patterns [-resolve_differences <filename>]`	Changes the fault coverage and therefore must rerun fault simulation if used
`update_faults [-direct_credit][-external <file_name>]`	When used together this is dependent on where the external file comes from. When `–direct_credit` is used alone that is safety ok.
`update_streaming_patterns [-load_scan_in <scan in pins to be made constant>]`	Must rerun fault simulation
`update_streaming_patterns [-remove <delete pattern number>] [-insert <insert pattern number>]`	Must rerun fault simulation
`write_faults <filename> -summary | -all`	Recommended command and similar command to `report_faults`

Table 2: Options Not Qualified for Safety-Critical Use

Options	Description
`add_net_connections`	
`analyze_faults`	
`remove_net_connections`	

Appendix B
Qualified Versions of Tools

Tool	Version
TetraMAX II ADV ATPG	N-2017.09